FACULTY OF SCIENCES

Automatic Web Form Generation based on Authorisation Specifications

Femke Brückmann
Student number: 01505066

Supervisors: Dr. Anastasia Dimou, Prof. dr. ir. Ruben Verborgh

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in de informatica

Academic year 2020-2021

GHENT UNIVERSITY

ACKNOWLEDGEMENTS

First of all, I would like to thank my promotors, dr. Anastasia Dimou and dr. Ir. Ruben Verborgh, as well as my counsellors Ben De Meester and Sven Lieber. They gave me the chance and freedom to research these interesting topics and guided me through every step of the process.

Secondly, I would like to thank Ms Helen Chuter for taking the time to answer some questions concerning a real-life application of the subject.

Of course, a huge thanks to all of my parents for giving me every possible opportunity to pursue this education in the best circumstances I could have ever wished. They have always supported me to the best of their abilities.

Next, I want to praise Sander for always being a source of encouragement and necessary distractions, continuously giving me the energy to keep going and tirelessly allowing me to voice my thoughts every single day. His support throughout this past year has been unmatched.

Furthermore, a big thanks to Pieter and Karel for spending their precious time reading and re-reading my thesis down to the last detail, offering suggestions and reassuring me countless times along the way.

Last but definitely not least, I would like to thank my friends Bert and Pieter for always believing in me throughout our years at Ghent University. I cannot praise them enough for celebrating the highs and pulling me through the lows. Without them, I would have never gotten this far.

Thank you all for making this possible.

Femke, 2021.

SUMMARY

Since the introduction of the General Data Protection Regulation (GDPR), businesses and organisations must comply with the extensive regulations associated with the processing of people's personal data. Nowadays, an enormous amount of this personal data is available on the Web in various forms. As this data is commonly ambiguously described, the Semantic Web aims to make data unambiguous and machine-readable. This Linked Data can be stored in Knowledge Graphs (KG). Forms can be generated on top of these KGs to enable easier enrichment of the data, increasing the machine-readability of data on the Web.

Various publications approach the automatic generation of forms on top of KGs. Additionally, there are numerous approaches to expressing the concepts and rules of the GDPR in the Semantic Web in the form of ontologies and models. However, access control in the context of consent in the GDPR has not yet been combined with form generation.

This thesis proposes an intermediate model that aims to act as an abstraction layer between expressed consent and a Knowledge Graph. The model applies a data subject's explicit and specific consent, in the form of an authorisation specification – expressed in the Privacy Preference Ontology (PPO) – to Shape Constraint Language (SHACL) shapes and forms generated on top of these shapes.

The application of this model has been evaluated by carrying out experiments to measure how well this model applies the expressed authorisation specification and to what degree Privacy Preference Ontology (PPO) and Social Semantic SPARQL Security for Access Control (S4AC) can be expressed using this model's features.

Applying eight combinations using Access Control List (ACL) access types resulted in 32 values expressed in Create, Read, Update, Delete (CRUD) access. The model applied 100% of these values correctly in the SHACL shape, and 93.75% was expressed correctly in the generated form. The model can be applied to forms without nesting and SHACL shapes with up to at least three levels of nesting, where properties can inherit the granted access from their parent if the data subject explicitly allows this behaviour. Consent expressed using PPO can be fully represented by the model, while 44% of S4AC's features can be mapped to the model.

SAMENVATTING

Sinds de introductie van de Algemene Verordening Gegevensbescherming (GDPR) moeten bedrijven en organisaties voldoen aan de uitgebreide voorschriften rond de verwerking van persoonsgegevens van mensen. Vandaag zijn enorm veel gegevens in verschillende vormen beschikbaar op het Web. Aangezien deze data meestal dubbelzinnig beschreven wordt, is het doel van het Semantisch Web om deze data ondubbelzinnig en machineleesbaar te maken. Deze zogenaamde Linked Data kan opgeslagen worden in Knowledge Graphs (KG). Formulieren kunnen dan gegenereerd worden voor deze KG's om het verrijken van de data te vereenvoudigen, wat de machineleesbaarheid vergroot.

Verschillende publicaties onderzochten het automatisch genereren van formulieren voor KG's. Daarnaast werden er talrijke manieren onderzocht om GDPR regels en concepten uit te drukken in het Semantisch Web in de vorm van ontologieën en modellen. Toegangscontrole en toestemming in de GDPR werden echter nog niet gecombineerd met het genereren van webformulieren.

Deze thesis stelt een intermediair model voor dat als abstractielaag moet fungeren tussen uitgedrukte toestemming en een KG. Het model past de expliciete en specifieke toestemming van de betrokkene – uitgedrukt in de vorm van een autorisatiespecificatie in de Privacy Preference Ontology (PPO) – toe op Shape Constraint Language (SHACL) shapes en formulieren gegenereerd op basis van deze shapes.

Het toepassen van dit model werd geëvalueerd door het uitvoeren van experimenten die meten hoe goed dit model de autorisatiespecificatie toepast en in welke mate PPO en Social Semantic SPARQL Security for Access Control (S4AC) uitgedrukt kunnen worden aan de hand van dit model.

Het uitvoeren van acht combinaties met Access Control List (ACL) toegangstypes resulteerde in 32 waarden uitgedrukt in Create, Read, Update, Delete (CRUD) toegang. Het model paste 100% van deze waarden correct toe in de SHACL shape terwijl in het gegenereerde formulier 93.75% correct werd toegepast. Het model kan toegepast worden op formulieren zonder geneste velden en op SHACL shapes met tot drie nestniveaus, waarbij de betrokkene kan kiezen of eigenschappen de verleende toegang van hun ouder erven. Toestemming uitgedrukt met PPO kan volledig voorgesteld worden aan de hand van het model, terwijl het model 44% van de functies van S4AC kan uitdrukken.

Automatic Web Form Generation based on Authorisation Specifications

Femke Brückmann

Supervisors: Dr. A. Dimou, Prof. Dr. Ir. R. Verborgh

Abstract— **Since the introduction of the General Data Protection Regulation (GDPR), the data subject's explicit and specific consent is required to process a natural person's data. Data on the World Wide Web is commonly stored ambiguously. The goal of the Semantic Web is to make the Web machine-readable to enable automatic processing using Knowledge Graphs (KG). In order to advance towards automated GDPR compliance regarding the data subject's consent, access control can be applied on forms generated on top of KGs. This thesis proposes an intermediate model that allows applying an authorisation specification to a form generated on top of a data graph and a shape graph. Experiments indicate that the main access control functionality is implemented, but more work is needed to improve the access control applied to the generated form.**

Keywords— **Form Generation, Knowledge Graph, Access Control, Consent**

I. Introduction

As of 2018, the introduction of the General Data Protection Regulation (GDPR) [3] has established regulations regarding the processing of a natural person's data and the data subject's consent. As of articles 6 and 7 of the GDPR, accessing, storing, or processing one's data requires the data subject's explicit consent. This requirement applies to all data collected on, for example, visited sites: Web usage stored as cookies or information filled out in a form. However, data acquired from forms is commonly stored in databases where it is ambiguously described. Therefore, machines do not understand the represented information and how it relates to other data.

The goal of the Semantic Web is to make the Web machine-readable by making the data unambiguous [1], allowing for automatic processing. Contrary to regular databases, entities and their relationships are semantically described as Knowledge Graphs (KG) represented using the Resource Description Framework (RDF) [6, 9]. Their meaning is unambiguous [1], allowing for machines to understand the data they are processing. Since the employed shape structure [4, 5] essentially describes what the data should look like, it enables automatic data processing and access control applications based on consent. One possible approach would be to automatically generate data editing interfaces on top of this data to simplify the enriching process of the KG [5]. GDPR compliance regarding consent has not been taken into account for existing form generation approaches. Over the past years, multiple publications [2, 7, 12] have generated forms on top of KGs to enrich them. However, Noy et al. [7][1] is the only publication that incorporates options for sharing created documents while granting either reading or writing access to certain people. Achieving and checking compliance is a demanding task for companies considering the number of directives in the GDPR, its extensiveness and the fact that it is fairly light on specifics [10]. Thus, businesses could be aided in automating the enforcement of GDPR compliance by applying access control to generated forms.

The focus of this dissertation is the application of access control to KGs to comply with the GDPR in the context of giving consent. Access control and privacy privileges will be applied to forms generated on top of these KGs, based on authorisation specifications expressed using the Privacy Preference Ontology (PPO) [8]. Thus, the data owner can specify which parts of their data are accessible, which types of access are granted and who can access this data using fine-grained access specifications.

II. Implementation

This thesis presents an intermediate model to tackle this problem. First, introductory steps are described before diving into the design and application of the intermediate model.

A. Schímatos

Since a recently published application called Schímatos[2] has already tackled form generation on top of KGs, this application is used as a base for the implementation. Schímatos is a form-based Web application that can be used to create and edit Resource Description Framework (RDF) [9] data constrained and validated by Shape Constraint Language (SHACL)[3] shapes [11, 12]. The user chooses the desired target and SHACL shapes using the *Sidebar Management*, which query

[1]WebProtégé – https://webprotege.stanford.edu/
[2]Schímatos – http://schimatos.github.io
[3]SHACL – https://www.w3.org/TR/shacl/

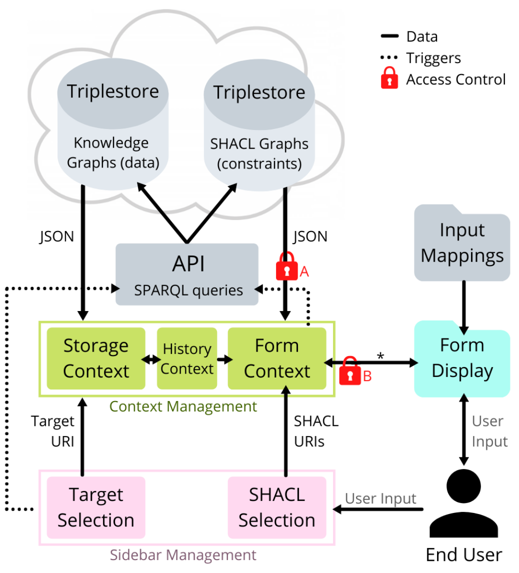


Figure 1: Schímatos Architecture.

the corresponding *triplestores* using SPARQL Protocol And RDF Query Language (SPARQL)[4] `ASK` queries. The resulting JavaScript Object Notation (JSON)[5] data is saved in the *Context Management*, where the *Form Context* transforms the JSON objects to allow the field generation in the *Form Display*. The application's architecture is visualised in fig. 1.

The first steps of the implementation process consist of creating a simple authentication system to allow the user to log in using a limited set of predefined user profiles. The user that is currently logged in will be referred to as the *requester*. Additionally, the user can insert a privacy preference expressed using PPO, which is assumed to be the data subject's explicit and specific consent to using their data.

B. Intermediate Model Design

The intermediate authentication specification model is a JavaScript object containing dictionaries of objects, centring around the *Policy* with its *UserAccessConditions* and *DataAccessConditions*. The primary reference for creating the structure of this intermediate model is PPO. Figure 2 visualises the intermediate model's design.

Each UserAccessCondition has an associated SPARQL `ASK` query to determine if the Policy applies to the requester based on their characteristics. On the other hand, DataAccessConditions contain properties indicating to which resources and data properties the Policy applies. The access granted to each Policy can be one of the basic CRUD operations: *Create*, *Read*, *Update* or *Delete*.

[4]SPARQL – https://www.w3.org/TR/sparql11-query/
[5]JSON – https://www.json.org/

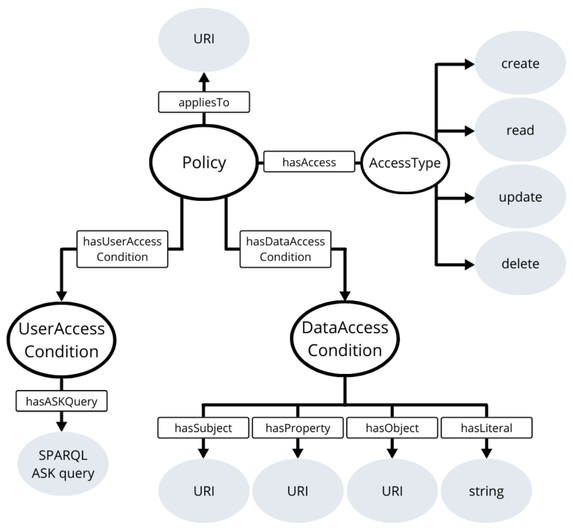


Figure 2: Intermediate Model.

This model aims to act as an abstraction layer between the expressed authorisation specification and the SHACL shape or form the specification is applied to. A specification expressed in PPO can therefore be mapped to this intermediate model.

C. Model Application

The application of this model is twofold. visualised in fig. 1 as access control options A and B, respectively. On the one hand, it can be applied to the JavaScript objects that generate the forms in Schímatos. On the other hand, it can be applied directly to the SHACL [5] shape used to generate the form and its constraints.

C.1 Form Generation

The first application option is based on creating an extra JavaScript object that indicates which types of access are granted to each field. Applying the authorisation specification essentially generates a new object containing the granted access types, which is taken into account while the form is being generated. This object contains four boolean values for every form field, each indicating one of the four CRUD access types: *canRead*, *canUpdate*, *canCreate* and *canDelete*.

Before the generation, the necessary information is fetched to determine the field properties. Depending on the corresponding access types, the allowed input fields and buttons are enabled and shown. For example, fig. 3 shows a generated form field where the requester has *Create*, *Read* and *Update* access, but cannot *Delete* values.

This step focused on implementing the access functionality, paving the way for a more generalised approach.

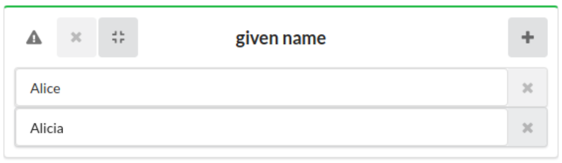


Figure 3: One generated form field.

C.2 SHACL Shapes

The intermediate model can also be applied to the SHACL constraint graph, applying the authorisation specification to the shape constraints before the form is generated. The model determines the applicable Policies and granted access types based on the property features and the requester's characteristics. These access types are then added as triples to the property shape, for example:

```
<http://schema.org/givenName>
    <http://example.org/access>
    <https://ns.inria.fr/s4ac/v2#Read> .
```

Since the triples are added directly to the SHACL shape, the shape with its applied access control can be used for other applications as well.

Due to implementation limitations in Schímatos and time constraints, these adjusted SHACL shapes were not implemented to replace the original shapes used to generate the forms. The following section will therefore consider these two application methods separately.

III. Evaluation

The functionality and performance of the intermediate model are evaluated using three experiments. First, the main features centred around granting access control are tested. The next experiment focuses on applying access control to nested shapes and access types inheritance from parents to their children. Finally, the mapping of Privacy Preference Ontology (PPO) and Social Semantic SPARQL Security for Access Control (S4AC) to the intermediate model is compared.

A. Results

Granting Access. In this experiment, eight resources are granted various combinations of ACL access types (*acl:Read*, *acl:Write* and *acl:Append*). These access options are then applied to the form generation and the SHACL shapes, resulting in 32 values expressed in CRUD. The model applied 100% of these values correctly in the SHACL shape, and 93.75% was expressed correctly in the generated form. In the latter case, granting only *acl:Append* access without *acl:Write* access results in some incorrectly disabled fields, since there is no distinction between input fields for new versus existing values.

Nesting. The second experiment explores the application of access control in nested shape constraints. The model can be applied to generated forms without nesting and to SHACL shapes with up to at least three levels of nesting. The data subject has the option to explicitly allow children to inherit granted access types from their parent.

Expressing Vocabularies. The classes and properties of PPO and S4AC were mapped to those of the intermediate model. In PPO, 16 out of 16 (100%) features could be expressed using the model, while for S4AC, this was only possible for 11 out of 25 features (44%). As mentioned earlier, the intermediate model is mainly based on PPO. On the other hand, concepts like access evaluation context and conjunctive/disjunctive condition sets are not included in the specification of the model.

B. Research Questions

Subquestion 1: Applying access control to a shape. Access control can be applied to a SHACL shape using an intermediate model to modify and add constraints to the given shape with up to three levels of nesting.

Subquestion 2: Applying explicit consent to form generation. A subject's explicit and specific consent can be applied to form generation without nesting using an intermediate model that adds access-related properties to the JavaScript objects for each field.

Research Question: Complying with GDPR's consent on generated forms. A subject's data shape can comply with the GDPR's definition of consent by using a fine-grained specification of access control permissions, as defined by the data subject using PPO.

IV. Conclusion and Future Work

This thesis proposed an intermediate model to express an authorisation specification. This model applies a data subject's explicit consent to SHACL shapes and the forms generated on top of these shapes. Three experiments indicated that the model could apply the consent expressed in PPO to shape constraints while taking into account nested shapes. However, applying the model to form generation requires some improvements in nesting and granting *acl:Append* access.

N-Quads. The implementation of the intermediate model could be expanded to support N-Quads and allow more options for expressing consent.

User-Friendly Consent Specification. Creating a more user-friendly way of expressing explicit and specific consent would allow inexperienced users to express their consent without understanding the underlying technologies.

Abstraction. A full abstraction between the consent expressed in any authorisation specification and the goal shape or the generated form could create a further generalised intermediate model. This generalisation could allow using different ontologies to express the data owner's consent.

Logical Condition Sets. Including some logical division between conjunctive and disjunctive condition sets in applying the intermediate model, combined with an indication of the condition set type, would increase the flexibility of the authorisation specifications.

Access Context. Another possible expansion for the model could support access context constraints, like temporal restrictions and a limited number of accesses. An additional description could inform the requester about the reason for access limitation.

Nesting. The model's application to the form generation might be improved by enhancing the inheritance of granted access types between parents and their children.

Strictness. Additional settings for the model might support more specific settings for strictness regarding combinations of denied and granted access. This approach could allow more flexibility for granting access.

New and Existing Fields. A distinction between fields of existing and new values could be made by adding an indication to the form fields before generating. This distinction could improve the application of the granted access in the form.

References

[1] Berners-Lee, T., Connolly, D., Stein, L.A., Swick, R.: The semantic web (August 2000), https://www.w3.org/2000/Talks/0906-xmlweb-tbl/text.htm

[2] Butt, A.S., Haller, A., Liu, S., Xie, L., et al.: Activeraul: A web form-based user interface to create and maintain rdf data. In: International Semantic Web Conference (Posters & Demos). pp. 117–120 (2013)

[3] Council of European Union: Regulation (EU) 2016/679 (2016), https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679

[4] Group, S.C.: Shex – shape expressions (2012), https://shex.io/

[5] Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl) (2017), https://www.w3.org/TR/shacl/

[6] Manola, F., Miller, E., McBride, B.: Rdf primer (2004), https://www.w3.org/TR/rdf-primer/

[7] Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protege-2000. IEEE intelligent systems **16**(2), 60–71 (2001)

[8] Sacco, O., Passant, A.: A privacy preference ontology (ppo) for linked data. In: LDOW (2011)

[9] W3C: Resource description framework (rdf) (2014), https://www.w3.org/2001/sw/wiki/RDF

[10] Wolford, B.: What is GDPR, the EU's new data protection law? (2020), https://gdpr.eu/what-is-gdpr/

[11] Wright, J.: Shacl form react (2021), https://github.com/schimatos/shacl-form-react

[12] Wright, J., Méndez, S.J.R., Haller, A., Taylor, K., Omran, P.G.: Schímatos: a shacl-based web-form generator for knowledge graph editing. In: International Semantic Web Conference. pp. 65–80. Springer (2020)

Webformulieren automatisch genereren op basis van autorisatiespecificaties

Femke Brückmann

Begeleiders: Dr. A. Dimou, Prof. Dr. Ir. R. Verborgh

Abstract— **Sinds de introductie van de Algemene Verordening Gegevensbescherming (GDPR) is expliciete en specifieke toestemming vereist voor het verwerken van persoonsgegevens. Data in het wereldwijde web wordt vaak dubbelzinning opgeslagen. Het doel van het Semantisch Web is om het web machineleesbaar te maken om data automatisch te kunnen verwerken aan de hand van Knowledge Graphs (KG). Om de toestemming van de betrokkene volgens de GDPR automatisch na te leven, kan toegangscontrole worden toegepast op formulieren die gegenereerd worden aan de hand van KG's. Deze masterproef stelt een intermediair model voor, waarmee een autorisatiespecificatie kan worden toegepast op een formulier dat is gegenereerd op basis van een *data graph* en een *shape graph*. Experimenten geven aan dat de belangrijkste functionaliteiten voor toegangscontrole zijn geïmplementeerd, maar er is aanvullend werk nodig om de toepassing van toegangscontrole op gegenereerde formulieren te verbeteren.**

Kernwoorden— **webformulieren, Knowledge Graph, toegangscontrole, toestemming**

I. Introductie

In 2018 heeft de Algemene Verordening Gegevensbescherming (AVG of GDPR) voorschriften geïntroduceerd met betrekking tot het verwerken van persoonsgegevens en het verlenen van toestemming hiertoe door de betrokkene. Artikels 6 en 7 van de GDPR stellen dat toegang, opslag en verwerking van persoonsgegevens uitdrukkelijke toestemming vereisen van de betrokkene. Deze vereiste is van toepassing op alle gegevens die verzameld worden, bijvoorbeeld op bezochte sites: webgebruik opgeslagen als cookies of informatie ingevuld in een formulier. Gegevens die verkregen worden uit formulieren, worden gewoonlijk echter opgeslagen in databanken waar ze dubbelzinnig worden beschreven. Machines hebben hierdoor geen notie van de weergegeven informatie en hoe deze zich verhoudt tot andere gegevens.

Het doel van het Semantisch Web is om het Web machineleesbaar te maken door data ondubbelzinnig te maken [1], wat automatische verwerking mogelijk maakt. In tegenstelling tot reguliere databanken worden entiteiten en hun relaties semantisch beschreven als Knowledge Graphs (KG), die voorgesteld worden met behulp van het Resource Description Framework (RDF) [5, 8]. Hun betekenis is ondubbelzinnig [1], wat ervoor zorgt dat machines de gegevens die ze verwerken ook kunnen begrijpen. Aangezien de gebruikte shape-structuur [3, 4] beschrijft hoe de gegevens eruit moeten zien, worden automatische gegevensverwerking en toepassingen op toegangscontrole op basis van toestemming mogelijk. Een mogelijke aanpak zou zijn om automatische gebruikersinterfaces voor het bewerken van gegevens te genereren bovenop deze gegevens om het verrijkingsproces van de KG te vereenvoudigen [4].

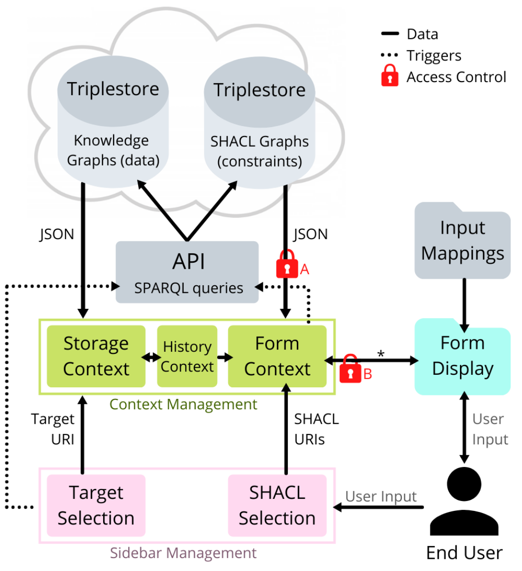
Naleving van de GDPR met betrekking tot toestemming werd niet eerder in rekening gebracht bij bestaande benaderingen voor het genereren van formulieren. In de afgelopen jaren hebben meerdere publicaties [2, 6, 11] formulieren gegenereerd op basis van KG's om deze te verrijken. Slechts één publicatie [6][1] bevat opties voor het delen van documenten met anderen, waarbij lees- of schrijftoegang verleend wordt. Voor bedrijven is het een veeleisende taak om de GDPR volledig na te leven en dit te controleren, gezien het aantal richtlijnen in de GDPR, de uitgebreidheid en de weinige details [9]. Bedrijven zouden dus geholpen kunnen worden bij het automatiseren van de naleving van de GDPR door toegangscontrole toe te passen op gegenereerde formulieren.

De focus van deze masterproef is het toepassen van toegangscontrole op KG's om te voldoen aan de GDPR bij het geven van toestemming. Toegangscontroles en privacyrechten zullen toegepast worden op formulieren die gegenereerd worden op basis van deze KG's, gebaseerd op de autorisatiespecificaties. Deze specificaties worden uitgedrukt aan de hand van de Privacy Preference Ontology (PPO) [7]. De betrokkene kan dus aangeven welke delen van hun gegevens toegankelijk zijn, welke toegang verleend wordt en wie de gegevens kan raadplegen met behulp van gedetailleerde toegangsspecificaties.

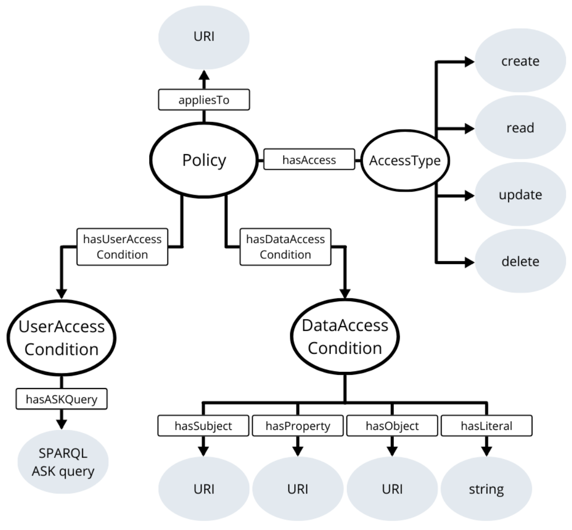
II. Implementatie

Deze masterproef stelt een intermediair model voor om dit probleem op te lossen. De voorbereidende stappen worden hieronder beschreven, gevolgd door het ontwerp en de toepassing van het tussenliggende model.

[1] WebProtégé – `https://webprotege.stanford.edu/`



Figuur 1: Architectuur van Schímatos. Naar [11].



Figuur 2: Intermediair Model.

A. *Schímatos*

Aangezien Schímatos[2], een recent gepubliceerde applicatie, reeds de mogelijkheid biedt om formulieren te genereren aan de hand van KG's, wordt deze applicatie gebruikt als basis voor de implementatie. Schímatos is een op formulieren gebaseerde webapplicatie die gegevens in Resource Description Framework (RDF) [8] kan aanmaken en bewerken. Hierbij wordt de vorm van de gegevens gewaarborgd aan de hand van de Shape Constraint Language (SHACL)[3] shapes [10, 11]. De gebruiker kiest de gewenste bron en SHACL shapes in de *Sidebar Management*, die de overeenkomstige *triplestores* zal aanspreken met behulp van SPARQL Protocol And RDF Query Language (SPARQL)[4] `ASK` queries. De resulterende JavaScript Object Notation (JSON)[5]-data wordt opgeslagen in de *Context Management*, waar de *Form Context* de JSON-objecten zal omvormen tot ze bruikbaar zijn om het formulier te genereren in de *Form Display*. De architectuur van deze applicatie is afgebeeld in fig. 1.

De eerste stappen van het implementatieproces bestaan uit het aanmaken van een eenvoudig authenticatiesysteem zodat de gebruiker kan inloggen aan de hand van een beperkt aantal voorgedefinieerde gebruikersprofielen. De gebruiker die op dit moment ingelogd is, zal de *aanvrager* genoemd worden. Daarnaast kan de gebruiker een privacyvoorkeur uploaden, uitgedrukt in PPO, wat wordt verondersteld de uitdrukkelijke en specifieke toestemming van de betrokkene te zijn voor het gebruik van hun gegevens.

[2]Schímatos – `http://schimatos.github.io`
[3]SHACL – `https://www.w3.org/TR/shacl/`
[4]SPARQL – `https://www.w3.org/TR/sparql11-query/`
[5]JSON – `https://www.json.org/`

B. *Design van het intermediaire model*

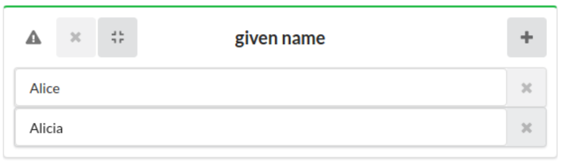
Het intermediaire model voor de authenticatiespecificatie is een JavaScript-object dat dictionaries van objecten bevat, opgebouwd rond de *Policy* met zijn *UserAccessCondition*s en *DataAccessCondition*s. De primaire referentie voor het maken van de structuur van dit model is de Privacy Preference Ontology (PPO). Figuur 2 visualiseert het ontwerp van dit intermediaire model.

Elke UserAccessCondition heeft een bijhorende SPARQL `ASK` query om te bepalen of een Policy van toepassing is op de aanvrager, gebaseerd op hun karakteristieken. Anderzijds bevatten de DataAccessConditions eigenschappen die aangeven op welke bronnen en gegevenseigenschappen de Policy van toepassing is. De toegang die verleend wordt aan een Policy, kan één van de basis-CRUD-bewerkingen zijn: aanmaken (*Create*), lezen (*Read*), aanpassen (*Update*) of verwijderen (*Delete*).

Dit intermediaire model zal fungeren als een abstractielaag tussen de uitgedrukte autorisatiespecificatie en de SHACL shape of het formulier waarop deze specificatie toegepast is. Een specificatie die uitgedrukt is in PPO kan dus worden vertaald naar dit model.

C. *Toepassen van het model*

Het model kan op twee manieren toegepast worden, die in fig. 1 weergegeven worden als opties A en B. Enerzijds kan het model toegepast worden op de JavaScript-objecten die de formulieren in Schímatos genereren. Anderzijds kan het model rechtstreeks toegepast worden op de SHACL shape die gebruikt wordt om het formulier en de bijbehorende validatie te genereren.



Figuur 3: Een gegenereerd veld.

C.1 Formuliergeneratie

De eerste manier van toepassen is gebaseerd op het aanmaken van een bijkomend JavaScript-object dat aangeeft welk type toegang verleend wordt aan elk veld. Het toepassen van deze autorisatiespecificatie genereert een nieuw object dat de verleende toegangstypes bevat. Tijdens het genereren van het fomulier zal rekening gehouden worden met dit object. Het object bevat vier booleaanse waarden voor elk formulierveld, die elk één van de vier CRUD-toegangstypen aangeven: *canRead*, *canUpdate*, *canCreate* and *canDelete*.

Alvorens het genereren zal de nodige informatie voor het bepalen van de veldeigenschappen opgevraagd worden. Knoppen en invoervelden worden zichtbaar en ingeschakeld naar gelang de overeenkomstige toegangstypes. Figuur 3 toont bijvoorbeeld een gegenereerd invoerveld waar de aanvrager *Create*-, *Read*- en *Update*-toegang heeft, maar geen waarden kan verwijderen.

Deze stap is gericht op het implementeren van de toegangsfunctionaliteit, ter voorbereiding van een meer algemene aanpak.

C.2 SHACL shapes

Het intermediaire model kan ook toegepast worden op de SHACL constraint graph, waarbij the autorisatiespecificatie zal toegepast worden op de shape constraints alvorens het formulier gegenereerd wordt. Het model bepaalt welke Policies toepasbaar zijn en welke toegangstypes hierbij toegekend worden op basis van de eigenschappen en de karakteristieken van de aanvrager. Deze toegangstypes worden dan als triples toegevoegd aan de property shape, bijvoorbeeld:

```
<http://schema.org/givenName>
    <http://example.org/access>
    <https://ns.inria.fr/s4ac/v2#Read> .
```

Aangezien de triples rechtstreeks toegevoegd worden aan de SHACL shape, zou deze shape met toegepaste toegangscontrole ook gebruikt kunnen worden voor andere doeleinden.

Door beperkingen bij de implementatie in Schímatos en tijdslimieten werden deze aangepaste SHACL shapes niet geïmplementeerd als vervanging voor de originele shapes in de formuliergeneratie. De volgende sectie zal daarom deze twee toepassingsmogelijkheden apart beschouwen.

III. Evaluatie

De functionaliteit en prestatie van het intermediaire model worden geëvalueerd aan de hand van drie experimenten. Eerst worden de voornaamste functies rond het verlenen van toegangscontrole getest. Het volgende experiment richt zich op het toepassen van toegangscontrole op geneste shapes en de overerving van toegangscontrole van ouders naar hun kinderen. Tot slot wordt de vertaling van de Privacy Preference Ontology (PPO) en de Social Semantic SPARQL Security for Access Control (S4AC) naar het intermediaire model vergeleken.

A. Resultaten

Toegang toekennen. In dit experiment krijgen acht bronnen verschillende combinaties van ACL toegangstypes (*acl:Read*, *acl:Write* en *acl:Append*) toegekend. Deze toegangsopties worden dan toegepast op de formuliergeneratie en de SHACL shapes, wat resulteert in 32 waarden uitgedrukt in CRUD. Het model past 100% van deze waarden correct toe op de SHACL shape, terwijl 93.75% ervan correct worden toegepast in het gegenereerde formulier. In het tweede geval zal het toekennen van enkel *acl:Append*-toegang zonder *acl:Write*-toegang resulteren in een paar onterecht uitgeschakelde velden, aangezien de applicatie geen verschil aanduidt tussen invoervelden voor nieuwe of bestaande waarden.

Nesten. Het tweede experiment onderzoekt de toepassing van toegangscontrole in geneste shape constraints. Het model kan toegepast worden op gegenereerde formulieren zonder nesten en SHACL shapes met maximaal drie nestniveaus. De betrokkene heeft de mogelijkheid om kinderen expliciet toe te staan verleende toegangstypes van hun ouder te erven.

Uitdrukken van ontologieën. De klassen en eigenschappen van PPO en S4AC werden vertaald naar deze van het intermediaire model. 16 van de 16 (100%) functies van PPO kunnen uitgedrukt worden aan de hand van dit model. Bij S4AC is dit slechts 11 van de 25 (44%). Zoals eerder vermeld is het intermediaire model vooral gebaseerd op PPO. Anderzijds worden concepten als de context van de toegangsevaluatie en conjunctieve/disjunctieve conditiesets niet in beschouwing genomen in de specificatie van het model.

B. Onderzoeksvragen

Subvraag 1: Toepassing van toegangscontrole op een shape. Toegangscontrole kan toegepast worden op een SHACL shape aan de hand van een intermediair model dat beperkingen aanpast en toevoegt aan de gegeven shape met tot drie geneste niveaus.

Subvraag 2: Toepassing van expliciete toestemming op gegenereerde formulieren. De expliciete en specifieke toestemming van de betrokkene kan toegepast worden op gegenereerde formulieren zonder nesting met behulp van een intermediair model dat toegangsgerelateerde eigenschappen toevoegt aan het JavaScript-object voor elk veld.

Onderzoeksvraag: Voldoen aan toestemming in de GDPR op gegenereerde formulieren. De data shape van een betrokkene kan voldoen aan de definitie van toestemming in de GDPR door gebruik te maken van fine-grained toelatingsspecificaties zoals door de betrokkene gedefinieerd in PPO.

IV. Conclusie en aanvullend werk

Deze thesis stelt een intermediair model voor om een autorisatiespecificatie uit te drukken. Dit model past de expliciete toegang van een betrokkene toe op SHACL shapes en de formulieren die hierop gegenereerd worden. Drie experimenten gaven aan dat het model, uitgedrukt in PPO, toegepast kan worden op shape constraints. Anderzijds heeft de toepassing op formuliergeneratie verbeteringen nodig op vlak van nesten en het toekennen van *acl:Append*-toegang.

N-Quads. De implementatie van het intermediaire model kan uitgebreid worden om N-Quads te ondersteunen, wat zou zorgen voor meer flexibiliteit bij het uitdrukken van toestemming.

Gebruiksvriendelijke toestemmingsspecificaties. Een meer gebruiksvriendelijke manier voor het uitdrukken van expliciete en specifieke toestemming zou het eenvoudiger maken voor onervaren gebruikers om hun toestemming uit te drukken zonder de onderliggende technologieën te begrijpen.

Abstractie. Een volledige abstractie tussen de uitgedrukte toestemming in eender welke autorisatiespecificatie en de doelshape of het gegenereerde formulier zou voor een verdere veralgemening van het intermediaire model zorgen. Deze veralgemening zou de betrokkene toestaan om hun toestemming uit te drukken aan de hand van verschillende ontologieën.

Logische conditiesets. Het implementeren van een logisch onderscheid tussen conjunctieve en disjunctieve conditiesets bij het toepassen van het intermediaire model, gecombineerd met een indicatie van het type conditieset, zou de flexibiliteit van de autorisatiespecificiatie vergroten.

Toegangscontext. Een andere mogelijke uitbreiding voor het model zou beperkingen op toegangscontext, zoals temporele restricties en een beperkt aantal toegangen, in acht nemen. Een bijkomende beschrijving zou de aanvrager kunnen informeren over de reden voor toegangsbeperking.

Nesten. De toepassing van het model op de formuliergeneratie zou verbeterd kunnen worden door het verbeteren van de overerving van verleende toegangstypes tussen ouders en hun kinderen.

Strengheid. Bijkomende instellingen voor het model zouden meer specifieke instellingen voor strengheid bij verschillende combinaties van toegekende toegangstypes kunnen toestaan. Deze aanpak zou voor meer flexibiliteit zorgen bij het toekennen van toegang.

Nieuwe en bestaande velden. Een onderscheid tussen invoervelden voor bestaande en nieuwe waarden door een aanduiding toe te voegen aan de velden alvorens ze te genereren. Dti onderscheid zou de toekenning van toegang in het formulier verbeteren.

Referenties

[1] Berners-Lee, T., Connolly, D., Stein, L.A., Swick, R.: The semantic web (August 2000), https://www.w3.org/2000/Talks/0906-xmlweb-tbl/text.htm

[2] Butt, A.S., Haller, A., Liu, S., Xie, L., et al.: Activeraul: A web form-based user interface to create and maintain rdf data. In: International Semantic Web Conference (Posters & Demos). pp. 117–120 (2013)

[3] Group, S.C.: Shex – shape expressions (2012), https://shex.io/

[4] Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl) (2017), https://www.w3.org/TR/shacl/

[5] Manola, F., Miller, E., McBride, B.: Rdf primer (2004), https://www.w3.org/TR/rdf-primer/

[6] Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protege-2000. IEEE intelligent systems **16**(2), 60–71 (2001)

[7] Sacco, O., Passant, A.: A privacy preference ontology (ppo) for linked data. In: LDOW (2011)

[8] W3C: Resource description framework (rdf) (2014), https://www.w3.org/2001/sw/wiki/RDF

[9] Wolford, B.: What is GDPR, the EU's new data protection law? (2020), https://gdpr.eu/what-is-gdpr/

[10] Wright, J.: Shacl form react (2021), https://github.com/schimatos/shacl-form-react

[11] Wright, J., Méndez, S.J.R., Haller, A., Taylor, K., Omran, P.G.: Schímatos: a shacl-based web-form generator for knowledge graph editing. In: International Semantic Web Conference. pp. 65–80. Springer (2020)

LAY SUMMARY

GDPR

In 2018, the *General Data Protection Regulation* (GDPR) was introduced in the European Union. This new law on data privacy and security includes new requirements for organisations that handle their users' or customers' personal data. *Personal data* is any piece of information that can relate to an identifiable person, including names, addresses, phone numbers, as well as data collected by visiting sites and using products. Accessing, storing or processing this personal data requires explicit and specific consent from the data owner. High fines are issued when this regulation is violated. Since the GDPR contains many rules, correctly applying all of them is a challenge for organisations.

Internet and the World Wide Web

The *Internet* is the worldwide system of interlinked computers that communicate between networks and devices. Essentially a network of networks, the Internet carries a huge range of services, including the *World Wide Web* (WWW). The World Wide Web, also known as the Web, is an information system where documents are linked using *hypertext*. Hypertext is text with references to other text that are immediately accessible by the reader. Additionally, data on the Web can be added and edited by anyone.

Semantic Web & Linked Data

The *Semantic Web* is an extension of the World Wide Web, created to fix some major issues. Even though the Web is good at displaying information for humans to read, machines and computers have a harder time understanding this data. The Semantic Web allows giving data a meaning, allowing machines to understand the given information. Additionally, Semantic Web tools can reach and manage this data if it is available in a standard format. By defining relationships as links between different pieces of information, a network of linked datasets is established: this is *Linked Data*. One such network of interlinked entities is a *Knowledge*

Graph (KG). In a KG, each entity is described alongside its relationship with other entities in the graph.

Defining these relationships is done using *ontologies*, vocabularies that define concepts and relationships in specific domains. For example, *Friend of a Friend* (FOAF) defines how people relate: friends, business partners, or family. The "mould" or so-called *shape* this data should fit in can be expressed using the *Shape Constraint Language* (SHACL). This language defines the requirements and restrictions the data should meet to be valid. For example, a person can have at most one birth date. Both the ontologies and shape constraints will return in this summary.

Solution

This thesis focuses on the combination of complying with the GDPR when it comes to consent and the generation of form interfaces on top of the data graphs that contain personal data. In a form generated on top of a person's data, the data owner could determine who can access or modify each part of their data. Thus, the automatic form generation with integrated access control would aid organisations in complying with the GDPR while making it easier to enter and modify data in KGs.

Implementation

This thesis proposes a model that acts as a layer between a person's expressed consent and the Knowledge Graph (KG) that contains their data. This intermediate model translates the expressed consent and can be applied in two ways: to both the shapes that constrain the data and the forms that are generated to edit the data. The expressed consent determines who can access or modify which part of the data. For each type of operation (Create, Read, Update, Delete, abbreviated as CRUD), a new property is added to a part of the shape's constraints. The added property types depend on which access is granted and who is requesting the access. This information can then be used to generate the form, determine which fields should be visible and which actions the requesting user can take on each piece of information.

Evaluation

The performance of this model was evaluated using three experiments, considering the two application methods separately.

The first experiment applies allowed access expressed in the data owner's consent. Three access types defined in *Access Control Lists* (ACL), namely *Read*, *Append* and *Write*, are coupled in eight different combinations. Then the translation from this expressed consent to the CRUD access types is checked. 100% of these values were translated correctly in the shape, while 93.75% was expressed correctly in the generated form.

The next evaluated aspect is how granted access is passed down from parent to child component. For example, if one is allowed to modify a person's address, are they allowed to modify the country of residence as well? The tests showed that access is passed down at least up to three "generations" in the SHACL constraints, where the data subject chooses if the children inherit their parent's access types. However, in the generated form, access is not passed down to the children.

The final experiment evaluated how well two ontologies, *Privacy Preference Ontology* (PPO) and *Social Semantic SPARQL Security for Access Control* (S4AC), could be translated to the model. PPO can be fully translated, while 44% of S4AC can be expressed using the model.

CONTENTS

GLOSSARY

conjunctive a conjunctive set of values is only True if and only if all of its values are true. 28, 30, 50

consent any freely given, specific, informed and unambiguous indication of the data subjects wishes by which they, by a statement or by a clear affirmative action, signify agreement to the processing of personal data relating to them. 11, 17

consistency a knowledge base KB is consistent if and only if its negation is not a tautology; in other words, if and only if there exists an interpretation which has KB as a logical consequence. 8

controller the natural or legal person, public authority, agency or other body which determines the purposes and means of the processing of personal data. 11, 17, 19

disjunctive a disjunctive set of values is only True if and only if at least one of its values is true. 28, 30, 50

HTTP cookie a small piece of data that a server sends to the user's web browser [26]. 11

personal data any information relating to an identified or identifiable natural person, called the data subject. 10

processing any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means. 10, 11, 17

processor a natural or legal person, public authority, agency or other body which processes personal data on behalf of the controller. 17, 19

ACRONYMS

ACL	Access Control List. 18, 27, 32, 36, 39, 40, 48
CRUD	Create, Read, Update, Delete. 29, 30, 32, 34, 36, 39, 40, 48
GDPR	General Data Protection Regulation. 1–3, 10–12, 16–20, 22, 37, 48, 49
HTML	HyperText Markup Language. 13, 21
IRI	Internationalized Resource Identifier. 6, 7
JSON	JavaScript Object Notation. 21, 22, 34
KG	Knowledge Graph. 1, 2, 7, 8, 22, 48, 52
OWL	Web Ontology Language. 4, 8, 12
PPO	Privacy Preference Ontology. 2, 18, 20, 23, 24, 26, 27, 30, 31, 36, 37, 39, 40, 46, 48–50
RDF	Resource Description Framework. 1, 4, 6–9, 12–18, 20, 21, 24–26, 31
RDFS	RDF Schema. 8
S4AC	Social Semantic SPARQL Security for Access Control. 18, 20, 23, 26, 28, 30, 34, 37, 46, 48, 50
SHACL	Shapes Constraint Language. 5, 8, 9, 12–16, 20–24, 32, 34, 35, 37, 39, 40, 42, 43, 45, 48, 49, 52, 54, 62, 63
ShEx	Shape Expressions. 5, 9, 15, 22
SPARQL	SPARQL Protocol And RDF Query Language. 9, 10, 14, 15, 18, 21, 22, 25, 27–30, 34, 37, 38
URI	Uniform Resource Identifier. 6, 7, 25, 29, 36
W3C	World Wide Web Consortium (W3C). 3–5, 9, 14, 17, 18, 20
WAC	Web Access Control. 18, 27, 30
WWW	World Wide Web. 3, 4, 6

CHAPTER 1

INTRODUCTION

As of 2018, the introduction of the General Data Protection Regulation (GDPR) [11] has established regulations regarding the processing of a natural person's data and the data subject's consent. Data may only be processed under specific lawful bases, one of which is consent ([11], art. 6, 7). Therefore, accessing, storing, or processing one's data requires the data subject's explicit consent. This requirement applies to all data collected on the visited sites: our Web usage is stored as cookies, to which users agree upon entering the site, or we fill out some information in a form. These forms are typically manually created since the acquired data is commonly stored in databases where it is ambiguously described. Therefore, machines do not understand the represented information and how it relates to other data.

The goal of the Semantic Web is to make the Web machine-readable by making the data unambiguous [3], allowing for automatic processing. Contrary to regular databases, entities and their relationships are semantically described as Knowledge Graphs (KG) represented using the Resource Description Framework (RDF) [24, 40]. Their meaning is unambiguous [3], allowing for machines to understand the data they are processing. Since the employed shape structure [17, 22] essentially describes what the data should look like, it enables automatic data processing and access control applications based on consent. One possible approach would be to automatically generate data editing user interfaces on top of this data to simplify the KGs' enriching process [22].

GDPR compliance regarding consent has not been taken into account for existing form generation approaches. Over the past years, multiple publications [5, 27, 49] generated forms on top of KGs to enrich them. Only one publication [27][1] incorporates options for sharing created documents while granting either reading or writing access to certain people. This sharing option is one step towards expressing consent regarding data access in the context of forms. Other publications did not take privacy preferences or consent into account. Achieving and checking compliance is a demanding task for companies considering the number of directives in the GDPR, its extensiveness and the fact that it lacks specificity [47]. Thus, businesses could be aided in automating the enforcement of GDPR compliance by applying access control to generated forms.

[1]WebProtégé – `https://webprotege.stanford.edu/`

The focus of this dissertation is the application of access control to KGs to comply with the GDPR in the context of giving consent. More specifically, access control and privacy privileges will be applied to forms generated on top of these KGs, based on authentication specifications expressed using the Privacy Preference Ontology (PPO). Thus, the data owner can specify which parts of their data are accessible, which types of access are granted and who can access this data using fine-grained access specifications. This solution is another step towards automatically applying the GDPR to linked data.

The remainder of this dissertation is structured as follows. Chapter 2 introduces concepts that this thesis builds upon. Then related work and research will be discussed in Chapter 3. The resulting research questions and methodology are described in Chapter 4. Then, the proposed solution and its implementation are presented in chapter 5. Chapter 6 describes the evaluation methods and their results. Finally, the dissertation concludes in Chapter 7 with a prospect on possible future work, followed by the bibliography and appendix.

CHAPTER 2

BACKGROUND

This chapter presents some of the key concepts and describes background information on what this dissertation builds upon to provide context. First, the Semantic Web and its technologies will be explained, followed by a brief explanation of the GDPR and its core principles.

2.1 Semantic Web and Linked Data

This section discusses the Semantic Web, including the World Wide Web and the Semantic Web Stack, along with the supporting technologies and used syntaxes.

2.1.1 World Wide Web

The Semantic Web cannot be explained without mentioning its foundation, the World Wide Web (WWW). The World Wide Web was invented by the English scientist Tim Berners-Lee in 1989 while employed at Conseil Européen pour la Recherche Nucléaire (CERN), where it started as a networked information project. The original proposal by Berners-Lee discusses how information loss affected the organisation's operations and describes a solution based on a distributed hypertext system [2]. Hypertext is text with references to other text, called hyperlinks, that are immediately accessible by the reader. Through these hypertext links and multimedia techniques, anyone can easily browse and contribute to the Web. W3C, an international community that develops open standards to ensure the long-term growth of the Web [46], defines the WWW as "the universe of network-accessible information, the embodiment of human knowledge" [38]. Making data available online, in a human-readable format, nevertheless does not mean that this data is essentially readable by machines as well.

2.1.2 Semantic Web

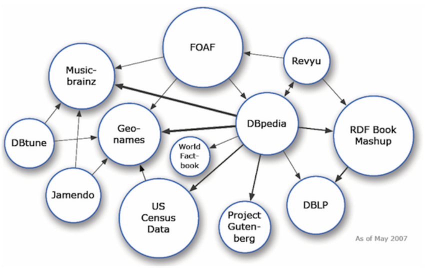
The Semantic Web is an extension of this World Wide Web. The term “Semantic Web” refers to the Web of linked data, which provides a common framework that allows data with a well-defined meaning to be shared and reused between diverse agents, which are entities, programs or applications that can perform actions on the Web. W3C has set the standards for this Web to make all available data interpretable by machines. To illustrate the growth of the Semantic Web throughout the years, fig. 2.1 visualises the Linked Open Data Cloud[1], a diagram that depicts publicly available linked datasets. Whereas in May 2007 the chart contained only 12 interlinked datasets, this number has grown to 1,301 in May 2021. It is important to note that the Semantic Web and the “regular Web” do not coexist side-by-side as two separate technologies. Instead, the Semantic Web lives on top of the World Wide Web as a semantic extension layer. This layer compromises a set of formats and technologies, providing formal descriptions of relationships and concepts [3], which will be explained later in this chapter. Additionally, it allows for recording how the data relates to real-world objects.

The vast amount of data on the Web should be made available in a standard format and be reachable and manageable by Semantic Web tools. The Web of Data is established by defining relationships among this data, which creates a collection of interrelated datasets called Linked Data [42]. The Semantic Web relies on technologies like the Resource Description Framework (RDF) [40] and Web Ontology Language (OWL) [39], to allow such functionality. The ultimate goal is to enable computers to perform tasks by analysing data and develop systems that can support trusted interactions.

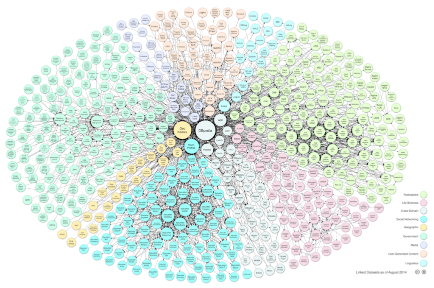
In the Semantic Web, *vocabularies* define the concepts and relationships used to describe a particular domain and are the basic building blocks for inference techniques. Inference in the Semantic Web is generally characterised by automatically discovering new relationships based on the data and information from the vocabulary [41]. Vocabularies are used to classify terms used in a particular application and define relationships and possible constraints [43]. A widely used example is Friend of a Friend (FOAF), devoted to linking people and their information on the Web [4]. The word “vocabulary” can often be interchanged with “ontology”. “Vocabulary” is more often used when strict formalism is unnecessary, while the latter is often used for a more complex and possibly quite formal collection of terms.

One aspect of interest about the Semantic Web is its open-world assumption: what is not known to be true is not necessarily false due to its open, distributed nature. In practice, this implies that conclusions can only be made from explicitly stated hypotheses. From the absence of a statement, one cannot infer that this statement is false. Interpreting ontology assertions as rules or conditions about certain data thus violates this open-world assumption.

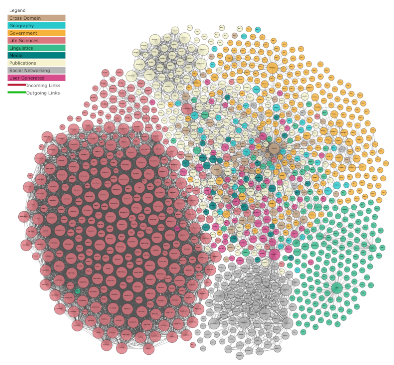
[1]Linked Open Data Cloud – `https://www.lod-cloud.net/`



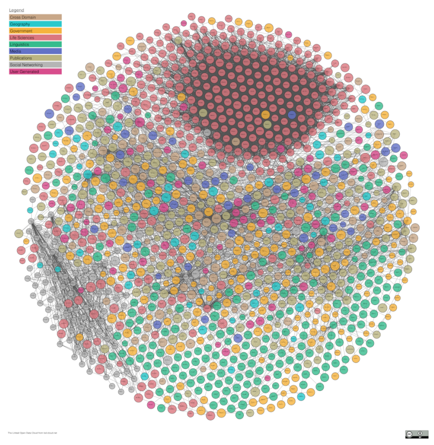
(a) May 1st, 2007.



(b) September 19th, 2011



(c) August 22nd, 2017.



(d) May 5th, 2021.

Figure 2.1: Linked Open Data Cloud throughout the years. (`https://www.lod-cloud.net/`)

Languages like SHACL[2] or ShEx[3] allow to describe graph-based data and explicitly define constraints and requirements that the data should satisfy. These languages will be explored further in the next section, among other technologies in the Semantic Web Stack.

2.1.3 Semantic Web Stack

This section outlines the technologies used in the relevant layers of the Semantic Web Stack, an overview of which can be found in fig. 2.2[4]. Since the notion of a Semantic Web evolves throughout the years, the depicted stack is a tweaked version [19]. The bottom layers contain well-known technologies used in the hypertext web that provide the basis for the Semantic Web. The middle layers of the stack contain technologies that W3C standardises to enable the building of Semantic Web applications and will be the focus of this dissertation. In contrast, the top layer technologies are not yet standardised and thus out of the scope of this thesis.

[2]SHACL – `https://www.w3.org/TR/shacl/`
[3]ShEx – `https://shex.io/shex-semantics/`
[4]Semantic Web Stack – `https://dbpedia.org/describe/?uri=http%3A%2F%2Fdbpedia.org%2Fresource%2FSemantic_Web_Stack` and [19]

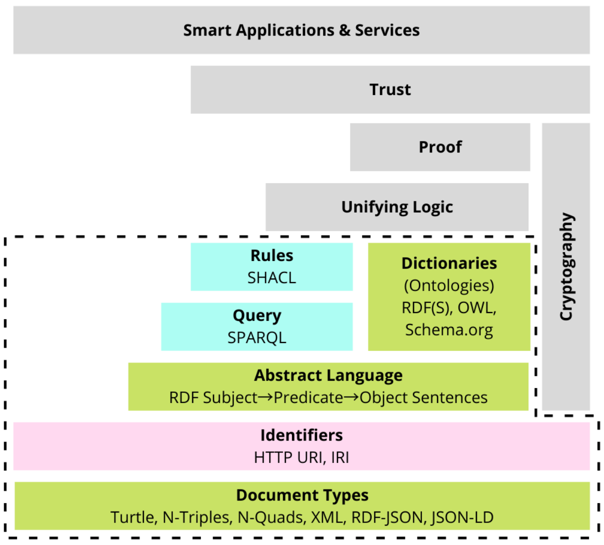


Figure 2.2: The Semantic Web Stack. The dashed area includes the technologies relevant in this thesis. Adapted from [19].

Some supporting technologies in the Semantic Web Stack are Uniform Resource Identifier (URI)[5] and Internationalized Resource Identifier (IRI)[6]. In the World Wide Web, and therefore in the Semantic Web, to identify and access individual objects. Uniform Resource Identifiers and Internationalized Resource Identifiers are parts of this universal set of identifiers [25]. The IRI defines a complement of the URI protocol by expanding the set of allowed characters to include most characters from the Universal Character Set. The valid IRI `https://päypal.com`, for example, would be an invalid URI. Since a mapping from IRIs to URIs is defined, one can use IRIs instead of URIs to identify resources [13]. A Uniform Resource Locator (URL) is a form of URI that maps onto an access algorithm using network protocols [25], such as `https://google.com`. URL is a subset of URI, identifying the location of the resource.

Next, the Resource Description Framework (RDF)[40] is a directed, labelled graph data format for representing information on the Web. Its goal is expressing the meaning of information using a *triple*, much like a sentence contains a subject, verb and object. For example, the statement

`http://www.example.org/index.html` `has a` **`creator`** `whose value is` **`John Doe`**

can be described using a triple [24]:

- The *subject* is the thing described by the statement.
  In this case, the URL `http://www.example.org/index.html`.
- The *predicate* is a specific property of the thing the statement describes.
  In this case, the word "creator".

[5]URI – `https://www.rfc-editor.org/rfc/rfc3986.html`
[6]URI – `https://www.rfc-editor.org/rfc/rfc5122.html`

- The *object* is the value of this property.
  In this case, the phrase “John Doe”.

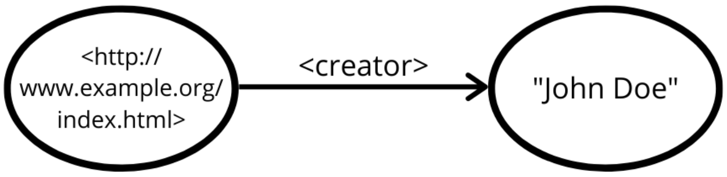


Figure 2.3: Graph visualisation of this triple.

Figure 2.3 shows a visual representation of the triple. In this directed graph, the object and subject are both nodes, while the relationship between them is categorised with a directed edge. Using a URI to name each part of a triple allows for structured and semi-structured data to be mixed and shared across different applications [40]. This triple structure will re-occur in later chapters. One format for storing these triples is *N-Triples*, a line-based, plain text format for encoding an RDF graph that is designed to be easy to parse and generate [1]. The simplest triple statement is a sequence of `(subject, predicate, object)` terms, for example:

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf>
  <http://example.org/#green-goblin> .
```

Listing 2.1: N-Triple example.

A serialisation similar to N-triples is *N-Quads*, which allows encoding multiple graphs. A simple example statement is the following, which is a sequence of `(subject, predicate, object)` terms and an optional blank node label or IRI labeling what graph the triple belongs to [6]:

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf>
  <http://example.org/#green-goblin> <http://example.org/graphs/spiderman> .
```

Listing 2.2: N-Quad example.

Knowledge Graphs (KG), also known as semantic networks, are used in many domains and have many definitions. One possible definition is that a KG represents a network of real-world entities expressed as triples and describes their relationships [14]. Figure 2.4 is a simple example of a KG. One example of an agency that often uses KGs is the National Aeronautics and Space Administration (NASA)[7]. Researchers and domain experts can enrich KGs by manually inserting information themselves or inferring new knowledge by mining relationships from multiple sources. However, enriching a KG requires experience with the technologies mentioned above, meaning that domain experts often need assistance from Semantic Web experts to do so.

[7]See `https://ntrs.nasa.gov/search?q=knowledge%20graph`

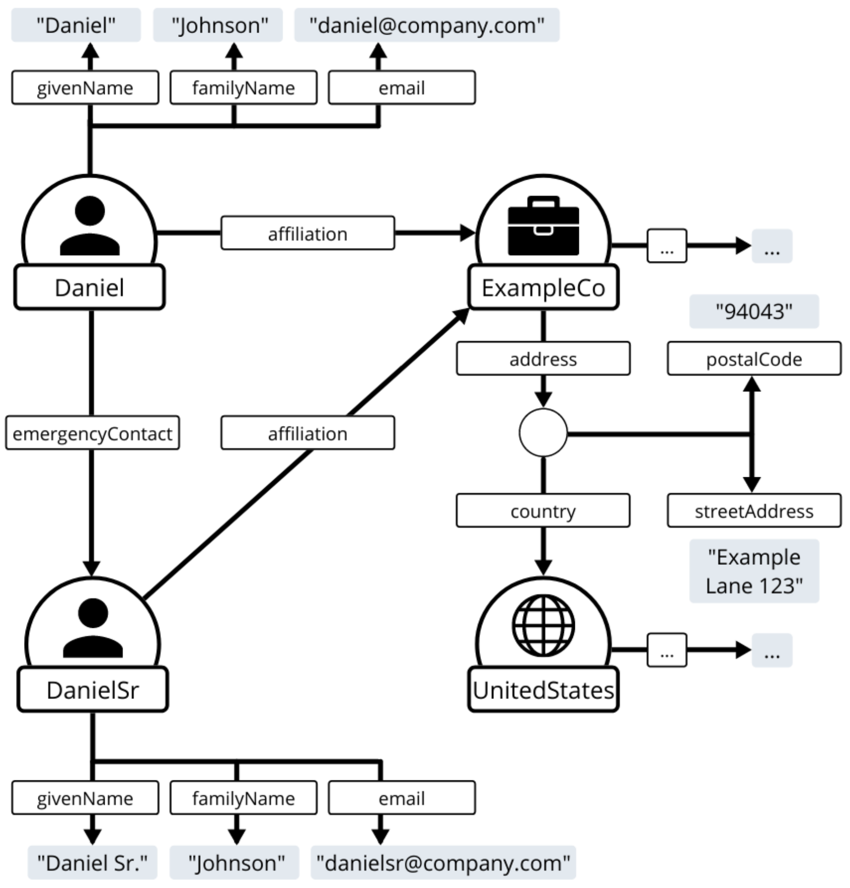


Figure 2.4: An example Knowledge Graph (KG).

RDF Schema (RDFS)[8] is a general-purpose language for representing simple RDF vocabularies on the Web. Web Ontology Language (OWL) builds on RDFS and is a computational logic-based language designed to convey complex knowledge about things, groups, and relationships. This language allows for defining structured ontologies that enable richer integration and interoperability of data since OWL is open and extensible and can be distributed across numerous systems [12]. Computer programs can use OWL documents, known as *ontologies*, e.g. to make implicit knowledge explicit or verify the consistency of expressed knowledge [39].

Shapes Constraint Language (SHACL) is the language for validating aforementioned RDF graphs against a set of conditions, provided as shapes and other constructs expressed in the form of an RDF graph. Possible conditions or constraints include that a specific shape property can only exist at most one time or that a certain string should match a given regular expression[9], using the following constraints respectively:

```
sh:maxCount 1 .
sh:pattern '^B' .
```

Listing 2.3: Two simple examples of SHACL constraints.

[8]RDFS – https://www.w3.org/TR/PR-rdf-schema/
[9]"^B" means that the string should start with “B”.

Note that the prefix `sh:` is an abbreviation for `http://www.w3.org/ns/shacl#` in the code examples throughout this dissertation.

In SHACL, RDF graphs are called *shape graphs*, while RDF graphs validated against a shape graph are called *data graphs*. SHACL shape graphs are used to validate the data graphs and are considered a description of the data graphs that satisfy its conditions. Validation takes a data graph and a shapes graph as input. It generates a validation report that indicates the conformance and all the validation results, optionally providing structural information on how the violations in the data graph can be identified and fixed. Besides validation, these descriptions could be used for building user interfaces and various other purposes [22]. Structural information expressed by SHACL is a significant aspect of the implementation in chapter 5. Another language for describing and validating RDF nodes and graph structures is ShEx [17]. SHACL and ShEx have similar goals and features, like the usage of shapes, node and property constraints. ShEx emphasises human readability with its compact syntax that follows traditional language design principles [44]. However, ShEx is not a W3C standard and will therefore not be used in this dissertation.

SPARQL Protocol And RDF Query Language (SPARQL)[10] is a set of specifications that provide languages and protocols to query and manipulate RDF graph content on the Web or in an RDF store [33]. It can be used to express queries across diverse data sources and can query graph patterns, support aggregation, subqueries, negation, create values by expressions, extensible value testing, and constraining queries by source RDF graph. Listing 2.4 is an example of a simple SPARQL query, intending to find the title of a book from a given data graph.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/> .
SELECT ?title
WHERE {
  <http://example.org/book/book1> dc:title ?title .
}
```

Listing 2.4: A simple SPARQL SELECT query.

The result of a SPARQL query is either a result set or an RDF graph. SPARQL has four query forms to form result sets or RDF graphs, including SELECT and ASK. ASK queries are used to test whether or not a query pattern has a solution but do not return any information about possible solutions [33]. Listing 2.5 is an example query used to determine an entity named "Alice" exists in the given data graph.

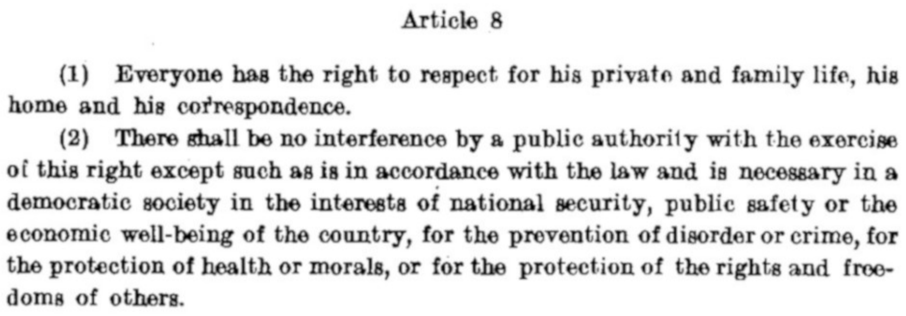
```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/> .
ASK  ?x foaf:name "Alice" .
```

Listing 2.5: A simple SPARQL ASK query.

[10]SPARQL – `https://www.w3.org/TR/sparql11-query/`

In the context of forms, SPARQL updates are used to execute the user's changes in the form to the actual data. SPARQL ASK queries can also be used to check certain information about the requesting user to determine if they should be granted access to the requested resource. This practice is further explored in chapter 5.

2.2 GDPR



Article 8

(1) Everyone has the right to respect for his private and family life, his home and his correspondence.

(2) There shall be no interference by a public authority with the exercise of this right except such as is in accordance with the law and is necessary in a democratic society in the interests of national security, public safety or the economic well-being of the country, for the prevention of disorder or crime, for the protection of health or morals, or for the protection of the rights and freedoms of others.

Figure 2.5: Article 8 from the official original European Convention on Human Rights text, 1950.

"Everyone has the right to respect for his private and family life, his home and his correspondence." This article includes the right to privacy in the 1950 European Convention on Human Rights (see fig. 2.5[11])[9]. With the vastly growing number of applications and the increasing amount of data on the Web, the right to privacy is a rising concern. The European Data Protection Directive from 1995 [10, 15] initially regulated the processing of personal data within the European Union (EU), incited by the fast progression of technology and the invention of the Internet. Each member state implemented their own law based on this ruling, which established minimal data privacy and security standards. However, considering the rapid evolution of technology and the accompanied amount of personal data usage on the Web, this directive was quickly proved insufficient. Banner ads started to appear online, financial institutions offered online banking, and social media applications gained popularity. These rapid advancements caused the need for "a comprehensive approach on personal data protection" [47].

The enforcement of the General Data Protection Regulation (GDPR) in May 2018 adds some new principles, defining rules regarding the protection of processing personal data and its free movement for a natural person. It aims to address several issues, such as creating a greater level of uniformity by eliminating inconsistencies in national laws, providing better privacy protection for individuals and better addressing contemporary privacy challenges created by the wide usage of the Internet and its numerous applications. This regulation applies to personal data processing, both automatically and as part of a filing system, even

[11]Source: `https://www.echr.coe.int/Documents/Archives_1950_Convention_ENG.pdf`

when transferred outside the EU. Europe signals its firm stance on privacy and security with the introduction of the GDPR and issuing high fines for violations, which can mount up to €20 million or 4% of the company's turnover [47].

For the average user, the introduction of the GDPR was most noticeable by the sudden prominent presence of dialogue boxes on visited websites requesting the user's "consent" for the usage of personal data for multiple purposes. From product usage and actions on Web pages to HTTP cookies, businesses gather user information and personal data daily.

The GDPR defines a set of rules organisations and businesses have to comply with when processing any personal data since the processing of personal data is only legal under a set of circumstances. Since the GDPR is such a vast set of rules, it can be challenging for a company to ensure that their applications and data usage comply with the regulation. Therefore, it would be helpful to automate this process and reduce its complexity. Since this regulation is too vast to cover every discussed aspect, this dissertation focuses on consent in the GDPR. The remainder of this section will thus expand on this facet. As an example, the data subject must give explicit, unambiguous consent to process the data, or the processing is necessary to comply with a legal obligation or perform a task in the public interest.

There are strict new rules about what constitutes a data subject's consent to process their information. Consent must be "freely given, specific, informed and unambiguous", compared to the earlier definition of simply being "unambiguous". The controller should keep evidence of consent, and any given consent can be withdrawn by the data subject whenever they want. Requests for consent must be presented in clear and plain language and distinctly distinguishable from other matters. For consent to be "specific", the data subject must be correctly notified about what kind of data will be processed, ranging from "name and date of birth" to different kinds of "data provided by cookies" (e.g. for targeted ads), as often prompted on websites nowadays.

A list of term definitions used throughout both the GDPR [11] and this dissertation can be found in the glossary.

CHAPTER 3

RELATED WORK

This chapter describes the state of the art technologies and related research on automatic form generation and access control. Related work on form generation regarding editing Knowledge Graphs in the Semantic Web is described in the first section. The second presents previous research on the topic of access control in the Semantic Web, as well as considered vocabularies.

3.1 Form generation

In the context of improving the ease of conforming to the GDPR in the Semantic Web, this section will explore previous attempts at generating forms on top of Knowledge Graphs and related tools and publications. Due to the size of the Semantic Web and the immense amount of available data, manually creating a form for each graph would require a tremendous manual effort. Automating this process and enabling form generation on top of a graph would simplify this procedure for all users.

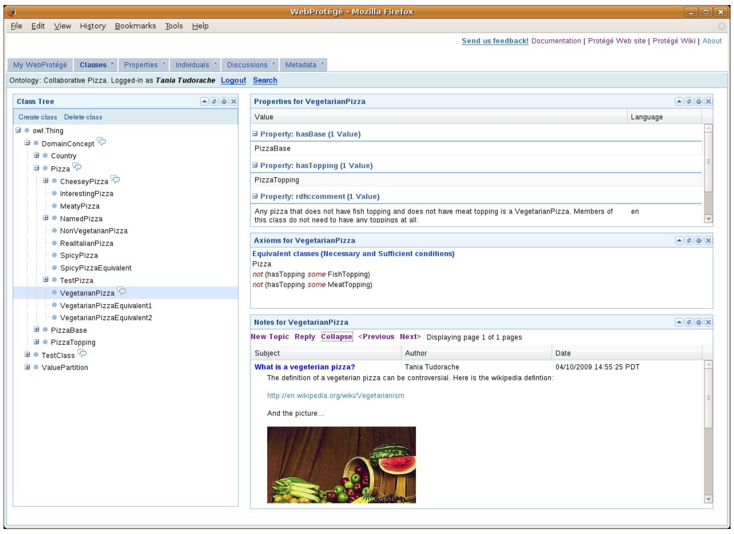
This section first discusses Resource Description Framework (RDF) authoring tools, continuing with other related work.

3.1.1 RDF Authoring Tools

In recent years, multiple form-based authoring tools have been proposed. Since Resource Description Framework (RDF) is the standard for information representation in the Semantic Web, this section will focus on RDF authoring tools, from mature ones before the introduction of Shapes Constraint Language (SHACL) to more recent publications from the past year.

One resource often mentioned in work related to form generation is *Protégé* [27], a mature graphical tool for ontology editing and knowledge acquisition. This tool is targeted towards people familiar with languages used in the Semantic Web. Its primary purpose is to create and validate the ontology schema. Instances can also be created, but this process requires detailed knowledge of RDF and Web Ontology Language (OWL).

In the Web version called *WebProtégé*[1], pictured in fig. 3.1, the created ontology can be either public or private. In the former case, the ontology as a whole can be shared with a select number of other users, who can be allowed either write or read access. This coarse-grained access control is one step towards giving explicit consent as to who can access specific resources.



Ontology: Collaborative Pizza. Logged-in as ***Tania Tudorache***

Property: hasBase (1 Value)
PizzaBase

Property: hasTopping (1 Value)
PizzaTopping

Property: rdfs:comment (1 Value)
Any pizza that does not have fish topping and does not have meat topping is a VegetarianPizza. Members of this class do not need to have any toppings at all. en

Equivalent classes (Necessary and Sufficient conditions)
Pizza
not (hasTopping *some* FishTopping)
not (hasTopping *some* MeatTopping)

Displaying page 1 of 1 pages

What is a vegeterian pizza? Tania Tudorache 04/10/2009 14:55:25 PDT
The definition of a vegeterian pizza can be controversial. Here is the wikipedia defintion:

http://en.wikipedia.org/wiki/Vegetarianism

And the picture…

Figure 3.1: WebProtégé.

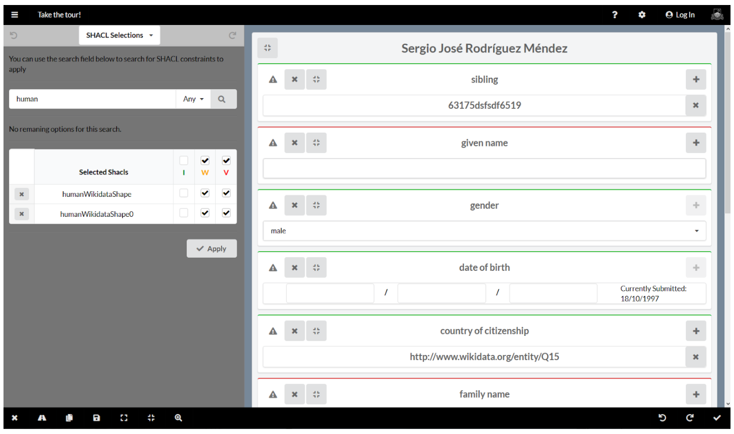
Another example of form-based RDF data-editing is *RDForms*[2], whose main task is to facilitate the construction of form-based RDF editors in a web environment. It relies on a templating mechanism that describes the HTML form generation and the mapping from specific RDF expressions to corresponding fields. Additionally, it can be used to present and validate RDF graphs.

FORMULIS [23] is a data-driven RDF editor, making input suggestions based on the data currently filled in. Their goal is to make the process easier for novice users by hiding RDF notations and offering guidance by dynamically computing intelligent filling suggestions. Additionally, form fields are suggested dynamically based on the fields and values already entered, which improves usability. The forms can be nested as deeply as necessary to create multiple interlinked entities at once. FORMULIS allows editing RDF data using generated forms and automatically generated form fields, allowing for easier and supported enrichment of Knowledge Graphs. However, FORMULIS is more targeted towards making dynamic input and field suggestions based on available data.

A recent publication is called *Schímatos* [49], a Web application to create and edit RDF data constrained and validated by shapes. Users can create and edit instance assertions, as well as create and edit shapes. The application allows to automatically generate Web forms from SHACL shapes, shown in fig. 3.2. The user can edit the generated Web form

[1]WebProtégé – `https://webprotege.stanford.edu/`
[2]RDForms – `https://rdforms.org/`

within the tool and update their SHACL definition using SPARQL updates. Some features, like adding new predicates with constraints to the shape and thus a new form to the field, require knowledge of the underlying languages from the user. Input validation is applied using SHACL constraints, like limiting values between certain boundaries and conforming to specific patterns.



You can use the search field below to search for SHACL constraints to apply

human

No remaning options for this search.

Selected Shacls	I	W	V
humanWikidataShape		✓	✓
humanWikidataShape0		✓	✓

Sergio José Rodríguez Méndez

sibling: 63175dsfsdf6519

given name:

gender: male

date of birth: / / Currently Submitted: 18/10/1997

country of citizenship: http://www.wikidata.org/entity/Q15

family name

Figure 3.2: Schímatos.

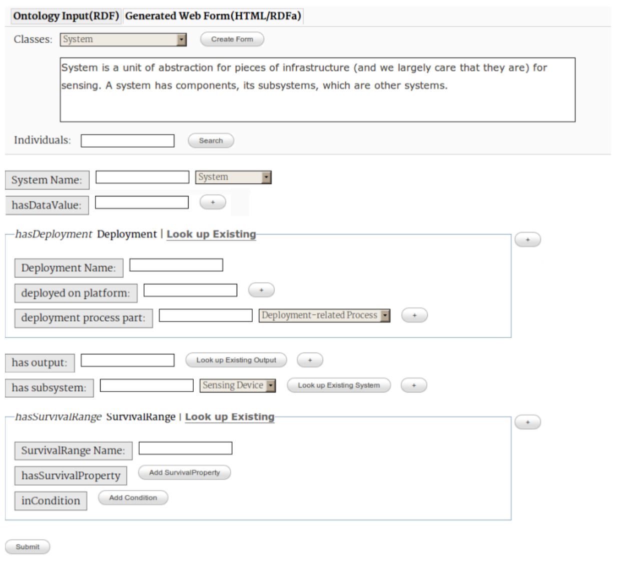
At the time of writing, the researchers behind Schímatos are working on redesigning internals for the form generation. This project is a form generator for SHACL constraints compliant to the W3C SHACL standard [48]. It is currently still in progress and has no related publication yet, but worth mentioning nonetheless because of the focus of generating fields based on SHACL constraints.

3.1.2 Supporting Tools & Mechanisms

The resources described in this subsection are different from the previous ones because they are not specific to RDF or do not generate forms themselves.

A Web service called *ActiveRaUL* [5, 18] allows the automatic rendering of Web form-based user interfaces from any input ontology (see fig. 3.3). Therefore, it is not bound to RDF usage and more versatile. ActiveRaUL operates on a model defined according to the RDFa User Interface Language (RaUL), a UI ontology. However, this tool was created before the introduction of SHACL in 2016, which means that for generating a form, the ontology assertions need to be interpreted as rules, violating the aforementioned open-world assumption of the Semantic Web. Schímatos, a publication that was described in the previous section and will be mentioned again later, makes use of ActiveRaUL in its implementation.

A resource worth mentioning that does not generate the RDF forms itself is the *Annotation Profile Model* [29]. This model is a configuration mechanism from which annotation tools



System is a unit of abstraction for pieces of infrastructure (and we largely care that they are) for sensing. A system has components, its subsystems, which are other systems.

Figure 3.3: ActiveRaUL.

can be automatically generated. It consists of a data capturing part and a presentation part, called the Graph Pattern Model and Form Template Model, respectively. The Graph Pattern model is both a query and a template language, heavily inspired by SPARQL and responsible for capturing and creating subgraphs of triples. On the other hand, the Form Template Model is a tree with references to variables of the Graph Pattern Model, providing order, grouping and deviations. The term “annotation” implies the human authoring of metadata. They mean to offer support for automatic generation of user-friendly form-based editors, depending on which role the user has: *annotation profile author*, *annotation profile facilitator*, or *end-user*. Annotation Profiles mainly target simplifying the editing process for end users. The Annotation Profile Model can thus form a basis for implementing form generation to which access control can be applied.

Lastly, *Fresnel* is a browser-independent presentation that provides an RDF vocabulary to encode information about how to present Semantic Web to users, what content to show and how to show it [32]. The vocabulary contains core RDF display concepts to promote the exchange of presentation knowledge. Nevertheless, though Fresnel aims to present Semantic Web content in a human-readable way, it does not generate forms.

3.1.3 Conclusion

Various publications already approached the practice of form generation in the context of the Semantic Web. Some of these publications are not applied to data shapes because they were published before the introduction of shape constraint languages like SHACL and ShEx. Only WebProtégé has access restrictions implemented, albeit coarse-grained, in the context of only being able to restrict a whole document compared to different specific resources.

This dissertation will focus on applying fine-grained access control on forms generated on top of Knowledge Graphs. Since Schímatos is a recent publication that automatically generates forms for RDF data from SHACL shapes, this application will be a foundation for the implementation.

3.2 Access Control

To comply with the GDPR and its regulation concerning the subject's consent and processing of their data, the data subject should give “specific and unambiguous” consent [47]. In other words, the subjects should specify under which exact conditions their data can be accessed. For example, specific colleagues or people with particular requirements might access a confined part of their personal data. Additionally, the subject also has to specify which actions are permitted: the user might be allowed to modify and delete some data or only to read it. *Access control* refers to mediating access to resources on the basis of identity and is generally policy-driven [36]. In this context, access control would be implemented as defining which (parts of the) data can be accessed, for what purposes, and by whom. This subsection describes related publications and goes deeper into vocabularies concerning access control in the Semantic Web.

As a starting point, Kirrane et al. [21] classify more than 70 articles based on privacy, security and policy issues to identify common trends and research gaps. Regarding effectiveness, for example, both access control enforcements and administration need to be effective in both performance and correctness. Unfortunately, there was no general access control benchmark available at the time of writing. For this thesis, this survey was mainly a starting point for gathering related work on the topic of access control. The identified shortcomings in state of the art, related publications and authors referenced in this survey were further explored to determine the direction of this dissertation.

Various methods have been explored in applying access control to data in the Semantic Web. One can make an initial distinction between coarse-grained and fine-grained access control considering the data that is being accessed. The former often means that access control is applied to the whole document or graph, whereas the latter indicates that it can be applied to specific triples or resources. On the other hand, there are different paradigms to implement access control. Role-based access control grants permissions by assigning one or multiple roles to users. On the other hand, attribute-based or policy-based access control uses policies combining attributes, like user or resource attributes.

For example, Padia et al. [28] describe a framework to support attribute-based fine-grained access control, which is done by representing the policies in a triple-based format. The access policies are enforced by rewriting the user's query before execution. They explore its feasibility and present an initial analysis on the relationship between access control policies, query execution time and size of the RDF dataset.

Some more recent publications take into account the GDPR and its obligations regarding processing and consent. For example, Kirrane et al. [20] present the *Scalable Policy-aware Linked Data Architecture For Privacy, Transparency and Compliance* (SPECIAL) consent, transparency and compliance demo system architecture. It uses Semantic Web technologies for the expression and evaluation of information for GDPR compliance. In addition, their system can be used to automatically check if existing data processing and sharing comply with the GDPR. This system gives data subjects more control and enables data controllers and processors to comply based on usage policies and events.

Vocabularies

As mentioned in chapter 2, vocabularies define the concepts and relationships of certain domains. Different vocabularies were considered to implement the access control in this dissertation. Previous research has been done and vocabularies have been created for expressing privacy preferences and consent. Since there are many available, practical options to use in this context, creating a new vocabulary is out of the scope of this dissertation. This thesis will use a previously defined vocabulary to express the authentication specifications, aiming to focus on applying this access control to shapes and not reinvent the wheel. The most relevant vocabularies will be discussed in this section.

An interesting option is *GConsent*, a consent ontology by Pandit et al. [30] based on an analysis of modelling data requirements related to the consent lifecycle for GDPR compliance. This ontology allows the modelling and representation of information related to compliance in an extensible and comprehensive manner. Most other ontologies are dated from before the employment of the GDPR, meaning that they do not take into account the entire lifecycle of consent, including withdrawal or modification. This ontology also includes distinctions in concepts like *Purpose*, *Medium* and *PersonalDataCategory*, which are meaningful and necessary in real-world applications but will not be considered in this thesis.

Similarly, Fatema et al. [16] propose their *Consent and Data Management Model* (CDMM). This consent ontology forms a semantic model of consent, making it specific and unambiguous as required by the GDPR (see section 2.2). They aim to improve the integration of data management across different information systems and helping controllers to demonstrate compliance. Their vocabulary offers definitions for *Purpose*, *Legal Basis* and *Data Controllers*, as well as *Consent* with expiry, withdrawal and provision. These concepts which will not be discussed in detail in this thesis since they are out of scope. Since GConsent and CDMM are more recent and W3C does not yet recommend their usage, they will not be used in this thesis.

With *Data Privacy Vocabulary* (DPV), Pandit et al. [31] aim to set a basis for establishing interoperable standards, addressing several gaps by complementing existing W3C standards. They provide a comprehensive and standardised way of annotating privacy policies, consent receipts and records of personal data handling. Again, the annotation of policies and consent receipts are not the focus of this dissertation.

A more mature option is the *Web Access Control* (WAC) vocabulary, which grants access to a whole RDF document based on the user-defined Access Control List (ACL). However, it does not provide fine-grained privacy measures specifying complex restrictions to access the data. *Social Semantic SPARQL Security for Access Control* (S4AC) [37] and *Privacy Preference Ontology* (PPO) [34, 35] both provide fine-grained privacy measures for the Semantic Web up to triple level. Fine-grained access allows the application of restrictions to a resource, statement or graph instead of being limited to restrictions on RDF graph level, which allows for giving the specific consent required by the GDPR. They either reuse concepts from or build on top of WAC. Both use SPARQL ASK queries for representing the access conditions.

The *SHI3LD* framework [7, 8] provides an authorisation mechanism for RDF stores, while considering the mobile context in which data consumption occurs. Therefore, it is grounded on two ontologies: S4AC deals with core access control concepts and PRISSMA focuses on the mobile context [8]. Considering the mobile context is out of the scope of this thesis.

Conclusion

Fine-grained access control has already been implemented to data in the Semantic Web by rewriting the user's query before execution. However, access control techniques have not been applied to shape constraint languages nor directly to forms or form generation. As discussed in the previous section, there are already multiple applications that allow the automatic generation of forms on top of RDF data. Applying access control to this form generation might allow more comprehensive adoption of the Semantic Web by supporting data subjects and data processors in complying with the GDPR.

Not to reinvent the wheel, multiple vocabularies were considered to implement access control in this dissertation. While younger ones are based on the definition of consent in the GDPR, the more mature W3C recommendations also provide fine-grained privacy measures. More specifically, PPO and S4AC will be further explored in Chapter 5.

CHAPTER 4

RESEARCH QUESTIONS AND METHODOLOGY

In this chapter, we define the research questions and related hypotheses identified through the research challenges and existing work described in the previous chapter. This thesis will comprise both form generation, as well as GDPR compliance, in the context of a person's private data and giving explicit consent.

4.1 Research Questions

As summarised in the previous chapter, little research can be found on the application of access control and consent on the combination of shape constraint languages and form generation. Recent work on vocabularies does take into account the GDPR and its restrictions, but they do not consider applying the expressed consent and access control to shapes and forms generated on top of these shapes. Bridging the gap between this framework and available implementations would benefit processors and controllers that should comply with this regulation and the data subject on giving their specific, explicit and unambiguous consent.

These findings lead to the research question that summarises the research topic of this dissertation:

Research Question. *How can we comply with the GDPR for giving explicit and specific consent while automatically generating forms on a Knowledge Graph?*

This research question can be split into two subquestions:

Subquestion 1. *How can we apply access control to a shape to satisfy the GDPR in the context of giving consent?*

Subquestion 2. *How can we apply the subject's explicit consent to a form generated on a Knowledge Graph?*

This first question focuses on applying the subject's authorisation preferences on the SHACL shape. On the other hand, the second question concentrates on applying these authorisation specifications on the form that is being generated on top of the relevant Knowledge Graph.

4.2 Research Methodology

Following the structure of the research questions, a hypothesis will be formulated for each of the subquestions.

Fine-grained access control is preferred over coarse-grained to precisely determine which resources the user can access to comply with the GDPR in the context of specific and explicit consent, which leads to the following hypothesis:

Main Hypothesis. *A subject's data shape can comply with the GDPR's definition of consent by using a fine-grained specification of access control permissions, as defined by the data subject using Privacy Preference Ontology (PPO).*

An intermediate model was created to map the subject's specific consent to a more general authorisation specification. In its turn, the model can be applied to the SHACL shape and the form generation. This leads to the following hypotheses:

Hypothesis 1. *Access control can be applied to a SHACL shape using an intermediate model to modify and add constraints to the given shape with nested properties.*

Hypothesis 2. *A subject's explicit and specific consent can be applied to form generation without nesting using an intermediate model that adds access-related properties to the JavaScript objects for each field.*

Schímatos will be used as a starting point for the form generation, as this is a recent publication that implements form generation and input validation. Schímatos also uses the SHACL standard to express conditions for validating Resource Description Framework (RDF) based Knowledge Graphs. Privacy Preference Ontology (PPO) was chosen as a starting vocabulary because it is a W3C recommendation that provides fine-grained privacy measures up to triple level. In addition, Social Semantic SPARQL Security for Access Control (S4AC) is similar to PPO in terms of application and specifications. It will thus be considered as a second language to test the intermediate model created for applying the authorisation specifications.

CHAPTER 5

IMPLEMENTATION

This chapter describes the steps taken and considerations raised for the application of access control to SHACL shapes in the context of form generation using Schímatos. The first section describes the starting point of the implementation, followed by the implementation of the authorisation specifications. This section will dive deeper into the goals, vocabularies and methodologies.

5.1 Schímatos

As mentioned in the chapter on related work (chapter 3), Schímatos [49] is a recently published form-based Web application that can be used to create and edit Resource Description Framework (RDF) data constrained and validated by shapes, as well as edit the shapes themselves. It is available to download as an HTML+CSS/JavaScript package[1] and can be run in the browser[2].

The main components in the architecture of Schímatos (fig. 5.1) are the *Sidebar Management*, the *Context Management*, the *Form Display* and the *triplestores*. The Sidebar Management allows the user to select data targets and shape graphs, which are queried from the corresponding triplestores using SPARQL Protocol And RDF Query Language (SPARQL) queries. The triplestores return the data in JavaScript Object Notation (JSON), a standardised data format. This data is stored in the Context Management and used to generate objects to create a form. Finally, the form is generated and visualised using the Form Display and the *Input Mappings*. At this point, the user can use the generated form to make modifications to the selected shape and target data.

In order to achieve a generated form for a given target using a specified shape, the user follows the next steps. First, the user queries a data graph to select the desired data target using the *Target Selection*. Next, the *SHACL Selection* is used to determine which conditions the chosen data graph should satisfy. The selected Shapes Constraint Language (SHACL) shape determines what the form should look like, including the form fields and their

[1] Schímatos package – `http://schimatos.org`
[2] Schímatos online – `http://schimatos.github.io`

validation. On the other hand, the selected data target is the person whose data will be represented in the generated form, so the user can then update this target person's data. Both the data graph and the shapes graphs are available from configurable triplestores. In addition, the sidebar component of the application contains *Target and SHACL Selection* components (fig. 5.2a and fig. 5.2b), which the user can utilise to select the desired data targets and SHACL shapes.

The triples obtained from these triplestores are transformed into JSON, and saved into the *Form and Storage Contexts*. The *Form Context* transforms the JSON objects to JavaScript objects with additional properties to generate each field in the *Form Display*. After the form on the selected entity has been generated using the selected SHACL shape(s), any current information on the selected target is visualised, and the user is prompted to fill out any missing information. Once the form is completed, the obtained data, and the class and datatype annotations, are submitted to a Knowledge Graph over a configurable SPARQL endpoint. The outline of the application's architecture can be found in fig. 5.1

Initially, Schímatos did not have access control implemented. This thesis aims to combine conforming with consent in the GDPR and form generation on top of KGs. Therefore, the data subject could be enabled to express their consent to accessing, processing and modification of specific parts of their data in Schímatos. This consent could then be applied to the generated forms. The red locks in fig. 5.1 indicate the two Access Control options implemented by this dissertation. These options, namely applying access control to the form in Schímatos and the SHACL shape using an intermediate model, will be described in section 5.2.4.

Forms in the application are automatically generated from SHACL shapes, automatically choosing a correct input field type and adding input validation. For example, the `sh:in` constraint generates a dropdown field with a limited set of value options to choose from, while `sh:pattern` automatically applies the desired regular expression to the user's input. On the other hand, the value range constraints `sh:minCount` and `sh:maxCount` allow or disallow the user to add new values to the property or indicate that at least a certain number of values is required for the given property for the form to be valid. The forms themselves and their SHACL definitons can be edited in the tool using SPARQL updates. The JavaScript package and the online version have a local instance of Wikidata[3] set as the default data graph and reuse existing ShEx files translated to SHACL.

[3]Wikidata – `https://www.wikidata.org/`

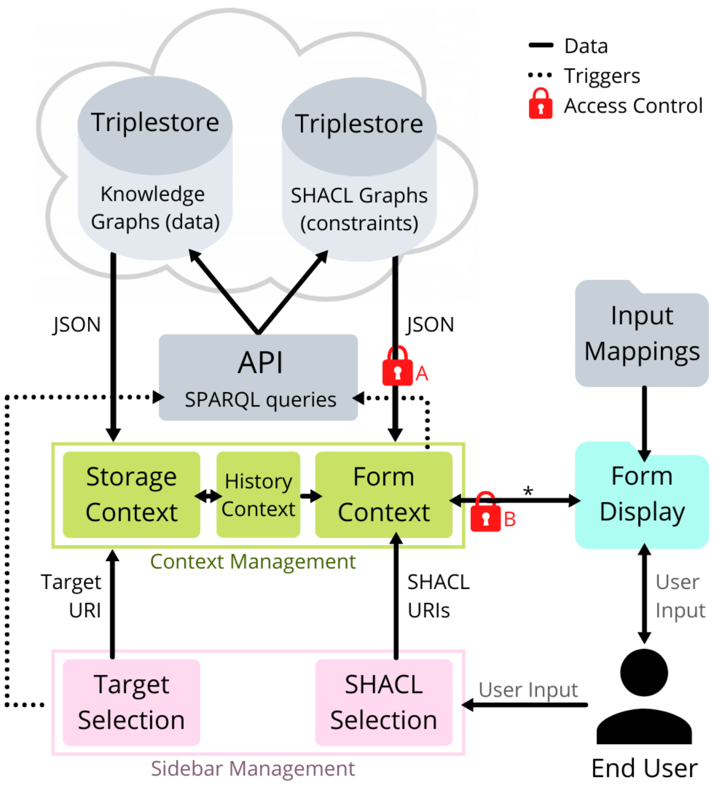


Figure 5.1: Main outline of the architecture of Schímatos. Adapted from [49].

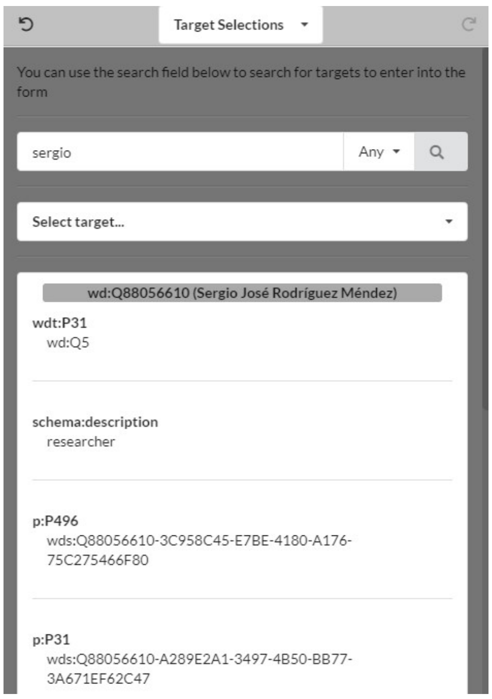
5.2 Authorisation Specifications

This section describes the parts implemented to achieve access control in the context of consent, applied to forms generated in Schímatos. The first section will clarify some goals. The following section will describe some of the priming steps, followed by an explanation of Privacy Preference Ontology (PPO) and Social Semantic SPARQL Security for Access Control (S4AC). Finally, this section will conclude with how the interpretation of these vocabularies led to an intermediate model for applying the expressed authorisation specification.

5.2.1 Goals

This dissertation aims to implement a data subject's explicit and specific consent to forms generated on top of Knowledge Graphs. This section will explore two methods of applying a user-created authorisation specification to SHACL shapes using an intermediate model. This model will act as a layer between the expressed consent and the target data or form by determining which access applies to which parts of the data for a given requester.

First, applying the access control to the form generated in Schímatos is developed as a starting point. Consequently, this approach is then used to create a more generalised



You can use the search field below to search for targets to enter into the form

sergio

wd:Q88056610 (Sergio José Rodríguez Méndez)

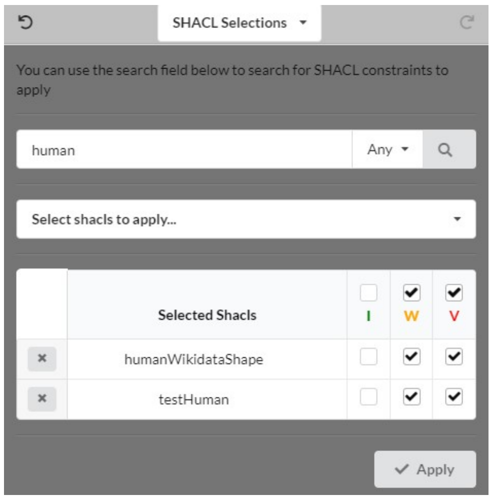
wdt:P31
wd:Q5

schema:description
researcher

p:P496
wds:Q88056610-3C958C45-E7BE-4180-A176-75C275466F80

p:P31
wds:Q88056610-A289E2A1-3497-4B50-BB77-3A671EF62C47

(a) Target Selection component.



You can use the search field below to search for SHACL constraints to apply

human

	Selected Shacls	I	W	V
		☐	☑	☑
×	humanWikidataShape	☐	☑	☑
×	testHuman	☐	☑	☑

(b) SHACL Selection Component.

Figure 5.2: Sidebar components in Schímatos.

method by applying the model to the underlying SHACL shape. The shape's constraints are then modified and enriched to accommodate the expressed consent and include the required access control. This generalised approach could later be used for other purposes and possibly different applications for generating forms. The application to the triples of the SHACL constraints is visualised as access control option A in fig. 5.1, while option B is the application to the objects that are used to generate the form. This intermediate model will be created based on PPO to act as an abstraction layer between SHACL and the authorisation specification.

5.2.2 First Steps

Schímatos does not have any privacy or authorisation specifications implemented. The first step is to create a simple authentication system, allowing to check who is currently using the application. This system is based on a Resource Description Framework (RDF) file parsed to create a fixed set of profiles. A simplified example of such a profile can be found in listing 5.1. *Schema.org* was chosen as the vocabulary to express the users' properties since it is a widely used collaborative community activity for creating and maintaining schemas for structured data on the Internet[4].

Saving user profiles in Turtle serialisation allows us to parse them into a local RDF triple store, which enables directly executing the ASK queries expressed in the authorisation specification

[4]Schema.org – `https://schema.org/`

on the profile of the current user. The fact that the ASK queries can be executed directly simplifies the implementation. Instead of an RDF store, a *Labeled Property Graph* is another storage option that can be queried. RDF stores are more widely used and implemented since RDF is a W3C standard for data exchange in the Web. The *RDFStore*[5] package allows creating an RDF store using the Turtle serialisation and querying its entries with SPARQL queries. Another library is *Comunica*[6], a modular knowledge graph querying framework that also allows to execute SPARQL queries in JavaScript apps. However, RDFStore has a more straightforward way of importing a local file as a store, while Comunica mainly supports using online sources as stores.

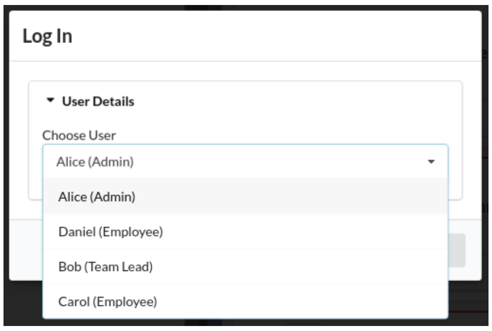
```
@prefix ex: <http://example.org/> .
@prefix schema: <http://schema.org/> .
ex:Daniel a schema:Person ;
  schema:givenName "Daniel" ;
  schema:familyName "Johnson" ;
  schema:email "daniel@company.com" ;
  schema:homeLocation [
    schema:streetAddress "Example Lane 123" ;
    schema:postalCode "94043" ;
    schema:addressCountry "US"
  ] ;
  ex:emergencyContact [
    schema:givenName "Daniel Sr." ;
    schema:familyName "Johnson" ;
    schema:email "danielsr@example.com" ;
  ] ;
  ex:ownsResources ( "https://example.org/Daniel );
  schema:hasOccupation ex:employee .
ex:employee a schema:Occupation .
```

Listing 5.1: Example user profile for an employee. Short version of listing 1 in appendix A.

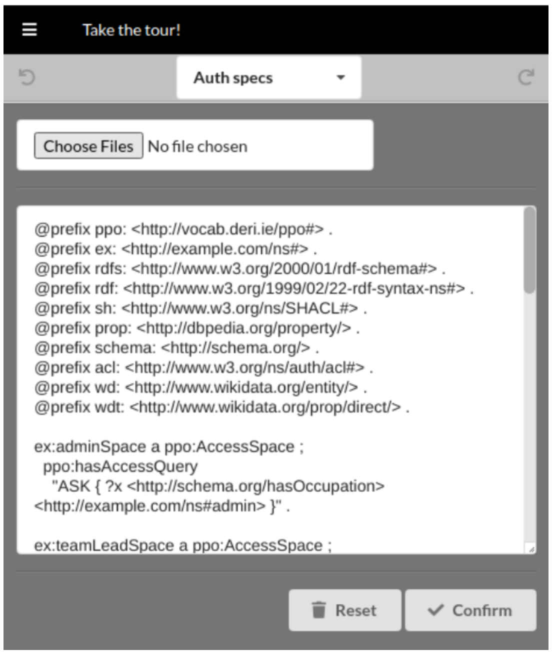
Each profile represents a person and has a given name and a unique e-mail address used to identify them. They each also have a list of owned resources, saved as URIs. In addition, other properties of this person, such as their occupation and home country, can be used in the authorisation process to determine if the current profile has access to a given resource.

While using Schímatos, the user has the option to log in by selecting a profile, visualised using their name and role, from the authentication dropdown (see fig. 5.3a). The user that is currently logged in, will be referred to as the *requester*. It is worth mentioning that the application can be used without logging in. The initial functionality without access control is thus retained to keep the original testing environment, where the user can perform any possible action to the presented form without limited access. The login mechanism allows to simulate access control from the point of view of various users with different roles. This

[5]RDFStore – `https://www.npmjs.com/package/rdfstore`
[6]Comunica – `https://comunica.dev/`



(a) The authentication dropdown.



```
@prefix ppo: <http://vocab.deri.ie/ppo#> .
@prefix ex: <http://example.com/ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sh: <http://www.w3.org/ns/SHACL#> .
@prefix prop: <http://dbpedia.org/property/> .
@prefix schema: <http://schema.org/> .
@prefix acl: <http://www.w3.org/ns/auth/acl#> .
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .

ex:adminSpace a ppo:AccessSpace ;
  ppo:hasAccessQuery
    "ASK { ?x <http://schema.org/hasOccupation>
<http://example.com/ns#admin> }" .

ex:teamLeadSpace a ppo:AccessSpace ;
```

(b) The authentication specification form.

Figure 5.3: Implemented authentication components in Schímatos.

authentication system is a simple auxiliary model used as an example to implement the actual access control for generated forms. Setting up an account system with passwords and security is outside of the scope of this dissertation.

Apart from logging in, it should also be possible to specify which privacy preferences must be taken into account while generating the forms. Since this thesis focuses on the actual application of the expressed consent, the preferred privacy specifications can be entered by the user (fig. 5.3b) and are processed in the local application. Saving these authorisation specifications in the local state simplifies the implementation process. The privacy specifications should be expressed in Turtle semantics and using Privacy Preference Ontology (PPO). Similarly to the user profiles mentioned earlier in this section, the privacy specifications are parsed from Turtle into a local RDF store, from which the intermediate model (section 5.2.4) will be built. By default, there is no privacy preference specified, retaining the original functionality provided by Schímatos. The access control implemented in this thesis is an illustration and can be turned on and off as desired.

5.2.3 Access Control Vocabularies

Before describing the intermediate authorisation specification model used in the implementation, two access control vocabularies will be discussed in more detail. Out of the various vocabularies discussed in section 3.2, Privacy Preference Ontology (PPO) and Social Semantic SPARQL Security for Access Control (S4AC) are the two mature W3C recommendations that provide fine-grained access control conditions.

Privacy Preference Ontology (PPO)

The first and most mature vocabulary that will be explored is Privacy Preference Ontology (PPO) [34]. It was developed on top of the existing Web Access Control (WAC)[45] vocabulary to provide fine-grained privacy measures, which WAC did not support. An overview of PPO can be found in fig. 5.4.

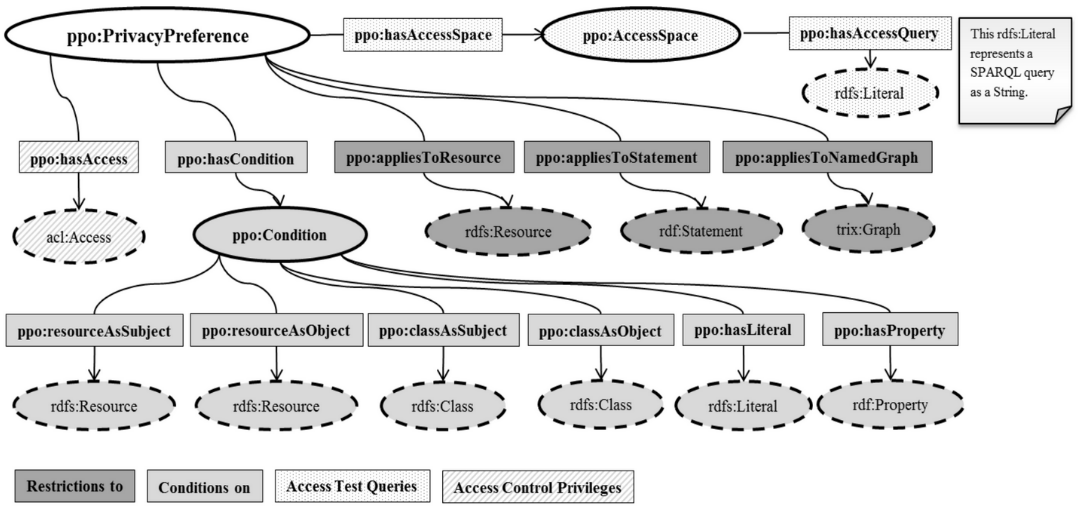


Figure 5.4: An overview of the Privacy Preference Ontology. Taken from [34].

PPO has classes for *Privacy Preference*s, *Condition*s and *Access Space*s. The main class, *PrivacyPreference*, contains properties to define multiple aspects of the privacy preference. *Condition*s are used to define restrictions within a privacy preference. For example, a condition can indicate if the privacy preference applies to statements that have a particular resource as their subject or that contain a certain literal. Indicating which requesters are granted access is defined by the *AccessSpace*. This AccessSpace contains SPARQL ASK queries with attributes and properties to satisfy. These queries are thus applied to the requester's profile and return either `True` or `False` whether the requester's information satisfies the graph pattern [34]. If the query returns a positive answer, the requester is granted access to the statement. The type of access is defined using the Web Access Control (WAC) vocabulary, which uses Access Control List (ACL) [45] Read and `Write` privileges. Lastly, each privacy preference also indicates to which resource, statement or named graph it applies and should thus be restricted.

The example query in listing 5.2 shows how a microblog post can be restricted to users that share an interest similar to the concept used to tag the post, which is Linked Data in this case [34]. Line 5 indicates to which resource the PrivacyPreference applies and the granted access is specified in line 6. Lines 7 to 10 describe the condition that the tag of the specified resource should be equal to `http://dbpedia.org/resource/Linked_Data` for the PrivacyPreference to be applicable. The AccessSpace of lines 11 to 14 defines the ASK query that is executed to determine if the requester will be granted access. In this case, the query checks if the requester is interested in the topic of Linked Data.

```
@prefix ppo: <https://vocab.deri.ie/ppo#> .
<http://www.example.org/pp2>
  a ppo:PrivacyPreference ;
  ppo:appliesToResource <http://smob.me/user/xyz/post1>;
  ppo:assignAccess acl:Read ;
  ppo:hasCondition [
    ppo:hasProperty tag:Tag ;
    ppo:resourceAsObject <http://dbpedia.org/resource/Linked_Data>
  ];
  ppo:hasAccessSpace [
    ppo:hasAccessQuery
      " ASK { ?x foaf:topic_interest <http://dbpedia.org/resource/Linked_Data> }"
  ].
```

Listing 5.2: PPO example.

Social Semantic SPARQL Security for Access Control (S4AC)

Another vocabulary is Social Semantic SPARQL Security for Access Control (S4AC) [37], a lightweight ontology that also allows specifying fine-grained access control policies. An overview of S4AC can be found in fig. 5.5.

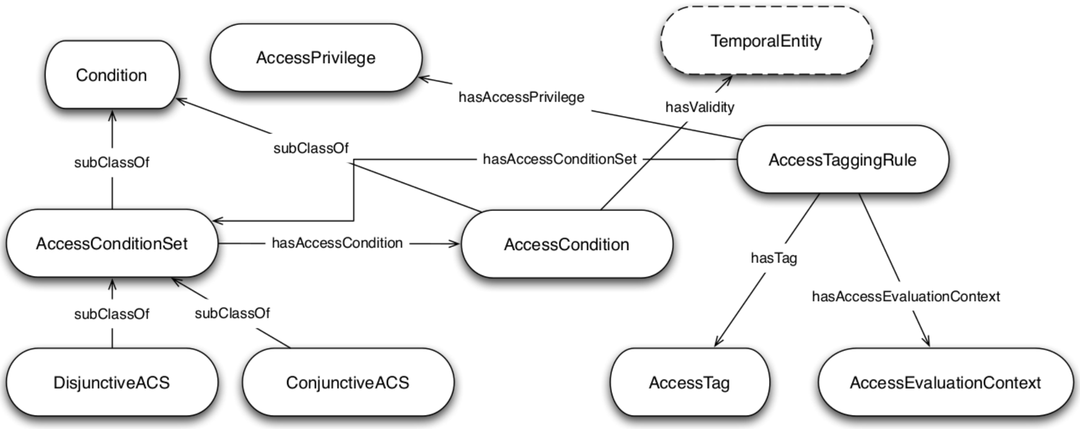


Figure 5.5: An overview of the S4AC Vocabulary. Taken from [37].

The core of S4AC is the *Access Condition* (AC), a SPARQL ASK clause that establishes which conditions should be met to gain access to a resource. An Access Condition is said to be verified if the ASK query returns `true`. Conditions can be further restrained by using *tags* and contextual information. For example, the former can indicate a social aspect of the accessing user, and the latter includes requirements concerning temporal features. An *Access Condition Set* (ACS) is a set of Access Conditions. S4AC distinguishes two subclasses: disjunctive and conjunctive ACS. A disjunctive ACS is verified if at least one of the contained access conditions is verified. For a conjunctive ACS to be verified, every contained access condition should be verified. Finally, an *Access Evaluation Context* (AEC) can be defined by binding conditions to distinct values to constrain the ASK query evaluation and provide an access evaluation context. For example,

`<?resource, <http://MyExample.net#doc>>`

binds the URI of the resource to `<http://MyExample.net#doc>` [37]. With this binding, the variable name `?resource` can be used in the ASK query, which will be expanded with a SPARQL `BIND` statement binding the variable name to its value. It is worth noting that these bindings, and thus the access conditions themselves, are not limited to the attributes of the requester. The type of privilege that is granted is defined by the *Access Privilege* and can be either `Create`, Read, `Update` or `Delete` (CRUD), the four basic functions of persistent storage in computer programming.

The rule in listing 5.3 defines a policy that only named graphs tagged “family” are constrained. `Update` access is granted to requesters if they have a `hasParent` relationship with the provider and if the resource is accessed after December 31st at 23:59 [37]. The AccessCondition of lines 6 to 16 describes the temporal validity in lines 7 to 11. The ASK query of lines 13 and 14 defines the requirement that the requesting user should have a `hasParent` relationship with the provider, who is the creator if the requested resource. The granted access is expressed in line 18. Finally, line 19 that the AccessTaggingRule applies to graphs tagged “family”.

```
1  @prefix s4ac: <https://ns.inria.fr/s4ac/v2#> .
2
3  <http://MyExample.net/expolicies>
4    a s4ac:AccessTaggingRule;
5    s4ac:hasAccessConditionSet [
6      s4ac:hasAccessCondition [
7        s4ac:hasValidity [
8          time:hasBeginning [
9            time:inXSDDateTime 2011-12-31T23:59:00
10         ];
11       ];
12       s4ac:hasQueryAsk [
13         ASK { ?resource dcterms:creator ?provider .
14         ?provider rel:hasParent ?user }
15       ];
16     ];
17   ];
18   s4ac:hasAccessPrivilege s4ac:Update;
19   s4ac:hasTag scot:Tag family@en.
```

Listing 5.3: S4AC example.

Comparison

Social Semantic SPARQL Security for Access Control (S4AC) differs from Privacy Preference Ontology (PPO) in several aspects. The former allows considering the context, like temporal validity and number of allowed accesses. Using S4AC, graphs can also be marked with sets of tags. Considering the access context and using tags to distinguish different “categories” of data would allow for additional access restriction options, but this is beyond the scope of this thesis. On the other hand, PPO does not offer dedicated tagging functionality and does not consider any access context. This makes PPO less extensive and fewer aspects have to be considered while implementing.

PPO is built upon WAC and does not distinguish different `Write` actions. While generating a form, a greater distinction is preferred when it comes to `Write` actions to determine which buttons and fields should be enabled in the form. On the other hand, S4AC uses CRUD and distinguishes `Create`, `Update` and `Delete` within the WAC `Write` class. This would allow for more nuanced implementation of access control in this thesis.

The ASK queries in PPO are applied to the requester’s profile. On the other hand, queries in S4AC are not specified to only apply to the requester’s attributes. In PPO, the resource conditions are expressed using separate clauses outside of the SPARQL query. These two definitions would therefore require different implementation approaches for checking the access conditions.

In addition, S4AC makes a distinction between disjunctive and conjunctive ACS, which PPO does not. This distinction enables more access restriction options and more flexibility, but would require more nuances in the implementation of access control in this dissertation. Based on the code example given in the publication, this thesis assumes that PPO applies conjunctive properties within a condition, while conditions within a PrivacyPreference are applied disjunctively.

This thesis will use PPO as a starting point since this ontology is more concise than S4AC. Some of S4AC’s features are out of the scope of this thesis, while others would require a higher implementation effort to achieve even more flexibility. In this dissertation, the functionality of PPO suffices for implementing access control.

5.2.4 Intermediate Authorisation Specification Model

As a means of representing authentication specifications in Schímatos, a model was created for internally parsing and applying authorisation specifications expressed by the data subject. This model is grounded on other ontologies and reuses concepts from Web Access Control and Privacy Preference Ontology mentioned above. The primary purpose of this model is to map specifications written in different privacy ontologies like PPO to this model. These specifications can then be applied to the generated form. Secondly, it aims to act as an abstraction layer between the authentication specification and the shape constraints.

Model Design

This section features a description of the intermediate authorisation specification model, along with the design choices and an illustrative example. The model is an intermediate model since it acts as a layer between the expressed authorisation specifications and the shape constraints those specifications apply to. This model is thus a layer between what is considered to be the data subject's explicit consent and the data that is being processed and modified. Privacy Preference Ontology (PPO) was used as the primary reference for the model design since the first and main focus was to apply authorisation specifications expressed in PPO to the shape constraints and the form generation.

The intermediate authentication specification model is a JavaScript object containing dictionaries of objects. The main object is the *Policy*, which applies to a resource with a given URI. Each Policy has a reference to one or multiple *UserAccessCondition*s and *DataAccessConditions*, both of which are discussed in the next paragraph. The intermediate model has a dictionary for each of these three object types.

The UserAccessConditions (UAC) are used to check if the requester has the right characteristics to be allowed access granted by the Policy. Each UAC has an associated SPARQL ASK query executed on the current user's profile to determine if the Policy applies to this user. Therefore, the data subject's consent can define which characteristics a person has to meet before gaining the associated type of access. On the other hand, DataAccessConditions (DAC) determine if the Policy applies to a certain triple or resource. Each Policy applies to data that meets the requirements described by its associated DAC. These conditions apply to data with an `rdf:subject`, `rdf:object`, `rdf:Property` or `rdfs:Literal`. Therefore, the data subject can accurately specify which exact resources should be granted the associated access type. The specifications of these two condition types contribute to the specificity and explicitness of the subject's consent.

As mentioned in chapter 2, N-Triples (`subject, predicate, object`) are not the only serialisation to express Resource Description Framework (RDF) statements: N-Quads (`subject, predicate, object, graph`) add the graph label to an RDF triple to indicate which graph the triple is considered to be a part of. Thus, if the data is expressed in N-Quads, the model could be extended to consider the context graph of the quad. Consequently, it is possible to determine whether or not the data is accessible to gain some more coarse-grained options. This would allow the data subject more freedom in the expression of their consent. However, this was not implemented yet and would be considered future work.

Mode	Allows
`Create`	create new information
Read	retrieve and read the contents
`Update`	modify existing information
`Delete`	delete information

Table 5.1: The four actions in the CRUD paradigm.

Mode	Allows
`acl:Read`	read the contents, including querying
`acl:Write`	overwrite the contents, including deleting, modifying and adding
`acl:Append`	add information to the end of it, but not remove or modify any other information
`acl:Control`	set the Access Control List for this themselves

Table 5.2: The modes of access in WAC as defined by the ACL vocabulary [45].

Finally, the access granted by the Policy is described using an internal *AccessType* object. After determining if a Policy applies to the current requester and a specific resource, the AccessType determines which type of access the data subject grants. The AccessType object thus allows the expression of explicit consent for the given action. This object represents the basic Create, Read, Update, Delete (CRUD) actions (table 5.1), as opposed to using the four access modes described by Access Control List (ACL) as classes, as seen in table 5.2 [45]. The `acl:Control` mode can be confusing, as it is an indirect way of specifying and allows only read-write mode, but not read-only. `acl:Control` allows the controller to read/write but does not deny read access to anyone else [45]. Additionally, Create, Read, Update, Delete (CRUD) operations allow for more fine-grained applications than the Access Control List (ACL) access classes and are widely used in the computer science world.

A visual representation of the intermediate model can be found in fig. 5.6. A code example can be found in listing 2 in appendix A.

Model Application

The intermediate model described in the previous section is implemented in two ways in this dissertation. Both options are built upon the same idea. In fig. 5.1, these two options are indicated as access control options A and B, respectively.

The first option is more tied to the specific implementation of Schímatos since the first goal was to apply the authorisation specification to the form generated using a SHACL shape. This first application method is based on creating an extra object that indicates which types of access are granted to each field. Schímatos uses multiple dictionaries with overarching keys in the process of generating a form. Applying the authentication specification here essentially generates a new object containing the access granted to each of the fields. This

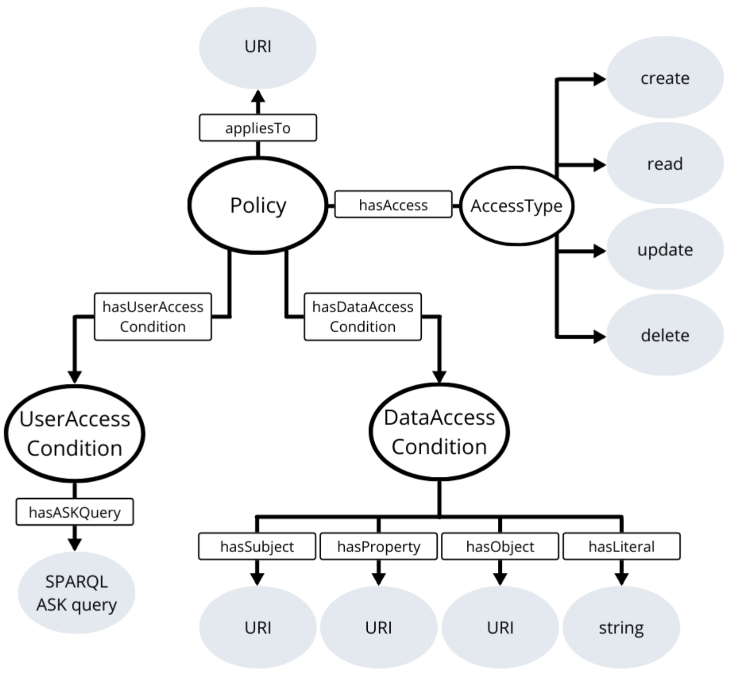


Figure 5.6: The intermediate model.

object is added to the set of objects that are taken into account while generating the form fields.

Before generating a field, the necessary information is fetched from these objects, which determines the type and other properties of the field. The granted access for this property defines whether the field is visible and enabled and if the corresponding buttons are enabled. Figure 5.7 is an example of a generated field where the requester has Read, `Update` and `Create` rights, but cannot `Delete` any values. Four distinct access options give the requester the following rights and functionality:

- canRead allows the requesting user to see the form field and its content. If this access is not granted, the user can only see the field's label, but not its value.
- can`Update` allows the requester to edit the field. If this access is granted, canRead access is assumed to be `True` as well. The field is thus visible and enabled. If canRead is granted but can`Update` is not, the field value is visible, but the field itself is disabled.
- can`Create` allows the requester to add new values to a property, as long as this property still meets its constraints. For example, if a property has a `sh:maxCount` of 2, the user will not be able to add a third value, as this would violate the original constraints. If enabled, the add-button allows for creating new values.
- can`Delete` allows the requester to delete the property's values if it has any. The delete button is enabled accordingly.

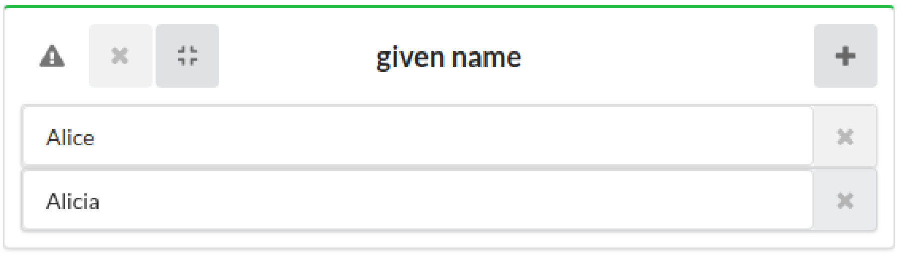


Figure 5.7: The requester has no `Delete` access to this shape property.

This first step focused on implementing the disabling and hiding of fields and their functionality in Schímatos, paving the way for a more generalised approach.

The model can be applied to the SHACL triples as a generalisation, applying the authorisation specification to the shape constraints before the form is generated. For each triple, the intermediate model determines which policies can be applied based on the requester and the triple's subject, predicate and object. If the requester does not have any access rights, the whole property is disregarded in the shape, which means that the corresponding field will not be generated. If the requester has access rights, the acquired access types are added as triples to the property's constraints. The example triple in listing 5.4 adds Read access to the `schema:givenName`[7] property.

```
<http://schema.org/givenName> <http://example.org/access> <https://ns.inria.fr/s4ac/v2#Read> .
```

Listing 5.4: Access triple granting Read access to `schema:givenName`.

The added access triples use Social Semantic SPARQL Security for Access Control (S4AC)'s CRUD classes and accompanying `hasAccessPrivilege` property to reuse predefined classes and properties from an existing vocabulary. If the requester should be allowed to see which fields they do not have access to, this method can also indicate that they have no rights instead of removing the triple from the shape altogether. Thus the field can be generated without visible values or functionality for the requester. In this case, the requester could only see the field's title and be aware that they have no access to this data.

In the actual implementation of this model application, the intermediate model is not applied directly to the shape triples because of the complexity of the implementation and the limited amount of time. Instead, they are applied to the JavaScript Object Notation (JSON) format of the SHACL shape, which are a direct conversion from the triples to a JavaScript Object acquired from the triplestore using a comprehensive SPARQL query. The triples added for access control are then part of the shape used to generate the form. So, for example, Read access can be added to the `givenName` property in listing 5.5, as seen in listing 5.6.

[7]Schema – `http://schema.org/`

```
{
  "datatype": "http://www.w3.org/2001/XMLSchema#string",
  "minCount": "1",
  "name": "given name",
  "pathType": "path",
  "path": "http://schema.org/givenName"
}
```

Listing 5.5: JSON representation of the original authentication specification.

```
{
  "datatype": "http://www.w3.org/2001/XMLSchema#string",
  "minCount": "1",
  "name": "given name",
  "pathType": "path",
  "path": "http://schema.org/givenName",
  "s4ac:hasAccessPrivilege": [
    "s4ac:Read"
  ]
}
```

Listing 5.6: Listing 5.5, now with an added access control triple.

The implementation of the form generation in Schímatos needs to be modified to consider these additional triples in the actual form generation to ensure that the buttons and fields are disabled and generated accordingly. In the current implementation, Schímatos still applies access control using the first implementation option. The modified SHACL shape from the second application option does not generate the form due to implementation limitations and time constraints.

Modified Settings

The implementation of the intermediate model allows for setting modifiers. For example, the boolean `defaultAllow` indicates if the requester has access to a triple for which no access rule has been defined.

Additionally, the boolean `strict` can be modified to change the way multiple access rules are handled. If `strict` is set to `True`, only the most restricting applicable rules are taken into account. For example, the following policies

- Allow Read+`Update` on mother and father's names
- Allow Read on father's name

would result in the granted access of option A in table 5.3. On the other hand, setting `strict` to `False` would allow the least restricting access. This case would then result in the granted access of option B in table 5.3 using the same policies. These modifiers can be adapted to the required functionalities.

Option	**Resource**	Create	Read	Update	Delete
A	mother's name	×	✓	✓	×
	father's name	×	✓	×	×
B	mother's name	×	✓	✓	×
	father's name	×	✓	✓	×

Table 5.3: Allowed access for the given example.

PPO Mapping

Privacy Preference Ontology (PPO) (fig. 5.4) is the main reference and starting point for creating the intermediate model, thus sharing multiple common aspects with this intermediate presentation. This model simplifies seven aspects of PPO in order to make them more abstract and applicable to the shapes used to generate the forms. For example, the model's `appliesTo` comprises PPO's `appliesToResource`, `appliesToStatement` and `appliesToNamedGraph`. PPO's `resourceAsSubject` and `classAsSubject` are reduced to `hasSubject`, while `hasObject` covers `resourceAsObject` and `classAsObject`. These simplifications were made because the intermediate model saves resources, classes and graphs using their Uniform Resource Identifier (URI).

Since PPO uses the Access Control List (ACL) access classes, these are mapped to the Create, Read, Update, Delete (CRUD) operations used in the intermediate model using the rules from table 5.4. So, for example, if the user has `acl:`Append privileges on a given resource they can execute `Create` and Read actions on this resource, but cannot `Update` or `Delete` any information. As mentioned before, `acl:Control` is complex and confusing and will thus be set equivalent to `acl:Write` to retain simplicity. Additionally, since this application concerns forms, each class that allows the user to modify the data in any way requires the form field to be visible, hence the Read action should be allowed.

CRUD	acl:Read	acl:Write	acl:Append	acl:Control
Create	×	✓	✓	✓
Read	✓	✓	✓	✓
Update	×	✓	×	✓
Delete	×	✓	×	✓

Table 5.4: The mapping from ACL to Create, Read, Update and Delete actions.

S4AC Mapping

As mentioned before, the main focus of the intermediate model was initially to apply the authorisation specifications to the shape constraints to modify the form generation to comply with the General Data Protection Regulation (GDPR)'s consent requirements. Additionally, as a possible adaptation, the model could act as an abstraction layer between the authorisation specification and the SHACL shape used to generate the form.. After applying the Privacy Preference Ontology (PPO)-based, user-created specifications to the shape constraints, Social Semantic SPARQL Security for Access Control (S4AC) (fig. 5.5) is the logical next step since PPO and S4AC have similar goals. This mapping would go hand in hand with attempting to make the model more of an abstract layer.

In the process of mapping an S4AC example to the intermediate model, the implementation of S4AC conditions turned out to differ from PPO, and thus from the initial intermediate model, more than expected. The main difference and greatest implementation effort is the different approach in the application of the conditions. In PPO, the access conditions for the requester were expressed using SPARQL ASK queries in an *AccessSpace*. In contrast, the conditions applied to the data were passed as separate properties of a *Condition*. S4AC uses SPARQL ASK queries to express the access conditions for both the requester and the requested resources.

An approach that could bridge the gap between these two options would be the usage of SPARQL ASK queries to determine the applicable policies for each part of the data. In this case, the PPO *Condition* could be converted to an ASK query that would be executed on the data. The current model applies the query related to the requester. The queries that apply to the requested data on the one hand and the requester, on the other hand, should be extracted to apply them to different data stores. Determining the feasibility of this extraction is considered to be future work.

Other aspects from S4AC that are not yet present in the intermediate model will be discussed in the section on future work (section 7.2). Later work can expand into these aspects and their implementations since time constraints did not allow these adaptations to be implemented in the actual intermediate model.

CHAPTER 6

EVALUATION

In this chapter, the performance of the intermediate model presented in the previous chapter will be evaluated. This evaluation consists of two parts. First, in section 6.1, the used data shapes and authorisation specifications are provided. This input data is then evaluated, and the results are discussed in section 6.2.

This thesis does not consider the complexity of the ASK queries in the UserAccessConditions of the intermediate model. These queries were implemented using the *RDFStore* JavaScript library[1]; thus, their complexity is limited by the package's functionality. Furthermore, as described in section 6.1, the SPARQL ASK queries are simple and only apply to the user's employment as an illustration.

6.1 Shapes and Authorisation Specifications

This section introduces the data and shapes that are used throughout the experiments explored in the next section. First, the different user profiles are described, as well as the data subject and requester, followed by the used authorisation specifications.

6.1.1 Users

In the following examples, four user profiles will be used: Alice is an *Admin*, Bob is a *Team Lead*, and Carol is an *Employee*. David, the data subject and consequently the owner of the data used in the examples, is an Employee as well (see listing 1). This distinction is made because the owner of the data inherently has different access rights than other employees. As mentioned in section 5.2.2, these four people are the four possible profiles the user can use to log in to test the authorisation specification applied in Schímatos. These three occupations (Admin, Team Lead and Employee) are used in the authorisation specification throughout this thesis.

The ASK queries used for defining the access spaces based on these occupations are provided in listing 6.1.

[1]RDFStore – `https://www.npmjs.com/package/rdfstore`

```
ex:adminSpace a ppo:AccessSpace ;
  ppo:hasAccessQuery
    "ASK { ?x <http://schema.org/hasOccupation> <http://example.org/admin> }" .
ex:teamLeadSpace a ppo:AccessSpace ;
  ppo:hasAccessQuery
    "ASK { ?x <http://schema.org/hasOccupation> <http://example.org/teamLead> }" .
ex:employeeSpace a ppo:AccessSpace ;
  ppo:hasAccessQuery
    "ASK { ?x <http://schema.org/hasOccupation> <http://example.org/employee> }" .
```

Listing 6.1: The ASK queries used to determine the access spaces.

These experiments assume that Daniel, mentioned above, is the data subject and owner. The requester in each case is Carol, a fellow employee of the company. The applicable `AccessSpace` is, therefore, `ex:employeeSpace`.

6.1.2 Authorisation Specifications

The consent expressed by Daniel is expressed using the Privacy Preference Ontology (PPO). These experiments assume that the expressed conditions apply to Daniels data and that Carol is the requester. The granted access will be visualised in tables to increase legibility.

As mentioned in section 5.2.4, Schímatos employs a different method for applying the access control to the generated form, limiting the access control application and producing different results in some cases. Therefore, applying the intermediate model to the SHACL shapes will be approached separately to the application to Schímatos in the following experiments.

6.2 Experiments

In this section, the experiments carried out on the intermediate model and the form generation in Schímatos will be explained. The first subsection describes the main features centred around access control, such as allowing Access Control List (ACL) operations and granting Create, Read, Update, Delete (CRUD) access. The next subsection explores the concept of nesting. The final subsection compares the mapping of different vocabularies to the intermediate model.

6.2.1 Granting Access

This section focuses on granting different combinations of access types and evaluating the different outcomes from both the intermediate model and the generated form.

Mode	**Allows**	**Visible/Enabled**
(no access)	Nothing	The property's input field is not visible, existing values are hidden, add/delete buttons are disabled
Read	Retrieving and reading the existing values of the property	The property's input field is visible but disabled, containing any existing values
`Write`	Overwriting and deleting existing property values, adding new values	The property's input field is visible and enabled, the add and delete buttons are enabled
Append	Adding property values, but not modifying or deleting existing values	The add buttons are enabled, an input field is visible and enabled, but existing values are not visible
Read + Append	Retrieving and reading existing property values, adding new values	existing values are visible but their input fields are disabled, an empty input field is enabled to add new values, the add button is enabled
Read + `Write`	Equal to `Write` (1)	
`Write` + Append	Equal to `Write` (2)	
Read + `Write` + Append	Equal to `Write` (1,2)	

Table 6.1: Possible combinations of ACL access types and their interpretation in form context.

Table 5.2 describes each access control type in ACL and table 5.4 shows the mapping from ACL to Create, Read, Update, Delete (CRUD) access types. The `acl:Control` type will be assumed to be equivalent to `acl:Write` control. An explanation of the ACL access types, possible combinations and how they could be interpreted in the context of a form can be found in table 6.1. Some additional remarks on the mode combinations that are equal to `acl:Write` access:

1. Values in a form cannot be modified without allowing `acl:`Read access since the modification of data values relies mainly on using input fields. Therefore, this thesis assumes that granting `acl:Write` access in the context of forms implies that the requester should also be able to `acl:`Read the values.
2. The `acl:`Append access is a subclass of `acl:Write`. While `acl:Write` allows every possible modification to the properties, `acl:`Append only allows adding new values. Since `acl:`Append is a subclass of `acl:Write`, granting `acl:Write` access consequently grants `acl:`Append access.

Table 6.2 shows eight possible combinations of Access Control List (ACL) access control modes from Privacy Preference Ontology (PPO). These examples show how the intermediate model is applied using different access control combinations, illustrating how the options interact and how the user can use these options to achieve the desired goal. These access options are applied using the intermediate model on the SHACL shape (table 6.3) and

Resource	acl:Read	acl:Write	acl:Append
schema:givenName	✓	×	×
schema:familyName	×	✓	×
schema:hasOccupation	×	×	✓
schema:email	✓	✓	×
schema:salary	×	✓	✓
schema:holiday	✓	×	✓
schema:addressCountry	✓	✓	✓
schema:postalCode	×	×	×

Table 6.2: Different combinations of granted access.

Resource	Create	Read	Update	Delete
schema:givenName	×	✓	×	×
schema:familyName	✓	✓”	✓	✓
schema:hasOccupation	✓	✓”	×	×
schema:email	✓	✓	✓	✓
schema:salary	✓	✓”	✓	✓
schema:holiday	✓	✓	×	×
schema:addressCountry	✓	✓	✓	✓
schema:postalCode	×	×	×	×

Table 6.3: The access applied in to the SHACL shape.

visualised on the form generated with Schímatos (table 6.4). As mentioned above, different access options are implemented. These access types are represented in the form generation using different components, like enabled or disabled buttons and hidden, disabled or visible fields. Cases where `acl:Append` is granted consequently since it is a subclass of `acl:Write` are indicated using quoted checkmarks (✓”).

The intermediate model and Schímatos apply these eight combinations of access control similarly, although the results should be clarified more in-depth. For example, the `hasOccupation` property only provides `acl:Append` access, meaning that new values can be added, but existing values should not be visible, removable, nor editable. However, if `acl:Append` access is granted in the current implementation, Read and `Create` access is inherently granted, while `Update` access is not. Granting `acl:Append` without `acl:Write` access causes the input field to remain disabled since there is no distinction between a field with an existing value and a newly added value. These cases are indicated in table 6.4 using

Resource	Add-button enabled	Value visible	Input field enabled	Delete button enabled
schema:givenName	×	✓	×	×
schema:familyName	✓	✓	✓	✓
schema:hasOccupation	✓	✓	×’	×
schema:email	✓	✓	✓	✓
schema:salary	✓	✓	✓	✓
schema:holiday	✓	✓	×’	×
schema:addressCountry	✓	✓	✓	✓
schema:postalCode	×	×	×	×

Table 6.4: The access applied in the form generated by Schímatos.

a single-quoted cross (×’). Creating this distinction to improve the application of different access types further is considered to be future work.

Therefore, access is granted correctly to the SHACL shape in 32 out of 32 values (100%) and 30 out of 32 values (93.75%) in the generated form.

6.2.2 Nesting

This subsection explores the concept of nesting in Shapes Constraint Language (SHACL) shapes.

```
ex:Shape a sh:NodeShape ;
  sh:property [
    sh:name "first property" ;
  ] ;
  sh:property [
    sh:name "parent" ;
    sh:property [
      sh:name "child" ;
    ]
  ] ;
  sh:property [
    sh:name "A: parent of B" ;
    sh:property [
      sh:name "B: child of A, parent of C" ;
      sh:property [
        sh:name "C: child of B and A" ;
      ]
    ]
  ] .
```

Listing 6.2: An example of a shape with nested properties.

Some properties of a shape might have properties themselves. In this thesis, this will be referred to as *nesting*. In the case of nesting, the *parent* is the property that encapsulates another property, which is the *child*. This child can also contain other properties and thus be a parent of these properties as well. Therefore, a property can simultaneously be a parent and a child.

An example shape to illustrate the concept of nesting can be found in listing 6.2. The example Shape has three properties. The `first property` is not nested since it does not contain any other properties. The second property is nested and the `parent` of an enclosing property `child`. The third property of Shape has multiple levels of nesting: property `B` is the child of property `A` but also has a child `C`. `C` is thus a direct child of `B` and an indirect child of `A`, while `A` is the parent of both `B` and `C`.

```
{
  "policies": {
    "ex:Policy-A": {
      "hasAccess": [ "read", "update" ],
      "hasUserAccessCondition": [ "ex:UAC-A" ],
      "hasDataCondition": [ "ex:DAC-A", "ex:DAC-B" ]
    },
  },
  "dataAccessConditions": {
    "ex:DAC-A": { "hasProperty": [ "schema:givenName" ] },
    "ex:DAC-B": { "hasProperty": [ "schema:familyName" ] },
  },
  "userAccessConditions": {
    "ex:UAC-A": { "hasAskQuery": [ "ASK { ... }" ] },
    "ex:UAC-B": { "hasAskQuery": [ "ASK { ... }" ] }
  }
}
```

Listing 6.3: A simplified example of an intermediate model instance.

Nesting in the context of the authorisation specification, i.e. nesting conditions, will not be considered. Nesting is not applicable to access conditions due to the structure of the objects and the concept of access control. Figure 5.6 and listing 6.2 illustrate and confirm these statements. An UserAccessCondition (UAC) or DataAccessCondition (DAC) can only be the child of a Policy. A UAC contains one ASK query and cannot be the parent of another condition, be it a UAC or a DAC. A DAC contains requirements that apply to data resources and cannot incapsulate another condition either. The only nested objects in the intermediate model are the UACs and DACs inherently nested in their parent Policy. A Policy can contain multiple UACs and DACs to combine conditions and make the Policy more specific. Policies are thus the parents of UACs and DACs, but cannot be the children of other objects. Therefore, the structure defined in the intermediate model does not allow for conditions nested within other conditions.

The simplified version of the used shape and the applied access specification can be found in listing 3 and table 6.5. In the first part of this experiment, the allowed access type will be limited to Read access for brevity and reduced cases. Table 6.6 explores the results of applying three combinations of Read and `Write` access on a parent and its two children using the intermediate model on the SHACL shape.

The SHACL shape is designed to handle different parent-child relationships and various combinations of granted and denied access. The following questions will be addressed in this experiment:

Property	Specified Access	Granted Access (SHACL)		Granted Access (Schímatos)
		no `passDown`	with `passDown`	
property A	✓	✓	✓	✓
property B	×	×	×	×
parent A	✓	✓	✓	✓
child A1	✓	✓	✓	×
child A2	×	×	✓”	n/a
parent B	×	×	×	×
child B1	✓	×’	×’	×’
child B2	×	×	×	n/a
parent C	✓	✓	✓	✓
child/parent C1	✓	✓	✓	×
child C1.1	✓	✓	✓	×
child C1.2	×	×	✓”	n/a
child/parent C2	×	×	✓”	n/a
child C2.1	✓	×’	✓”	n/a
child C2.2	×	×	✓”	n/a
parent D	✓	✓	✓	✓
child/parent D1	✓	✓	✓	×
child/parent D1.1	✓	✓	✓	×
child D1.1.1	✓	✓	✓	×

Table 6.5: The specified access and resulting granted access when applying the intermediate model to the SHACL shape and Schímatos.

Property	Specified Access	Granted Access (SHACL)	
		no `passDown`	with `passDown`
parent A	R	R	R
child A1	W	R	R
child A2	×	×	R
parent B	W	W	W
child B1	R	R	W
child B2	×	×	W
parent C	×	×	×
child C1	R	×	×
child C2	W	×	×

Table 6.6: The specified access and resulting granted access when applying the intermediate model to the SHACL shape, using a combination of Read (R) and `Write` (W) access.

- Can access control be applied to nested properties?
  - What is the limit of nested levels?
- If parent and child have different access types, how are they applied?
  - If more access is granted to the parent than the child, does the child inherit his parent's access type?
  - Correspondingly, if more access is granted to the child, does the child still inherit the parent's access?

Applying the intermediate model to the SHACL shape results in granted access shown in table 6.5, from which the following conclusions can be made:

- Access control can be applied to nested properties.
  - This experiment uses a limit of three nested levels to illustrate the inheritance of granted access.
- If the parent and child properties have different access types, the data subject can choose if the more restricted child should inherit the access granted by the parent property.
  By leaving this choice to the data subject, they can explicitly indicate how their expressed consent for access should be applied. The `passDown` setting can be set to `True` if more restricted child properties can inherit access. By default, this setting is set to `False` to avoid wrongly granted access.
  - If the parent has Read access while the child does not, access is not passed down from parent to child. In case the `passDown` setting is enabled by the data subject, the child inherits any access from the parent. These cases are indicated in table 6.5 using quoted checkmarks (✓") since these are instances where the child's access was not given explicitly by the data subject.
    Similarly, if the parent (parent B in table 6.6) has `Write` access while the child (B1) only has Read access, the child's granted access will not be modified. Therefore, access to the child is not granted unless explicitly stated.
  - If the parent has no Read access, but the child does, the child inherits the parent's denied access and can thus not be read. Three such cases are indicated in table 6.5 using a single-quoted cross (×'). If the parent (parent A in table 6.6 has Read access while the child (A1) has `Write` access, the child's granted access will be restricted to Read. This behaviour is implemented to avoid accidentally granting access to resources that are part of restricted data.
    If the data subject enables the `passDown` setting, the parent's more restricted

access shall be passed down to the child, even when access to the child has been explicitly granted.

Out of the 19 elements without `passDown`, 17 instances are applied correctly. At the same time, two cases (×′) can be interpreted and may be considered either correct or incorrect depending on the meaning accepted by the data subject. If the data subject has enabled the `passDown` setting, 18 out of the 19 cases are applied as specified by the data subject, while one case (×′) is open for interpretation. For these three interpretable cases, another option setting could be created for the user to allow access to these cases explicitly.

The form generated by Schímatos can be found in fig. 1, while the granted access is expressed in table 6.5. Since the only granted type of access is Read, the buttons and input fields are disabled in the generated form. It is noteworthy that Schímatos only generates the first child of a property due to an unfixed error in the original form generation of the application. These non-generated cases are indicated as “n/a” in the table. In this web application, the form generation is based on a set of JavaScript objects, while the object that determines the applied access control is based on these original objects. Unfortunately, these objects were not optimised to support the generation of nested properties, which explains that only the first child is generated. On the other hand, the implementation and representation of these nested objects in Schímatos make applying access rules to the nested fields a more complex task.

6.2.3 Expressing Vocabularies

This section focuses on the vocabularies that can be expressed using the intermediate model. For both Privacy Preference Ontology (PPO) (fig. 5.4) and Social Semantic SPARQL Security for Access Control (S4AC) (fig. 5.5), this section will evaluate and compare how many features from each ontology can be expressed.

The classes and properties of PPO and S4AC mapped to the properties of the intermediate model can be found in table 6.7 and table 6.8, respectively. Here we see that in PPO 16 out of 16 features (100%) can be expressed, while for S4AC, this is 11 out of 25 (44%).

The main reason S4AC has fewer equivalent features in the intermediate model is described in section 5.2.4. S4AC introduces concepts that were either out of scope, like the access evaluation context and its related properties (`hasAccessEvaluationContext`, `hasVariable`, `hasDescription`, ...), or the nuance differences between S4AC’s `AccessCondition` and the intermediate model’s `DataAccessCondition` and `UserAccessCondition`. Additionally, the model makes no distinction between conjunctive and disjunctive condition sets.

Class/Property	**Intermediate Model**
PrivacyPreference	Policy
AccessSpace	UserAccessCondition
Condition	DataAccessCondition
hasAccessSpace	hasUserAccessCondition
hasAccessQuery	hasASKQuery
hasAccess	hasAccess
appliesToResource	appliesTo
appliesToStatement	appliesTo
appliesToNamedGraph	appliesTo
hasCondition	hasDataAccessCondition
resourceAsSubject	hasSubject
resourceAsObject	hasObject
classAsSubject	hasSubject
classAsObject	hasObject
hasProperty	hasProperty
hasLiteral	hasLiteral
Total: 16	16

Table 6.7: Features of PPO mapped to the intermediate model.

Class/Property	**Intermediate Model**
AccessCondition	UserAccessCondition & DataAccessCondition
AccessPrivilege	AccessType
AccessPolicy	Policy
AccessEvaluationContext	×
AccessConditionSet	×
DisjunctiveAccessConditionSet	×
ConjunctiveAccessConditionSet	×
Create	s4ac:Create
Read	s4ac:Read
Update	s4ac:Update
Delete	s4ac:Delete
Value	×
Variable	×
hasAccessPrivilege	hasAccess
hasAccessConditionSet	×
hasAccessEvaluationContext	×
hasAccessCondition	hasUserAccessCondition
isAccessConditionOf	×
hasName	×
hasVariable	×
hasVarName	×
hasValue	×
hasDescription	×
hasQueryAsk	hasASKQuery
appliesTo	appliesTo
Total: 25	11

Table 6.8: Features of S4AC mapped to the intermediate model.

CHAPTER 7

CONCLUSION

The main objective of this dissertation was to automatically generate forms on a Knowledge Graph while complying with the General Data Protection Regulation (GDPR) for giving explicit and specific consent. These goals were addressed by implementing an intermediate model that applies the expressed access control and consent in two ways. On the one hand, this model was applied to the JavaScript objects that generate the forms in Schímatos. On the other hand, it was applied directly to the Shapes Constraint Language (SHACL) shape used to generate the form and its constraints.

Experiments show that these two methods produce mixed results due to their differences in implementation. First, granting different types of Access Control List (ACL) access was explored. Applying eight combinations using ACL resulted in 32 values expressed in Create, Read, Update, Delete (CRUD) access. The model applied 100% of these values correctly in the SHACL shape, and 93.75% was expressed correctly in the generated form. Next, Schímatos provides limited support for nesting and granted access in nested fields. On the other hand, the intermediate model supports nesting at least up to three levels, where the data subject has the option to explicitly allow the children to inherit the parents' granted access types. Finally, mapping the features of both Privacy Preference Ontology (PPO) and Social Semantic SPARQL Security for Access Control (S4AC) to those of the intermediate model demonstrated that PPO could be fully mapped to this model, while the model could express 44% of S4AC's features.

The remainder of this chapter will first revisit the research questions and hypotheses established in chapter 4. Finally, opportunities for future work will be explored in section 7.2.

7.1 Fulfilment of Research Objectives

This section will first explore the answers to the subquestions and evaluate their hypotheses before answering the main research question and assessing its hypothesis.

Subquestion 1. *How can we apply access control to a shape to satisfy the GDPR in the context of giving consent?*

Hypothesis 1. *Access control can be applied to a SHACL shape using an intermediate model to modify and add constraints to the given shape with nested properties.*

Section 6.2.1 demonstrated that access expressed by the data subject in Privacy Preference Ontology (PPO) could be applied to a SHACL shape using the intermediate model described in section 5.2.4. This model adds constraints to the given shape to indicate the access granted to its properties. As seen in section 6.2.2, access control can be applied to up to three levels of nested properties. This hypothesis can therefore be accepted.

Subquestion 2. *How can we apply the subject's explicit consent to a form generated on a Knowledge Graph?*

Hypothesis 2. *A subject's explicit and specific consent can be applied to form generation without nesting using an intermediate model that adds access-related properties to the JavaScript objects for each field.*

As with the first subquestion, section 6.2.1 indicated that access expressed in PPO could be applied to forms generated in Schímatos using the intermediate model designed in section 5.2.4. The model creates a new JavaScript object that contains the access granted for each of the properties used to generate the form fields.

In this case, however, applying the access control is limited by the application's implementation in the context of nested properties. Therefore, the hypothesis can be accepted for generated forms without nested properties.

After discussing the two subquestions, these answers can be combined to answer the main research question:

Research Question. *How can we comply with the GDPR for giving explicit and specific consent while automatically generating forms on a Knowledge Graph?*

Therefore, the following main hypothesis can be accepted:

Main Hypothesis. *A subject's data shape can comply with the GDPR's definition of consent by using a fine-grained specification of access control permissions, as defined by the data subject using PPO.*

7.2 Remaining Challenges and Future Directions

Various opportunities for future work have been mentioned in earlier sections. This final section will elaborate on these remaining challenges.

N-Quads. The intermediate model currently employs N-triples in its implementation. The implementation of the intermediate model could be expanded to support N-Quads and allow more options for expressing consent.

User-Friendly Consent Specification. The current implementation assumes that the data subject is familiar with Turtle semantics and can express his consent using Privacy Preference Ontology (PPO). Creating a more user-friendly way of expressing explicit and specific consent would allow inexperienced users to express their consent without knowing the underlying technologies.

Abstraction. Future studies could address the abstraction created by the intermediate format. As mentioned in section 5.2.4 and section 6.2.3, the current intermediate model does not allow for a full abstraction between the consent expressed in any authorisation specification and the goal shape or the generated form. The current model is more tied to Privacy Preference Ontology (PPO) and not easily generalisable to other ontologies like Social Semantic SPARQL Security for Access Control (S4AC). Later research could attempt to further generalise the intermediate model by adapting the application of the access conditions as described in section 5.2.4.

Access Context. Along with these generalisations, the intermediate model could be expanded to allow taking into account the access context. For example, features could be added to allow checking the number of allowed accesses and time constraints. Additionally, a description could be added to inform the user why access has been denied without revealing the actual access conditions.

Logical Condition Sets. Social Semantic SPARQL Security for Access Control (S4AC) supports the distinction between conjunctive and disjunctive condition sets. Including some logical division in applying the intermediate model and an indication of the condition set type would increase the flexibility of the authorisation specifications.

Nesting. The application of the authorisation specification to nested properties could be improved to increase the correctness of the model, mainly in the forms generated by

Schímatos. Since Schímatos does not allow complete functionality of nested fields and the generation is based on a set of JavaScript objects, access is not correctly granted. This implementation could be reworked to apply the authorisation specifications to both the parent fields and the nested children.

Strictness. The intermediate model could be expanded to support more specific settings for strictness, allowing more flexibility for the data owner concerning the granted access. These settings would allow the data subject to specify how strict the model should approach combinations of denied and granted access.

New and Existing Fields. As for the implementation of the granted access in the form generation, a distinction could be made to improve the application of `acl:Append` access. For example, the current implementation does not enable the input field if the `Update` access is not granted along with the `Create` access granted by `acl:Append`. An improvement would be to add an indication to the generated fields, indicating if they are linked to existing values in the data shape or newly added fields by the requester for adding a new value. This additional indication would allow enabling the new input fields without compromising the Read and `Update` access of the existing values.

LIST OF FIGURES

LIST OF TABLES

LIST OF EXAMPLES

BIBLIOGRAPHY

[1] Beckett, D.: Rdf 1.1 n-triples (February 2014), `https://www.w3.org/TR/n-triples/`

[2] Berners-Lee, T.: Information management: A proposal. Tech. rep. (1989)

[3] Berners-Lee, T., Connolly, D., Stein, L.A., Swick, R.: The semantic web (August 2000), `https://www.w3.org/2000/Talks/0906-xmlweb-tbl/text.htm`

[4] Brickley, D., Miller, L.: Foaf vocabulary specification 0.99 (January 2014), `http://xmlns.com/foaf/spec/`

[5] Butt, A.S., Haller, A., Liu, S., Xie, L., et al.: Activeraul: A web form-based user interface to create and maintain rdf data. In: International Semantic Web Conference (Posters & Demos). pp. 117–120 (2013)

[6] Carothers, G.: Rdf 1.1 n-quads (February 2014), `https://www.w3.org/TR/n-quads/`

[7] Costabello, L., Villata, S., Delaforge, N., Gandon, F.: Linked data access goes mobile: Context-aware authorization for graph stores. In: LDOW-5th WWW Workshop on Linked Data on the Web-2012 (2012)

[8] Costabello, L., Villata, S., Gandon, F.: Context-aware access control for rdf graph stores. In: ECAI. vol. 242, pp. 282–287 (2012)

[9] Council of Europe: European Convention on Human Rights (1950), `https://www.echr.coe.int/Documents/Convention_ENG.pdf`

[10] Council of European Union: Directive 95/46/ec (1995), `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:31995L0046`

[11] Council of European Union: Regulation (EU) 2016/679 (2016), `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679`

[12] Daly, J.: World wide web consortium issues web ontology language candidate recommendations (August 2003), `https://www.w3.org/2003/08/owl-pressrelease.html.en`

[13] Dürst, M., Suignard, M.: Internationalized resource identifiers (iris). Tech. rep., RFC 3987, January (2005)

[14] Education, I.C.: Knowledge graph (2021), `https://www.ibm.com/cloud/learn/knowledge-graph`

[15] European Data Protection Supervisor (EDPS): The history of the general data protection regulation (2018), `https://edps.europa.eu/data-protection/data-protection/legislation/history-general-data-protection-regulation_en`

[16] Fatema, K., Hadziselimovic, E., Pandit, H.J., Debruyne, C., Lewis, D., O'Sullivan, D.: Compliance through informed consent: Semantic based consent permission and data management model. In: PrivOn@ ISWC (2017)

[17] Group, S.C.: Shex – shape expressions (2012), `https://shex.io/`

[18] Haller, A., Groza, T., Rosenberg, F.: Interacting with linked data via semantically annotated widgets. In: Joint International Semantic Technology Conference. pp. 300–317. Springer (2011)

[19] Idehen, K.U.: Semantic web layer cake tweak, explained (July 2017), `https://medium.com/openlink-software-blog/semantic-web-layer-cake-tweak-explained-6`

[20] Kirrane, S., Fernández, J.D., Dullaert, W., Milosevic, U., Polleres, A., Bonatti, P.A., Wenning, R., Drozd, O., Raschke, P.: A scalable consent, transparency and compliance architecture. In: European Semantic Web Conference. pp. 131–136. Springer (2018)

[21] Kirrane, S., Villata, S., dAquin, M.: Privacy, security and policies: A review of problems and solutions with semantic web technologies. Semantic Web **9**(2), 153–161 (2018)

[22] Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl) (2017), `https://www.w3.org/TR/shacl/`

[23] Maillot, P., Ferré, S., Cellier, P., Ducassé, M., Partouche, F.: Nested forms with dynamic suggestions for quality rdf authoring. In: International Conference on Database and Expert Systems Applications. pp. 35–45. Springer (2017)

[24] Manola, F., Miller, E., McBride, B.: Rdf primer (2004), `https://www.w3.org/TR/rdf-primer/`

[25] Masinter, L., Berners-Lee, T., Fielding, R.T.: Uniform resource identifier (uri): Generic syntax. Network Working Group: Fremont, CA, USA (2005)

[26] MDN Contributors: Using http cookies (2021), `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies`

[27] Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protege-2000. IEEE intelligent systems **16**(2), 60–71 (2001)

[28] Padia, A., Finin, T., Joshi, A., et al.: Attribute-based fine grained access control for triple stores. In: 3rd Society, Privacy and the Semantic Web-Policy and Technology workshop, 14th International Semantic Web Conference (2015)

[29] Palmér, M., Enoksson, F., Nilsson, M., Naeve, A.: Annotation profiles: Configuring forms to edit rdf. In: International Conference on Dublin Core and Metadata Applications. pp. 10–21 (2007)

[30] Pandit, H.J., Debruyne, C., OSullivan, D., Lewis, D.: Gconsent-a consent ontology based on the gdpr. In: European Semantic Web Conference. pp. 270–282. Springer (2019)

[31] Pandit, H.J., Polleres, A., Bos, B., Brennan, R., Bruegger, B., Ekaputra, F.J., Fernández, J.D., Hamed, R.G., Kiesling, E., Lizar, M., et al.: Creating a vocabulary for data privacy. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 714–730. Springer (2019)

[32] Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A browser-independent presentation vocabulary for rdf. In: International Semantic Web Conference. pp. 158–171. Springer (2006)

[33] Prud’hommeaux, E., Seaborne, A.: Sparql query language for rdf (2008), `https://www.w3.org/TR/rdf-sparql-query/`

[34] Sacco, O., Passant, A.: A privacy preference ontology (ppo) for linked data. In: LDOW (2011)

[35] Sacco, O., Passant, A., Decker, S.: An access control framework for the web of data. In: 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 456–463. IEEE (2011)

[36] The OWASP Foundation: Access control (2021), `https://owasp.org/www-community/Access_Control`

[37] Villata, S., Delaforge, N., Gandon, F., Gyrard, A.: An access control model for linked data. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 454–463. Springer (2011)

[38] W3C: About the world wide web (1997), `https://www.w3.org/WWW/`

[39] W3C: Web ontology language (owl) (2013), `https://www.w3.org/OWL/`

[40] W3C: Resource description framework (rdf) (2014),
https://www.w3.org/2001/sw/wiki/RDF

[41] W3C: Inference (2015),
https://www.w3.org/standards/semanticweb/inference

[42] W3C: Linked data (2015),
https://www.w3.org/standards/semanticweb/data

[43] W3C: Ontologies (2015),
https://www.w3.org/standards/semanticweb/ontology

[44] W3C: Shacl-shex-comparison (March 2017),
https://www.w3.org/2014/data-shapes/wiki/SHACL-ShEx-Comparison

[45] W3C: Webaccesscontrol (2019),
https://www.w3.org/wiki/WebAccessControl

[46] W3C: About w3c (2021),
https://www.w3.org/Consortium/

[47] Wolford, B.: What is GDPR, the EUs new data protection law? (2020),
https://gdpr.eu/what-is-gdpr/

[48] Wright, J.: Shacl form react (2021),
https://github.com/schimatos/shacl-form-react

[49] Wright, J., Méndez, S.J.R., Haller, A., Taylor, K., Omran, P.G.: Schímatos: a shacl-based web-form generator for knowledge graph editing. In: International Semantic Web Conference. pp. 65–80. Springer (2020)

APPENDIX

User Profile

```
@prefix ex: <http://example.org/> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:employee a schema:Occupation ;
  schema:name "Employee" ;
  schema:estimatedSalary 1000.

ex:Daniel a schema:Person ;
  schema:givenName "Daniel" ;
  schema:familyName "Johnson" ;
  schema:telephone "+32490000000" ;
  schema:email "daniel@company.com" ;
  schema:homeLocation [
    schema:streetAddress "Example Lane 123" ;
    schema:postalCode "94043" ;
    schema:addressCountry "US"
  ] ;
  ex:emergencyContact [
    schema:givenName "Daniel Sr." ;
    schema:familyName "Johnson" ;
    schema:telephone "+32490000001" ;
    schema:email "danielsr@example.com" ;
  ] ;
  ex:holiday (
    "2021-08-07T00:00:00Z"^^xsd:dateTime
    "2021-08-08T00:00:00Z"^^xsd:dateTime
    "2021-08-09T00:00:00Z"^^xsd:dateTime
  );
  ex:ownsResources (
    "https://example.org/Daniel"
  );
  schema:hasOccupation ex:employee .
```

Listing 1: Example user profile for an employee named David.

Intermediate Model

```
{
  "policies": {
    "http://example.com/ns#readBasicInfo": {
      "appliesTo": [ "http://example.com/ns#Daniel" ],
      "hasAccess": [ "read" ],
      "hasUserAccessCondition": [
        "http://example.com/ns#teamLeadSpace",
        "http://example.com/ns#employeeSpace"
      ],
      "hasDataCondition": [
        "http://example.com/ns#givenNameCondition",
        "http://example.com/ns#familyNameCondition",
        "http://example.com/ns#emailCondition"
      ]
    },
    "http://example.com/ns#editEverything": {
      "appliesTo": [ "http://example.com/ns#Daniel" ],
      "hasAccess": [ "read", "create", "update", "delete" ],
      "hasUserAccessCondition": [
        "http://example.com/ns#adminSpace"
      ],
      "hasDataCondition": [
        "http://example.com/ns#danielsData"
      ]
    },
    ...
  },
  "dataAccessConditions": {
    "http://example.com/ns#givenNameCondition": {
      "hasProperty": [ "http://schema.org/givenName" ]
    },
    "http://example.com/ns#familyNameCondition": {
      "hasProperty": [ "http://schema.org/familyName" ]
    },
    "http://example.com/ns#emailCondition": {
      "hasProperty": [ "http://schema.org/email" ]
    },
    "http://example.com/ns#danielsData": {
      "hasSubject": [ "http://example.com/ns#Daniel" ]
    },
    ...
  },
  "userAccessConditions": {
    "http://example.com/ns#employeeSpace": {
      "hasAskQuery": [
        "ASK { ?x <http://schema.org/hasOccupation> <http://example.com/ns#employee> }"
      ]
    },
    "http://example.com/ns#adminSpace": {
      "hasAskQuery": [
        "ASK { ?x <http://schema.org/hasOccupation> <http://example.com/ns#admin> }"
      ]
    },
    "http://example.com/ns#teamLeadSpace": {
      "hasAskQuery": [
        "ASK { ?x <http://schema.org/hasOccupation> <http://example.com/ns#teamLead> }"
      ]
    }
  }
}
```

Listing 2: An example of an authentication specification expressed using the intermediate model.

Nesting

SHACL Shape

```
@prefix ex: <http://example.org/> .
@prefix sh: <http://www.w3.org/ns/SHACL#> .

ex:AuthNesting a sh:NodeShape ;
  sh:property noNesting1 ;
  sh:property noNesting2 ;

  sh:property nesting1Y [
    sh:property nestingLevel1Y ;
    sh:property nestingLevel1N ;
  ] ;
  sh:property nesting1N [
    sh:property nestingLevel1Y ;
    sh:property nestingLevel1N ;
  ] ;

  sh:property nesting2 [
    sh:property nestingLevel1Y [
      sh:property nestingLevel2Y ;
      sh:property nestingLevel2N ;
    ] ;
    sh:property nestingLevel1N [
      sh:property nestingLevel2Y ;
      sh:property nestingLevel2N ;
    ] ;
  ] ;

  sh:property nesting3 [
    sh:property nestingLevel1Y [
      sh:property nestingLevel2Y [
        sh:property nestingLevel3Y ;
      ] ;
    ] ;
  ] .
```

Listing 3: Simplified version of the SHACL shape used in the nesting experiment.

Generated Form

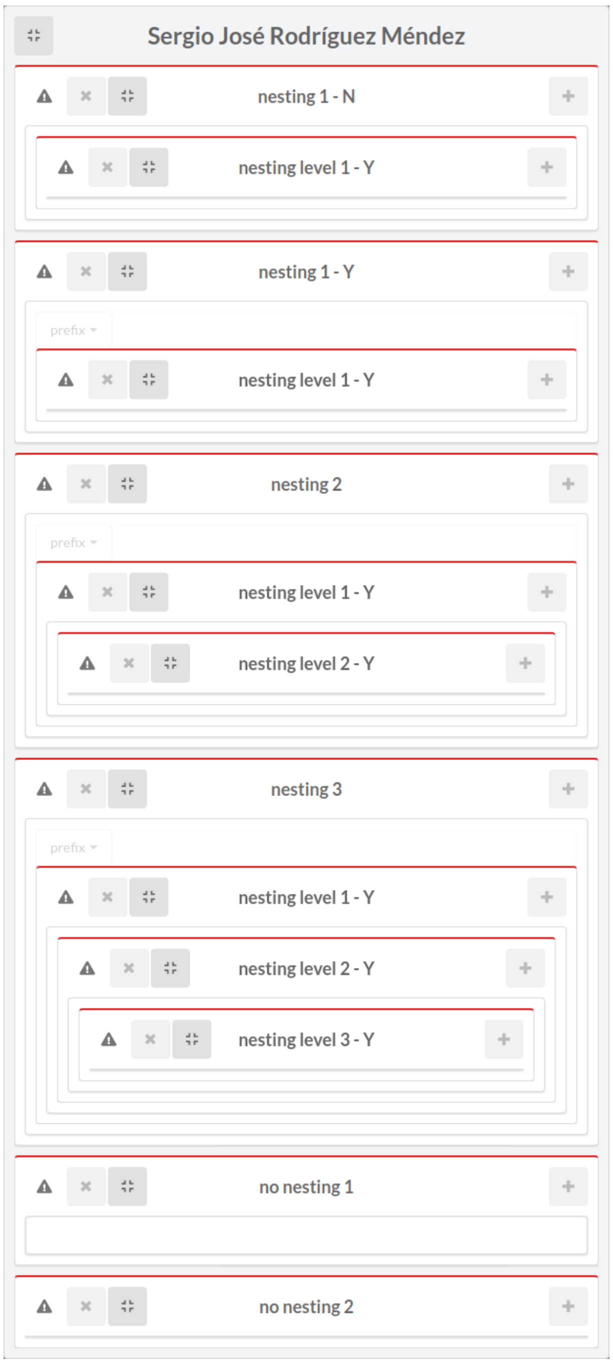


Figure 1: Form generated using the SHACL shape used in the nesting experiment.