POD-QUERY: Schema Mapping and Query Rewriting for Solid Pods

Maarten Vandenbrande[1,†], Maxime Jakubowski[2,†], Pieter Bonte[1,4], Bart Buelens[3], Femke Ongenae[1] and Jan Van den Bussche[2]

[1]IDLab, Ghent University – imec, Belgium
[2]Data Science Institute, Hasselt University, Belgium
[3]Flemish Institute for Technological Research (VITO), Belgium
[4]Departement of Computer Science, KU Leuven Kulak, Belgium

Abstract

We envisage a decentralized Web architecture in which access to data in Solid pods is mediated by Web agents. Leveraging methods from information integration and data exchange, Web agents can give clients access to views defined by schema mappings. Queries formulated by clients over the view are rewritten by the agent to queries that work correctly over the base data. We demonstrate POD-QUERY, our prototype implementation of such an architecture, where schema alignment is specified by RDF-to-RDF mappings written in RML, and SPARQL queries are rewritten based on these mappings. We demonstrate our system in a personal health scenario.

Keywords

Decentralized Web, Personal data management, Web agent, Schema alignment

1. Background

For many years there has been interest in the management of personal data in a decentralized manner, e.g., ongoing societal movements such as MyData [https://mydata.org], and work on personal information management systems [1]. A more recent example is the massive uptake of Mastodon and the Fediverse [https://spectrum.ieee.org/mastodon-social-media]. Of course, more broadly, support for federated linked data has always been a goal of the Semantic Web [2].

An instrumental advance in this direction is Solid, a Web protocol for accessing "pods", i.e., personal data vaults [https://solidproject.org]. Solid essentially describes authenticated HTTP access to linked data containers. Like the Fediverse, Solid was initially inspired by social network applications [3], but also has a broader applicability in the decentralized Web of data. There is ambition to become a W3C standard.

Solid pods, following the LDP protocol, serve resources structured in nested containers, comparable to files in a UNIX directory structure. Resources can be RDF or non-RDF (i.e., blobs); we will focus here on resources that are RDF graphs. Clients, identified by a Web ID, can be given access to specific resources in the pod. Access can be read or write; we will focus here on read access. When a client is given read access to some resource, they can simply

ISWC 2023 Posters and Demos: 22nd International Semantic Web Conference, November 6–10, 2023, Athens, Greece

† These authors contributed equally and should both be considered first author.

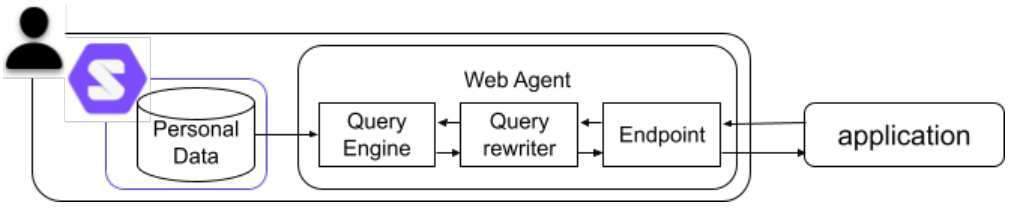

Figure 1: Schematic overview of the POD-QUERY system.

download (HTTP GET) the corresponding RDF graph. For some types of applications, however, and especially with larger graphs, we may want to allow clients to submit SPARQL queries instead. This natural extension to Solid is being developed or supported by several servers (Inrupt, Comunica [4]), and is adopted in our demo system (using Comunica).

2. Problem statement and approach

The problem motivating this work is that of *dealing with multiple parties or clients*. Assuming a world where Solid pods are used routinely, there will be multiple parties that we are willing to share data with, e.g., governments, health providers, recommendation services, social networks. With each party we want to share different data, and, moreover, each party would likely prefer to see that data according to a specific vocabulary or schema. However, the pod stores our base data in several RDF graphs that are organized according to our own, fixed, internal schema. Therefore, we cannot simply grant a party access to some set of graphs coming straight out of the pod, as these will likely include data that we are not willing to share with said party, and moreover, the data will not conform to the party's desired schema.

Towards a solution, we propose here the POD-QUERY system. The user can specify different views over their personal data for every requesting party. This way, the user can control exactly what the party can query. These views can hide data as well as transform into the right shape.

3. The POD-QUERY System

POD-QUERY (Fig. 1) realizes views on Solid pods through RDF-to-RDF schema alignment specified in RML, and rewriting SPARQL queries based on these RML mappings.

Formally, consider a pod D owned by A, and a party B with which A is willing to share some of its data. B specifies a schema S_B, stating requirements to which they expect the data to conform. We are not fixing the schema language in this paper; the requirements can be very loose, merely suggesting the use of certain vocabularies, or can be quite strict, taking the form of SHACL constraints, for example. POD-QUERY consists of the following components:

View: A defines, over the data in D, a view V_B which contains the RDF graph that B will be allowed to read. This view conforms to S_B.

Schema mapping: V_B is specified as a schema mapping, from data conforming to whatever local schema used in D, to RDF graphs conforming to S_B. In our approach, we use RDF-

to-RDF mappings written in RML https://rml.io [5], but alternatives could be SPARQL CONSTRUCT queries, or N3Logic rules.

Query rewriting: B can simply download the view V_B, which may be computed on demand or may be materialized by A (in which case it must also be kept up-to-date under updates to D). However, for larger data, and frequent access, we want to make it possible for B to be able to pose arbitrary SPARQL queries over the view V_B. If V_B is materialized, these queries can be directly answered by a SPARQL engine. Otherwise, however, the client query Q must be rewritten into an equivalent query Q' that works over the local schema.

Web agent, mediator: Keeps track of which views are used for which parties, executes or materializes the schema mappings, and applies query rewriting. This agent has full access to D, but may appear to B as a virtual Solid pod; the view V_B is then a resource to which B is granted access.

Each schema mapping is based on a SPARQL query that can be executed over a pod to result in a set of variable bindings. Each binding gives rise to generated triples, which can involve newly created IRIs through IRI templates [6]. Our system POD-QUERY generates, from any input RML mapping document V and any input client SPARQL query Q, a SPARQL query Q' such that $Q'(D) = Q(V(D))$, for every pod dataset D.

An example of rewriting is given in Figure 2. Our SPARQL–RML Rewriter software is publicly available as an independent module https://github.com/MaximeJakubowski/SRR. The consolidated POD-QUERY system software is available as well https://github.com/maartyman/solid-aggregator-server/tree/query-rewriting, https://github.com/maartyman/solid-agent.

4. The demo

Video: https://vimeo.com/846598633, Online: https://podquery-demo.vito.be

We focus on a personal health data sharing scenario, inspired by the We Are platform in Flanders [https://we-are-health.be]. Citizens are asked to fill a health questionnaire known as GGDM. As this pertains personal information, answers to the questions are stored in their pod using a designed GGDM vocabulary. Now assume a regional research survey (RRS) which asks people access to their GGDM data in order to study diabetes. Alice is willing to participate, but only wants to share selected info. Moreover, for her diabetes status, she refers to her health record, which was directly filled in her pod at the hospital. This record using the FHIR vocabulary [7], however. Thus, Alice instructs her Web agent to invoke two schema mappings defining her view for RRS: (1) directly retrieve only selected GGDM answers; and (2) transform my diabetes status from FHIR to GGDM.

Now RRS, contacting Alice's Web agent, may come with a query to retrieve all available GGDM answers, on condition that her diabetes status is positive. POD-QUERY will automatically rewrite this query correctly, checking diabetes status in FHIR and returning only the answers (e.g., eating habits and exercising) that Alice instructed to share. For another example, RRS may ask how many GGDM answers Alice makes available. In general, arbitrary client queries can be posed, but will be rewritten to answer only what Alice makes available, possibly from diverse sources.

```
# Schema mapping in RML
:answerSource a rml:LogicalSource ;
 rml:source :sparqlService ;
 rml:referenceFormulation ql:JSONPath ;
 rml:iterator "$.results.bindings[*]" ;
 rml:query
 """
 SELECT ?question ?completedQuestion
 ?answer ?date ?person ?session
 WHERE { ?session
  prov:atTime ?date ;
  prov:wasAssociatedWith ?person .
 ?completedQuestion
  sur:answeredIn ?session ;
 sur:hasAnswer ?answer ;
 sur:completesQuestion ?question .
 FILTER (?question IN (ggdm:question1,
 ggdm:question2,ggdm:question3,
 ggdm:question4,ggdm:question5,
 ggdm:question6,ggdm:question6-1,
 ggdm:question6-2,ggdm:question7,
 ggdm:question7-1,ggdm:question7-2,
 ggdm:question7-3,ggdm:question7-4,
 ggdm:question7-5,ggdm:question7-6,
 ggdm:question7-7,ggdm:question7-8,
 ggdm:question7-9,ggdm:question7-10,
 ggdm:question8-1,ggdm:question9-1,
 ggdm:question10,ggdm:question11,
 ggdm:question12,ggdm:question13,
 ggdm:question14)) }
 """
:completedQuestionTriplesMap
 a rr:TriplesMap ;
 rml:logicalSource :answerSource ;
 rr:subjectMap [
  rml:reference
   "completedQuestion.value" ;
  rr:termType rr:IRI ] ;
 rr:predicateObjectMap [
  rr:predicate sur:answeredIn ;
  rr:objectMap [
   rml:reference "session.value" ;
   rr:termType rr:IRI ] ] ;
 rr:predicateObjectMap [
  rr:predicate sur:hasAnswer ;
  rr:objectMap [
   rml:reference "answer.value" ;
   rr:termType rr:IRI ] ] ;
 rr:predicateObjectMap [
  rr:predicate sur:completesQuestion ;
  rr:objectMap [
   rml:reference "question.value" ;
   rr:termType rr:IRI ] ] .

:sessionTriplesMap a rr:TriplesMap ;
 rml:logicalSource :answerSource ;
 rr:subjectMap [
  rml:reference "session.value" ;
  rr:termType rr:IRI ] ;
 rr:predicateObjectMap [
  rr:predicate prov:atTime ;
  rr:objectMap [
   rml:reference "date.value" ;
   rr:datatype xsd:date ] ] ;
 rr:predicateObjectMap [
  rr:predicate prov:wasAssociatedWith ;
  rr:objectMap [
   rml:reference "person.value" ;
   rr:termType rr:IRI ] ] .
#
# Client query in SPARQL
# How many GGDM questions are available?
#
SELECT (
COUNT(DISTINCT ?completedQuestion) )
WHERE {
?completedQuestion sur:answeredIn ?session
}
#
# Rewritten query
#
SELECT (
COUNT(DISTINCT ?completedQuestion) )
WHERE {
{SELECT ?rvar28 ?rvar24 ?rvar29
        ?rvar27 ?rvar25 ?rvar26
 WHERE {
 ?rvar26 prov:atTime ?rvar27 ;
 prov:wasAssociatedWith ?rvar25 .
 ?rvar24 sur:answeredIn ?rvar26 ;
 sur:hasAnswer ?rvar29 ;
 sur:completesQuestion ?rvar28
 FILTER (?rvar28 IN (ggdm:question1,
 ggdm:question2,ggdm:question3,
 ggdm:question4,ggdm:question5,
 ggdm:question6,ggdm:question6-1,
 ggdm:question6-2,ggdm:question7,
 ggdm:question7-1,ggdm:question7-2,
 ggdm:question7-3,ggdm:question7-4,
 ggdm:question7-5,ggdm:question7-6,
 ggdm:question7-7,ggdm:question7-8,
 ggdm:question7-9,ggdm:question7-10,
 ggdm:question8-1,ggdm:question9-1,
 ggdm:question10,ggdm:question11,
 ggdm:question12,ggdm:question13,
 ggdm:question14)) } }
BIND(?rvar26 AS ?session)
BIND(?rvar24 AS ?completedQuestion)
}
```

Figure 2: Example of query rewriting. Prefix definitions in RML and SPARQL have been omitted.

5. Related work, concluding remarks

The application of views is, of course, very familiar from database systems. Furthermore, techniques for schema mapping and query rewriting have been developed in information integration and data exchange [8, 9]. For example, the successful Ontop system applies SPARQL-to-SQL rewriting [6].

Several researchers have explored the use of views and query rewriting for privacy [10, 11, 12]. The utility of expressive views as a bridge from the local vocabulary of pod data to client vocabularies was also pointed out by Arndt and Van Woensel [13], but query rewriting was not considered.

The novelty of our work lies in showing the applicability of these techniques and adapting them to the new Solid pod context. Furthermore, our use of RML to specify RDF-to-RDF

mappings is new, as is our SPARQL-to-SPARQL rewriting algorithm. Our algorithm (details to be published separately) involves an optimization technique that checks satisfiability on sets of equalities coming from matched triple patterns in the BGP against the RML mapping heads.

The design of interfaces for configuring the schema mappings in Web agents is an important direction for future research.

Acknowledgments

This work was supported by the Flanders AI Research Program, and also partly funded by the SolidLab Vlaanderen project (Flemish Government, EWI and RRF project VV023/10) and the FWO Project FRACTION (Nr. G086822N). We are indebted to Thomas Delva for his contributions to an earlier stage of this work, and to Bart Bogaerts, Anastasia Dimou, and Ruben Verborgh for inspiring discussions.

References

[1] S. Abiteboul, B. André, D. Kaplan, Managing your digital life, CACM 59 (2015) 32–35.
[2] P. Hitzler, A review of the Semantic Web field, CACM 64 (2021) 76–83.
[3] E. Mansour, A. Sambra, et al., A demonstration of the Solid platform for social web applications, in: 25th WWW Companion, 2016, pp. 223–226.
[4] R. Taelman, J. Van Herwegen, M. Vander Sande, R. Verborgh, Comunica: A modular SPARQL query engine for the Web, ISWC, 2018, pp. 239–255.
[5] A. Dimou, M. Vander Sande, et al., RML: A generic language for integrated RDF mappings of heterogeneous data, LDOW, 2014.
[6] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, Semantic Web 8 (2017) 471–487.
[7] D. Bender, K. Sartipi, HL7 FHIR: An agile and RESTful approach to healthcare information exchange, CBMS, 2013, pp. 326–331.
[8] A. Doan, A. Halevy, Z. Ives, Principles of Data Integration, Morgan Kaufmann, 2012.
[9] M. Arenas, P. Barceló, L. Libkin, F. Murlak, Foundations of Data Exchange, Cambridge University Press, 2014.
[10] A. Bonifati, U. Comignani, E. Tsamoura, Exchanging data under policy views, EDBT, 2021.
[11] S. Oulmakhzoune et al., Privacy query rewriting algorithm instrumented by a privacy-aware access control model, Ann. des Télécommunications 69 (2014) 3–19.
[12] M. Goncalves, M. Vidal, K. M. Endris, PURE: A privacy aware rule-based framework over knowledge graphs, DEXA, 2019, pp. 205–214.
[13] D. Arndt, W. Van Woensel, Decentralization rules: Linking Solid pods in different vocabularies using N3, First DKG Workshop, 2022. https://paul.ti.rw.fau.de/~ec69etyl/2022/dkg-22/DKG-22_paper_7.pdf.