

# Using Notation3 reasoning for crawling Linked Data in Solid Pods

Anon Ymous<sup>1</sup>

<sup>1</sup>Anon Research

## Abstract

Although Solid provides semantic data through a defined interface, the interoperability of applications is still an unsolved challenge. This poster paper presents a rule-based agent for discovering, collecting and processing differently structured Linked Data on Solid Pods. Declarative rules for link traversal and RDF querying are expressed in the logic language Notation3. These rule-based programs are executed in an *N3 Solid crawler* that combines Notation3 reasoning and an HTTP client for fetching RDF resources.

## Keywords

Social Linked Data (Solid), Notation3 (N3), Linked Data, rule-based link traversal

## 1. Introduction

A fundamental architecture design principle of Social Linked Data (Solid)<sup>1</sup> is the separation of data storage and applications (apps). Personal data is stored on decentralized, user-controlled data stores called Pods that provide data to several interchangeable apps. Even though Solid offers semantic data through a standardized interface [1], the interoperability of apps is still an unsolved challenge. The first main difficulty is the regulation of access to prevent corruption of the data for another app. The second main problem is the agreement of multiple apps on a shared data structure. In Solid, the data structure is not only defined by the vocabulary of the RDF triples but also by the partitioning of linked data in different LDP [2] containers of the Pod. Triples may reside in a single RDF document or link to other documents in the same or different LDP containers. The Solid community group is drafting a specification for Solid application interoperability [3] that includes technical measures for regulating how different agents and apps work with the same datasets on a Pod. This approach is based on a contract, an RDF description of agents, apps, access authorization and data structures called shape trees. However, these complicated contracts define strict rules which might discourage app developers.


This poster paper proposes a rule-based agent for improving Solid application interoperability by means of a combination of Notation3 (N3) [4, 5] reasoning and link following. The N3 language is used to describe declarative rules for fetching, processing and querying RDF resources from Solid Pods or the Web. A Notation3 reasoning and link traversal agent executes crawling rules and applies inference and query rules to the successively collected RDF datasets. The logical reasoning allows the application of RDFS and OWL entailment rules or the evaluation

---

*The 1st Solid Symposium Poster Session, co-located with the 2nd Solid Symposium, May 02 – 03, 2024, Leuven, Belgium*

✉ anon@ymous.com (A. Ymous)

ORCID 0000-0000-0000-0000 (A. Ymous)

 © 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

<sup>1</sup><https://solidproject.org/>

of equivalence or translation statements, e.g. for ontology matching. The link following component fetches required RDF resources depending on the results of the rule execution instead of hard-coded control flows. The rule-based crawler can be embedded into Solid apps to improve the interoperability between different data structures and to simplify the development process.

Section 2 briefly introduces Notation3 and implementations related to the N3 Solid crawler presented in section 3. The application scenarios are explained in section 4. The conclusion chapter gives an outlook on the expansion of the concept.

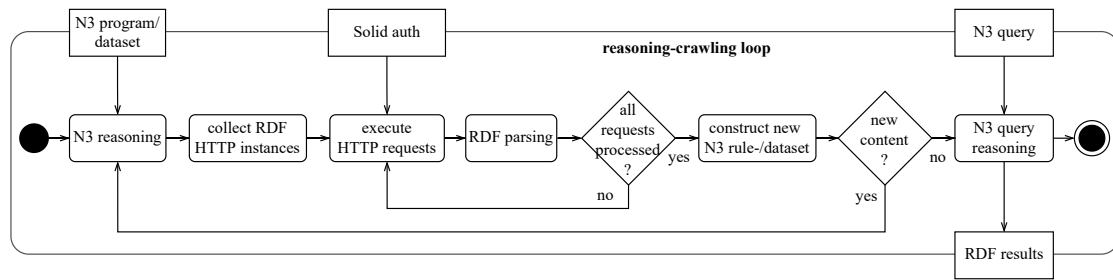
## 2. Related Work

Notation3 Logic [6] aims to implement decision-making abilities by extending the representational abilities of RDF and allowing operating and reasoning over information on the Web [4]. Notation3 (N3) is an assertion and logic language and a superset of RDF adding formulae, logical implication and functional predicates [5]. Negated forms and built-in functions [7] allow expressing logical rules with limited RDF processing. There are partial and full implementations of N3 and its built-in functions in different programming languages [8] with Cwm [9] being the first N3 reasoner. EYE is a generic reasoning engine for N3 that performs forward and backward chaining along Euler paths [10, 11]. The N3 Solid crawler uses EYE JS [12], a distribution of the EYE reasoner [11] for JavaScript using an SWI-Prolog WebAssembly.

The application of rule-languages such as N3 for the programming of user agents for Linked Data is not new [13]. Linked Data-Fu [14] is a "data processing system for data integration and system interoperation with Linked Data" that uses a rule language based on N3 and executes HTTP requests for RDF crawling or updating the state of resources in LDP [2], but not Solid. In comparison to the N3 Solid crawler, Linked Data-Fu executes deduction rules instead of an N3 reasoner and uses a subset of SPARQL for querying. Similar to the presented solution, Arndt and Van Woensel demonstrated the execution of N3-based ontology alignment rules for integrating RDF from decentralized Solid Pods that is modeled with two ontologies [15].

## 3. N3 Solid Crawler

The N3 Solid Crawler combines N3 reasoning and fetching of additional RDF resources in a loop. The activity sequence of the *reasoning-crawling loop* is shown in figure 1. The process starts with reasoning over the initial N3 program, a document which includes RDF statements and N3 rules for logical processing of RDF data. The program may already contain RDF nodes of type *http:Request* [16] or infer these nodes during reasoning. Through reasoning, the N3 rule in listing 1 collects *http:Request* instances which describe HTTP requests for fetching additional RDF documents. The N3 built-in functions *log:semantics* and *log:content* already provide a way to import public RDF resources from the Web [7]. However, in addition to the lack of Solid authentication, these functional properties do not allow the detailed modeling of HTTP requests [16]. An HTTP client then executes each of the HTTP request descriptions while an RDF parser processes the results. The HTTP client uses Solid authentication to fetch private RDF resources from Solid Pods. The N3 Solid crawler keeps track of the URIs that have been fetched to prevent requesting the same URI in the next iteration of the loop. The new RDF content is then merged



**Figure 1:** Activity diagram of the reasoning-crawling loop

with the initial N3 program to create the input for the next iteration of the reasoning-crawling loop. If there is no new RDF content from HTTP requests, a final reasoning step applies a N3 query rule for generating the RDF results of the process. Without a query, the result of the reasoning-crawling loop is the aggregated RDF dataset with inferred triples.

---

```

@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix httpm: <http://www.w3.org/2011/http-methods#>.
{
  (?req_uri
  {
    ?req a http:Request; http:mthd httpm:GET; http:requestURI ?req_uri.
  }
  ?req_uri_list) log:collectAllIn [].
} => {
  :GetRequestQuery :foundRequestUris ?req_uri_list.
}.

```

---

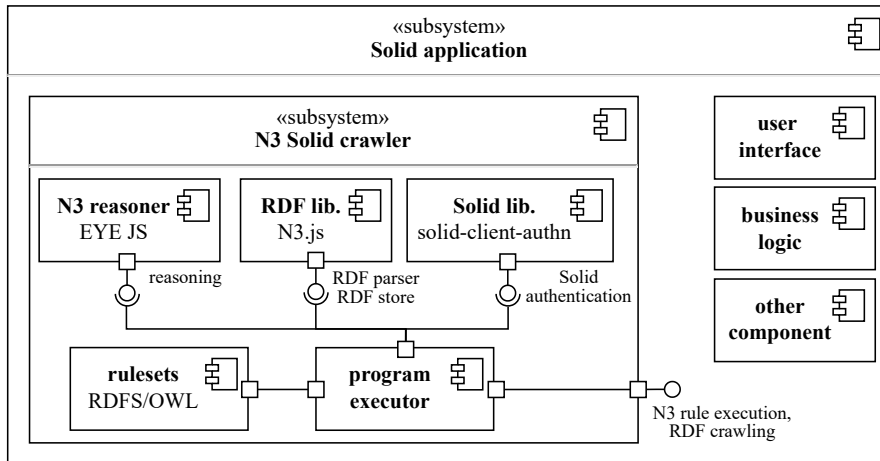
Listing 1: N3 rule for collecting RDF objects representing HTTP requests

The N3 Solid crawler is implemented as a library that can be embedded into client-side or server-side Solid apps for better application interoperability and developer experience. The components of the N3-based crawler for Solid Pods are illustrated in figure 2.

Declarative programming with N3 rules has advantages over traditional imperative and procedural programming of Solid apps. Instead of hard-coding the processing and decisions for crawling RDF, the N3 language allows to define logical rules for RDF processing and for when and how to follow links. Furthermore, the reasoning component executes formal logic, e.g. mapping expressions for ontology alignment [15]. This simplifies gathering RDF that is modeled with similar, but different ontologies. In comparison to bloated programming code with many if-clauses, the declarative N3 rules improve the reuse and maintenance of discovery and crawling operations for data on Solid Pods.

## 4. Application Scenarios

Application scenarios for the N3-rule-based Solid crawling range from discovery to collection and processing of partitioned information on Solid Pods and the Web.



**Figure 2:** Components of an N3 Solid crawler embedded in a Solid application

An example for the discovery of information is the crawling of profile information. A Solid profile is a description of a social agent’s identity that may include links to the agent’s storage, extended profile or preferences documents [17]. The starting point for this scenario is the WebID of an agent or a document that defines a *vcard:Group* of agents. Thus, the N3 example program appended in listing 2 includes RDF statements that describe an HTTP GET request to the WebID. An N3 rule is used to fetch any linked extended profile documents. The rule body specifies a condition, an RDF triple pattern that checks if a variable object is connected to the agent’s URI via the property *rdfs:seeAlso*. If this condition is true, the rule implies the creation of triples that describe an HTTP request to the variable that matched the URI of the extended profile. Another rule can be added to import any *solid:TypeIndex* documents linked in the profile. Solid type indexes [18] are registries for discovering data on the Solid Pod. Both link traversal rules are applied independently during N3 reasoning so that it is irrelevant if the initial or extended profile document includes the link to the type index.

An example for the collection and processing of RDF data is the calculation of an average value of sensor measurements, shown in listing 3 in the appendix. The necessary collection, string and math operations are expressed by means of the N3 built-in functions [7].

## 5. Conclusion and Future Work

This poster paper presented the concept of an N3 Solid crawler that combines N3 reasoning and link following for the discovery, collection and simple processing of RDF resources from Solid Pods and the Web. The author plans to implement the concept as a reusable Typescript library for Solid applications.

Future research includes the modeling of more extensive programs with N3. The concept could be extended with write capabilities and an abstract state machine to allow CRUD operations on Solid resources [13]. Notation3 logic has also been used to define a finite state machine for task network models [19].

## References

- [1] S. Capadisli, Solid technical reports, 2024. URL: <https://solidproject.org/TR/>.
- [2] A. Malhotra, S. Speicher, J. Arwe, Linked Data Platform 1.0, W3C Recommendation, W3C, 2015. <https://www.w3.org/TR/2015/REC-ldp-20150226/>.
- [3] J. Bingham, E. Prud'hommeaux, E. Pavlik, Solid Application Interoperability, Editor's Draft, Solid community, 2023. <https://solid.github.io/data-interoperability-panel/specification/>.
- [4] J. D. Roo, et al., Notation3 Language, Draft Community Group Report, W3C N3 Community Group, 2024. <https://w3c.github.io/N3/spec/>.
- [5] T. Berners-Lee, D. Connolly, Notation3 (N3): A readable RDF syntax, W3C Team Submission, W3C, 2011. <https://www.w3.org/TeamSubmission/n3/>.
- [6] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, J. Hendler, N3logic: A logical framework for the World Wide Web, *Theory and Practice of Logic Programming* 8 (2008) 249–269. doi:10.1017/S1471068407003213.
- [7] D. Arndt, P.-A. Champin, T. Duval, D. Tomaszuk, W. V. Woensel, P. Hochstenbach, Notation3 Builtin Functions, Draft Community Group Report, W3C N3 Community Group, 2023. <https://w3c.github.io/N3/reports/20230703/builtins.html>.
- [8] W3C N3 Community Group, List of N3 implementations, 2023. URL: <https://github.com/w3c/N3/blob/master/implementations.md>.
- [9] T. Berners-Lee, Cwm, 2009. URL: <https://www.w3.org/2000/10/swap/doc/cwm.html>.
- [10] R. Verborgh, J. D. Roo, Drawing conclusions from Linked Data on the Web: The EYE reasoner, *IEEE Software* 32 (2015) 23–27. URL: <https://josd.github.io/Papers/EYE.pdf>.
- [11] J. D. Roo, Euler Yet another proof Engine, 2023. URL: <https://eyereasoner.github.io/eye/>.
- [12] J. Wright, J. D. Roo, EYE JS - a distribution of EYE reasoner in the JavaScript ecosystem using WebAssembly, 2024. URL: <https://github.com/eyereasoner/eye-js>.
- [13] T. Käfer, A. Harth, Rule-based programming of user agents for Linked Data, in: T. Berners-Lee, S. Capadisli, S. Dietze, A. Hogan, K. Janowicz, J. Lehmann (Eds.), *Workshop on Linked Data on the Web co-located with The Web Conference 2018, LDOW@WWW 2018*, Lyon, France April 23rd, 2018, volume 2073 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018.
- [14] A. Harth, et al., *Linked Data-Fu*, 2017. URL: <https://linked-data-fu.github.io/>.
- [15] D. Arndt, W. V. Woensel, Decentralization rules: Linking Solid Pods in different vocabularies using Notation3, in: *DKG-22: 1st Workshop on Consumers of Distributed Knowledge Graphs in Digital, Industry and Space*, 2022. URL: [https://github.com/doerthe/oslo-to-fhir/blob/main/paper/Decentralisation\\_rules.pdf](https://github.com/doerthe/oslo-to-fhir/blob/main/paper/Decentralisation_rules.pdf).
- [16] J. Koch, C. A. Velasco, P. Ackermann, HTTP Vocabulary in RDF 1.0, W3C Working Group Note, W3C, 2017. <https://www.w3.org/TR/HTTP-in-RDF10/>.
- [17] T. Berners-Lee, S. Capadisli, V. Balseiro, T. Turdean, J. Zucker, Solid WebID Profile, Editor's Draft, Solid working group, 2024. <https://solid.github.io/webid-profile/>.
- [18] J. Zucker, V. Balseiro, S. Capadisli, T. Berners-Lee, T. Turdean, Type Indexes, Editor's Draft, Solid working group, 2023. <https://solid.github.io/type-indexes/>.
- [19] W. Van Woensel, S. Abidi, K. Tennankore, G. Worthen, S. S. R. Abidi, Explainable decision support using task network models in Notation3: Computerizing lipid management clinical guidelines as interactive task networks, in: *Artificial Intelligence in Medicine*, Springer International Publishing, Cham, 2022, pp. 3–13.

## A. Example N3 Programs

---

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix httpm: <http://www.w3.org/2011/http-methods#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix solid: <http://www.w3.org/ns/solid/terms#>.

# seed request
[] a http:Request; http:mthd httpm:GET;
  http:requestURI <https://id.inrupt.com/myusername>.
# rule for fetching extended profile
{
  ?webid a foaf:Person; rdfs:seeAlso ?extprofile.
  _:b log:notIncludes { ?extprofile a ?type }.
} => {
  [] a http:Request; http:mthd httpm:GET; http:requestURI ?extprofile.
}.
# rule for fetching public type index
{
  ?webid solid:publicTypeIndex ?ptindex.
  _:d log:notIncludes { ?ptindex a solid:TypeIndex }.
} => {
  [] a http:Request; http:mthd httpm:GET; http:requestURI ?ptindex .
}.
}
```

---

Listing 2: Section of an N3 program for crawling Solid profile information

---

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix sosa: <http://www.w3.org/ns/sosa/>.

{
  ?sensor a sosa:Sensor.
  (?tempvalue
  {
    ?observation a sosa:Observation; sosa:madeBySensor ?sensor; sosa:hasSimpleResult ?tempvalue.
  }
  ?tempvalues) log:collectAllIn [].
  ?tempvalues math:sum ?sum; math:memberCount ?count.
  ?count math:greaterThan 0.
  (?sum ?count) math:quotient ?avg.
} => {
  ?sensor <http://example.org/averageValue> ?avg.
}.
}
```

---

Listing 3: Section of an N3 program for calculating the average value of temperature measurements of different sensors