

# Using Notation3 reasoning for crawling Linked Data in Solid storages

Sascha Meckler<sup>1</sup>

<sup>1</sup>Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits IIS, Germany

## Abstract

Although Solid provides semantic data through a defined interface, the development of applications that are able to discover and consume decentralized data is still a complex task. This paper presents the concept of a rule-based client for discovering, collecting and processing differently structured Linked Data on Solid storages. 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 public and private RDF resources from Solid storages and the Web.

## Keywords

Social Linked Data, decentralized data storage, Solid Pod, rule-based link traversal

## 1. Introduction

A fundamental architecture design principle of Solid<sup>1</sup>, originally Social Linked Data, is the separation of data storage and applications (apps). Personal data is stored on decentralized, user-controlled data stores also known as Solid 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 interlinked data in different LDP [2] containers of a Solid storage. Triples may reside in a single RDF document or link to other documents in the same or different LDP containers. In summary, the Solid environment is characterized by a decentralized storage of a large number of small, interlinked RDF and non-RDF sources. The Solid community group is drafting a specification for 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 kind of contract, an RDF description of agents, apps, access authorization and data structures called shape trees. However, these complicated descriptions define rigid, inflexible rules which might discourage app developers.


To create applications that are able to discover and consume differently structured data on Solid storages, this poster paper proposes a rule-based client which combines Notation3 (N3)

---

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

✉ [sascha.meckler@iis.fraunhofer.de](mailto:sascha.meckler@iis.fraunhofer.de) (S. Meckler)

ORCID  0000-0003-4846-1626 (S. Meckler)

 © 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/>

[4, 5] reasoning and link following. The N3 language is used to describe declarative rules for fetching, processing and querying RDF resources from Solid storages or the Web. A Notation3 reasoning and link traversal component 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 and therefore the evaluation 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 is meant to be embedded into Solid apps to improve flexible processing of different data structures and to simplify the development process.

Section 2 briefly introduces Notation3 and its implementations related to the N3 Solid crawler and describes the relation to Link Traversal Query Processing. The concept of the N3 Solid crawler and the components of a possible implementation are presented in Section 3. The application scenarios are explained in Section 4 while 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 interoperability 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 storages that is modeled with two ontologies [15].

Although the N3 Solid crawler is focused on practical applications for the Solid ecosystem, the underlying concept has overlaps with existing research about querying the Web of Linked Data, in particular Link Traversal Query Processing (LTQP). LTQP adopts the follow-your-nose principle of Linked Data during query execution, in which new RDF documents are continuously crawled by following links between documents. The declarative query language LDQL [16] separates the selection of the query-relevant region of the Web and the evaluation of a query over the collected documents. Link path expressions, so-called link patterns, specify the range of link traversals while the final query is defined with SPARQL 1.1 graph patterns [16]. In the N3 Solid crawler concept, link following is defined by N3 rules that generate RDF triples

representing HTTP requests while the separate query is expressed with graph patterns wrapped inside an N3 rule. The application of LTQP for decentralized Solid storages can be optimized by exploiting the existing structural information of the Solid environment such as the Solid type index [17] or LDP [2] containers [18]. With these "closed-world" assumptions, LTQP algorithms are able to deliver first results within one second and can guarantee complete results in the scope of Solid storages [18]. A traversal-based query engine for Solid was implemented and tested as a module for the Comunica SPARQL framework [19]. In the presented N3-based concept, the discovery and query strategy is defined by the users themselves. In comparison with LTQP for Solid [18], the repeated N3 reasoning comes with a performance penalty but features implicit type knowledge throughout the link traversal and querying.

### 3. N3 Solid Crawler

The N3 Solid crawler combines N3 reasoning and the loading 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. In contrast to the link patterns of LDQL [16], the crawler does not match patterns to decide which links to follow. Instead, a link extractor component collects RDF nodes of type *http:Request* [20] which may already exist in the initial program as so-called "seed requests" or which are inferred through reasoning. For example, the N3 rule in Listing 1 creates an instance of *http:Request* which describes an HTTP GET request for fetching the public type index document linked from a Solid profile document. A more detailed example for discovering Solid profile information is presented in Section 4. 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 [20], e.g. the specification of HTTP headers. The collected HTTP request descriptions are stored in a link queue that keeps

---

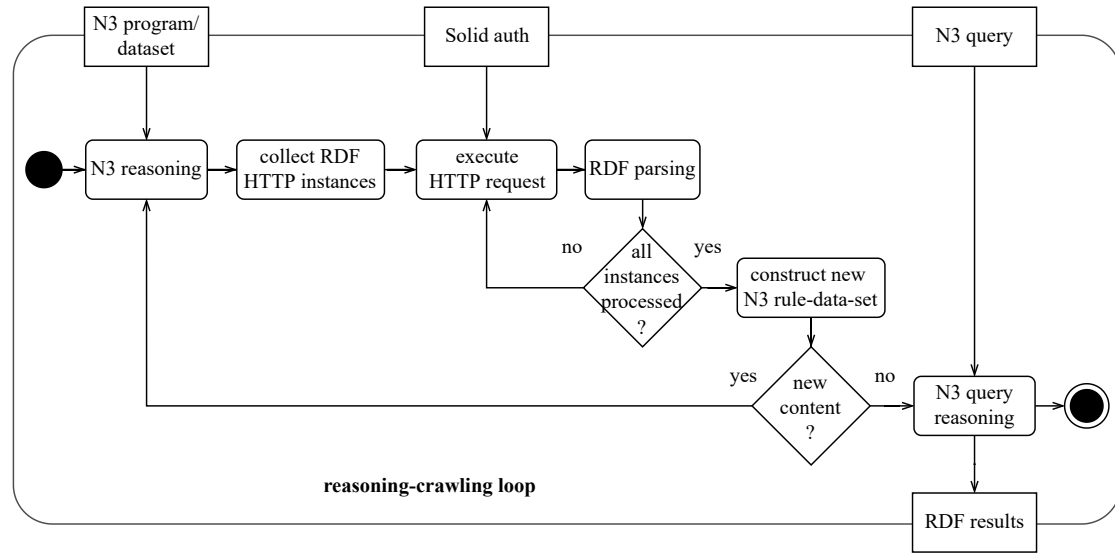
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix httpm: <http://www.w3.org/2011/http-methods#>.
@prefix solid: <http://www.w3.org/ns/solid/terms#>.
{
  ?webid solid:publicTypeIndex ?ptindex.
  _:d log:notIncludes { ?ptindex a solid:TypeIndex }.
} => {
  [] a http:Request; http:mthd httpm:GET; http:requestURI ?ptindex .
}.
```

---

Listing 1: N3 rule for the definition of an HTTP requests for fetching a type index document

track of the URIs that have been fetched to prevent requesting the same URI in the next iteration of the loop. An HTTP client then executes each of the HTTP request objects while an RDF parser processes the results. The HTTP client uses Solid authentication to fetch private RDF resources from Solid Pods. The new RDF content is then merged with the initial N3 program

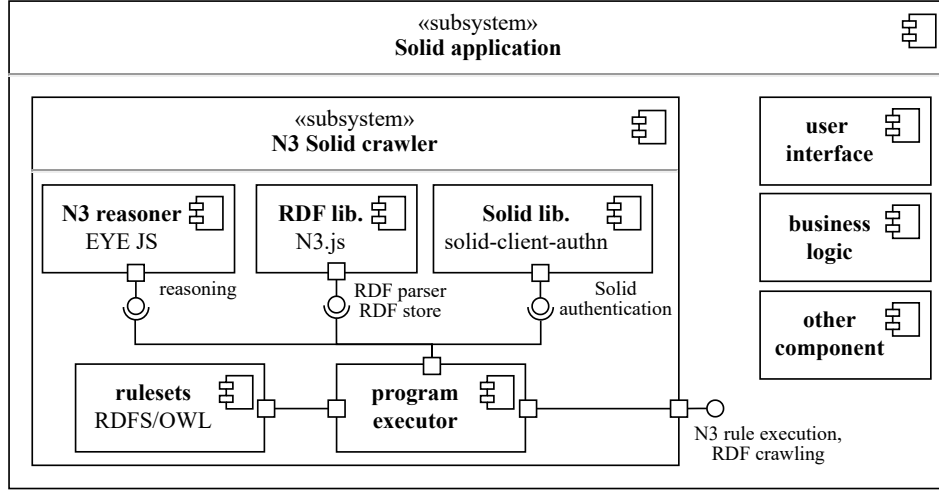
to create the input for the next iteration of the reasoning-crawling loop. If every *http:Request* instance has been processed and there is no new RDF content, a final reasoning step applies an N3 query rule for generating the RDF results of the process. Without a query, the result of the reasoning-crawling loop is the aggregated RDF dataset with inferred triples. In this simple configuration, results are not available until the end of the process. However, the process could return incomplete intermediate results by applying the N3 query reasoning step in every iteration of the reasoning-crawling loop. The evaluation of the N3-based query after every batch of HTTP requests is a compromise between early feedback and performance.



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

The N3 Solid crawler concept can be implemented as a library for client-side or server-side Solid applications. The necessary components of the N3-based crawler are illustrated in Figure 2. The reasoning-crawling loop is implemented in a so-called program executor component that uses the EYE JS, an RDF/JS and a Solid library. The crawler should provide predefined N3 rule sets for RDFS and OWL entailment rules [11], e.g. for the OWL-LD subset [21].

By embedding the rule-based crawler, Solid apps benefit from the advantages of declarative programming with N3 rules instead of traditional imperative and procedural programming. Instead of hard-coding decisions and control flows, 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 scattered data on Solid storages. In summary, the application of reasoning during the discovery and processing of RDF contributes to application interoperability. If a Solid app is more flexible in handling data that is modeled with similar vocabularies or data that is partitioned in different ways, the app is more likely to work with data sets created by other apps.



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

## 4. Application Scenarios

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

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 [22]. The starting point for this scenario is the WebID of an agent. Thus, the N3 example program appended in Listing 2 includes RDF statements that describe an initial HTTP GET request to a WebID. A first N3 rule is used to fetch any linked extended profile document. 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* and an N3 formula that verifies that the current dataset not already includes information about the extended profile. 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 is added to import any *solid:TypeIndex* documents linked in the profile. Solid type indexes [17] are registries for discovering specific data on a Solid storage. 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. Instead of a given WebID, the N3 program could also start with an RDF document which describes a *vcard:Group* and fetch profile information for each member of this group. Furthermore, the N3 program can provide context for the reasoning, for example in the form of equivalence and type relations. If the program defines the subclass relation between *foaf:Person* and *foaf:Agent* and imports RDFS entailments, the first traversal rule also matches Solid profiles in which the WebID is defined as *foaf:Person*.

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 presents 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 storage and the Web. The author plans to implement the concept as a reusable, isomorphic Typescript library for Solid applications. With the implementation, the performance can be set in relation to existing querying techniques for Solid such as LTQP [18, 19].

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 [23].

**Resource Availability Statement:** A demo implementation<sup>2</sup> of the N3 Solid crawler concept and the poster<sup>3</sup> which was presented at the 2<sup>nd</sup> Solid Symposium 2024 are available on the Solid server of Fraunhofer IIS.

## Acknowledgments

This work has been partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) through the "Antrieb 4.0" project (Grant No. 13IK015B).

## 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. De 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. Van 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>.

<sup>2</sup><https://solid.iis.fraunhofer.de/SCS-DS-IoT/public/2024/n3solidcrawler/>

<sup>3</sup>[https://solid.iis.fraunhofer.de/SCS-DS-IoT/public/2024/n3solidcrawler/Poster\\_N3SoC.pdf](https://solid.iis.fraunhofer.de/SCS-DS-IoT/public/2024/n3solidcrawler/Poster_N3SoC.pdf)



- [10] R. Verborgh, J. De 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. De Roo, Euler Yet another proof Engine, 2023. URL: <https://eyereasoner.github.io/eye/>.
- [12] J. Wright, J. De 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. Van 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] O. Hartig, J. Pérez, Ldql: A query language for the web of linked data, *Journal of Web Semantics* 41 (2016) 9–29. doi:<https://doi.org/10.1016/j.websem.2016.10.001>.
- [17] 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/>.
- [18] R. Taelman, R. Verborgh, Link traversal query processing over decentralized environments with structural assumptions, in: T. R. Payne, et al. (Eds.), *Proceedings of the 22nd International Semantic Web Conference*, Springer, 2023, pp. 3–22. doi:10.1007/978-3-031-47240-4\_1.
- [19] R. Taelman, et al., Link traversal for Comunica, 2024. URL: <https://github.com/comunica/comunica-feature-link-traversal>.
- [20] 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/>.
- [21] B. Glimm, A. Hogan, M. Krötzsch, A. Polleres, Owl: Yet to arrive on the web of data?, *ArXiv abs/1202.0984* (2012). URL: <https://api.semanticscholar.org/CorpusID:585128>.
- [22] 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/>.
- [23] 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 rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@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:Agent; 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 .
}.
# context for reasoning
foaf:Person rdfs:subClassOf foaf:Agent.
```

---

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