HTWK Leipzig University of Applied Sciences

Data Model

Florian Gudat

Version ea5e943, 2024-06-17

Table of Contents

Version

ea5e943, 2024-06-17

Editor

Florian Gudat

Module

Mastermodul (C533.2 Compulsory module)
https://modulux.htwk-leipzig.de/modulux/modul/6291

Module Supervisor

Prof. Dr.-Ing. Jean-Alexander Müller

Lecturer

Herr Prof. Dr. rer. nat. Andreas Both
Herr M. Sc. Michael Schmeißer

Institute

Leipzig University of Applied Sciences

Faculty

Computer Science and Media

Acronyms

ACL	Access Control List
ACP	Access Control Policy
API	Application Programming Interface
CRUD	Create, Read, Update and Delete
CSS	Community Solid Server
DNS	Domain Name System
DPC	Data Privacy Cockpit
DPV	Data Privacy Vocabulary
DPoP	Demonstration of Proof-of-Possession
ESS	Enterprise Solid Server
GDPR	General Data Protection Regulation
HTTPS	Hypertext Transfer Protocol Secure
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IEC	International Electrotechnical Commission
IE	Information Engineering
IP	Internet Protocol
ISO	International Organization for Standardization
LTS	Long-Term Support
N/A	Not Applicable
ODRL	Open Digital Rights Language
OIDC	OpenID Connect
OSI	Open Systems Interconnection
RDF	Resource Description Framework

ROA	Resource-Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
ShEx	Shape Expressions
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAC	Web Access Control

Namespaces

This enumeration lists the prefixes and the associated namespace. It will be used as the standard syntax for RDF prefixes. When a prefix is followed by a colon symbol, the part after the colon can be appended to the namespace URI, thereby creating a new URI.

acl	http://www.w3.org/ns/auth/acl#
al	*dynamic* (see Custom Vocabulary)
claim	urn:claim# (see Custom Vocabulary)
ex	*example*
foaf	http://xmlns.com/foaf/0.1/
http	http://www.w3.org/2011/http#
interop	http://www.w3.org/ns/solid/interop#
ldp	http://www.w3.org/ns/ldp#
pim	http://www.w3.org/ns/pim/space#
rdfs	https://www.w3.org/2000/01/rdf-schema#
solid	http://www.w3.org/ns/solid/terms#
st	http://www.w3.org/ns/shapetrees#

The prefixes and namespaces enumerated above are applicable to diagrams, listings, and inline listings throughout the entirety of the document.

Notation

The diagrams in this document were generated using Asciidoctor Diagram 2.3.1[1] and its bundled PlantUML[2] version.

Unless otherwise specified, all framed diagrams will use the UML 2.5.1[3] standard, constrained by the limitations of PlantUML. As defined in the standard, the following abbreviations will be utilized to identify the type of UML diagram:

cmp component diagram

sd interaction diagram

stm state machine diagram

In addition to the UML abbreviation, the following abbreviations are used to identify non-UML diagrams:

dm data model diagram; The information structure is entirely based on RDF, and will be presented as entity-relationship diagrams in Clive Finkelstein's IE[4] notation, with some additional elements. The text in the double angle brackets will define the entity type (e.g., [Dataset], [Thing], …). The path labels indicate potential routes through the graph structure, while the number within the bracket indicates the branch that has been taken.

wbs work breakdown structure; This diagram is a decompositional diagram[5], intended for use in hierarchical structures, originally designed as a project management tool. In this context, it is used to illustrate any kind of hierarchical structure.

All diagrams and figures presented in this work were created by the author. Any discrepancies have been highlighted in the corresponding figures.

[1] https://docs.asciidoctor.org/diagram-extension/latest/

[2] https://plantuml.com

[3] https://www.omg.org/spec/UML/2.5.1

[4] https://plantuml.com/en/ie-diagram

[5] https://plantuml.com/en/wbs-diagram

Chapter 1. Solid Application Interoperability

The Solid Specification outlines the overall framework of the system-wide data model. Additionally, the Solid Application Interoperability Specification[1], an extension to the Solid ecosystem, addresses application-independent design and a uniform mechanism for data discovery. It should be noted that the Specification has not yet been fully matured or implemented by any Solid Provider. However, it can be used in part without a full implementation of the Solid extension. The Editor's Draft of November 7th, 2023 introduces a mechanism for discovering registered data without requiring knowledge of the physical structure of the file system or HTTP endpoints. An application only needs to be aware of the profile document and follow the suggested references in the specification. Figure 1 illustrates these entities and relations. `DataType` and `DataElement` represent a selectable data type and element, respectively.

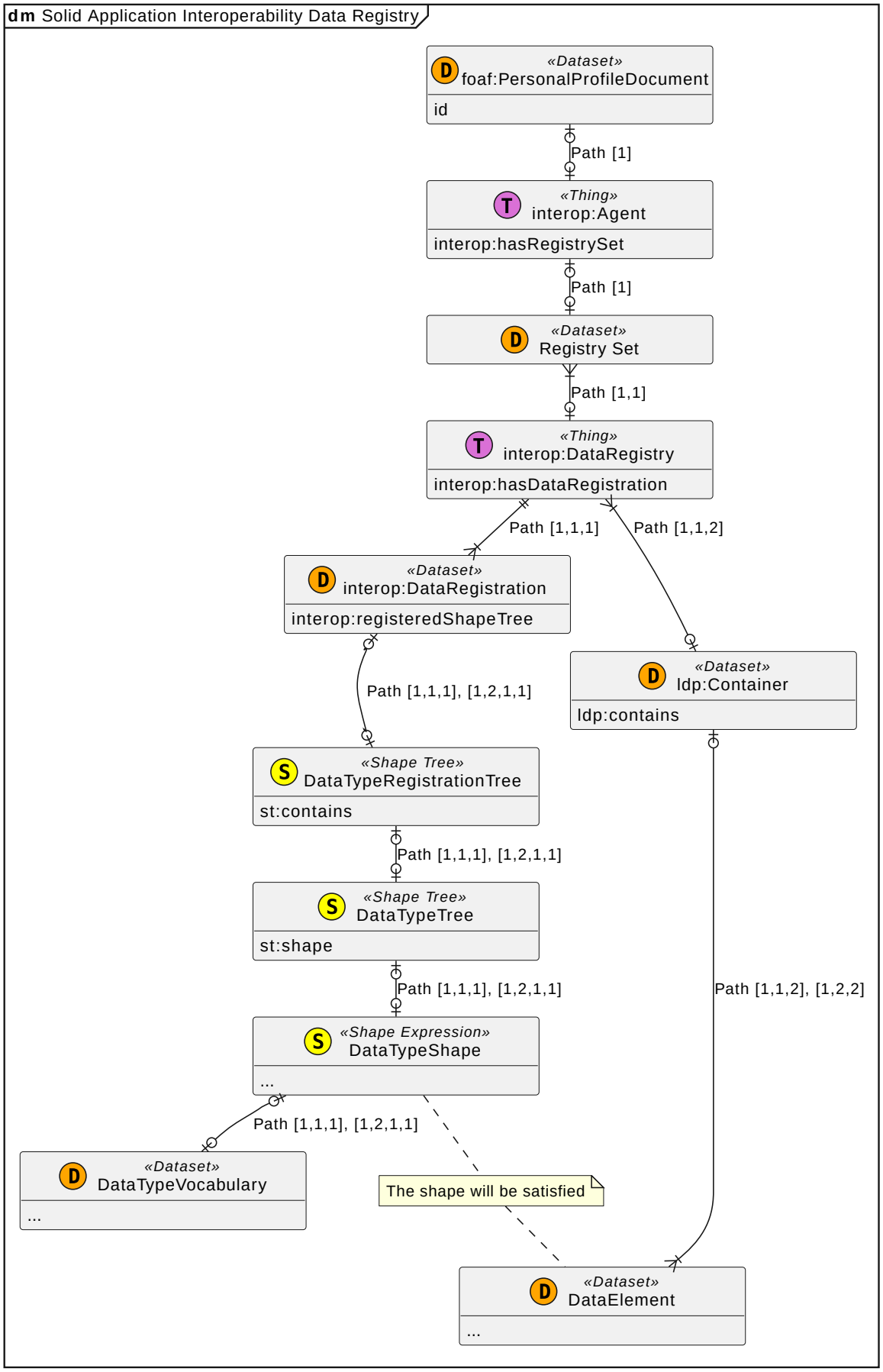

Figure 1. IE diagram of the Solid Application Interoperability Data Registry

The entities and relations in Figure 1 represent a partially implemented data registry component of the Solid Application Interoperability specification by Bingham et al. (2023). As described in the specification, an agent must declare an interop:Agent in the personal profile document to participate in the Solid Application Interoperability Specification. From there, one can follow the specified path, starting with the registry set, which is referred from the declared agent. The registry set contains an `interop:DataRegistry`, which refers to an `interop:DataRegistraion`. As the `interop:DataRegistraion` resource is a resource container (`ldp:Container`), all contained resources will apply the `interop:DataRegistraion` attributes. These attributes are defined in the registered ShapeTree, which is referred to from the `interop:DataRegistraion`. The registered ShapeTree defines the shape of the contained resources, by referring to the Shape. The Shape will point to a Shape Expression once more. The Shape Expression defines the data types of the predicates utilized in the vocabulary.

[1] https://solid.github.io/data-interoperability-panel/specification/

Chapter 2. Entity-Relationship Model

The Entity-Relationship Model is based on the Solid Application Interoperability specification and describes the logical arrangement of the system's data. This selected part of the specification can be used without further modification while the specification is still a draft. However, the current state of the specification does not fully satisfy the needs of a claiming mechanism. Figure 2 illustrates the additions to the model that are necessary to enable this mechanism. The full model can be found in Appendix A.

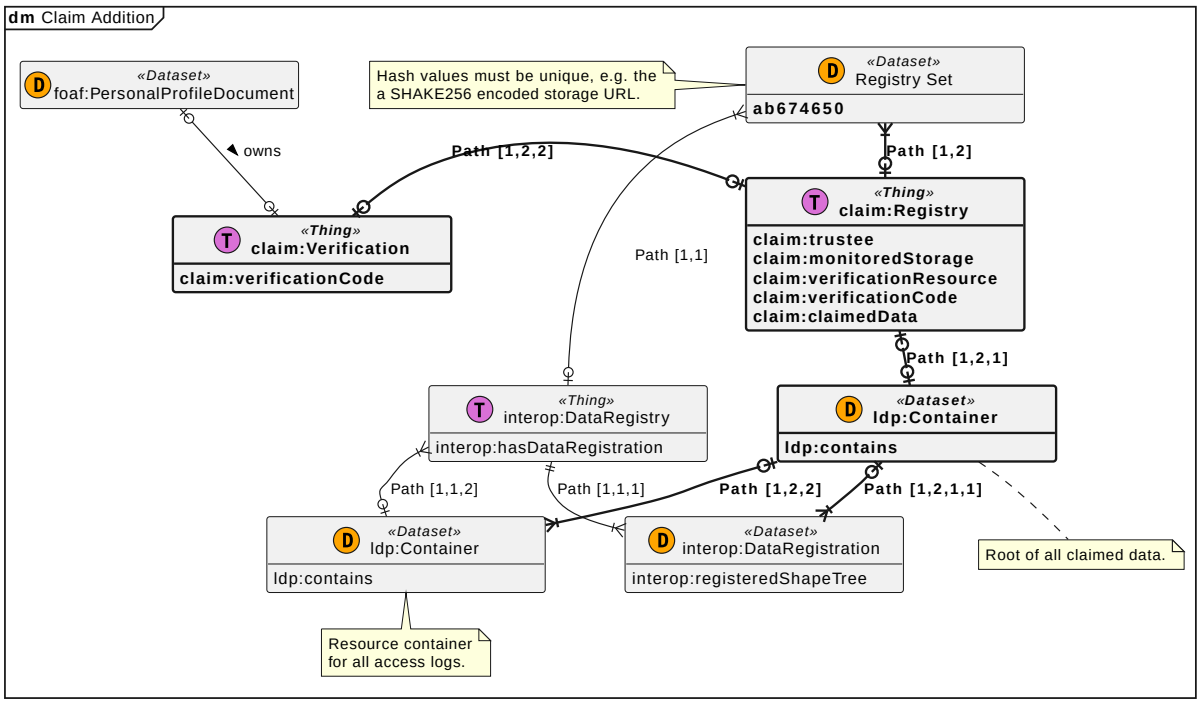

Figure 2. IE diagram of the Claim addition to the Solid Application Interoperability Data Registry

The model meets the requirements of the Solid Application Interoperability Specification, which is omitted in Figure 2. However, as described in Solid Application Interoperability, the data discovery will begin from the personal profile document. This document declares an `interop:Agent`, which refers to a registry set. This declaration informs agents that the data model will be modeled according to the specification. The registry set contains a list of all references to data registrations. A data registration is a resource container that contains all resources in a given tree[1] and shape[2].

As the interoperability specification does not address the handling of multi-agent data or require agents to participate in the specification, some enhancements have been made to the data model. In Figure 2, the highlighted additions (bold) to the model include the requirement that Things contained by the registry set must have a unique identifier based on the claimed item, which in this case is the hashed (SHAKE256) storage URL. The Thing's type is claim:Registry, a newly introduced Claim Vocabulary that will be explained in detail in the Custom Vocabulary section.

The `claim:Verification` is a resource within the observed storage that serves to verify using a verification code. This code must correspond to the verification code within the `claim:Registry` to authorize the trustee's access to the claimed data.

[1] https://shapetrees.org/TR/specification/

[2] https://shex.io/shex-semantics/

Chapter 3. Information Retrieval

The DPC agent captures, manages, and presents client data. Data can only be retrieved through the DPC agent. Figure 2 shows five paths through the data structure. Two of the paths are alternative paths that lead to the same leaf of the graph. The resulting data that can be received is:

AccessLogShape	following path [1,1,1], [1,2,1,1]
AccessLog	following path [1,1,2], [1,2,2]
Verification	following path [1,2,2]

The bracketed numbers indicate which branch to follow to access the described data.

Chapter 4. Custom Vocabulary

The vocabulary provided by the [Solid Ecosystem] does not cover all the necessary information that has been introduced in the data model. An custom RDF vocabulary for the claim process and logging of access has been introduced to support this.

4.1. Claim Vocabulary

Besides the ACL vocabulary, that allows the access granting of resources of a web-server, there are several other model for processing of restricted data. The ODRL Information Model[1] for instance, aims to standerize the permission, prohibition, and obligation of general content. The DPV[2] however enables expressing machine-readable metadata about the use and processing of personal data, with focus on the GDPR[3]. In order to prevent the creation of another information structure besides the model inherited from the Solid Application Interoperability and the limited options of integrating these model into the Solid Application Interoperability a custom vocabulary for the claiming mechanism has been introduced.

Listing 1. Custom Vocabulary: Claim

```
<#Registry>
    a          rdfs:Class ;
    rdfs:label "A registry entry for data that has been the subject of a trustee
claim"@en .

<#Verification>
    a          rdfs:Class ;
    rdfs:label "A verification resource, located in monitored storage" .

<#trustee>
    a          rdf:Property ;
    rdfs:label "The WebID reference of the agent requesting access to the claimed
data"@en .

<#monitoredStorage>
    a          rdf:Property ;
    rdfs:label "The observed storage reference"@en .

<#verificationResource>
    a          rdf:Property ;
    rdfs:label "The reference to the verification resource in the monitored
storage"@en .

<#verificationCode>
    a          rdf:Property ;
    rdfs:label "A random hash in the registry and verification resource"@en .

<#claimedData>
    a          rdf:Property ;
    rdfs:label "The reference to the resource container of all claimed data
```

```
resources"@en .
```

The claiming vocabulary presented in Listing 1, provides an illustrative example of how such a vocabulary might be constructed. However, in the implementation, this approach has not been employed in an effective manner. For the purposes of mocking, the URL references were sufficient, as the data was not fetched from the vocabulary. Nevertheless, the semantics presented in the listing are accurate. Uses of the RDF vocabulary are shown in Claim Registry (Data Model) and Verification (Data Model).

4.2. Access Log Vocabulary

The access log vocabulary is a dynamically generated vocabulary from the agent, produced in its own context. For instance, the DPC agent generates it at `http://proxy.localhost:4000/dpc/ns/log`. This enables each agent to bring its own vocabulary if necessary. The vocabulary is a condensed and human-readable form of the HTTP Vocabulary[4]. In order to facilitate comprehension by non-expert users, the vocabulary was introduced in a simplified form, as illustrated in Listing 2.

Listing 2. Custom Vocabulary: Access Log

```
<#AccessLog>
    a          rdfs:Class ;
    rdfs:label "AccessLog"@en .

<#date>
    a          rdf:Property ;
    rdfs:label "Accessed at"@en .

<#accessor>
    a          rdf:Property ;
    rdfs:label "Accessing agent"@en .

<#application>
    a          rdf:Property ;
    rdfs:label "Accessing application"@en .

<#resource>
    a          rdf:Property ;
    rdfs:label "Accessed resource"@en .

<#action>
    a          rdf:Property ;
    rdfs:label "Action"@en .
```

The relationship between the HTTP Vocabulary and the data is that the majority of the data originates from a regular request object. For instance, `al:resource`

matches the `http:absolutePath` property. As the vocabulary is custom, additional processing has been introduced. `al:action`, which matches the `http:methodName`, such as `POST`, `GET`, etc., has been converted to CRUD operations. `al:accessor` is a part of the serialized authorization header, equivalent to `http:RequestHeader`. Finally, the `al:application` property is intended to display the application name that appears when a Solid application is requesting data access. When granting access, the token is stored and associated with each authorized request. However, this technique is only effective when using the [Authorization Code Flow] authorization method.

It is also noteworthy that `al:accessor` and `al:application` may be absent in certain instances. To illustrate, if a resource is accessible to the general public, there is no authorized request and thus no requesting agent or logged-in Solid application.

The utilization of the RDF vocabulary is illustrated in Access Log (Data Model).

[1] https://www.w3.org/TR/odrl-model/

[2] https://w3c.github.io/dpv/dpv/

[3] https://eur-lex.europa.eu/eli/reg/2016/679/oj

[4] https://www.w3.org/TR/HTTP-in-RDF/

Chapter 5. Serialized Data Model

Before looking at the serialized model, it is important to understand the structure of the HTTP endpoints. The storage URLs for the HTTP APIs will begin with a storage identifier added as a suffix to the base URL. Figure 3 shows the storage URLs at the second level. The data of the corresponding agent will be represented below this node.

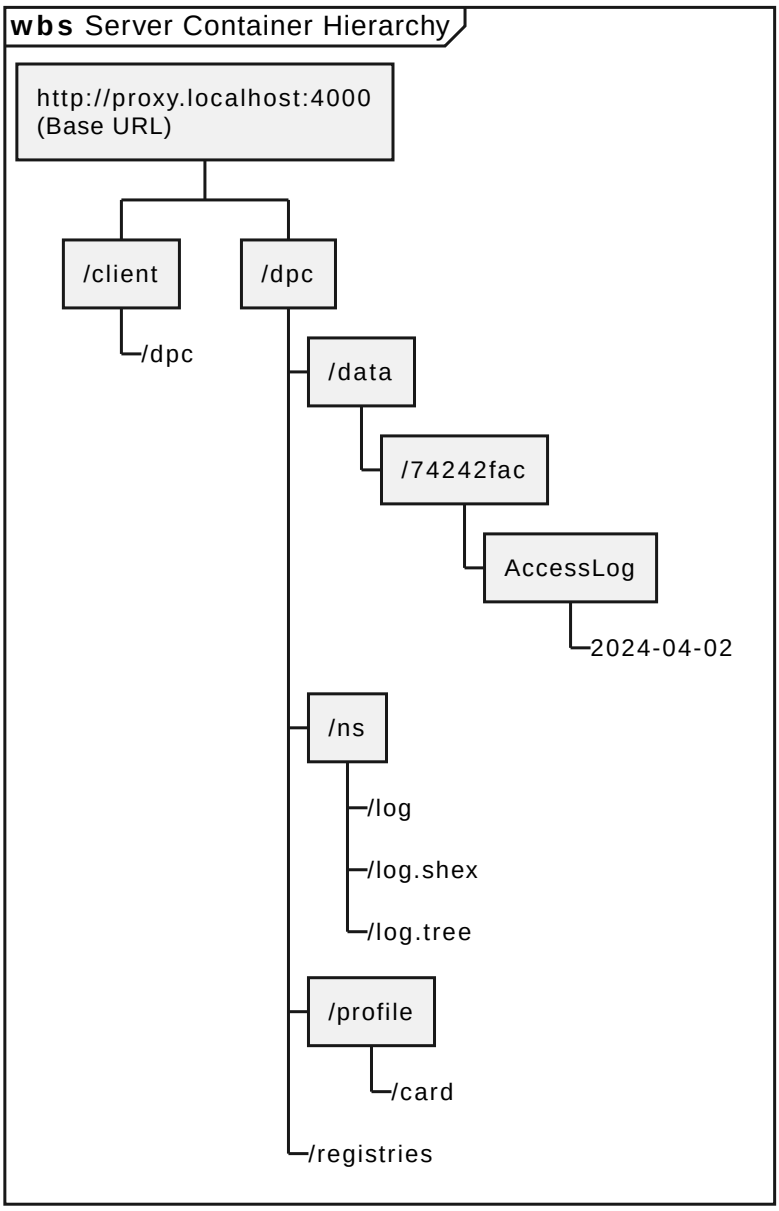

Figure 3. The structure of the HTTP endpoints

The two most commonly used serialization formats for RDF-based data in data-driven web-based systems are `text/turtle` and `application/ld+json`. This inspection does not focus on data storage, as the [Solid Provider] is considered replaceable. However, HTTP APIs use Turtle as the exchange format for communica-

tion, which will be displayed below. As part of the structural hierarchy shown in Figure 3, all resources and listings refer to the data model shown in Figure 2.

5.1. Personal Profile Document (Data Model)

To participate in the Solid Application Interoperability Specification, an `interop:Agent` must be declared in the profile document. This node will also refer to the registry set. Listing 3 presents the corresponding RDF fragment.

Listing 3. `interop:Agent` *[Thing] at http://proxy.localhost:4000/dpc/profile/card*

```
<#id>
    a                       interop:Agent ; ①
    interop:hasRegistrySet <http://proxy.localhost:4000/dpc/registries> . ②
```

① Declaration as `interop:Agent`.

② Reference to the registry set.

Followed Path [1]

5.2. Registry Set (Data Model)

The registry set contains an entry for each agent who has claimed data captured by the DPC agent. This captured data will be referred to as data registration. The subject of the RDF triple, however, must be unique and built based on the claimed subject. In this case, it will be the hashed storage URL. Listing 4 presents the corresponding RDF fragment.

Listing 4. `interop:DataRegistry` *[Thing] at http://proxy.localhost:4000/dpc/registries*

```
<#ab674650> ①
    a                           interop:DataRegistry;
    interop:hasDataRegistration
<http://proxy.localhost:4000/dpc/data/74242fac/AccessLog/>.
```

① The hashed (SHAKE256) storage URL.

Followed Path [1,1]

5.3. Data Registration (Data Model)

The Shape Tree data is referenced in the data registration. As it is a container resource (see Container (Access Logs from Data Model)), all child resources will satisfy the referenced Shape Tree. Listing 5 presents the corresponding RDF fragment.

Listing 5. `interop:DataRegistration` *[Thing] at http://proxy.localhost:4000/dpc/data/74242fac/AccessLog/*

```
<>
    a                           interop:DataRegistration ;
    interop:registeredBy        <http://proxy.localhost:4000/dpc/profile/card#id> ;
    interop:registeredAt        "2024-04-02T16:00:09.959Z"^^xsd:dateTime ;
    interop:registeredShapeTree
<http://proxy.localhost:4000/dpc/ns/log.tree#AccessLogRegistrationTree> . ①
```

① The referenced Shape Tree.

Followed Path [1,1,1], [1,2,1,1]

5.4. Shape Trees (Data Model)

Both Shape Trees, `AccessLogRegistrationTree`, and `AccessLogTree` define the contents of the referring [container_resource]. The `AccessLogRegistrationTree` defines the resources that contain Shape Tree Resources in a given shape. The referenced Shape Expression declares the form of the shape. Listing 6 presents the corresponding ShapeTree fragment.

Listing 6. Shape Tree at http://proxy.localhost:4000/dpc/ns/log.tree

```
PREFIX st: <http://www.w3.org/ns/shapetrees#> .
PREFIX log-shex: <http://proxy.localhost:4000/dpc/ns/log.shex#>.

<#AccessLogRegistrationTree>
  a st:ShapeTree ;
  st:expectsType st:Container ;
  st:contains <#AccessLogTree> . ①

<#AccessLogTree>
  a st:ShapeTree ;
  st:expectsType st:Resource ;
  st:shape log-shex:AccessLogShape . ②
```

① The internal reference to `AccessLogTree`

② The reference to the Shape Expression (Data Model)

Followed Path [1,1,1], [1,2,1,1]

5.5. Shape Expression (Data Model)

ShEx defines the schema for every literal associated with a predicate of the vocabulary. The RDF vocabulary will not be listed further. Listing 7 presents the corresponding ShEx fragment.

Listing 7. Shape Expression

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX shx: <http://www.w3.org/ns/shex#>
PREFIX log: <http://proxy.localhost:4000/dpc/ns/log#>

<#AccessLogShape> {
  log:date xsd:dateTime ;
  log:accessor IRI ;
  log:application xsd:string ;
  log:application xsd:string ;
  log:resource xsd:string ;
  log:action xsd:string
}
```

Followed Path [1,1,1], [1,2,1,1]

5.6. Container (Access Logs from Data Model)

As explained in the Data Registration (Data Model) section, this container resource corresponds to the `interop:DataRegistration` definition. The files contained within it meet the specified definitions. For example, the file dated 2024-04-02 will be referred to as Access Log (Data Model), matching the Shape Expression (Data Model). Listing 8 presents the corresponding RDF fragment.

Listing 8. `ldp:Container` *[Thing] at http://proxy.localhost:4000/dpc/data/74242fac/AccessLog/*

```
<>
    a           ldp:Container, ldp:BasicContainer, ldp:Resource ;
    ldp:contains <2024-04-02> .
```

Followed Path [1,1,2], [1,2,2]

5.7. Access Log (Data Model)

The access log is a resource that contains the actual data and satisfies the shape as defined in the Shape Expression (Data Model). Listing 9 presents the corresponding RDF fragment.

Listing 9. `ldp:Container` *[Thing] at http://proxy.localhost:4000/dpc/data/74242fac/AccessLog/2024-04-02*

```
@prefix al:      <http://proxy.localhost:4000/dpc/ns/log#>.
```

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

<#1712073817394>
    a              al:AccessLog ;
    al:date        "2024-04-02T16:03:37.426Z"^^xsd:dateTime ;
    al:accessor    "http://proxy.localhost:4000/dpc/profile/card#me" ;
    al:application "Data Privacy Cockpit" ;
    al:action      "READ" ;
    al:resource    "/client/dpc" .
```

Followed Path [1,1,2], [1,2,2]

5.8. Claim Registry (Data Model)

The claim registry is an custom extension of the `interorp:DataRegistry` within the registry set. It refers to the root container for claimed data and the verification resource. Listing 10 presents the corresponding RDF fragment.

Listing 10. `claim:Registry` *[Thing] at http://proxy.localhost:4000/dpc/registries*

```
<#ab674650>
    a                           claim:Registry;
    claim:trustee               <http://proxy.localhost:4000/client/profile/card#me>;
    claim:monitoredStorage      <http://proxy.localhost:4000/client/>;
    claim:verificationResource  <http://proxy.localhost:4000/client/dpc#verification>;
    claim:verificationCode      "66097db6e9c3c234eb35f8ca66b5e4d829c6d...";
    claim:claimedData           <http://proxy.localhost:4000/dpc/data/74242fac/>.
```

Followed Path [1,2]

5.9. Container (Claimed Data from Data Model)

This resource contains all claimed data. When using the Solid Application Interoperability Specification, it primarily refers to data registrations and their corresponding containers. Listing 11 presents the corresponding RDF fragment.

Listing 11. `ldp:Container` *[Thing] at http://proxy.localhost:4000/dpc/data/74242fac/*

```
<>
    a           ldp:Container, ldp:BasicContainer, ldp:Resource ;
    ldp:contains <2024-04-02> .
```

Followed Path [1,2,1]

5.10. Verification (Data Model)

The verification resource shown in Figure 3 is the only resource stored by the client and will be used for comparison purposes. The verification code will be compared to the verification code of the claim registry. If they are equivalent, access to the claimed data will be granted. Listing 12 presents the corresponding RDF fragment.

Listing 12. `claim:Verification` *[Thing] at at http://proxy.localhost:4000/client/dpc*

```
<#verification>
    a                              <urn:claim#Verification> ;
    <urn:claim#verificationCode> "66097db6e9c3c234eb35f8ca66b5e4d829c6d..." .
```

Followed Path [1,2,2]

Each of the mentioned resources must have a corresponding ACL. The lists have been intentionally omitted for simplicity. The DPC agent requires read and write access to all of these resources. The only exception is the verification resource, which only needs to be read.

Appendix A: Full Logical Data Model

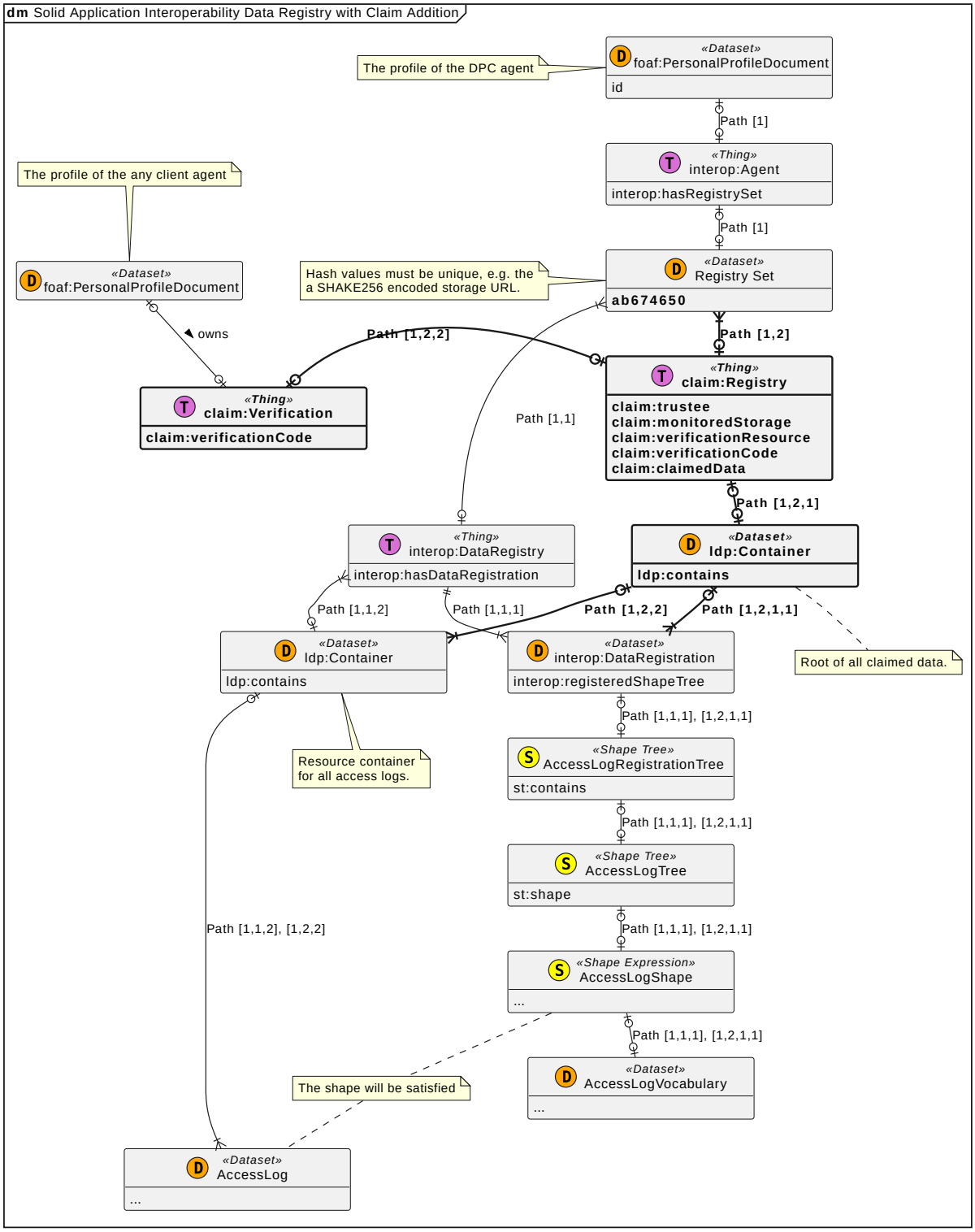

Figure 4. IE diagram of the full logical data model

Appendix B: Community Solid Server Configuration

Listing 13. Community Solid Server Configuration

```
{
  "@context": "https://linkedsoftwaredependencies.org/bundles/npm/@solid/community-
server/^7.0.0/components/context.jsonld",
  "import": [
    "css:config/app/init/default.json",
    "css:config/app/main/default.json",
    "css:config/app/variables/default.json",
    "css:config/http/handler/default.json",
    "css:config/http/middleware/default.json",
    "css:config/http/notifications/all.json",
    "css:config/http/server-factory/http.json",
    "css:config/http/static/default.json",
    "css:config/identity/access/public.json",
    "css:config/identity/email/default.json",
    "css:config/identity/handler/default.json",
    "css:config/identity/oidc/default.json",
    "css:config/identity/ownership/token.json",
    "css:config/identity/pod/static.json",
    "css:config/ldp/authentication/dpop-bearer.json",
    "css:config/ldp/authorization/webacl.json",
    "css:config/ldp/handler/default.json",
    "css:config/ldp/metadata-parser/default.json",
    "css:config/ldp/metadata-writer/default.json",
    "css:config/ldp/modes/default.json",
    "css:config/storage/backend/file.json",
    "css:config/storage/key-value/resource-store.json",
    "css:config/storage/location/pod.json",
    "css:config/storage/middleware/default.json",
    "css:config/util/auxiliary/acl.json",
    "css:config/util/identifiers/suffix.json",
    "css:config/util/index/default.json",
    "css:config/util/logging/winston.json",
    "css:config/util/representation-conversion/default.json",
    "css:config/util/resource-locker/file.json",
    "css:config/util/variables/default.json"
  ],
  "@graph": [
    {
      "comment": "The updated OIDC configuration.",
      "@type": "Override",
      "overrideInstance": {
        "@id": "urn:solid-server:default:IdentityProviderFactory"
      },
      "overrideParameters": {
        "@type": "IdentityProviderFactory",
        "config": {
          "claims": {
            "openid": [
              "azp"
```

```
      ],
      "webid": [
        "webid"
      ]
    },
    "clockTolerance": 120,
    "cookies": {
      "long": {
        "signed": true,
        "maxAge": 86400000
      },
      "short": {
        "signed": true
      }
    },
    "enabledJWA": {
      "dPoPSigningAlgValues": [
        "RS256",
        "RS384",
        "RS512",
        "PS256",
        "PS384",
        "PS512",
        "ES256",
        "ES256K",
        "ES384",
        "ES512",
        "EdDSA"
      ]
    },
    "features": {
      "claimsParameter": {
        "enabled": true
      },
      "clientCredentials": {
        "enabled": true
      },
      "devInteractions": {
        "enabled": false
      },
      "dPoP": {
        "enabled": true
      },
      "introspection": {
        "enabled": true
      },
      "registration": {
        "enabled": true
      },
      "revocation": {
        "enabled": true
      },
      "userinfo": {
        "enabled": false
```

```
            }
          },
          "scopes": [
            "openid",
            "profile",
            "offline_access",
            "webid"
          ],
          "subjectTypes": [
            "public"
          ],
          "ttl": {
            "AccessToken": 3600,
            "AuthorizationCode": 600,
            "BackchannelAuthenticationRequest": 600,
            "ClientCredentials": 172800000,
            "DeviceCode": 600,
            "Grant": 1209600,
            "IdToken": 3600,
            "Interaction": 3600,
            "RefreshToken": 86400,
            "Session": 1209600
          }
        }
      }
    },
    {
      "comment": "The new expiration time for inactive locks, in milliseconds.",
      "@type": "Override",
      "overrideInstance": {
        "@id": "urn:solid-server:default:ResourceLocker"
      },
      "overrideParameters": {
        "@type": "WrappedExpiringReadWriteLocker",
        "expiration": 172800000
      }
    }
  ]
}
```

Bibliography

Bingham, J., Prud'hommeaux, E., & Pavlik, elf. (2023). *Solid Application Interoperability*. https://solid.github.io/data-interoperability-panel/specification/

Colophon

Built with Asciidoctor PDF 2.3.17, Asciidoctor Bibtex 0.9.0 and Asciidoctor Diagram 2.3.1 on linux-musl.

Repository	https://github.com/guddii/SEACT/tree/main
Revision	https://github.com/guddii/SEACT/commit/ea5e94359349f1fed086a61c7cfe386089ee4658
Build	https://github.com/guddii/SEACT/actions/runs/9552829495