How to use the Data Type Registry?

RDA Datafabric - IG

Ulrich Schwardmann

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)

Am Fassberg, 37077 Göttingen
ulrich.schwardmann [at] gwdg.de

27 August 2020, video conference
Content

1. FAIR Digital Objects

2. Type Registration
   Life Demo

3. Profiles

4. Questions
How to Foster automation in the Data Domain?

- The amount of data needs **Automation** and
- the **Heterogeneity and Complexity of Data** are the mayor obstacles for automation

There are two classical approaches to ease automation and overcome Heterogeneity and Complexity

- **Abstraction**
  - this is the FDO concept

- **Standardization**
  - this should be covered by the Data Type Registry
  - a lightweight, domain specific and federated approach for standardization
Reusability by Abstraction

- needs knowledge about basic properties of data
  - **Metadata** is often unavailable, not connected to data or not interpretable

- Registration:
  
  **bind metadata and data with PID to a digital object**

- For reuse provide as much of this knowledge before access to the data

- **PID Information Types**
  - are additional metadata, stored in the PID database
  - similar to Mime Types, but much more flexible
  - Examples are *checksum*, *mime type*, *reference information*, *versioning* (relative and absolute), *embargo time*, *expiration date*, *add. metadata location*, *basic Dublin Core*, *access restrictions and methods*, *data and table column formats*, *collection description*, ...
Reusability by Abstraction

- needs knowledge about basic properties of data
  - **Metadata** is often unavailable, not connected to data or not interpretable

- Registration:
  - bind metadata and data with PID to a digital object

- For reuse provide as much of this knowledge before access to the data

- **PID Information Types**
  - are additional metadata, stored in the PID database
  - similar to Mime Types, but much more flexible
  - Examples are *checksum*, *mime type*, *reference information*, *versioning* (relative and absolute), *embargo time*, *expiration date*, *add. metadata location*, *basic Dublin Core*, *access restrictions and methods*, *data and table column formats*, *collection description*, ...
Interoperability by Standardization

- needs standardization and policies first of all on the metadata (types), but also on references and operations
- usually standardization is made for technical interoperability, but a long and difficult process
- policies are often simpler, but technically intangible, because part of an agreement between humans
- one needs simple machine readable definitions, made for specific domains, but reusable by others
  - linked data goes into this direction, but the end points are in general not machine readable URLs
- here so called **Data Type Registries (DTRs)** come into play
  - to provide lightweight standardization for metadata
  - and attach a PID to each such definition for disambiguation
The ePIC Data Type Registry

- Features
  - Definition of PID Information Types
  - Hierarchical types and automated schema extraction
  - Access via REST API, Browser

- Based on CORDRA software

- GWDG is provider on behalf of ePIC

- Who can use the service?
  - Public, authorization needed only for type definition

- Overview: http://dtr.pidconsortium.eu/

Policies for a PID InfoType life cycle:

- in preparation (21.T11148),
  - http://dtr-test.pidconsortium.eu/
- candidate, approved, deprecated (21.11104)
  - http://dtr-pit.pidconsortium.eu/
Hierarchies in Metadata

Example: geographic coordinate.

```
21.T11148/ada87a981a168d9a4ccf
Type Name: geographic-coordinate
Type: PID-InfoType
```

```
21.T11148/d2a773ae817d7d07c19d
Type Name: longitude
Type: PID-InfoType
```

```
21.T11148/f71615c9c8ee58437e3
Type Name: altitude
Type: PID-BasicInfoType
```

```
21.T11148/5fccecdf1d079c4a85c9
Type Name: latitude
Type: PID-InfoType
```
Hierarchies in Metadata

Example: geographic coordinate.
Hierarchical Type Definitions

- types are often dependent from each other, how exactly?
- to exactly describe objects by data types one needs:
  - a distinction between derived objects and basic objects
    - concept of basic info types and info types
  - a more exact description of the type dependencies
  - additionally a schema inspired dependency model
  - grounding by exact definition of basic types (regexp, ...)
  
- in consequence:
  - possibility to derive JSON/XML schemas for the types
    - automated server side schema derivation at ePIC DTR
  - one type defines in an exact way its whole dependencies
    - inside objects of a certain type one can use new names for its parts (instead of type identifiers and type names)
  
- see also Schwardmann, U.: Automated schema extraction for PID information types
  - PID: http://hdl.handle.net/21.11101/0000-0002-A987-7
Special Technical Registration Issues

- **Abbreviated Form**: the need for keys that are just numbers, strings, or arrays
  - since all types have a name usually the values of types would be objects with that name as parameter
  - *Abbreviated Form* equals True allows unnamed values

- **Omit Names of Subtypes**: avoid over-specified objects
  - all subtypes come with a name in their own definition and a name as subtype in the type they are used in
  - in some cases the values should specify both of these names in nested objects, in others this would be an over-specification

  - Example: many types reuse "unicode-string" for certain fields like name, but one would not like to have a name specification like
    ```json
    { "name" : { "unicode-string" : "Ulrich" } }
    ```
Types vs. Linked Data

- An Example of a type: `isPreviousVersionOf`
  - Such a type is stored as key-value pair in the PID (`pid-do1`) of a digital object
  - as key-value pair consisting of the `type` and the PID of the previous version (`pid-do2`)

This gives a triple:

- `pid-do1 type pid-do2`
- Digital-Object-1 `isPreviousVersionOf` Digital-Object-2

Thus one has a relation:

```
subject predicate object
```

with types as predicates.

- Types can be represented by PIDs again (DTR), so the SPO becomes a PID triple
- A converter of FAIR-DOs with registered types into JSON-LD representation exists.

was shown in the contexts of PID for Instruments
Kernel Information Type Profile

<table>
<thead>
<tr>
<th>Property identifier</th>
<th>Content format</th>
<th>Cardinality</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  PID</td>
<td>Handle</td>
<td>1..n</td>
<td>Global identifier for the object; external to the PID Kernel Information</td>
</tr>
<tr>
<td>2  KernelInformationProfile</td>
<td>Handle</td>
<td>1</td>
<td>Handle to the Kernel Information type profile; serves as pointer to profile in DTR. Address of DTR federation expected to be global (common) knowledge.</td>
</tr>
<tr>
<td>3  digitalObjectType</td>
<td>Handle</td>
<td>1</td>
<td>Handle points to type definition in DTR for this type of object. Distinguishing metadata from data objects is a client decision within a particular usage context, which may to some extent rely on the digitalObjectType value provided.</td>
</tr>
<tr>
<td>4  digitalObjectLocation</td>
<td>URL</td>
<td>1..n</td>
<td>Pointer to the content object location (pointer to the DO). This may be in addition to a pointer to a human-readable landing page for the object.</td>
</tr>
<tr>
<td>5  digitalObjectPolicy</td>
<td>Handle</td>
<td>1</td>
<td>Pointer to a policy object which specifies a model for managing changes to the object or its Kernel Information record, including particularly object access and modification policies. A caller should be able to determine the expected future changes to the object from the policy, which are based on managed processes the object owner maintains.</td>
</tr>
<tr>
<td>6  etag</td>
<td>Hexstring</td>
<td>1</td>
<td>Checksum of object contents. Checksum format determined via attribute type referenced in a Kernel Information record.</td>
</tr>
<tr>
<td>7  dateModified</td>
<td>ISO 8601 Date</td>
<td>0..1</td>
<td>Last date/time of object modification. Mandatory if applicable.</td>
</tr>
<tr>
<td>8  dateCreated</td>
<td>ISO 8601 Date</td>
<td>1</td>
<td>Date/time of object creation</td>
</tr>
<tr>
<td>9  version</td>
<td>String</td>
<td>0..1</td>
<td>If tracked, a version for the object, which must follow a total order. Mandatory for all objects with at least one predecessor version.</td>
</tr>
<tr>
<td>10 wasDerivedFrom</td>
<td>Handle</td>
<td>0..n</td>
<td>PROV-DM: an update or construction of an entity.</td>
</tr>
<tr>
<td>11 specializationOf</td>
<td>Handle</td>
<td>0..n</td>
<td>PROV-DM: shares all as it presents more information than the latter.</td>
</tr>
<tr>
<td>12 wasRevisionOf</td>
<td>Handle</td>
<td>0..n</td>
<td>PROV-DM: entity is a revision of the latter.</td>
</tr>
<tr>
<td>13 hadPrimarySource</td>
<td>Handle</td>
<td>0..n</td>
<td>PROV-DM: something produced by an entity that experiences a state at a specific time of the thing.</td>
</tr>
<tr>
<td>14 wasQuotedFrom</td>
<td>Handle</td>
<td>0..n</td>
<td>PROV-DM: an entity, such as a person or thing.</td>
</tr>
</tbody>
</table>
| 15 alternateOf       | Handle         | 0..n        | PROV-DM: some entity that serves or is served by the other thing.
RDA KernelInformationType Profile at ePIC

ePIC Specific Consequences on the Typing:

- In order to express a Kernel Information Profile as an InfoType in the ePIC DTR we need a different **DTR schema**
  - **KernelInformationProfile**: 21.T11148/532ce6796e2828dd2be6
  - A Kernel Information Profile is then an instance of the DTR schema KernelInformationProfile
    - An example instance of such a **DTR type** is **recommendedKernelInformationProfile**: 21.T11148/0c5636e4d82b88f86132

- A PID, fulfilling a concrete Kernel Information profile, has to have the properties, as described in this profile.
ePIC KernelInformationProfile

identifier: "21.T11148/0c5636e4d82b88f86132"
name: "recommendedKernelInformationProfile"
description: "Recommended Kernel Information profile, describing which attributes must or may be included in a conforming default Kernel Information record. (context: KernelInformation)"

properties:
0:
  name: "KernelInformationProfile"
  identifier: "21.T11148/076759916209e5d62bd5"

1:
  name: "digitalObjectType"
  identifier: "21.T11148/1c699a5d1b4ad3ba4956"

2:
  name: "digitalObjectLocation"
  identifier: "21.T11148/b84578129056b83046284"

3:
  name: "digitalObjectPolicy"
Many Thanks

Questions ???

Contact at ePIC:

- support [at] pidconsortium.eu

Contact at GWDG:

- **Ulrich Schwardmann**
  T: 0551 201-1542, E: ulrich.schwardmann [at] gwdg.de

Further reading: Schwardmann, Ulrich, 2020, ”The ePIC PID Information Type Registry”, DOI: 10.25625/9DNRSJ