Use Cases that may require matrix parameters, and rebuttals.

By Markus (reviewed by Joe)

This document continues a discussion which was held during the "Matrix Parameters & Query Parameters" session at the face-to-face meeting of the W3C DID Working Group in Amsterdam in late January 2020. The assignment following the meeting was:

1. To present a “killer use case” for matrix parameters in DID URLs.
2. To explore if/how that use case can be implemented without matrix parameters.

If you are not familiar with the topic, please review the session slides and minutes first. In those materials, this set of use cases was also referred to as the "Data Hierarchy Portability" use case.

For more information, also see the "Why matrix parameters?" RWoT#10 paper.

Web Address Portability Use Cases Summary

A number of use cases that are hereby collectively referred to as "Web Address Portability" use cases all have to do with the need to have a persistent address for arbitrary web resources (website, image file, document, blog post, etc.). This address must remain stable even if the underlying network location of the resource changes (e.g. a file is moved from one server to another).

Concrete example use cases:

1. I want to print a DID URL on my business card (or add it to my email signature, etc.), which should always point to my current profile picture. I want to update the network location of that profile picture without having to re-print my business cards.
   E.g. my DID URL should today point to this network location:
   https://socialnetwork.com/user/123/profile?image
   And in a week from now it should point to this network location:
   https://mywebsite.nl/me/photo.jpg

2. I have a Nextcloud instance hosted by a third party hosting provider, and I want to be able to link to files on my Nextcloud using web addresses that don't break if I eventually decide to self-host my Nextcloud. I want those links to automatically work for the entire hierarchy of resources at the network location.
   E.g. it is today hosted at this network location:
   https://nextcloudsuperhost.com/files/*
   And tomorrow I want to self-host it at this location:
   https://216.3.128.12:8000/files/*

Note: Use case 2 above not only requires a stable address for a single web resource, but for an entire information space underneath the DID URL, including all standard URI syntax components (path, query, fragment).

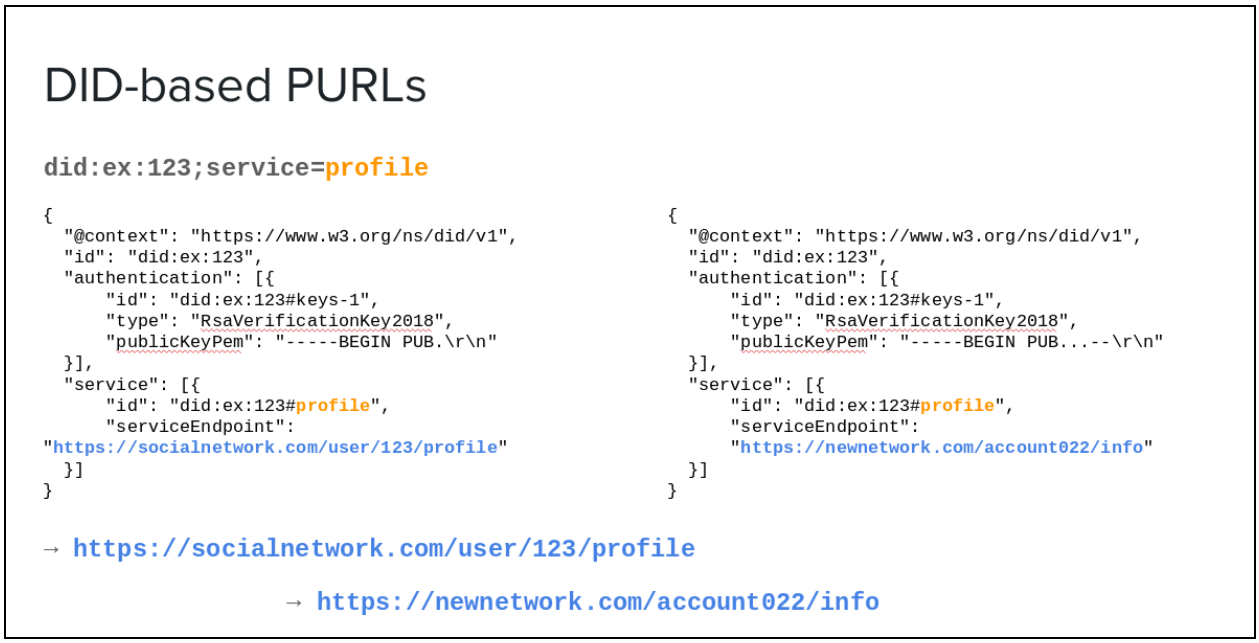
Note: Such DID URLs may become "brittle", if the information hierarchies at different network locations are not strictly equivalent.

(Many similar use cases can be constructed, e.g. in example 1 use a personal résumé document instead of a profile picture, or in example 2 use a blog or a plain website instead of a Nextcloud instance).

This functionality combines two concepts that are well-known in web architecture: Web Linking and PURLs, essentially building an indirection layer for the web that offers all the advantages of DIDs (persistence, cryptographic verifiability, decentralization).

Implementing these use cases with matrix parameters

The following examples show how the above "Web Address Portability" use cases can be implemented with matrix parameters.



Basically, the "service" matrix parameter is used to select a service endpoint from the DID document, and the remaining DID URL (including path, query, fragment) is appended to the service endpoint URL (re-using well-defined "relative URI" functionality).

Implementing these use cases without matrix parameters

The same use cases can potentially be implemented without matrix parameters, using "special" query parameters instead, as shown in the following example:

Web Address Portability without matrix parameters?

- With matrix parameter:

  `did:ex:123;service=files/myresume/doc?version=latest#intro`

- With query parameter:

  `did:ex:123/myresume/doc?_did_service=files&version=latest#intro`

- Result:

  `https://selfhosted.me:8080/myresume/doc?version=latest#intro`

Web Address Portability without matrix parameters?

- With matrix parameter:

  `did:ex:123;service=files/myresume/doc?compact#intro`

- With query parameter:

  `????`

- Result:

  `https://selfhosted.me:8080/myresume/doc?compact#intro`

This approach has several downsides:

1. The order in which different DID URL syntax components have to be processed is not obvious: To dereference the DID URL, the "special" query parameter _did_service would have to be processed first, then the path component plus the remaining query string

(with the "special" parameters removed) would have to be appended to the service endpoint URL.
2. Therefore, standard "relative URI" processing (see https://tools.ietf.org/html/rfc3986#section-5.2) cannot be used; instead a custom algorithm has to be used for processing the URI components (path, query, fragment).
3. The DID Core specification would "step on" the standard URI syntax components (path, query, fragment), which other URI schemes (e.g. HTTP URLs) leave completely open for use by developers and applications however they see fit.
4. This approach would essentially require DID URL query strings to always follow the "name=value" pattern, whereas standard URIs (see RFC3986) do not have this constraint.

Use as "Ersatz-DNS"[1]

In order to remove a dependency on another weak link in Internet architecture, services in the DID documents can be designed to use only IP addresses instead of domain names[2]. Since DID documents and service selection already provide a cryptographically verifiable indirection mechanism, DNS as a less secure indirection layer becomes unnecessary.

Other Matrix Parameters

This document only explores one matrix parameter and its use. Several more have been proposed (and accepted by the W3C Credentials Community Group). For example, matrix parameters can be used to identify a specific version of a DID document (as opposed to the most recent one)[3], or to supply hints for being able to resolve a DID to its DID document correctly[4].

In some situations, the same functionality is applicable to both the DID document AND the resource identified by the DID URL (e.g. version of the DID document vs. version of the resource, hash of the DID document vs. hash of the resource). In general, it is very useful to have two parameter passing mechanisms that cleanly address different parts of the DID URL dereferencing process.

---

[1] Credits to Sam Smith for coming up with this term.
[2] Tim Berners-Lee has called DNS the "Achilles heel" of the Web.
[3] See the "version-id" and "version-time" parameters.
[4] See the "initial-values" parameter: https://github.com/decentralized-identity/sidetree/blob/master/docs/protocol.md#unpublished-did-resolution

To summarize, matrix parameters are used for influencing the DID resolution process, while other DID URL syntax components (path, query, fragment) have exactly the same meaning as in standard URIs (see RFC3986)[5].

A Reconsideration

From Joe Andrieu
When we started this exercise, it was my understanding that what was unique about the portable hierarchy use case was the challenge of parameter aggregation, specifically how are resolvers supposed to handle situations where the DID URL and the selected service endpoint have query and/or path parts.

The problem is that all of the algorithms that have been described for this have fundamental problems which make DID URLs particularly fragile if aggregation is supported by default.

By fragile, I mean that the entire point of DID Documents is to allow the controller the flexibility to change the material related to a DID *without* breaking any of the existing DID-URLs extant in the universe. In particular, if one has a DID-URL like
did:example:joe;service=avatar
And a service endpoint like

```
{
 "id": "did:example:joe#avatar",
  "type": "image",
  "serviceEndpoint": "https://pic.example.com/joe/image.png"
}
```

Then the the DID URL would remain a valid pointer to an image if the service endpoint is updated to

```
{
 "id": "did:example:joe#avatar",
  "type": "image",
  "serviceEndpoint": "https://example.com/joe/images?name=image.png"
}
```

In both cases, the service endpoint points to a PNG and the original DID-URL works great.

However, if a user were to set up their system in a slight different way, starting with a DID-URL like

[5] It is interesting to note that URN resolution (see https://tools.ietf.org/html/rfc8141#page-12) also uses a dedicated syntax component ("r-component") for passing parameters to a resolution process, and a different syntax component for standard query parameters.

did:example:joe;service=avatar:joe/image.png

And a service endpoint like

```
{
 "id": "did:example:joe#avatar",
  "type": "image",
  "serviceEndpoint": "https://pic.example.com/"
}
```

This works the same as the first example, because the aggregation rules are aligned with the service endpoint.

However, the the DID URL would break if the service endpoint is updated to point to a new service with a query-based lookup of the file name.

```
{
  "id": "did:example:joe#avatar",
   "type": "image",
   "serviceEndpoint": "https://example.com/joe?file="
}
```

I have not found ANY aggregation rules (nor variations on that service endpoint) which could salvage this situation. It is, I believe an intractable problem. Which is why web redirection does not guarantee that path and query parts be retained after a redirect.

If we are to allow path and query terms in a service endpoint AND in the DID-URL, we WILL encourage people to accidentally create extremely fragile URLs. And, IMO, just like XHTML allowed developers to create broken html files--and then brutally failed when those files weren't perfect--we will be enabling millions of developers to fail with neither clear guidance nor graceful recovery. I expect many will decide these crazy DID URLs are just too complicated, just like XHTML was abandoned.

My opinion has long been that either service endpoints OR DID-URLs needed to NOT have path and query parts, and since the service endpoints are generally NOT under control of the DID Controller, but the DID-URLs constructed based on those endpoints generally are, then path and query terms in the DID-URL should be ignored. Specify a service but not with extra parts. But the creators of this approach weren't having it.

After many hours of discussion with Drummond and Markus, I understood that the reason ignoring the DID-URL query & path parameters is problematic is because of the use case described above: the desirability to have a complete hierarchy of resources that can be relocated by simply updating the DID Document. In those cases, retaining the path part--at a minimum--would be highly desirable.

THAT is a good use case.

But it is not an argument for matrix parameters.

For two reasons.

First, the aggregation pattern can readily be specified in the service property, by adding an “aggregationType” property so that any given service endpoint can be typed to handle aggregation appropriately. There is no need to put this in a matrix parameter (in case anyone thought that might be a useful thing). This extra redundancy gives, IMO, the appropriate flexibility to different kinds of service endpoints. For those that WANT that fragile yet powerful technique of magically aggregating path & query parts from the DID-URL with the service endpoint, that can be explicitly requested. Everyone else, IMO, should probably get a default behavior that ignores the path & query part on redirect, just like the normal behavior of the web.

Second, what is presumed in the previous discussion is that there are *multiple* service endpoints and that one must somehow choose between them.

There is no concrete use case explaining why this is required for DIDs to function.

If we restrict the number of service endpoints to zero or one, and guarantee that dereferencing ALWAYS returns a redirect to the service endpoint if present, then we can still support hierarchical portability EVEN without a matrix parameter specifying which service to select--because there is only one.

I have argued elsewhere that both service endpoints and verification methods are privacy disasters waiting to happen. To the extend that these values are exposed publicly, they will be indexed, catalogues, and analyzed to the likely detriment of everyone who so foolishly exposed all that information in their DID Document. This is even worse for those DID methods that store DID Documents on chain. It is particularly problematic that even network information as stripped bare as an IP address can be regarded as PII in the GDPR framework. Some will argue that “it depends” but part of what it depends on is how correlatable that IP address is to a specific individual. This suggests to me that it is imperative to separate such correlative exposure as much as possible and remove everything from the DID Document that isn’t absolutely required.

In particular, having multiple service endpoints explicitly associates a DID across multiple services, creating a completely unnecessary point of correlation. Rather than encouraging people to have a single DID, at which all sorts of sundry service endpoints are aggregated, use a different DID for each service.

did:example:123 is for my avatar
did:example:abc is for my instant messenger

The practice of using a single DID for both of these wildly different services, is directly linking information that has no need to be associated and leaking correlation. It will encourage individuals to use a single DID to refer to themselves generally, which will turn DIDs into PII and make any such DIDs stored on an immutable ledger a GDPR violation.

If instead, we use DIDs freely without baked in correlation of services, then it is extremely hard to argue that a given DID is PII. The resource on the other end of the service endpoint might have PII, but the DID itself may not even refer to a person.

To my analysis, even a single verification method or endpoint is both unnecessary--both can be found by using the DID to look up this information in a public directory, making that directory the appropriate management point for any PII that it might present.

Since we can address the use case that triggered this exercise with a single service endpoint--and since we have no compelling use cases for why multiple service endpoints are appropriate, we still don't have a compelling use case for matrix parameters. On the contrary, we've potentially identified an extremely problematic privacy problem with the general approach of putting a number of convenience properties in what, in its essential form, need only present the *current* cryptographic material for secure interactions with the DID Subject.

So, yes, I would support removing service endpoints and verification methods from the DID Document. Although I supported those in the past, it was because I had not fully considered the privacy implications.

However, even if we choose to support the use case of a portable hierarchy, we can do that with just a single service endpoint and without matrix parameters.