

Pension Dashboards Programme

API Baseline Design

Document Version History

Version	Date	Author	Comment
0.1	21/06/2021	Nick Georgallides	Initial draft with only Find API
0.2	20/07/2021	Nick Georgallides	Second draft detailing the rest of the APIs
0.3	31/08/2021	Nick Georgallides	Third draft post MP comments
0.4	20/10/2021	Nick Georgallides	Includes authorise API, obtain PeIs API and obtain PeI configuration API
0.5	17/11/2021	Nick Georgallides	Updates post MP comments and feedback
0.6	25/11/2021	Nick Georgallides	Updates post MP comments and feedback
0.7	04/02/2022	Nick Georgallides	Added sequence diagrams
0.8	05/05/2022	Nick Georgallides	Introspection payload added
0.9	25/05/2022	Nick Georgallides	Holdername GUID added in PeI examples
0.10	14/06/2022	Nick Georgallides	View response updated to include "view_data" JWT claim, technical standards added into section 12
0.11	13/07/2022	Nick Georgallides	"kid" parameter added to JWT header – section 12.2 Behaviour of re-registration of an existing resource at C&A- section 4.6.1

1 Introduction

1.1 Purpose

This document outlines the API baseline design for PDP specific APIs.

1.2 Scope

The following areas will be covered in the document:

- A summary of each API
- Hosting
- Format
- HTTP method
- Authorization
- Expected responses
- Error handling

1.3 Transaction monitoring

All interfaces will carry a unique transaction identifier GUID (`request_id`) for logging, audit and monitoring purposes, the detail of which will be published in future documentation.

The transaction identifier is issued by the party which initiates the transaction.

For transactions which PDP initiates, PDP will generate the identifier; for transactions which the data/dashboard provider initiates, the provider will generate the identifier.

2 Find API

2.1 Summary of the Find API

The find API is exposed by data providers and its purpose is to receive the find requests which are initiated by the Pension Finder Service (PFS) containing the pension owner's PII data needed for a data provider to use in order to determine a match within the internal records. The find input data will also carry the relevant consents given by the pension owner during the find process and the user account token needed for obtaining the PAT (protected API token) needed for the data providers to use the UMA specific protection API to register PeIs upon a successful find with the consent and authorization service.

2.2 Pension Finder Service (PFS)

The pension finder service is orchestration middleware, it distributes the find request across the data provider endpoints by invoking their find APIs and manages the low-level interactions to achieve message delivery to the data providers.

2.3 Hosting

Each data provider connected to the ecosystem will be required to host their find API within their domain.

2.4 Format

The find API will be a REST API using JSON encoded as UTF-8.

2.5 Authorization

This is a closed ecosystem with all end point connections secured using private PKI certificates issued by the PDP Governance Register to suitably enrolled organisations. This enables connecting entities to establish a mutual TLS connection with the central

API Baseline Design

infrastructure. For find requests only the PFS will be responsible for invoking the find API endpoints. There is an assumption made by the Pension Dashboard Programme that there are no additional API security requirements for the find API.

2.6 HTTP Method

The PFS will be restricted to only make HTTP POST requests to each data provider find endpoint with the body of the request containing the find parameters as a signed JWTs.

Summary of the Find Request data parameters sent to data provider find endpoint:

user_token	A combination of the following expressed as a JWT: <ul style="list-style-type: none">• Verified identity details such as name, date of birth and postcode. These will have been selected by the C&A service from the verified details supplied by the Identity Service.• Identity details asserted by the user at the C&A's Consent user interface e.g., National Insurance Number.
user_account_token	The User Account Token is an OAuth2 authorization grant, expressed as a JWT (JSON web token) which can be exchanged for the PAT
consents_token	A set of user consents for subsequent processing using the supplied identity details expressed as a JWT
request_id	Transaction identifier GUID used for monitoring and reporting purposes

For example, the PFS makes the following HTTP POST request using mTLS:

```
POST /find HTTP/1.1
Host: www.dashboard.aviva.com
Content-Type: application/json;charset=UTF-8
{
  "user_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
  "user_account_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
  "consents_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
  "request_id": "040dbfd1-40db-4caa-96ea-bf3c7d230877"
}
```

2.7 Response

If the find request sent from the PFS succeeds, the data provider (UMA Resource Server) will respond with a *202 Accepted HTTP* status implying acknowledgment of the find request. Following this the PFS will not be involved in any request back from the data provider.

HTTP/1.1 202 Accepted

If a match is found then the data provider will issue a PeI in the agreed format alongside it's description and register this directly to the consent & authorization service via the UMA protection API. If no match is found then there is no further action required by the data provider other than the option to hash failed find requests in order to avoid repeating them. This optional requirement is discussed in more detail in the data provider technical guide.

2.8 Error Handling

If the find request sent from the PFS fails then data provider is expected to respond with the appropriate HTTP status code and error message. The body of the response for an error must contain an error code which correlates to a message so that the test harnesses and logs can be automated

4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a POST request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Status	Message	Note
400	Bad Request	Potential bad requests examples: <ul style="list-style-type: none"> • PFS is not sending a HTTP POST • Parsing error by data provider • Schema not configured correctly

5xx

The 5xx (Server Error) class of status code indicates that an exception occurred during the elaboration of a request. An indication about the nature of the error SHOULD be included together with an indication if the error is temporary or permanent.

Status	Message	Note
500	Internal Server Error	
503	Service Unavailable	This status is temporary used when the service is down for maintenance or is overloaded by requests

3 Obtain PAT API

3.1 Summary of the Obtain PAT API

During Find the PFS includes a valid User Account Token issued by the authorization server (part of the C&A) in the Find request sent out to the data provider find interface endpoints. The User Account Token is a OAuth2 authorization grant, expressed as a JWT (JSON web token) which can be exchanged for the PAT, which is an OAuth2 access token, as per the OAuth2 standard by the presentation of this token to the authorization server's token endpoint. Data providers will be a client of the authorization server.

The obtain PAT API (i.e. the authorization server's OAuth2 token endpoint) will be exposed by the authorization server and will follow a standard implementation of <https://datatracker.ietf.org/doc/html/rfc6749#section-4.5> using

urn:ietf:params:oauth:grant-type:jwt-bearer in accord with <https://datatracker.ietf.org/doc/html/rfc7523#section-2.1>

3.2 Hosting

The API will be hosted on the authorization server (AS) as the request is made is to the authorization server's token endpoint.

3.3 Format

The obtain PAT API will be a REST API with character encoding of UTF-8 in the HTTP request entity-body.

3.4 Authorization

The data provider will be a client of the authorization server and will need to register it's software with it before a request can be made. This will ensure the communication channel is encrypted and will establish dynamic trust between both parties.

3.5 HTTP Method

The data provider MUST use the HTTP "POST" method when making access token requests to the authorization server's token endpoint, by sending the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type

REQUIRED. Value MUST be set to " urn:ietf:params:oauth:grant-type:jwt-bearer".

assertion

REQUIRED. MUST contain a single JWT

scope

REQUIRED. Specifies the scope of the access request. MUST be uma_protection

For example, the data provider makes the following HTTP POST request using mTLS:

```
POST /token HTTP/1.1
```

```
Host: as.pdp.com
```

```
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
```

```
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3jwt-bearer&
assertion=PHNhbWxwOl...[omitted for brevity]...ZT4&scope= uma_protection
```

3.6 Response

If the request for an access token is valid, the authorization server generates an access token (PAT) as a structured JWT bound (in accordance with [Section 3 RFC8705](#)) to the resource server (data provider certificate)

For example, a successful token response may look like the following with all the properties of the PAT token encoded in the payload:

```
HTTP/1.1 200 OK
```

Content-Type: application/json;charset=UTF-8

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
  "token_type": "pension_dashboard_pat"
}
```

3.7 Error Handling

If the access token request is invalid, such as the redirect URL didn't match the one used during authorization, then the server will return an error response. The error handling will follow standard OAuth 2 failure codes as per [RFC6749](#).

Error responses are returned with an HTTP 400 status code (unless specified otherwise), with error and error_description parameters. The error parameter will always be one of the values listed below.

- `invalid_request` – The request is missing a parameter so the server can't proceed with the request. This may also be returned if the request includes an unsupported parameter or repeats a parameter.
- `invalid_grant` – The authorization code (or user's password for the password grant type) is invalid or expired. This is also the error you would return if the redirect URL given in the authorization grant does not match the URL provided in this access token request.
- `invalid_scope` – For access token requests that include a scope (password or client_credentials grants), this error indicates an invalid scope value in the request.
- `unauthorised_client` – This client is not authorized to use the requested grant type. For example, if you restrict which applications can use the Implicit grant, you would return this error for the other apps.
- `unsupported_grant_type` – If a grant type is requested that the authorization server doesn't recognize, use this code. Note that unknown grant types also use this specific error code rather than using the `invalid_request` above.

The entire error response is returned as a JSON string, similar to the successful response. Below is an example of an error response.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "error": "invalid_request",
}
```

4 Register Pel API

4.1 Summary of the Register Pel API

When the data provider (UMA resource server) has determined a match within their internal records and obtained the PAT token to enable access to the UMA protection API hosted in the authorization server then the data provider will be able to access the API to register PeIs (resources) and place them under protection of an authorization control on

API Baseline Design

behalf of the pension owner (resource owner) and manage them over time. Protection of a resource at the authorization server begins on successful registration and ends on successful deregistration.

The authorization server MUST support the following five registration options and MUST require a valid PAT for access to them; any other operations are undefined by this specification. Here, rreguri stands for the resource registration endpoint and _id stands for the authorization server-assigned identifier for the web resource, returned by the authorization server when the create resource operation was performed, corresponding to the resource at the time it was created, included within the URL returned in the Location header. Each operation is defined in its own section below.

Create resource description: POST rreguri/
Read resource description: GET rreguri/_id
Update resource description: PUT rreguri/_id
Delete resource description: DELETE rreguri/_id
List resource descriptions: GET rreguri/

The resource server must persist the following, for each Resource Owner following a create resource operation:

- Resource _id – index of the registered resource (PeI)
- Resource owner's PAT – access token to API
- Authorization Server 'AS URI' which issued the PAT (at which the resource _id is registered) - address of the authorization server token endpoint

The resource server should also persist these items in a manner which it can locate them using the inbound URL of the view request.

4.2 Hosting

The API will be hosted on the authorization server which is part of the C&A service. The resource server will use this API at the authorization server's resource registration endpoint to create, read, update, and delete resource descriptions, along with retrieving lists of such descriptions. The descriptions consist of JSON documents that are maintained as web resources at the authorization server. The authorization server should declare this endpoint in the discovery document so that the resource server knows the endpoint.

4.3 Format

The API will be a REST API using JSON encoded as UTF-8.

4.4 Authorization

The data provider will need to use the relevant PAT specific to the pension owner to authorise it's use of the API when making a request to the authorization server to create, read, update or delete a resource (PeI).

4.5 HTTP Method

This will be depending on the type of request i.e. create, read, update, delete, list. These is covered in the relevant sections below.

4.6 Resource Description

A resource description is a JSON document that describes the characteristics of a resource sufficiently for an authorization server to protect it. A resource description will have the following parameters:

Resource scopes – required

These will be ["value", "owner", "delegate"]. All three scopes must be used for all registrations so that the resource owner can subsequently delegate access if required without further resource server activity.

Name – required

This is the URN of the resource, i.e. of the pension asset at the resource server. It will be used as the unique name against which the authorization server will apply authorization protection and is the representation of the pension asset as is available to a Pension Dashboard client. It is a URN of the form: 'urn:pei:<holder-name GUID>':<asset GUID>'

Type – required

This is a URI of the type of all 'name' parameters used in the Pension Dashboard ecosystem. It is required for future extension, e.g. to support resources which have wider scope options than defined here, or for specialised RS-AS relationships in the future.

Description – required.

Data standards for the PD ecosystem will define 'type' and 'name' mandated here. It may be that human readable derivations of 'name' will be aided by this 'description'. Format of this field TBC.

4.6.1 Create Resource Description

The resource server must use the HTTP "POST" method when registering the resource with the authorization server. The resource server must register each pension asset as a separate resource (to enable delegation and access at the most granular level). The request must contain the required parameters.

Example of a resource registration request message with a PAT in the header:

```
POST /rreg/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IjEwMDEyMzQ1NjY3ODk5IiwiaWF0IjoiMTYxMjM0NTY3ODk5InQ.SfIKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c

...
{
  "resource_scopes":["value", "owner", "delegate"],
  "name": "urn:pei:0e55140a-87d3-41cf-b6f7-bc822a4c3c3b:6e29eeb8-814c-
44a6-a43f-b4830f3f4590",
  "type": "http://pdp.gov/uma/PEI",
  "description" : "Aviva Pension"
}
```

If the request is successful, the resource is thereby registered and the authorization server MUST respond with an HTTP 201 status message that includes a Location header and an resource_id parameter. The resource_id parameter is issued by the authorization server for each registered resource, i.e. as a result of each UMA registration of a PeI, initiated by the resource server, authorised by the PAT. The resource_id is the common index between authorization server and resource server associated with each PeI.

Example resource registration response message:

API Baseline Design

Example of a update request adding a description parameter to a resource description that previously had none, with a PAT in the header:

```
PUT /rreg/9UQU-DUWW HTTP/1.1
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
...
{
  "resource_scopes":["value", "owner", "delegate"],
  "name": "urn:pei:0e55140a-87d3-41cf-b6f7-bc822a4c3c3b:6e29eeb8-814c-
44a6-a43f-b4830f3f4590",
  "type": "http://pdp.gov/uma/PEI",
  "description" : "Aviva Pension"
}
```

Form of a successful response, not containing the optional user_access_policy_uri parameter:

```
HTTP/1.1 200 OK
...
{
  "resource_id":"9UQU-DUWW"
}
```

4.6.4 Delete Resource Description

The resource server must use the HTTP Delete method to delete a previously registered resource description. If the request is successful, the resource is thereby deregistered and the authorization server MUST respond with an HTTP 200 or 204 status message.

Form of a delete request, with a PAT in the header:

```
DELETE /rreg/9UQU-DUWW
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
...
```

Form of a successful response:

```
HTTP/1.1 204 No content
...
```

4.6.5 List Resource Description

The resource server must use the HTTP GET method in order to get a list of all previously registered resource identifiers for this resource owner. The authorization server MUST return the list in the form of a JSON array of resource_id string values.

The resource server can use this method as a first step in checking whether its understanding of protected resources is in full synchronization with the authorization server's understanding.

Form of a list request, with a PAT in the header:

```
GET /rreg/ HTTP/1.1
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
```

...

Form of a successful response:

```
HTTP/1.1 200 OK
```

...

```
[ "KX3A-39WE",
  "9UQU-DUWW"
]
```

4.7 Error Handling

If the request fails because the resource server does not have a valid access token (i.e. PAT is missing or has expired) then the authorization server responds with an HTTP 401 status code as the request cannot be authenticated.

If a request is successfully authenticated, but is invalid for another reason, the authorization server produces an error response by supplying a JSON-encoded object with the following members in the body of the HTTP response:

error – REQUIRED except as noted. A single error code. Values for this parameter are defined throughout this specification.

error_description – OPTIONAL. Human-readable text providing additional information.

error_uri – OPTIONAL. A URI identifying a human-readable web page with information about the error.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
```

...

```
{
  "error": "invalid_resource_id",
  "error_description": "Permission request failed with bad resource ID.",
  "error_uri": "errors to be defined later in design"
}
```

If the request to the resource registration endpoint is incorrect, then the authorization server instead responds as follows:

If the referenced resource cannot be found, the authorization server MUST respond with an HTTP 404 (Not Found) status code and MAY respond with a not_found error code.

If the resource server request used an unsupported HTTP method, the authorization server MUST respond with the HTTP 405 (Method Not Allowed) status code and MAY respond with an unsupported_method_type error code.

If the request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed, the authorization server MUST respond with the HTTP 400 (Bad Request) status code and MAY respond with an invalid_request error code.

5 View API

5.1 Summary of View API

Enables a dashboard (client) to retrieve pension details on behalf of a requesting party (i.e pension owner or delegate) by dereferencing the PeI which resolves to a URL and making a HTTP GET request to access the pension details. It is this URL which is an UMA protected resource and if the request is authorised via the UMA protocol then the data provider will respond back to the dashboard with the pension details encoded within the data payload as a JWT.

5.2 Hosting

Each data provider connected to the ecosystem will be required to host their view API within their domain.

5.3 Format

The view API will be a REST API using JSON encoded as UTF-8.

5.4 Authorization

The dashboard client will authenticate itself with the authorization server at run time as defined in [rfc8705 section 2](#) . In addition to this, all end point connections are secured using mutual TLS.

5.5 HTTP Method

The dashboard will make a HTTP GET request to the data provider's view endpoint by dereferencing the PeI which resolves into a URL. The request will carry sufficient information to identify the resource owner at the data provider and the type of access being attempted:

- An identifier for the 'pension resource' owned by the resource owner at the resource server which is being accessed
- The type of requesting party (owner or delegate)

The assumed design of the 'unique dereferenceable identifier' for each individual pension resource is of the following form; and the access can carry a query parameter asserting the nature of the requesting party user:

- 'urn:pei:'<holder-name GUID>': '<asset GUID>'
- ?user=owner (the default if absent) or ?user=delegate (for advisers or guidance staff)

The asset GUID can be used as a key within the data provider to locate both the internal asset and the pension owner (i.e. resource owner's identifier and PAT) with which it is associated and persisted by the data provider after create resource operation. The data provider will use the PAT associated with the pension owner to coordinate with the authorization server to introspect the RPT (Section 6) or to obtain a permissions ticket (Section 7) if necessary.

Example of a view request carrying an RPT:

```
GET/pei/b1c832df301a431aab330c7ef88275de?user=owner HTTP/1.1
Host: www.dashboard.aviva.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IjE2MzQ1NiJ9.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IjE2MzQ1NiJ9.
```


API Baseline Design

REQUIRED. An array referencing zero or more strings representing scopes to which access was granted for this resource. Each string MUST correspond to a scope that was registered by this resource server for the referenced resource. Pension Dashboard valid granted scopes (in a Requesting Party Token, RPT) must be a list of two containing "value" and "owner" or "delegate".

exp

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this permission will expire. If the token-level exp value pre-dates a permission-level exp value, the token-level value takes precedence.

token_type

OPTIONAL. Type of the token as defined by PDPs UMA Profile, will be pension_dashboard_rpt

exp

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire. If the token-level exp value pre-dates a permission-level exp value, the token-level value takes precedence.

iss

OPTIONAL. String representing the issuer of this token.

Example of a response containing the introspection object:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...
{
  "active": true,
  "permissions": [
    {
      "resource_id": "658b9e38-dd91-4e35-93ca-5154aba7321e0",
      "resource_scopes": [
        "value", "owner"
      ],
      "exp": 1651001341
    }
  ],
  "token_type": " pension_dashboard_rpt",
  "exp": 1651001341,
  "iss": https://claimsgathe.sandbox.k8s.dev.pensiondashboard.org/am/oauth2
}
```

Resource Servers are responsible for access to the resource – the introspection response needs to be compared with what is stored internally (i.e. resource_id, scopes and expiry times) in order to determine whether the access attempt is valid and relates to the correct resource.

6.7 Error Handling

If the request to the introspection endpoint is incorrect, then the authorization server instead responds as follows:

If the referenced resource cannot be found, the authorization server MUST respond with an HTTP 404 (Not Found) status code and MAY respond with a `not_found` error code.

If the resource server request used an unsupported HTTP method, the authorization server MUST respond with the HTTP 405 (Method Not Allowed) status code and MAY respond with an `unsupported_method_type` error code.

If the request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed, the authorization server MUST respond with the HTTP 400 (Bad Request) status code and MAY respond with an `invalid_request` error code.

7 Permission API

7.1 Summary of Permission API

If the view request made by the dashboard is without an RPT or is accompanied by an invalid RPT then the resource server will coordinate with the authorization server to request one or more permissions (resource identifiers and corresponding scopes) on the dashboard's behalf and receive a permissions ticket on return. The resource server must request scopes "value" and either "delegate" (if the inbound call explicitly requested this), or, "owner" (by default or as explicitly requested), but not both. The permissions ticket is used by the dashboard to initiate the UMA Grant protocol with the authorization server to obtain a new RPT in order to authorise it's view request.

7.2 Hosting

The API will be hosted on the authorization server which is part of the C&A service. The authorization server will declare this endpoint in the discovery document so that the resource server knows the endpoint.

7.3 Format

The API will be a REST API using JSON encoded as UTF-8

7.4 Authorization

The data provider will need to use the relevant PAT specific to the pension owner to authorise it's use of the API when making permissions requests to the authorization server.

7.5 HTTP Method

The resource server will call the Permission API using HTTP POST method. The body of the HTTP request message contains a JSON object for requesting a permission for single resource identifier.

The object format is derived from the resource description format specified in Section 4.6; it has the following parameters:

resource_id

REQUIRED. The identifier for a resource to which the resource server is requesting a permission on behalf of the client. The identifier MUST correspond to a resource that was previously registered.

resource_scopes

REQUIRED. An array referencing zero or more identifiers of scopes to which the resource server is requesting access for this resource on behalf of the client. Each scope identifier MUST correspond to a scope that was previously registered by this resource server for

the referenced resource. Pension Dashboard valid requested scopes (in a permission ticket) must be a list of two containing "value" and "owner" or "delegate".

Example of an HTTP request for a single permission at the authorization server's permission endpoint, with a PAT in the header:

```
POST /perm HTTP/1.1
Content-Type: application/json;charset=UTF-8
Host: as.pdp.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjM5MDIyNDg5MCI6ImV4cCI6MTIzNDU2Nzh9.XliAWL97BMJx4V3WIIZESvEWhw7DGGUTJAOpIGYv_H0
...
{
  "resource_id": "KX3A-39WE",
  "resource_scopes": ["value", "owner"],
}
```

7.6 Response

If the authorization server is successful in creating a permission ticket in response to the resource server's request, it responds with an HTTP 201 (Created) status code and includes the ticket parameter in the JSON-formatted body as signed JWT. Regardless of whether the request contained one or multiple permissions, only a single permission ticket is returned.

For example:

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
...
{
  "ticket": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjM5MDIyNDg5MCI6ImV4cCI6MTIzNDU2Nzh9.XliAWL97BMJx4V3WIIZESvEWhw7DGGUTJAOpIGYv_H"
}
```

The PMT token is a JWT¹ as defined in [JWT] and contains the following claims.

iss

REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within dashboard ecosystem of the AS issuing the JWT.

sub

REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within scope of iss of the Resource Owner identifier at the AS (*derived from the PAT used in the initial permission ticket request*).

aud

REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within the scope of the dashboard ecosystem of the authorization server.

¹ PDP's profile of UMA suggests the use of structured tokens however this is not mandatory and is up to the choice of the UMA implementors

iat

REQUIRED. Registered claim name. Defined [JWT]. Profiled: time of issue.

exp

REQUIRED. Registered claim name. Defined [JWT]. Profiled: time of expiry.

jti

REQUIRED. Registered claim name. Defined [JWT]. Profiled: using jti as the unique token identifier.

Public claim name. *none*

rs

REQUIRED. Private claim name. The identifier of the Resource Server. *(Derived from the PAT used in the initial permission ticket request.)*

owner

OPTIONAL. Private claim name. The identifier of the RO at the Resource Server. *(Derived from the PAT used in the initial permission ticket request.) May be of use to the RS to minimise lookup time of resource_id to derive the owner of the resource.*

permissions

REQUIRED. Private claim name. UMA permission as defined in [UMAGrant] and [UMAFedAuthz] using scopes defined in this profile.

rqp

OPTIONAL. Private claim name. Structured representation (JSON object) of the *pension_dashboard_rqp* which was presented with the permission ticket (if any) in a previous call to the AS. Claim must be present when the permission ticket is presented for the second or subsequent time by a dashboard client. The AS must populate this claim with the contents of the *pension_dashboard_rqp* which was presented in the call for which it is reissuing a permission ticket, having checked that the content is in accord with the same requesting party presenter.

assuredID

OPTIONAL. Private claim name. Structured representation (JSON object) of the identity of the requesting party (as uniquely represented at the AS) and the asserting identity provider reference. Claim is present when the AS reissues it after assured identification and if necessary, confirmation of the assured professional status of a 'delegate' requesting party.

The token must be signed by the issuer. It MUST be encrypted for the AS.

The token may be persisted by the AS to enable correlation across presentations of such tokens.

As per [UMAGrant] 5.5 permission tickets are single use: the AS must issue a new token with a new jti for every iteration of the permission process. The AS must ensure that authorization process and any dependent tokens are revoked if a permission ticket is replayed.

The token will always be presented to the token or claims interaction endpoints at the AS by the DB client so it does not need to be bound further than application level checking by the AS which must ensure that the *pension_dashboard_rqp* details match across related calls.

API Baseline Design

```
"consents_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SfIKxwRJSMeK
KF2QT4fwpMeJf36POk6yJV_adQssw5c",
"resource_id": "KX3A-39WE",
"pei": "urn:pei:f1c72611-438b-4f72-a4b5-ec7e69000c31:8ff2063a-48bd-4ed7-
bcf8-7c3b8f89626d"
}
```

8.6 Response

If the request sent by the C&A is success then the data provider will respond with a HTTP *202 Accepted* status implying acknowledgment of the PAT refresh request.

HTTP/1.1 202 Accepted

Following that the resource server requests a new PAT by quoting the user account token in its request the authorization server token endpoint – see Section 3 Obtain PAT.

8.7 Error Handling

If the PAT refresh request sent by the authorization server is fails then data provider is expected to respond with the appropriate HTTP status code and error message. The body of the response for an error must contain a message in verbose and plain language where it describes, accordingly with the error catching process, what went wrong and possibly how to amend so as to issue a new, valid request.

4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a POST request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Status	Message	Note
400	Bad Request	Potential bad requests examples: <ul style="list-style-type: none">• authorization server is not sending a HTTP POST• Parsing error by data provider• Schema not configured correctly

5xx

The 5xx (Server Error) class of status code indicates that an exception occurred during the elaboration of a request. An indication about the nature of the error SHOULD be included together with an indication if the error is temporary or permanent.

Status	Message	Note
500	Internal Server Error	

9 Authorise API

9.1 Summary of Authorise API

The dashboard initiates the authorization 'dance' using this Oauth specific API to AS token endpoint to obtain a new access token (RPT). The dashboard will need to provide a set of claims needed for the AS to assess the current authorization context when

interacting with the AS token endpoint. Once this process is successful the AS in its response will issue a new access token (RPT) for the Dashboard to authorise it's view requests in the future.

9.2 Hosting

The API will be hosted on the authorization server which is part of the C&A service. The dashboard will have obtained the `as_uri` from the resource server following a failed view request.

9.3 Format

The API will be a REST API using JSON encoded as UTF-8.

9.4 Authorization

The dashboard client will authenticate itself with the authorization server at run time as defined in [rfc8705 section 2](#). In addition to this, all end point connections are secured using mutual TLS.

9.5 HTTP Method

The dashboard MUST use the HTTP "POST" method when making access token requests to the authorization server's token endpoint. It sends the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type

REQUIRED. MUST be the value `urn:ietf:params:oauth:grant-type:uma-ticket`.

ticket

REQUIRED. The most recent permission ticket received by the client (dashboard) as part of this authorization process. Permission ticket is a structured JWT.

claim_token

REQUIRED. The client must provide a claim of type `pension_dashboard_rqp`. The contents of this token must represent the current requesting party user at the dashboard client.

claim_token_format

REQUIRED. The claim (above) is of a specific type `'pension_dashboad_rqp'`. UMA requires this parameter to match the claim(s).

scope

REQUIRED. The dashboard client request MUST contain the requested pension dashboard scopes: `value` and `owner/delegate`

pct

OPTIONAL. The client MUST provide its existing PCT for the requesting party (i.e. for the combination of the client and its user), for the resource the client is seeking to access for that requesting party, if it has one, even if it knows that the PCT has expired.

rpt

OPTIONAL. The client SHOULD provide its existing RPT for the resource it requested in the previous call to the same RS for the same requesting party, if it has one, even if it knows that this RPT is expired. *(Although the profile will not upgrade RPTs, this is included to enable flexibility for possible future extensions.)*

For example, the dashboard client makes the following HTTP POST request using mTLS:

API Baseline Design

```
POST /token HTTP/1.1
Host: as.pdp.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
...
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Auma-ticket
&ticket=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
&claim_token= aGVsbG8gd29ybGQ
&claim_token_format= this type name needs agreed URI format, based on the profile's domain
&scope=value owner
&pct=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
&scope=value owner
&pct=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

9.6 Response

If the permission request is successful the Authorization Server will issue the RPT and optionally a PCT if required.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c ",
  "token_type": "pension_dashboard_rpt",
  "upgraded": false,
  "pct": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"
}
```

9.7 Error Handling

If the permission request is unsuccessful the Authorization Server will respond with the appropriate error.

Example of a need_info response with a hint to redirect the requesting party to a claims interaction endpoint:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
...
{
  "error": "need_info",
  "ticket": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"
}
```

API Baseline Design

```
FtZSI6IkpvaG4gRG9IiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNjIzOTgwZGF1udPrSdTRkLf3QzRoHFff_T6BzRqISADIZoZ-c5sI ",  
"redirect_user":"https://as.pdp.com/rqp_claims?id=2346576421"  
}
```

Example when the client was not authorised to have the permissions:

```
HTTP/1.1 403 Forbidden  
Content-Type: application/json;charset=UTF-8  
Cache-Control: no-store  
...  
{  
  "error":"request_denied"  
}
```

10 Obtain Pels API

10.1 Summary of Obtain Pels API

In order for the dashboard, whether used by the owner or their delegate, to obtain the owner's PeIs following the completion of Find, the dashboard will pull the PeIs from the owner user's C&A account via an API hosted at the authorization server.

10.2 Hosting

The API (an UMA Resource Server) will be hosted on the authorization server which is part of the C&A service.

10.3 Format

The API will be a REST API using JSON encoded as UTF-8.

10.4 Authorization

The dashboard client will authenticate itself with the authorization server at run time as defined in [rfc8705 section 2](#). In addition to this, all end point connections are secured using mutual TLS.

10.5 HTTP Method

The Dashboard will make a HTTP GET request to the obtain PeI endpoint for that specific user's C&A account – i.e. the dashboard attempts to GET the URL `https://CA.ObtainPeIs/<userGUID>` with a parameter of the user is 'owner' or 'delegate'. The 'userGUID' is the unique identifier for the user's C&A account. As it is a protected resource the dashboard will need to quote an appropriate access in order for the request to be successful.

Example of a obtain PeIs request carrying an RPT:

```
GET/a7f542a22c8647b3b62c2fb9a81c2495?user=owner HTTP/1.1  
Host: www.CA.ObtainPeIs.com  
Authorization: Bearer  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9IiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNjIzOTgwZGF1udPrSdTRkLf3QzRoHFff_T6BzRqISADIZoZ-c5sI",  
EFOqwxjF_xSJyQQ.
```


API Baseline Design

maintained from the Governance Register. The master will be accessible via an API by registered dashboard providers.

11.2 Hosting

The API will be hosted on the Governance Register

11.3 Format

The API will be a REST API using JSON encoded as UTF-8.

11.4 Authorization

The dashboard client will authenticate itself with the authorization server at run time as defined in [rfc8705 section 2](#). In addition to this, all end point connections are secured using mutual TLS.

11.5 HTTP Method

The Dashboard will make a HTTP GET request to the obtain PeI configuration endpoint with the parameters containing the variable name (i.e. pei) and corresponding value (i.e. holdertype) – i.e. the dashboard attempts to GET the URL `https://GR.ObtainPeIsConfig?pei="26d93ebc-0dfd-43c0-bfee-2b8f8ad7a742,728f9722-88c1-42f3-965a-d2faab8967e8,..."` this will be for a specific set of PeIs. The dashboard can alternative request all of the PeI configuration from the Governance Register by using the appropriate query parameter in the HTTP GET request – i.e. the dashboard attempts to GET the URL `https://GR.ObtainPeIsConfig?pei=all`.

11.6 Response

The Governance Register will respond with the corresponding hostname for each of holdertype.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...
{
  "26d93ebc-0dfd-43c0-bfee-2b8f8ad7a742": "aviva.dashboard/pei",
  "728f9722-88c1-42f3-965a-d2faab8967e8": "scottishwidows.dashboard/pei"
}
```

11.7 Error Handling

If the request sent by the dashboard fails then Governance Register is expected to respond with the appropriate HTTP status code and error message.

400 Bad Request

Potential bad requests examples:

- Dashboard is not sending a HTTP GET
- Parsing error by governance register
- Schema not configured correctly

503 Service Unavailable

- Service is down for maintenance or overloaded by requests

12 Technical Standards

12.1 Dashboard Redirection Protocols

Dashboards expose no interfaces (other than their redirection endpoint which is used to unwind a previous redirection to the relevant Consent and Authorization redirection interface).

Redirection from dashboard to the C&A is a vital constituent of these processes and for dashboard designers it is vital to understand in connection with the strict 'APIs'.

Dashboards must redirect their user agents to either of C&A's interfaces – UMAGrant.ClaimsRedirection or ConsentandControl.Redirect. The ClaimsRedirection interface is part of the UMA authorization flow. The Consent.Redirect interface handles requests of type:

- Find (incl pullPeis)
- Refresh PAT (RS failure case)
- Consent
- Account deletion (GDPR)

For every interaction made by the dashboard to the C&A it has to mint a new RQP token.

Formal description of the token. The RqP token is a JWT as defined in [JWT] and profiled here.

REQUIRED iss. Registered claim name. Defined [JWT]. Profiled: unique identifier within dashboard ecosystem of the dashboard instance issuing the JWT. (This is Dbi.)

REQUIRED sub. Registered claim name. Defined [JWT]. Profiled: unique identifier within scope of iss, of the requesting party which is authenticated to iss at the time the JWT is issued. (This is user@dbi.)

REQUIRED aud. Registered claim name. Defined [JWT]. Profiled: unique identifier within the scope of the dashboard ecosystem of the authorization server.

REQUIRED iat. Registered claim name. Defined [JWT]. Profiled: time of issue.

REQUIRED exp. Registered claim name. Defined [JWT]. Profiled: time of expiry.

REQUIRED jti. Registered claim name. Defined [JWT]. Profiled: using jti as the unique token identifier.

Public claim name. none

REQUIRED role. Private claim name. The iss states the role in which the requesting party is acting. String value. One of "owner" or "delegate".

The token MUST be signed by the issuer. The token MAY be encrypted for the AS.

The token must always be presented to the token or claims interaction endpoints at the AS by the DB client along with an AS issued token. It does not need to be bound.

12.2 JWT Signing and Verification

After successfully onboarding to the CDA you will receive a crypto package which will contain a signing certificate and embedded within it the private key which will be used to generate the signature using RS256 as the signing algorithm to sign a JWT.

In order to verify the signature of a JWT the CDA will expose a centralised JWKS endpoint which you will be able to call to obtain the corresponding public key to verify the signature by using the "kid" parameter to match on which is found in the JWT header.

Data providers will need to ensure they pass their assigned "kid" parameter as part of the JWT header when returning pension details back to a dashboard. The "kid" will be generated as part of onboarding and its format will be a GUID. It will be sent to providers as part of the crypto package.

Example of JWT header containing "kid" parameter:

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "ec1abf89-225b-49c2-ab87-1d425ac70f8d"
}
```

12.3 Pension identifier format

The pension identifier (PeI) shall have a standardised format across all providers. It is expressed in the form of a URN (uniform resource name) that provides a location-independent, globally-unique, persistent identifier, with a defined namespace. PeIs are issued by the data provider (or by the obligated entity) and must be associated with zero or one pension asset. PeIs (more specifically the PeI.assetGUID) can be associated with a pension asset at any time prior to registration of the asset with the C&A Service. Once associated with a pension asset a PeI should not be reused for a different pension asset.

It is a URN of the form: `urn:pei:'<holder-name GUID>':<asset GUID>`.

An example of a possible PeI is:

```
urn:pei:f1c72611-438b-4f72-a4b5-ec7e69000c31:8ff2063a-48bd-4ed7-bcf8-7c3b8f89626d
```

Both the Holder-name GUID and the Asset GUID are globally unique identifiers (see rfc4122 <https://www.ietf.org/rfc/rfc4122.txt>) of 32 hex digits (128 bits) allocated 'randomly' by standard methods.

Holder-name is a globally unique string which can be dereferenced to an endpoint (i.e. URL) which serves view requests, composed of the view endpoint and the asset GUID making the query URL. If properly authorised, the pension details associated with that asset ID are served by the full URL.

Holder-name must be associated with the part-scheme which is responsible for the specific pension asset. A part scheme may have more than one holder-name (e.g., if the data provider wants to share the view load over multiple view endpoints) but this is likely to be rare.

More than one holder-name will usually dereference to the same endpoint. (I.E., a data provider will have one or a few endpoints each of which serve many but different part-schemes.)

```
urn:pei:f1c72611-438b-4f72-a4b5-ec7e69000c31:8ff2063a-48bd-4ed7-bcf8-7c3b8f89626d
```

example View URL constructed by dereferencing the holdername to the base URL :
<https://testISP.co.uk/8ff2063a48bd4ed7bcf87c3b8f89626d>

12.4 Data Providers (UMA Resource Servers)

Data providers (UMA resource servers) will operate the UMA FedAuthz protocols, which include introspection, permission and register API calls, as described in the above sections.

The resource server must persist the following for each Resource Owner following a create resource operation:

- resource _id – index of the registered resource (PeI)
- resource owner’s PAT – access token to the Protection API
- authorization server 'AS URI' which issued the PAT (at which the resource _id is registered) - address of the authorization server token endpoint

The resource server should also persist these items in a manner which allows it to locate them using the inbound URL of the view request.

Resource servers are responsible for access to the resource – introspection response needs to be compared with what is stored internally for the resource in order to ensure access request is authorised.

13 Appendix

13.1 Glossary

Term	Definition
Resource Owner (i.e. Pension Owner)	The resource owner is a user or legal entity that is capable of granting access to a protected resource
Client (i.e. Dashboard)	The client is an application that is capable of making requests with the resource owner's authorization and on the requesting party's behalf
UMA Resource Server (i.e.Data Provider)	The resource server hosts resources on a resource owner's behalf and is capable of accepting and responding to requests for protected resources
UMA Authorization Server	Part of the C&A – the authorization server protects resources hosted on a resource server on behalf of resource owners. It manages and applies the resources owner’s policy
PMT - Permission Token	Gives permission for a dashboard to initiate the authorization protocol in order to authorise a retrieval request
RQP – Requesting Party	Identifies the current user and their role in session at the dashboard when requesting access. Can be either a pension owner or a delegate

API Baseline Design

RPT – Requesting Party Access Token	Needed by the dashboard to authorise its view call to the data provider endpoint in order to retrieve pensions details related to the PeI. Each PeI will have a unique RPT (i.e. one to one association)
PCT – Persistent Claims Token	This claim binds the asserted user at dashboard and role to that user’s assured identity at the ecosystem and, where applicable, professional status. In doing so it acts as a medium-term authenticator correlator helping reduce user friction in subsequent sessions when attempting to retrieve pension details
PAT – Protection API Token	OAuth2 token scope UMA protection which represents the Resource Owner’s authorization for the RS to manage federated authorization at the AS (permits APIs to register, obtain PMT, introspect RPTs)
User Account Token	OAuth2 authorization grant issued by the authorization server to the resource server so that it can be exchanged at the authorization server’s token endpoint for a PAT

13.2 Client Registration

Registration of OAuth clients (i.e. dashboards and data providers) is required for the operation of the UMA profiles. The 'governance register' (which includes the UMA Authorization Server as the software entity managing software client registration) will provide services for client registration.

Static registration may be more secure but dynamic registration may be more expected by industry participants, may involve fewer manual processes, and is necessary to support public client types (especially non-confidential i.e. SPAs, native apps).

In order for PDP to offer dynamic client registration it requires the AS to support dynamic registration [ODynClient] and to provision clients appropriately. The AS needs to support software statements to bootstrap the registration process securely. Client authentication to the AS will be defined in accord with dynamic registration requirements.

The alpha phase of the programme will focus on static client registration. Dynamic client registration will be looked at once the requirements to support this have been refined with the supplier.

13.3 Overview of PDP's Primary Components

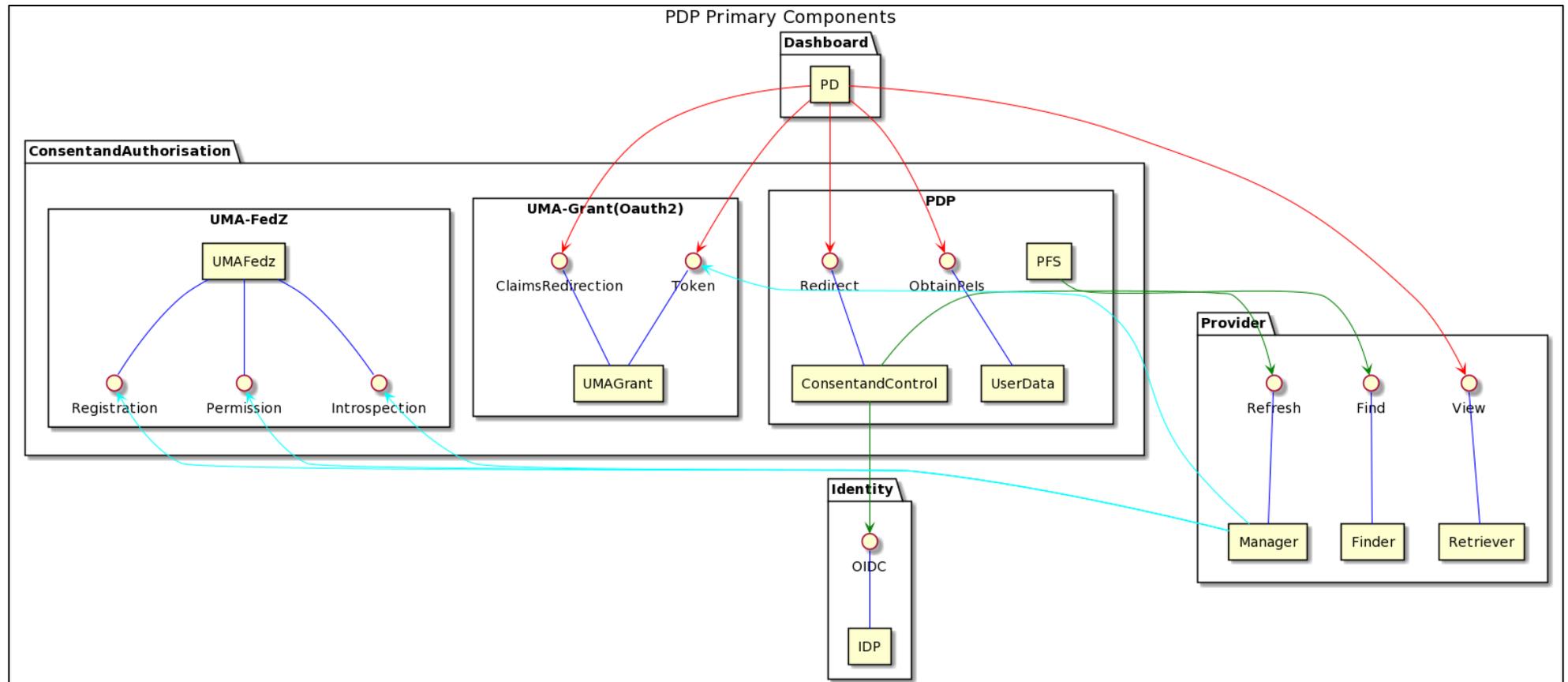
This diagram shows each component involved in the protocol interactions in this document. Components are presented in packages ; each package corresponds to a participant in the ecosystem: dashboard, C&A service, data provider, identity service. Within a package the components expose interfaces (blue lines) and call interfaces of other components (lines of other colours dependent upon the caller). E.g. the Data Provider interfaces are called by Dashboards (red) and by C&A components (green).

Calls within packages are not shown on this diagram: e.g. PDP.ConsentandControl – handling redirections – deeply interacts with UserData and the UMA Authorization components; e.g. PDP.ConsentandControl interacts with PFS when it initiates a search, passing consents and search parameters and Authorization Server grants.

Dashboards expose no interfaces (other than their redirection endpoint which is used to unwind a previous redirection to the relevant Consent and Authorization redirection interface).

PDs must redirect their user agents to either of C&A's interfaces – UMAGrant.ClaimsRedirection or ConsentandControl.Redirect. The ClaimsRedirection interface is part of the UMA authorization flow. The Consent.Redirect interface handles requests of type 'Find', 'Consent' and 'Refresh' .

API Baseline Design



The Authorization Server is not explicitly shown on the diagram: it is part of the Consent and Authorization package and comprised of the UMA components and *some* of the functions of ConsentandControl and UserData.

The **two UMA2 interfaces** can be seen: Federated Authorization is called by Data Provider, and UMA Grant by the Dashboard.

Note that, since an UMA authorization server is a specialisation of an OAuth2 authorization server, the Token interface, part of UMA-Grant(OAuth2), is *also* the same endpoint called by the Data Provider when it needs to obtain a PAT (Protection API Access Token), which

is an OAuth2 access token with scope `uma_protection`, subsequently used against the UMA Federated Authorization Protection API (UMAFedz).

Apart from UMA support itself, the PDP package shows the **other primary components of the C&A Service**. The user-facing Consent services manage user interaction and policy. UserData contains the policy and registered PeIs for each user; it exposes the interface for dashboards to obtain PeIs for their users. The ConsentandControl component also manages user and authorization processes and repositories (UserData, selectable names for user directed find, etc). PFS performs search orchestration after the ConsentandControl component has established the context, which includes proving user identity using the Identity Service, establishing UserData, and other tasks as necessary.

The **Identity Service** interface (IDP.OIDC) is simplified in this diagram (it is also a specialisation of the standard OAuth2 Authorization Service endpoints).

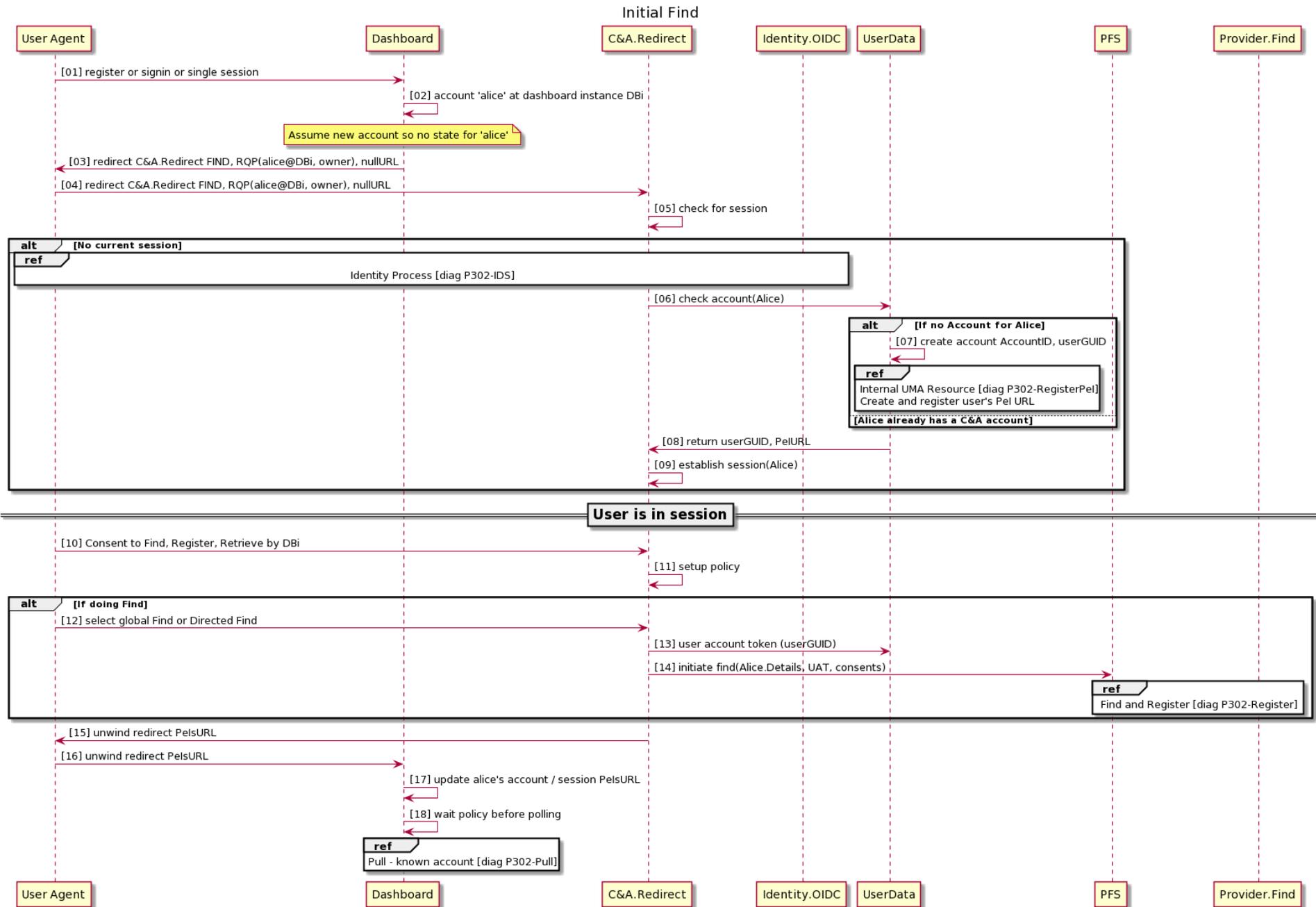
14 Sequence Flows

14.1 Initial Find

This section presents a flow for a new user at a dashboard, initiating a find activity at the C&A.

In this case the dashboard knows it has a new user (unless it is a stateless dashboard when it *assumes* it has a new user) and redirects to the C&A to process a find operation. The condition around Step 12ff is discussed in a later section on obtaining PeIs URL from dashboards. In the simple 'new find' variant, these find steps will be performed, specifically the PFS will be invoked, Step 14, to orchestrate calls to several/all data providers.

API Baseline Design



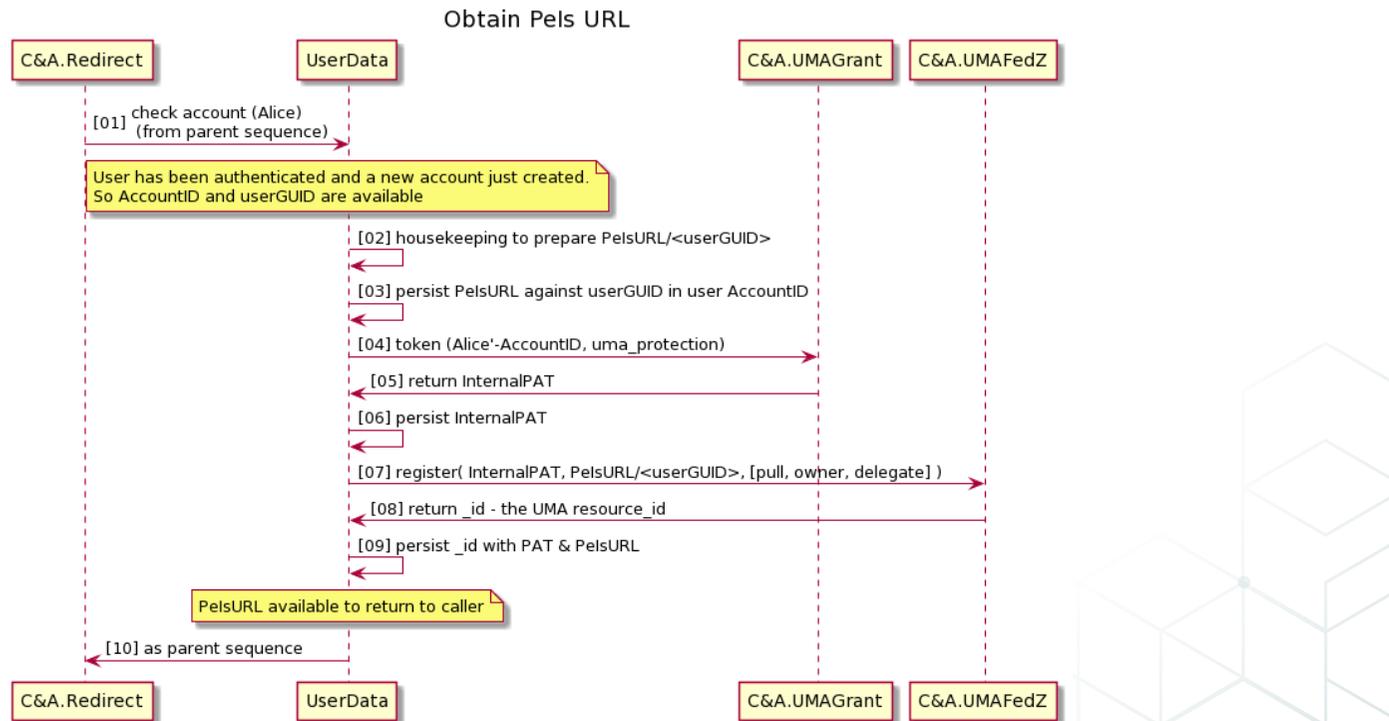
API Baseline Design

14.1.1 Internal Register Obtain Pels-URL

This sequence is nested in the above flow. After creating a C&A Account at step 07 this sequence registers the protected resource for the owner user's PullPeIs resource. In this case both the UMA Authorization Server and the UMA Resource Server which presents the PullPeIs resources are in the *same security domain*, i.e. are part of the C&A Service. Accordingly, there is the possibility of significant optimisation (customisation) of the relationship between the AS and the PullPeIs Resource Server, yet to still offer the same external UMA2 grant based access to the protected resource by dashboards.

The following takes an 'UMA2 Federated Authorization' approach.

In Step 04 the parameter Alice'-AccountID is Alice's identity token, available from the parent sequence, along with the new derived AccountID, from which a token can be constructed in the trusted UserData RS. This token is used as an authorization grant for the Authorization Server to issue an internal PAT. In implementation, this could use the OAuth2 JWT-bearer grant. Since the AS and the RS (the UserData component) are coupled and in the same domain, the AS could check that Alice's account (AccountID) exists (not shown on the diagram) or simply trust the internal caller and signature, and issue the requested access token (the InternalPAT) at step 05.



14.2 Pull Pels

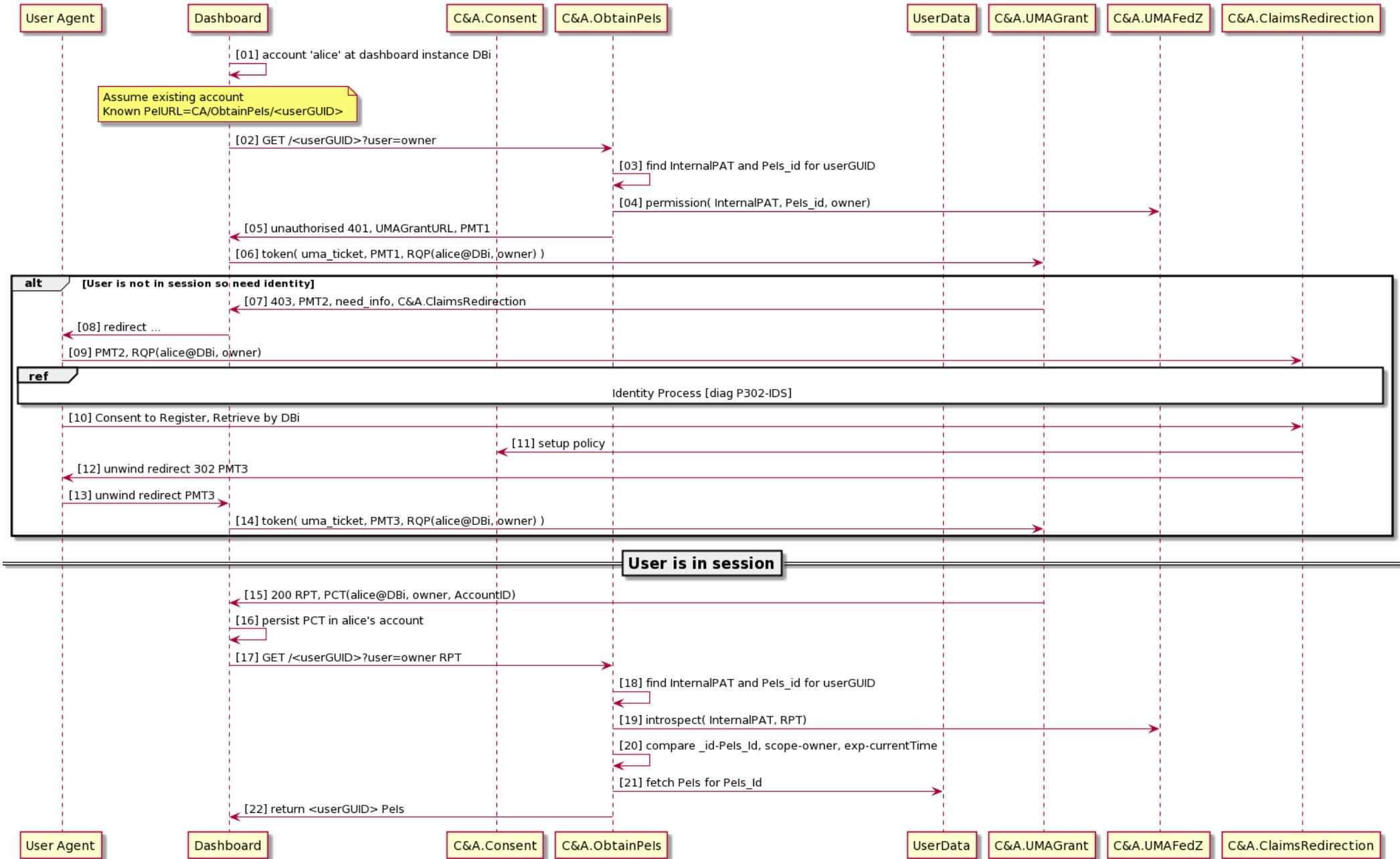
Once the dashboard knows the user's C&A account in the form of the relevant protected PeIs resource, it can pull the PeIs as follows. The dashboard attempts to GET the URL `https://CA.ObtainPeIs/<userGUID>` with a parameter of the user is 'owner'. Initially, this will fail because the dashboard does not quote an appropriate access token, but it will trigger the UMA authorization dance which obtains the RPT which then succeeds in the subsequent call.

Whilst it is tempting (and valid) to read this flow as a direct successor to the Initial Find flow above, remember that this also stands alone and can be performed whenever necessary.

- Immediately after find new
- For up to the defined SLA after an initial find as the (slower) data providers find pension records
- After repeat finds (user-controlled timing, user directed finds)
- When a delegate (financial adviser or guidance officer) is initialising their records for the client in which they have the client-specific PeIsURL in their communication from the C&A Server.
- When an owner-user initialises a new dashboard by importing PeI data from a previous one, the UMA PeIs resource would bootstrap the process of initialising the new dashboard. (New PD attempting to obtain the PeIs would initiate the whole of this process, including new identity assertion, account update, new consent and policy for the new dashboard.)

API Baseline Design

Pull Pels



14.3 Provider Registers Pels

This flow is referenced from Initial Find flow above.

The PFS's only role is to orchestrate the calls, step 01 in the diagram below, the receipt of each 'find' call should be acknowledged using low level http 202 code; note this has nothing to do with whether the find actually results in the location of a pension. When a pension is found it is registered at the C&A service by the Data Provider using the UMA Protection API and an access token issued for the purpose, a PAT.

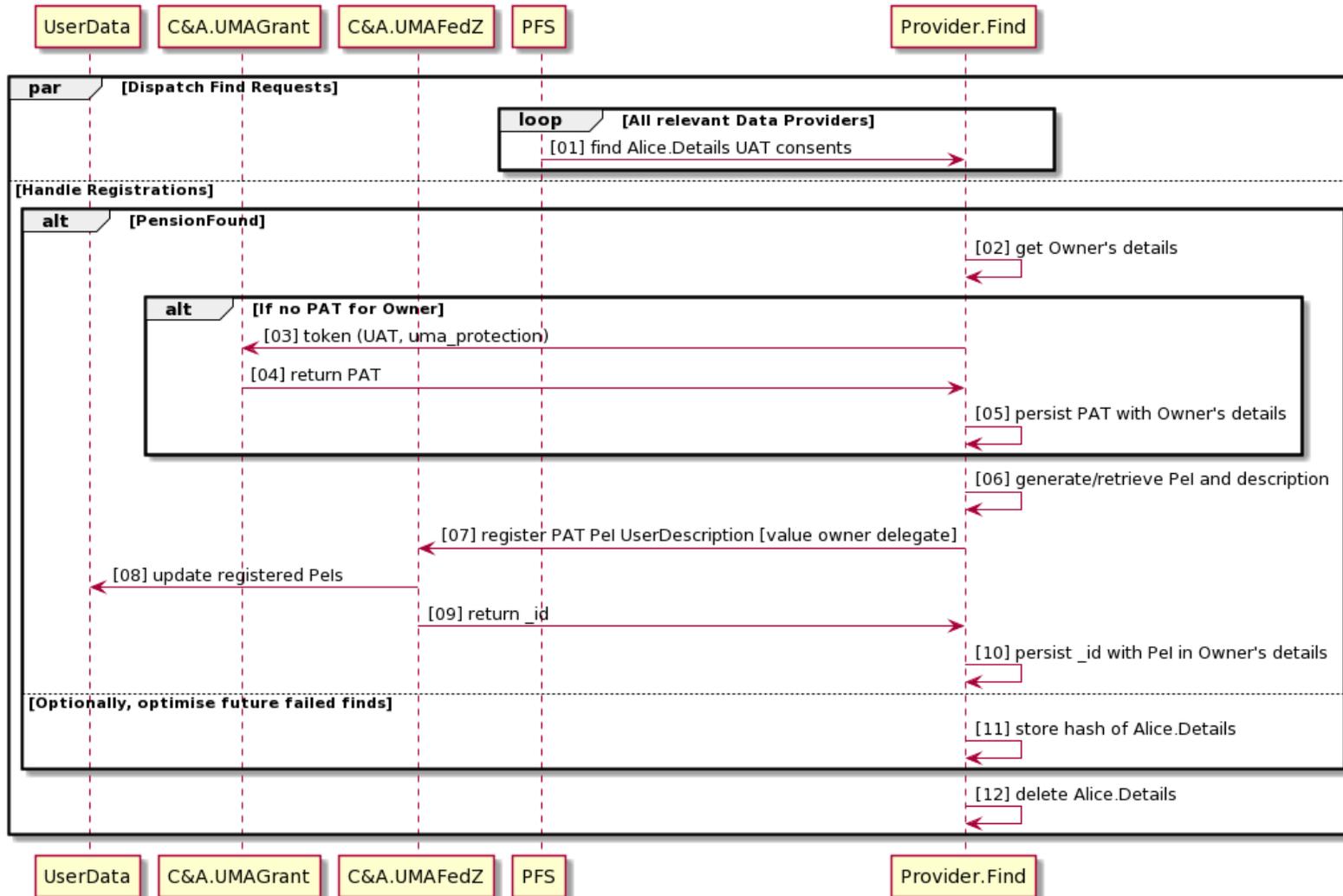
If no pension is located (Step 11) the data provider may store a hash of the some of the details used in the search so that when a subsequent search is initiated by the same user the provider does not waste resources in running a repeat search.

Consents in this flow are assumed to permit Find and Register. If Register is not permitted then the Data Provider could still contact its customer using its existing details as it knows that the customer is interested in their pension (but this is a business decision for the provider).

If a pension is located (and consents permit) the provider should register the corresponding PeI with the C&A server as per the flow (steps 02ff).

API Baseline Design

Register Pels



API Baseline Design

Note that Alice's personal details originated from the IDP / C&A Service are never retained by the Data provider whether or not the find is successful, step 12. This is to ensure that the data provider cannot use them for any other purpose than finding the pension, in accord with PDP policy.

The case where the find failed has been mentioned above. However, in cases where the find succeeded (so steps 02-10 apply) a data provider will need to arrange for **step 02 'get Owner's Details'** to function correctly, for view and for find, whether or not the same user has executed a find operation in the past. The PD ecosystem requires that these details are:

- Any identifier the provider wants to use for the owner's account
- Any internal keys or similar which enable the provider to locate relevant internal records associated with the owner and their assets
- The user's authorization server (Of course in the PD ecosystem at least initially this will be the one C&A Server, but theoretically this might change over time if users appoint other 'open finance' components, so the actual C&A AS URL should be kept in the RS from the outset; in any case it is required to be returned to the dashboard in the UMA protocol.)
- PAT when one exists
- Associated PeIs and their descriptions, including a map of inbound URLs to the relevant PeI.
- UMA _id for each PeI

Data providers will need to consider how to retrieve this data in two cases:

- On the basis of the inbound View URL. This is a map of URL->PeI->_id and from that to the owner & PAT.
- On the basis of subsequent 'find' requests for the *same* user. The provider needs to decide whether to repeat the find operation itself, which should result in the same internal owner identifier and internal keys (and thus find the above details), or keep a hash of the inbound personal details and use that to index the above details.

Step 03 shows the use of a temporary credential as an **OAuth2 grant** of type JWT-bearer, as described in section 3.5.

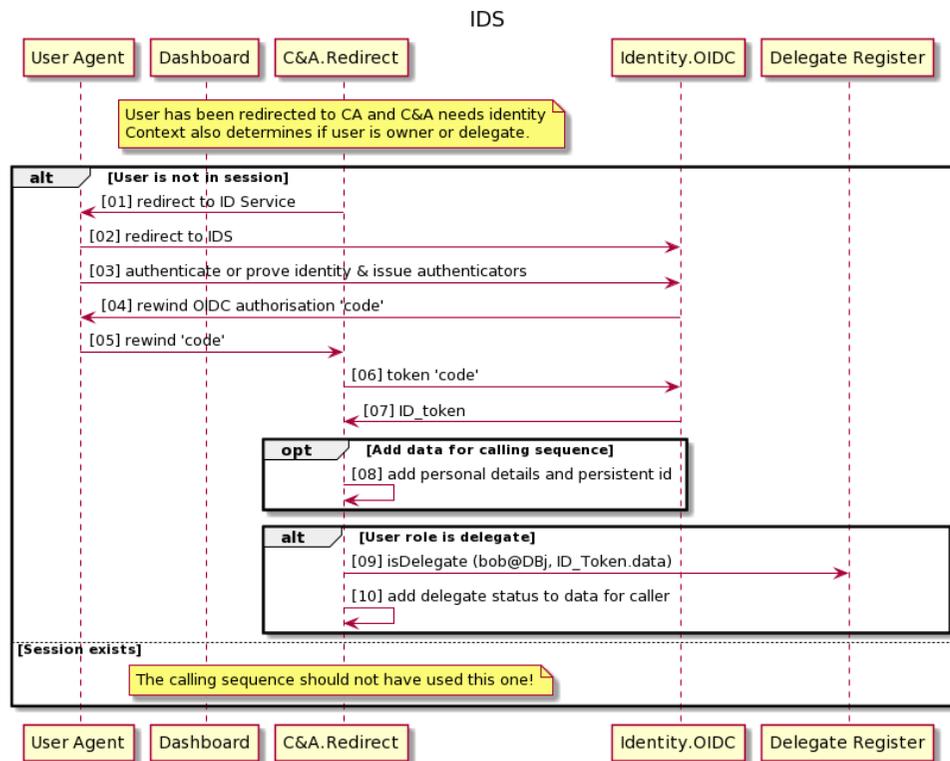
Step 06 'generate or retrieve PeI' also indicates a decision for providers. A PeI is a unique URI (not URL) identifying a 'asset' / 'pot' / 'pension' uniquely and persistently. It is *independent* of the owner of the asset. Thus it can be allocated *at any time* from the creation of the asset until its initial registration with the PD C&A service. There are choices open to data providers in their business processes and in their preparation for PD integration based on the properties of the PeI, its behaviour across providers, and the provider's decisions on processing 'maybe' or 'uncertain finds'. These matters need to be discussed with industry representatives and PDP should issue advice or mandation.

API Baseline Design

14.4 Identity Process

Whenever the C&A Service needs to determine a user's identity it uses the Identity Service, which is (assumed to be) a federation of external Identity Providers, interfaced via some form of hub. The details of this are to be discussed in separate documents / PDP procurements. The purpose of this section is to present how this service integrates with the C&A.

There are two key use cases, dependent upon the 'role' of user. The user may be a pension *owner* or the user may be a *delegate* of one or more owners. Both user roles need identity proofing. PDP has not yet determined the Level of Confidence (according to GPG45) required for proofing of each role. The Identity Service will issue authenticators to each user to a certain standard (GPG44) to enable reauthentication without re-proofing. In addition, delegates will have to have their status as a valid delegate checked. This check will probably be recorded by the PDP governance register which will surface an API to be used for verification in the following flow.



API Baseline Design

The implementation of the standard OIDC Identity service will comprise of an authorization endpoint (handling redirection of the user agent to so the user can authenticate and prove their identity) and a token endpoint from which the identity assertion (ID Token) will be retrieved, using OIDC authorization code flow. OIDC Core also defines a 'user_info' endpoint to obtain additional claims, in addition to those which may be in the ID Token, if the Pension Dashboard Identity Profile or Standards requires it. In addition, it is likely that the OIDC interface will be brokered via a hub component (not shown in this document) so that a federation of Identity Providers can be supported.

The context of this sequence (e.g. Initial Find above) will determine details of the information required from the Identity process. For example, the call will determine whether the user is a delegate on a delegate dashboard (e.g. bob@DBj) or an owner user. The sequence should only be called when the caller has checked there is no current session for the owner user. After this flow the caller may also create the C&A account for the owner user and establish the required session as appropriate, using the temporary data noted in the flow.

The dashboard may have already authenticated the user using the ecosystem ID Service (see P301 6.1 for discussion). The fact that the user is authenticated by an IDP may be added to the redirect parameters (probably adding an element to the RQP so an RQP is comprised of: alice@DBi, owner, and idp=<IDP>). Given that PDP mandate that the IDPs in its federation must maintain an open session (for a controlled period) then the redirection to the ID Service in the above flow can immediately return the OIDC grant code without requiring the user to reauthenticate at the Open ID provider (IDP).

14.5 View

When a dashboard has obtained PeIs for its user, it will usually seek to view the pension details, by GETting the *dereferenced* PeI. This action will initially fail, as there is no access token, initiating the UMA2 Grant authorization dance.

Dereferencing a PeI means taking the PeI which is a URI (format "urn:pei:<holdertype>:<assetidentifier"> e.g. "urn:pei:aJ001:WGF45920EJH348ASEWQ0284") and breaking it into components, looking up the holdertype, e.g. 'aJ001' in a configuration table to derive a host name, e.g. 'aviva99.dashboard/pei', and composing a URL, e.g. <https://aviva99.dashboard/pei/WGF45920EJH348ASEWQ0284>. It is this URL which is an UMA protected resource; the URL of the pension details to be accessed by an http GET with appended role parameter, eg '?user=owner'.

The Governance Register API will map the <holdertype> element of the PeI to the current URL of the provider's view endpoint (example above). It is recommended / required for performance reasons that dashboards incrementally build a cache of the configuration table for those 'holdertypes' used in their users' PeIs. A cache entry might be deleted if a resulting hostname is not resolved or periodically based on PD ecosystem standards.

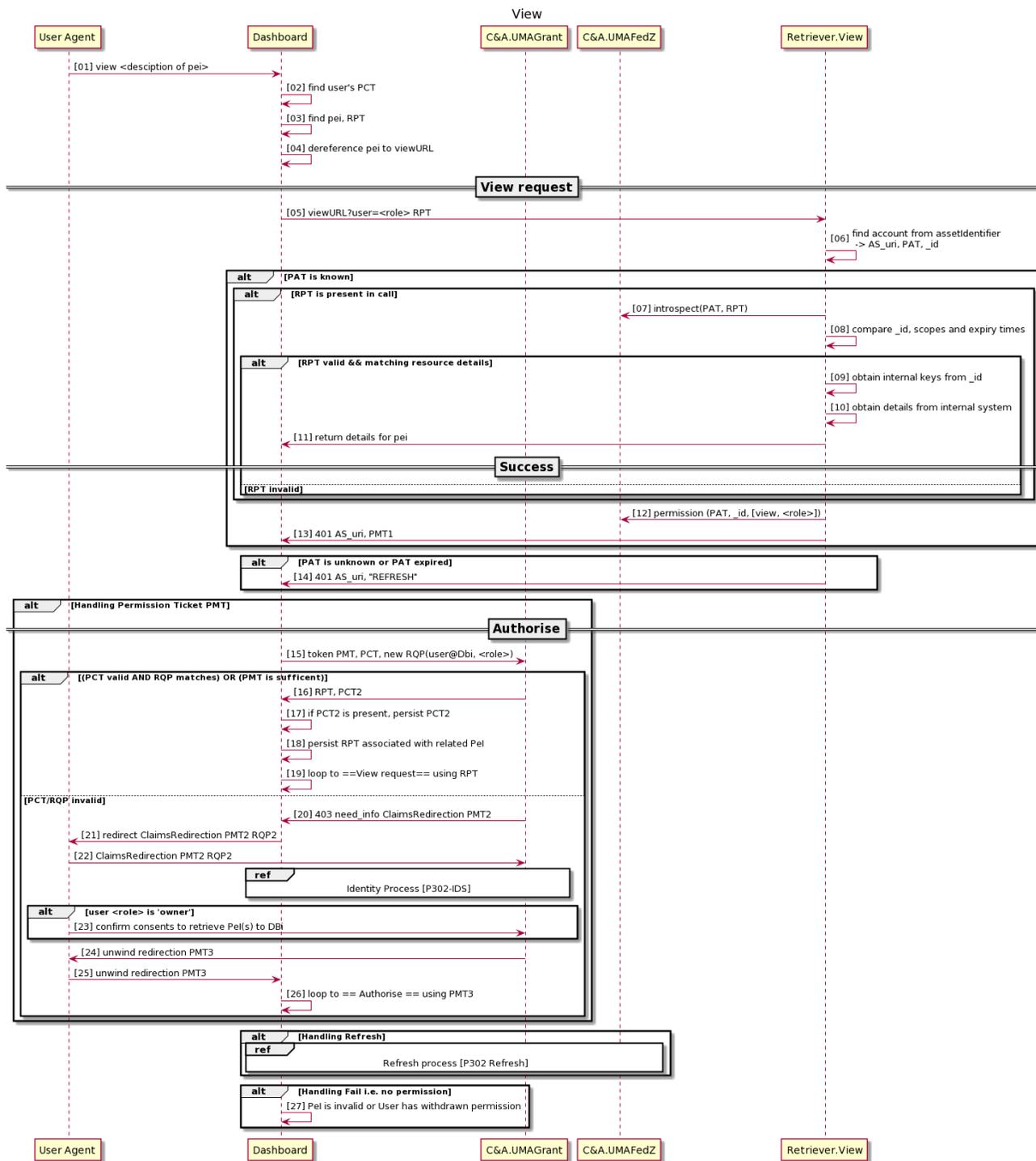
Dashboard preparation. A user may select one or more descriptions of their pensions to view (Step 01). The dashboard finds related control information whether this be from a persistent store (for dashboards which have accounts & appropriate security controls) or from its current session. For each user description of an asset there will be a PeI and may be an existing access token, RPT. If the user has been authenticated to the Identity Service recently from that dashboard there may be a PCT available from that user's account at the dashboard. The PeI is dereferenced as discussed above, deriving the protected resource for the asset.

Authorised View Access The 'happy path' in which the provider locates the UMA control information based on analysis of the URL (step06) and there is a valid access token (07) which matches the user's role (owner or delegate) and other information (08), results in the api returning the pension details associated with the PeI to the dashboard.

There are three types of **authorization failure**:

- Those for which the AS can provide an UMA permission token (PMT) to the RS which is returned to the dashboard.
- The case in which the PAT has expired, so the UMA AS cannot respond to the provider's RS request for a permission ticket. This is handled by a PDP-specific error message requesting 'REFRESH' processing which is discussed in a later section.
- Final failure in which the RS or AS does not return a PMT nor a specific instruction. In this case the dashboard must assume that the PeI is invalid or that the user has withdrawn consent for that dashboard (in spite of having been directed to the C&A as part of the above automatic handling processes).

Well-behaved dashboards respond to these conditions as per these flows.



API Baseline Design

The **Authorization process** (Step 15) uses the current PMT (always present), and current PCT (if there is one) and a new RQP token, minted by the dashboard for every relevant call to the C&A service. The RQP (Requesting Party token) is an assertion by the dashboard that a specific user known to it (eg 'alice') is present at the interface of a specific dashboard instance (e.g. DBi), and that the user is acting in a specific role (i.e. owner or delegate).

The AS makes the authorization determination based on the state (contained in the PMT at step 15 and in the AS) and the other parameters. If the authorization request is successful, the AS returns an access token for the specific protected resource (RPT) and it may issue a new or replacement PCT. The PCT (Persistent Claims Token) binds the user at the dashboard to the C&A's view of the identity (derived from the external Identity Service) and the user's validated role. When an authorization request is successful the dashboard should retry the view request.

If the AS determines that it cannot grant access, but that it can continue the authorization process, it issues another PMT and requests that the client (dashboard) redirects the user to its Claims endpoint for an interactive session with the user. There the appropriate identity uplift occurs and if the user is an 'owner' the appropriate consents are refreshed. (Details of interaction between the claims redirection code and the Consent and UserData components are omitted here, but see also the sequences Initial Find and Pull PeIs above.) The AS also needs to consider whether the policy of the owner for a specific PeI and specific delegates matches the recently uplifted user. If identity uplift was successful and policy matches PeI and person, the AS issues another PMT and unwinds the redirection back to the dashboard. The dashboard uses the new PMT to re-attempt the authorization call.

API Baseline Design

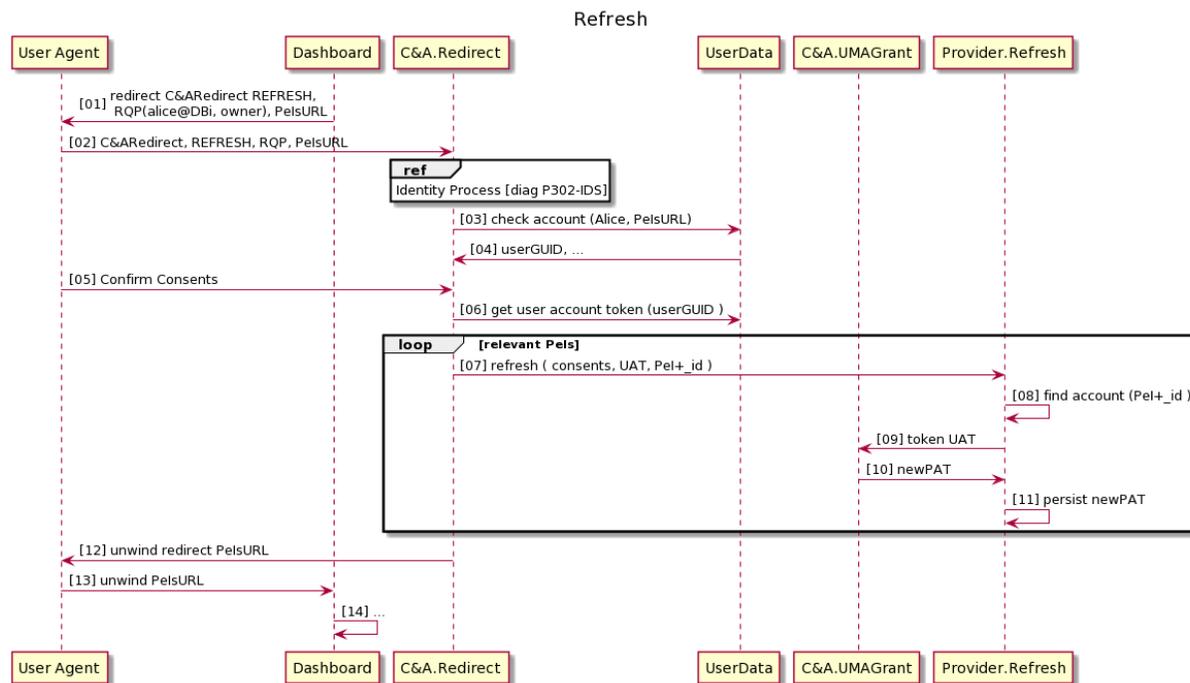
14.6 Refresh Process

Only owner-users can perform refresh (not delegates); same is true of the other redirect purposes – Find and Consent.

Refresh processing may be entered as a result of failure of a view attempt in which a PAT has expired, or in *routine* use of the C&A Consent service by the user (e.g. in adding dashboards, e.g. changing consents, e.g. stepping up identity after PCT expiry after 90 days, e.g. user directed find or repeat find).

The owner-user is redirected to the C&A service so that consents can be confirmed or modified and, if necessary, the identity can be uplifted. If the user is in session at the C&A then there may be no reason to perform a reauthentication. (This is not shown on the diagram but simply skips the Identity Process.)

The flow shows the account details being checked, including the dashboard-provided user's PeIsURL. There are error conditions related to dashboard and user which can be corrected as part of this process (not shown) e.g. the user's correct PeIsURL is returned as described in the section 'Initial Find', e.g. the RQP can be used to check that the user has consent policy for that dashboard.



API Baseline Design

Irrespective of the dashboard's stated purpose for the redirection (REFRESH, repeat FIND or CONSENT), or the purpose of the redirection to the Claims redirection endpoint during authentication (to seek periodic step-up), the C&A has the *opportunity* to run the loop to refresh PATs at the relevant Providers of the PeIs of the user. Clearly if the redirection from a dashboard was for REFRESH, this implies that the dashboard had received an error from a provider during an attempted View, but even in this case the C&A needs to manage the process appropriately: for example if a user has withdrawn consent for any dashboard to view a PeI, or for registrations of PeIs from that provider, there is no need to attempt the refresh.

Note that, unlike the initial find process in which the owner-user's personal details are sent to perform a search at the provider, there is no need for these biographic details in performing a refresh. All that is needed are the parameters shown at step 07. The 'relevant PeIs' are those which a) are registered for the user's account, b) the RS hosting one or more of these PeIs is not subject to a current (user-directed, repeat) find² for the same user, c) the RS has not already been contacted³ in the same refresh loop. For each relevant PeI, which has the relevant resource `_id`, the C&A can determine the provider RS and hence which Provider's Refresh end point to call.

The above flow assumes that the user does approve refreshing the PAT and that the dashboard which initiated the refresh is still consented to receive the owner's PeIs and other data. If not the return redirection (steps 13, 14) would not carry the current PeISURL and may have a specific error code so the dashboard knows the user has withdrawn consent.

² Because if a find is happening for that user for the same RS, the RS will have a separate opportunity to refresh the PAT.

³ Because only one PAT is needed for each user at an RS (not per PeI if the same user has more than one PeI).