OAuth Dynamic Registration/Association

UMA Requirements
UMA Requirement #1

• The client needs to be uniquely identifiable by the authorization server
  – In order for an authorization server to do proper user-delegated authorization and prevent unauthorized access it must be able to identify clients uniquely. As is done today in OAuth, the client identifier (and optional secret) should thus be issued by the authorization server and not simply accepted as proposed by the client.
UMA Requirement #2

• The authorization server must collect metadata about a client for later user interaction
  – In order for the authorization server to describe a client to an end-user in an authorization step it needs information about the client. This can be the client name at a minimum, but today servers usually request at least a description, a homepage URL, and an icon when doing manual registration.
UMA Requirement #2 contd.

authorize an application to access protected data.

Home » Authorize access request

An application that you've been just using (http://smartfetch.appspot.com/uma/oauth_callback) would like to get access to some protected information on your behalf:

Protected data

- All my photos
- This data is located at Gallerify.me
  http://www.gallerify.me

This application will be granted following permissions:

- Permissions to edit data
- Permissions to view data

Allow this application to access the protected information on your behalf?

- Deny
- Allow

http://www.smartam.net
UMA Requirement #2 contd.

The application Gallerify.me at http://www.gallerify.me would like to have access to your account and to:

- Validate access requests to your albums.
- Register albums in order to protect them.

Gallerify.me
Online photo management tool

Allow Gallerify.me access to your account?

Deny
Allow

http://www.smartam.net
UMA Requirement #3

• The authorization server must have the option of strongly authenticating the client and its metadata
  – In order to prevent spoofing of clients and enable dynamic building of strong trust relationships, the authorization server should have the option to verify the provided information. This might be solved using message signature verification; relatively weaker authentication might be achieved in a simpler way by pulling metadata from a trusted client URL.
UMA Requirement #4

• Dynamic client registration must be possible from both web-server applications and applications with other capabilities and limitations, such as native applications
  – Various UMA Hosts
  – Various UMA Requesters
  – Each instance of a native application that is installed and run by the same user may need the option of getting a unique client identifier.
UMA Requirement #5

• Transaction integrity must be ensured in large deployments where data propagation can be an issue
  – When a client sends information to a server endpoint, it might take time for this data to propagate through big server installations that spread across various data centers. Care needs to be taken that subsequent interactions with the user after the registration process, such as an authorization request, show the correct data.
  – Interactions between Hosts/Requesters and the same Authorization Manager
UMA Requirement #6

• In line with design principles and requirements defined by UMA
  – DP1: Simple to understand, implement in an interoperable fashion, and deploy on an Internet-wide scale
  – DP6: Able to be combined and extended to support a variety of use cases and emerging application functionality
UMA Requirement #6 contd.

– DP8: Avoid adding crypto requirements beyond what existing web app implementations do today.
– DP10: Complexity should be borne by the authorization endpoint vs. other endpoints.
Dynamic Reg. – Push Flow

CLIENT | (A) | Registration Request | with Metadata | AUTHORIZATION SERVER

| (B) | Registration Response | with Client ID Info |

POST /register HTTP/1.1
Host: server.example.com
Content-Type: application/json

```
{
  type: "push",
  client_name: "Online Photo Gallery",
  client_url: "http://onlinephotogallery.com",
  client_description: "Uploading and also editing capabilities!",
  client_icon: "http://onlinephotogallery.com/icon.png",
  redirect_url: "https://onlinephotogallery.com/client_reg"
}
```

HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

```
{
  client_id: "5UO9XcL4TQTa",
  client_secret: "WdRKN3zeTc20"
}
```

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

```
{
  "error": "unauthorized_client",
  "description": "This client is not on the white list of this Authorization Server."
}
```
Dynamic Reg. – Pull Flow

Figure 2: Client Registration Flow with Pushed URL and Pulled Metadata

POST /register HTTP/1.1
Host: server.example.com
Content-Type: application/json

{
  type: "pull",
  url: "http://onlinephotogallery.com"
}

HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "client_id":"5U09XcL4TQTa",
  "client_secret":"WdRKN3zeTc20"
}

<XRD>
  <Host>http://onlinephotogallery.com</Host>
  <Property type="http://oauth.net/client/name">
    Online Photo Gallery
  </Property>
  <Property type="http://oauth.net/client/description">
    Really nice online photo gallery!  
  </Property>
  <Property type="http://oauth.net/client/uri" href="http://onlinephotogallery.com">Client URI</Property>
  <Link rel="http://oauth.net/client/icon" href="http://onlinephotogallery.com/icon.png">Client Icon</Link>
</XRD>