FHIR Consent Management Using UMA/OAuth 2.0

Version 2.1

Mohammad Jafari, Kathleen Connor,
John M. Davis, Christopher Shawn

November 27, 2017
# Table of Contents

1. **Introduction** ........................................................................................................................................ 1

2. **Use-Case** .......................................................................................................................................... 1

3. **Overview of the Solution** .................................................................................................................... 1
   1.1 Consent ............................................................................................................................................ 1
   1.2 Patient Information .......................................................................................................................... 2
   1.3 UMA/OAuth 2.0 .............................................................................................................................. 2

4. **Components** ...................................................................................................................................... 2

5. **Flow** .................................................................................................................................................. 4
   1.4 Preconditions .................................................................................................................................... 4
   1.5 Sequence .......................................................................................................................................... 5

6. **Issued and Lessons Learned** ............................................................................................................... 7
   1.6 Issuer of Trusted Claims .................................................................................................................. 7
   1.7 Consent Resource .......................................................................................................................... 7
   1.8 Whose Consent? .............................................................................................................................. 8
   1.9 Finding Applicable Consents ......................................................................................................... 8
   1.10 Conflict Resolution ....................................................................................................................... 8
   1.11 Issues with UMA ........................................................................................................................... 9
   1.12 Moving Towards Cascading Authorization ................................................................................ 10

7. **Authorization for Apps** ..................................................................................................................... 10
   1.13 App Registration ........................................................................................................................... 10
   1.14 Patient Consent Form .................................................................................................................... 11
   1.15 Case-by-Case Approval ................................................................................................................ 11
List of Figures

FIGURE 1. HIGH-LEVEL OVERVIEW OF THE FLOW 4
FIGURE 2. SEQUENCE FROM THE INTERCEPTOR’S STANDPOINT. 6
FIGURE 3. SEQUENCE FROM THE AS STANDPOINT. 7
FIGURE 4. DIFFERENT PERCEPTION OF THE CONSENT RESOURCE. 8
1. INTRODUCTION

This document discusses the design details of the demonstration project “FHIR Consent Management Based on UMA/OAuth 2.0” which was initially presented at the HL7 FHIR Connectathon in September 2016 in Baltimore and subsequently updated in November 2017 to extend its use-case to access by mobile and web applications.

It is focused on the core concepts and the main flows of a Consent Management System backed by FHIR which leverages UMA/OAuth to communicate authorization decisions. At a later stage in the project, we expanded the design and added the Cascaded Authorization Services which is discussed in more general terms in a separate document. This document does not go into the details of the Cascading Authorization and remains focused on the enforcement of patient consent directives.

2. USE-CASE

The use-case for this demonstration is a typical exchange scenario in which a Client wants to access some patient’s data residing at a Custodian Organization. The aim of this demonstration is to honor patient’s preferences about sharing their information in the course of this use case.

While the initial implementation and demonstration was focused on the case in which the Client is another healthcare organization, we will also discuss the case of smaller clients such as mobile and web applications.

3. OVERVIEW OF THE SOLUTION

The main goals of this solutions are as follows:

- Develop an authorization technology that integrates the generic OAuth/UMA authorization with the authorization decisions made by patient in the form of patient Consent.

- Providing a mechanism for decoupling the authorization decisions pertaining to the patients’ consents from the resource server where the information is stored and outsource it to a FHIR-based Consent Directive Management System (CDMS).

The CDMS stores and processes patient consents and provides an authorization service based on User-Managed Access (UMA)/OAuth 2.0. This provides a mechanism for the Custodian Organization to separate (or even outsource) handling of patient consents and only inquire about consent authorization decisions when needed.

On the other hand, both the Client and the Custodian organization will be able to rely on a common standard for authorization, namely, OAuth, while patient Consent preferences are transparently incorporated into the result of such authorization decisions under the hood. In other words, the client and custodian are relieved from the burden of collecting, maintaining, and processing patient Consents and can simply rely on the standard OAuth service to take care of that.

1.1 Consent

Patients’ preferences can be recorded and stored in various standard or proprietary forms. We use the latest FHIR Consent resource which is part of the ongoing development of the HL7 FHIR standard.
1.2 Patient Information
We also assume that at the Custodian Organization, patient information is either stored in a standard FHIR Server or is accessible via a FHIR Adapter where the information can be requested according to the FHIR API.

1.3 UMA/OAuth 2.0
The UMA protocol is based on a three-step flow for managing authorization for access to resources:

- Registering resources in the form of Resource Groups with an Authorization Server (AS). In the upcoming UMA 2.0, this has moved out of the core to the Federated UMA profile.

- Setting up the policies governing access to these resource groups by the suitable party with the authority to make such decisions. UMA considers the form in which these policies are formulated and the mechanism in which they are set up in the server out of the scope of UMA.

- Providing a process by which a client can obtain approval of the AS in the form of a Requesting Party Token (RPT) prior to access to resources. Only clients who can provide such an Access Token will be allowed to access the data and their access will be subject to the Scopes determined by the AS. A key step in this process is providing certified Claims by the Client signed by a trusted authority to assert Client’s identity and attributes.

In order to leverage the UMA protocol for authorization, our solution uses the following mappings:

- The patient takes the role of the UMA Resource Owner.

- Each resource type for each patient constitutes an UMA Resource Group; e.g. Alice/Immunization is the resource group containing all the immunizations of patient Alice.

- FHIR Consent resources are treated as the policies governing access to the patient’s information. Note that such consents could be filed either by the patient or by others on the patient’s behalf. It can also be either explicit (e.g. by filling a form) or implicit (e.g. created implicitly by clicking a ‘share’ button).

  The UMA server fetches and consumes FHIR consents and relies on them as authorization policies.

- Confidentiality clearances are modelled as UMA scopes.

4. COMPONENTS
The following major components were developed in the demonstration:

- **Authorization Server (AS):** A server which offers the standard capabilities of UMA including:

  o Resource group registration end-point,

---

1 Note that in our implementation we used patient identifiers (rather than names) in forming the resource group ID.
- **Consent Handler:** The consent handler is tightly bound to the UMA AS and is in charge of looking up and fetching consent directives and processing them per request, eventually returning an authorization decision including authorized scopes to the UMA AS. The consent handler includes a module for Consent Discovery which looks for a given patient’s consents based on their identifier in a number of pre-introduced FHIR Consent Repositories.

- **Authorization Interceptor:** A component that monitors all the requests received by the FHIR resource server at the Custodian Organization and handles the authorization and labeling before letting the server send out any data in response.

- **Security Labeling and Privacy Preserving Service (SLS/PPS):** The SLS/PPS are in charge of:
  - Tagging outgoing FHIR resources with security labels based on the labeling rules,
  - Removing any resource from the outgoing FHIR bundle for which the Client’s presented Access Token does not have enough scopes, and
  - A High-Watermark label on the outgoing FHIR bundle based on its content.

- **Client Library:** A FHIR client library which encapsulates the logic of handling the UMA/OAuth authorization protocol, thereby enabling the Client to focus on their business logic while the authorization in handled transparently. This enables client-side developers to set up the authorization configurations and then work with the FHIR server while the authorization steps are handled under the hood.

- **Demo Graphical User Interface:** A single-page graphical user interface to demonstrate the process of the fetching some patient’s data, the ensuing AuditEvent resource, and the authorization steps taking place in the process.
5. FLOW

This section discusses the flow of the use-case while providing details of the authorization steps taking place in order to enforce the patient’s consent. Figure 1 shows a high-level overview of the flow. Figure 2 and Figure 3 provides further detailed sequence diagram from the viewpoint of the Interceptor and the Authorization Server.

1.4 Preconditions

The following assumptions are made for this flow:

- We assume the patient has filed their Consent with a FHIR Consent Directive Management System and it is stored in a FHIR server available and known to the AS. In our demonstration, we used a simple configuration setting on the AS side to specify the URIs to all the possible FHIR Consent repositories.

- We assume that the FHIR resource server has been equipped with the Authorization Interceptor. In our demonstration, this required adding the interceptor to the FHIR server application space and setting the FHIR server to use it for incoming/outgoing requests/responses.

- We assume that the AS and the Authorization Interceptor have been introduced and have established a trust relationship. In this demo, this will take place by setting up the relevant endpoints of the AS (the permission registration and introspection) with the configurations of the resource server.
1.5 Sequence

- **Client Request:** The client sends a request to the FHIR Resource Server. This request is picked up by the Authorization Interceptor and the would-be server response is determined. The response is often a bundle including one or more resources.

- **Pre-Authorization:** If the client does not provide an Access Token:
  - **Inspect the Would-Be Server Response:** The Authorization Interceptor evaluates the would-be server’s response to check whether any protected resources are in the bundle. If there are no protected resources, the request is permitted. The current demo handles this simply by looking up the resource type in a list of protected resources from the configuration. This may, however, be expanded to be a decision depending on more complex factors.
  - **Permission Registration:** The Authorization Interceptor sends a permission registration request to the AS informing the AS what resource groups are being requested and what scopes are required in order for the request to go through. The AS returns a Ticket denoting the success of this registration.
  - **Redirect and Ticket:** The Authorization Interceptor sends a response to the client redirecting it to the AS. This response includes the Ticket returned by the AS in the previous step which acts as some form of a confirmation code for the client to use when requesting for an access token.
  - **Request Access Token:** The client sends a request for an Access Token to the AS providing the Ticket received in the previous step, as well as a secure Claims Token which includes assertions claims about its identity and purpose of use.
  - **AS Decision:** The AS makes an assessment of this request by:
    - Collecting the identity and purpose of use claims
    - Recovering the requested permissions corresponding to the presented Ticket,
    - Querying the Consent Handler to find, fetch, and evaluate any Consent resources that could be applicable to this request and to return an authorization decision including granted scopes.
  - **Issuing Access Token:** If the result of the assessment step is to permit, the AS issues an Access Token and records the corresponding granted scopes alongside other information such as time-to-live and the address of the referenced Consents which are the basis of the decision. If the assessment step returns with a deny, the AS will inform the client with a suitable response.

- **Post-Authorization:** If/when the client provides an Access Token:
  - **Validation and Scopes:** The Authorization Interceptor runs the Introspection Protocol to verify the validity of the token and get the associated scopes. If the token is invalid or expired the request is denied.
  - **SLS Invocation:** The Authorization Interceptor invokes the Security Labeling Service to tag the resources in the response bundle with sensitivity and confidentiality labels according to the labeling rules.
  - **PPS Invocation:** The Authorization Interceptor compares the confidentiality labels on the response data with the granted confidentiality clearances to the client
(in the form of authorized scopes). Any resource for which the client does not have sufficient clearance is removed from the response.

- **High-Watermark**: The Authorization Interceptor invokes the SLS High-Water Mark service to compute the high-watermark labels on the response bundle based on its remaining content.
- **Record Audit**: The Authorization Interceptor creates an AuditEvent resource recording the information required for the accounting of disclosure.
- **Response**: The Authorization Interceptor returns the modified and labeled response back to the client.

**Request Processing at ACS Interceptor**

![Diagram of request processing]

- **HTTP 401 Unauthorized + UMA Server address + UMA ticket**
- **HTTP 403 Forbidden**
- **Invoke Organizational SLS and PPS**
- **Register Permission**
  - patient name + resource type + requested scopes
- **Introspect access token**
  - valid? patient name + resource type + granted scopes (clearances)
- **Redact any resource with a label which is not in the granted scopes (clearances)**
- **Invoke high-watermark on this bundle**

*Figure 2. Sequence from the Interceptor’s standpoint.*
6. ISSUED AND LESSONS LEARNED

In this section, we review some of the issues or points for further extensions as well as some of the lessons learned in this project.

1.6 Issuer of Trusted Claims

When a Client requests for an Access Token from the Authorization Server, it needs to provide a signed token (in the form of a JSON Web Token (JWT)) to prove its identity, e.g. the Client’s identifier, Organization name, etc. The party that issues this assertion must be trusted by the Authorization Server so that its assertions are accepted.

In the initial demonstration, we did not specify who issues and signs these Claim tokens. In the simplest form, the Custodian Organization can issue these tokens after identifying and onboarding partner organizations. Alternatively, the Authorization Server can rely on third-party identity providers for issuing such tokens. At the end of the day, the issuer must be known to the Authorization Server and needs to have a trust relationship with it. As we will discuss later, in the case of smaller clients, such as mobile or web Apps, this role can be assumed by a globally known authority that performs authorization, on-boards, and verification for Apps, such as the App store.

1.7 Consent Resource

An important issue raised in the course of our analysis was whether the consent represents the patient preferences “as they file it” or whether it is considered a computable policy compiled from the raw form which the patient fills, perhaps in the form of a questionnaire. Figure 4 depicts these two different viewpoints.
For the purpose of the initial demo, we assumed that the Consent resource represents a computable and self-containing policy. This presumes that the tools used to assist the patient to author the consent will eventually compile all the patient choices into a computable consent policy ready to be parsed and relied on as an authorization policy by the UMA server.

**Figure 4. Different perception of the Consent resource.**

### 1.8 Whose Consent?

One of the interesting questions in the design of this demo was finding the patient who has the authority to determine policies about a given resource. In this demo, we simply used a query mechanism to find the patient to whose FHIR Compartment the requested resource belongs. This relies on an algorithm that simply does a graph navigation from the target resource until it finds a patient. In general, the problem may be more nuanced and might require more complex processing.

### 1.9 Finding Applicable Consents

To find a patient’s applicable consent, we relied on a Consent Discovery component within the Consent Handler module. A simple instantiation of this component was used in the original demo which accepts the URI for a number of FHIR repositories where it can look for a patient’s consents by a simple FHIR search query based on that patient’s identifier. This component will look for the consent based on the patient identifier (which is different from the FHIR URI of the patient resource in one FHIR server) and is assumed to be uniquely identifying a patient across different consent repositories. Moreover, this component will also look for Consents filed by others (e.g. a legal guardian) which would apply to the resources of the patient in question.

In general, this mechanism can be expanded to include more sophisticated forms of consent lookup, such as direct Consent push by the requester in which the Client provides a copy of the patient’s consent as an evidence for proving it has earned the patient’s approval for the requested access.

### 1.10 Conflict Resolution

When the Consent Handler module finds all of a patient’s Consents, it processes this set to separate those consents that are valid and applicable to the given request, for example by matching the purpose of use and other factors.

It is possible, however, that multiple consents with potentially contradictory decisions are found. The Consent Handler relies on a Conflict Resolution component to resolve these conflicts and
distill them into a single ultimate decision. In the initial demo, a simple implementation of this component was used based on a Most-Recent-Overrrides policy which prioritizes the decision expressed by the Consent files at the most recent date. We acknowledge that in general, the conflict resolution may be more complicated.

1.11 Issues with UMA

We identified a number of issues in the UMA specifications; our initial demo used some workarounds to solve these issues and sent the feedback the UMA workgroup. Some of these issues are resolved in the current draft for UMA 2.0.

1.11.1 Registering Multiple Permissions

The permission registration API only accepts a single Permission object which presumes only one resource set is accessed per request. This is problematic when many resource sets are being requested, for example, in a search request. Moreover, the Introspection API, which returns the accepted permissions (scopes) associated with an Access Token, already returns an array of Permissions. As a workaround, we used a mechanism to encode multiple Permission objects into a single compound Permission to be unwound by the UMA server upon receipt. However, this is an issue that needs to be fixed by UMA in their future iterations.

1.11.2 Wildcard Scopes

UMA specifications require that the Resource Server determine the scopes required by the Client’s access request and register a set of UMA Permissions with the AS.

In some applications like the case of our demonstration, this can unnecessarily leak some information about the patient’s information to the AS. For example, suppose that Alice has a Restricted medication, while Bob does not have any Restricted medication. When a client requests access to Alice’s medications, the Permission object registered by the RS will include the scope Restricted, while in the case of Bob, it will not include that scope. From observing this pattern of permission registration, the AS can tell that Alice likely has a protected medical condition but Bob does not.

Moreover, determining these scopes, and thereby deciding the required scopes to fulfill a request can be additionally costly to the Resource Server, especially if it does the labeling on the fly.

To avoid these issues, in our demonstration, we decided to ask for all possible scopes for each request and leave it up to the AS to grant the minimal authorized subset of these scopes. This would prevent the AS from noticing any difference between the labels assigned to different patient’s records. This was, however, possible because we were working with the Confidentiality labels and there is a limited set of such labels in the FHIR specifications. In case this set would not be limited, or had a very large number of labels, this approach could become problematic.

A better solution would be for UMA specifications to provide a mechanism to not specifically enumerate for scopes in the Permission registration step, and instead allow a way to state “anything” instead. This could enable the Resource Server to simply leave it up to the AS to determine what scopes could be granted.

1.11.3 Negative scopes

UMA specifications currently only support modelling granted scopes with the implication that any scope that is not granted is denied. This contrasts with an alternative approach where the
server can explicitly deny some scope with the implication that any other scope which is not explicitly denies is granted.

In the FHIR Consent resource, there are cases where the patient can express their consent by a default opt-in and then exceptions to explicitly deny some scopes. For example, when the patient files an opt-in consent for Organization A, with the exception of any resource marked as Restricted or Very Restricted.

Modeling a denial in the form of granted permission requires a closed-world assumption in order to model a denied scope in the form of the lack of a grant. For example, in the previous example, granting all confidentiality clearances other than the two explicitly denied can imply which ones have been denied. This requires that the Consent Handler run an algorithm to find the complement of a denied scope and grant that set.

Note that, this is only possible because the set of all possible confidentiality labels is known and has a limited number of members. This may not be the case, for example, in the case of the value sets.

Having a mechanism to explicitly prohibit a scope makes it a lot more efficient to encode such cases where the closed-world assumption cannot be made, i.e. the system cannot determine the set of all possible values, e.g. compartment labels. In our demonstration, we considered using a special modelling of scopes (e.g. prefixed with a ‘~’). However, since scopes are a fundamental concept in the specifications, eventually we decided to stick to the letter of the standards and not implement negative scopes. We made a closed-world assumption, as confidentiality labels are fully enumerated in the FHIR specifications, and implementation negative scopes by subtracting the set of denied scopes from the set of all possible confidentiality scopes.

1.12 Moving Towards Cascading Authorization

One of the concerns raised about the original design presented at the HL7 Connectathon in September 2016 was the fact that while the authorization logic for patient consent policies were outsourced to an UMA/OAuth server, that server only accounted for the patient consents and the logic for handling other applicable policies such as organizational and jurisdictional policies was still within the Authorization Interceptor. This was not desirable because ideally we wanted a complete separation of authorization logic from the resource sever. Moreover, this design also had the undesirable complexity of handling one group of authorization policies using UMA/OAuth 2.0 while leaving other authorization policies to other technologies. To ameliorate these issues and to improve the design and its uniformity, we decided to expand the Authorization Service into multiple Cascading Authorization Servers each of which could be in charge of handling one type of policies. This provides the capability to not only separate authorization logic from the resource server, but also to separate the authorization logic pertaining to different kinds of authorization policies from each other, thereby eventually leading to both a more functionally-capable, as well as a more elegant and consistent design.

7. AUTHORIZATION FOR APPS

The architecture discussed in this document can be used for authorizing mobile and web applications to access patient’s information by applying the following considerations.

1.13 App Registration

Mobile and Web Apps are usually developed in an App ecosystem where developers submit their Apps to an App Store which makes them available to clients after an approval process. The approval process often includes identifying and authentication the developers of the App and
verifying its code to make sure it does what it claims. Since App stores are often known and trusted entities, the process of App registration is the best place to issue the credentials for an App. These claims are verified and signed by the App store in the form of a token (JWT). An example of such a claim is:

**App ID:** x2y3z4.z4y3x2  
**App Developer:** VHA  
**Purposes of Use:** Treatment

Note that the fine-grained purpose of use depends on the details of the business process in which the data is used. However, each App will have a limited number of broader (e.g. HIPAA level) purposes which can be verified and asserted as a Claim by the App store at the time of verification. Since the App store often conducts a general verification of the App functionality and code, it is in the perfect position to verify whether the business workflows in an App are aligned with the claimed purposes of use. For example, the App store can refuse to approve the claim of a Research or Travel Planning App to use the data for the purpose of Treatment.

### 1.14 Patient Consent Form

In the case of exchange between different organizations the patient sets general authorization policies without getting manually involved in approving individual requests. Unlike that, in the case of App authorization, there is a reasonable expectation by the Patient to have the option to authorize certain Apps manually and on a case-by-case basis. This is also practically possible, because unlike the exchange between organizations, the Patient is usually present and actively involved at the time of App authorization and can be redirected to authorize an App on-the-fly based on a simple OAuth flow via a web interface.

The patient consent, therefore, must provide a mechanism for the patient to specify this. In other words, aside from the other decisions factors available to patients in formulating their consent, (e.g. Purpose of Use, Requesting Organization ID, etc.) the patient also needs to have the option to state that for certain Apps manual authorization on a case-by-case basis is required. For example, a patient consent can include the following rules:

- **Consent 1:** Grant [all scopes] to any App by [VA] for the purpose of [Treatment] expires on [2018/10/10],
- **Consent 2:** Grant [all scopes except Restricted, Very Restricted] to any App by [Microsoft] for the purpose of [Treatment], expires on [2018/10/10],
- **Default Consent:** For any other App, ask for manual approval on a case-by-case basis, label all outgoing data with [no-redisclosure].

This is also a process improvement over the simple OAuth flow in which every case must be approved manually by the patient as the patient in this architecture can grant blanket authorizations to some trusted Apps without manual involvement.

### 1.15 Case-by-Case Approval

When a patient manually approves an App, it often takes the form of clicking ‘Approve’ on a web form. This will be in the form of a web form such as:

**App X, developed by organization Y, wants to access resource type Z, and asking for the following scopes:**
Depending on the design of the App, there may be the option for the user to uncheck some of the checkboxes for some scopes. It is up to the App's designers to decide which scopes are optional and which scopes are required, i.e. if the user does not grant that scope, the App will not be able to function properly, so the user has to decline access to the App altogether. For example, when Alice, the patient, decides to sign up for an immunization App for travel planning, granting access to immunization resources of normal confidentiality (Alice/Immunizations/*/N) is required, because otherwise the App would not have the basic information to function upon. Granting access to (Alice/Immunizations/*/R) could be optional, i.e. Alice can choose to not grant that scope.

Once the patient approves this consent form, most OAuth systems store this in a proprietary format in a database. We propose that this event be captured and stored by as a FHIR consent. The entry will look similar to the following:

- Grant scope [Immunization/*/Normal] to App ID [x1y2z3.z4y3x2] expires on [2018/10/10].

This will be subject to other default obligations set by the patient (e.g. label with ‘no-redisclosure’) and once this Consent is expired, the App will have to be approved again manually.