Solution Overview – Holding Area

Overview

The UniWeb system is a new system for managing CV data for professors in a centralized repository. Some of the data that will be stored in this system is currently being held in other systems. Our goal is to allow users to control the flow of data from those systems into the CV system, to ensure that only the correct data is moved into the system. The primary user objective is to make managing their electronic CV easier, by leveraging existing data.

More specifically, the Holding Area project is designed to meet the following business needs.

- A logged in user should be able to see a list of their own data that has been updated in various systems (see Integration section below) that can be mapped into the UniWeb system.
- A logged in user should be able to approve a change from that list and have it updated in UniWeb.

As this is a new project, it is intended to follow the recently discussed and adopted best practices and standards, in order to ensure that new systems are robust and offer quality to our users.

Integration

What follows is a list of systems that are planned to be integrated with the UniWeb system through the holding area, along with their expected update frequency.

1. Stipends will provide information on courses taught for UGME and PGME (weekly).
2. Mainport – Royal College portal will provide information on learning activities (monthly).
3. Qtips will provide academic work experience information based on appointments, re-appointments and cross-appointments (monthly).
4. Entrada (CPD and BAF) will provide information on courses taught and taken (monthly).
5. Academic promotions will provide information on academic work experience based on promotions (yearly).
6. Faculty department websites (Psychiatry, Surgery, Internal Medicine, and Obstetrics-Gynecology) will provide information on supervising students and event participation (weekly).
7. Physician Annual Review will be integrated into this system (to be defined better).

UniWeb Integration

The team has confirmed that writing to UniWeb does work as documented. However, there remain several challenges around the integration that must be addressed.

Of these, the most notable one is writing an update to a data set. This type of data is basically a list of items. An example of this would be updating a course that is taught or a work experience item. Since there are multiple systems providing similar types of data, and since that data is in flux, it is likely that we will need to handle updating a specific item at some point. The UniWeb API allows this to happen by clearing the entire data set and then rewriting it all. There is no method for targeting a specific item within a data set. This is likely to lead to concurrency conflicts in the future, causing data loss within UniWeb; as a trivial example, imagine two concurrent API calls that want to read the data set, delete the set, and rewriting all the items back with one change. Should the delete from one be called before the read from the other, the second call will have no item to update and may write back a blank set. Data set updating is therefore not thread-safe.

The next challenge is also surrounding data sets, and duplicate data between systems. In at least one case (academic work experience), information is pulled from two different systems (Stipends and Promotions). The ideal system would be able to detect that these are the same items and not create multiple objects for them. However, this poses significant challenges as UniWeb does not provide primary keys for existing data entries nor any deduplication facility of its own.

Solution Overview

Requirements Overview

Any acceptable solution will have to address the following broad requirements:

1. Allow users to be authenticated into the system.
2. Integrate with multiple different services to pull data.
    a. Find the proper user ID for the current user for the given system.
    b. Access a service API to pull certain types of information about that user.
    c. Monitor for changes in those types of information regularly (no more than weekly).
3. Allow users to control what data is pushed to UniWeb.
    a. Browse the list of changed data points.
    b. Accept them to be pushed into UniWeb.
4. Integrate with UniWeb to write data.
    a. Map data pulled from different services into the UniWeb API.
    b. Deduplicate and unify data from different services.
    c. Detect items that already exist within UniWeb, and update them instead of creating a new item.
    d. Write creates or updates to UniWeb in a thread-safe manner.

Open Questions

The following questions should be addressed in scoping the project:

Business

1. How do users login? What authentication source can we use?
    a. Do we need to limit which users have access to this tool?
2. How likely is it that data will need to be changed / de-duplicated?
3. Can users interact with change requests in any other way? For example:
    a. Can they ignore them? If so, should they be able to see what they have ignored?
    b. Can they cancel them if they haven't been processed yet (if there is a delay)?
    c. Can they request we revert them if they have been?
4. How much of a delay between accepting a change request and seeing it in UniWeb is acceptable?
5. How do we de-duplicate data between services and with existing data in UniWeb?
    a. How much control will/should the user have in making these decisions?
    b. How do we handle conflicting field updates?
6. What constitutes a change?
    a. Is it a change in source data? A change in what would be written to UniWeb?

Technical

1. Which integration points require additional services to be built?
2. Does data need to be transformed between the source services and UniWeb (e.g. date formats)?
3. Do we have sufficient information available to associate all of the accounts together?
4. Is the data mapping too complex to do in configuration?
5. Should we be queuing requests to UniWeb?

Proposed Components

RESTful Framework

As will be seen in the other components, new RESTful services will be developed as part of this project. Since there are few good examples of such services already written, it would be best if best practices were outlined for these services before we started. These best practices would address the following concerns:

• Intra-service calls • Code framework • Resource naming • Meta data distribution	• Security (HTTPS only) • Authentication / Authorization • Logging / Auditing	• Deployment • Monitoring • Testing

Identity Provider

In order to map data between services, it is essential to know how user IDs map between systems. In order to resolve this problem permanently, a micro-service should be built that allows a system to map one user ID into another. As this system will have to manage a user set already, providing user authentication and authorization through the same application makes sense. The broad requirements for this system would be:

- Authenticate that a user is who they say they are, and provide a token for clients to continue to use.
- Given a token, verify that the token is still valid.
- Look up an alternative user ID for a given user (or token).
- Manage the alternative user IDs for users.
- Look up whether or not a user (or token) has access to a given system or element.
- Manage which users have access to which systems.

Note that, as we are dealing with semi-sensitive information (user IDs from different systems), we should make sure there is appropriate security in place to prevent unauthorized requests.

Information Exchange

The ideal method for managing information exchange between systems would be to write a configurable tool for doing so. By defining how data flows between systems in configuration instead of code, future additions can be pushed out quickly and seamlessly while also involving a minimum of developer time. Such configuration would have to include:

- Defining a list of RESTful resources that can be used.
    - The system should be able to pull which type of ID is needed, and be able to get that ID from the identity provider.
    - Most support will probably be for a GET of listings and then a GET of each item.
    - UniWeb Write API should be included in here.
- Defining how resources are mapped between systems.
- Defining how resource properties are mapped (and possibly transformed).
- Defining how de-duplication is done (and how any conflicts are handled).
- Defining how change is detected.

As an alternative to the above, configuration could be implemented through code so that it is more flexible (but deployment will be required for changes).

The broad interface for this system would be:

- Triggering a new check for changes on a given service (different services are scheduled differently).
- Browsing a list of changes for a given user.
- Approving a change for a given user.
- Support for other operations on data as needed (see open questions).

In addition, the best way to write to external services is through queuing (to prevent load problems on their end and responsiveness problems on our user's front end). Approving a change should be an asynchronous operation therefore, with a service to trigger pushes.

UniWeb Integration

In order to avoid the problems in UniWeb, the best option is to layer it behind a new RESTful API that hides these concerns from service users. This new API will focus on implementing the resources required for integration first:

- Members
  - Courses Taught
  - Events Participated
  - Academic Work Experience
  - Presentations
  - Student Supervision

Service Integration

For each of the other integrations, it would be ideal to have a RESTful API available so that mapping can be configured. Such a service will have to be created where it does not exist.

User Interface

Finally, a user interface will need to be constructed to allow users to monitor the changes in their data and approve them to be moved into eCV. The broad requirements for this interface will be:

- Ability to browse changes to their personal data.
- Ability to approve moving those changes along the mapping into the UniWeb system.

Work Plan (Tentative)

Please note that estimates in italics are subject to change based on the business use cases and will be refined as the project progresses. It is also assumed that service integrations already exist for the given systems. Physician Annual Review is not included as it lacks specifications at the moment.

Step	Name	Description	Owner	Estimate
1	Create business use cases	Define all the interactions that need to happen. For this project, this should include at least the following: • User authentication and authorization • Browsing of change requests • Manipulating change requests (each action from each valid states) • Reading and detecting changes from each system • Writing to UniWeb after approval of request Estimate: there will be about 25 of these, two hours each.	Edy	50 hours
2	Design micro-services framework	Can happen concurrently to initial planning work. Define how we want to go about creating a micro-services environment for Medtech.	Robert / Jim	50 hours
3	Formalize architecture	Based on business use cases, formalize the architecture of the project in a specifications document for each component. Two hours each.	Robert	*50 hours*
4	Estimation	Based on the architecture and the complexity of the requirements, an estimate will be prepared for how long the entire project will take (refining the ones prepared here).	Robert	6 hours

5	Resource Analysis	Define what the resources will look like. Estimate there will be about twenty of these, at two hours each.	Robert	*40 hours*
6	UI Prototyping	Define what the user interface will look like, using mock services.	Robert	*60 hours*
7	Identity Provider	Analyze, design, build, and test the identity provider service.	Robert	*120 hours*
8	Information Exchange	Analyze, design, build, and test the information exchange service.	Robert	*240 hours*
9	UniWeb Integration	Analyze, design, build, and test the UniWeb integration component.	Robert	*90 hours*
10	UI Integration	Integrate the user interface with the information exchange and identity provider components	Robert	*30 hours*
11	Integrate with MOCOMP	Leverage an existing REST service or provide a micro-service on top of it, and integrate it into the information exchange service.	Robert	*90 hours*
12	Integrate with Stipends	Leverage an existing REST service or provide a micro-service on top of it, and integrate it into the information exchange service.	Robert	*90 hours*
13	Integrate with Qtips	Leverage an existing REST service or provide a micro-service on top of it, and integrate it into the information exchange service.	Robert	*90 hours*
14	Integrate with Entrada (CPD and BAF)	Leverage an existing REST service or provide a micro-service on top of it, and integrate it into the information exchange service.	Robert	*90 hours*
15	Integrate with Academic Promotions	Leverage an existing REST service or provide a micro-service on top of it, and integrate it into the information exchange service.	Robert	*90 hours*
16	Integrate with Faculty department portals	Leverage an existing REST service or provide a micro-service on top of it, and integrate it into the information exchange service.	Robert	*90 hours*
17	Testing & Deployment	This will be iterative throughout development.	Robert	*42 hours*

Based on the above tentative work plan, it will be about seven weeks until we can start development (though this depends on the scope of the business cases as well). This puts us at the end of the year, meaning development can begin in January 2016. Assuming the problem is no more complex than estimated, there are about two full quarters of development. This means we will be finished delivering (except Physician Annual Review) in fall 2016 (September-October), with the first iteration shipping in early spring (May-June). A functional prototype (no real data) would probably be available for review in March. It is important to note that there are still many assumptions to validate and that this timeline could change depending on them. To summarize this approximate timeline:

Target Date	**Deliverable**
Late November 2015	Business use cases, RESTful architecture best practices
Christmas	Specifications, refined estimate
March 2016	Functional prototype demo
Spring 2016	First integration live
Fall 2016	Six integrations live (MOCOMP, Stipends, Qtips, Entrada, Academic Promotions, Faculty portals)

Conclusion

This project provides us with an excellent opportunity to move towards a robust micro-services environment with thin-client front-ends. While adapting the team to those patterns may take a bit longer than hacking it together, the result will be a robust system that can be used as an exemplar for future development while also providing an efficient and useful tool for users to control their CV material.