perMIT and the CMU Billing Use Case

There are several ways that one could satisfy the CMU Billing Use case using perMIT. This document will attempt to outline a couple of different approaches.

Without Rules

Without using rules driven privileges, or what perMIT has been calling Implied Authorizations, it is possible to support most, if not all of the CMU Billing Use Case. The design of the qualifier heirarchies are potentially influenced by the data model of the application. However, the counter view is that data model of the application may also be influenced by the qualfier hierarchies already defined and in use, supporting other applications.

The application that supports viewing of bills would need to do the following, when used by a billing administrator:

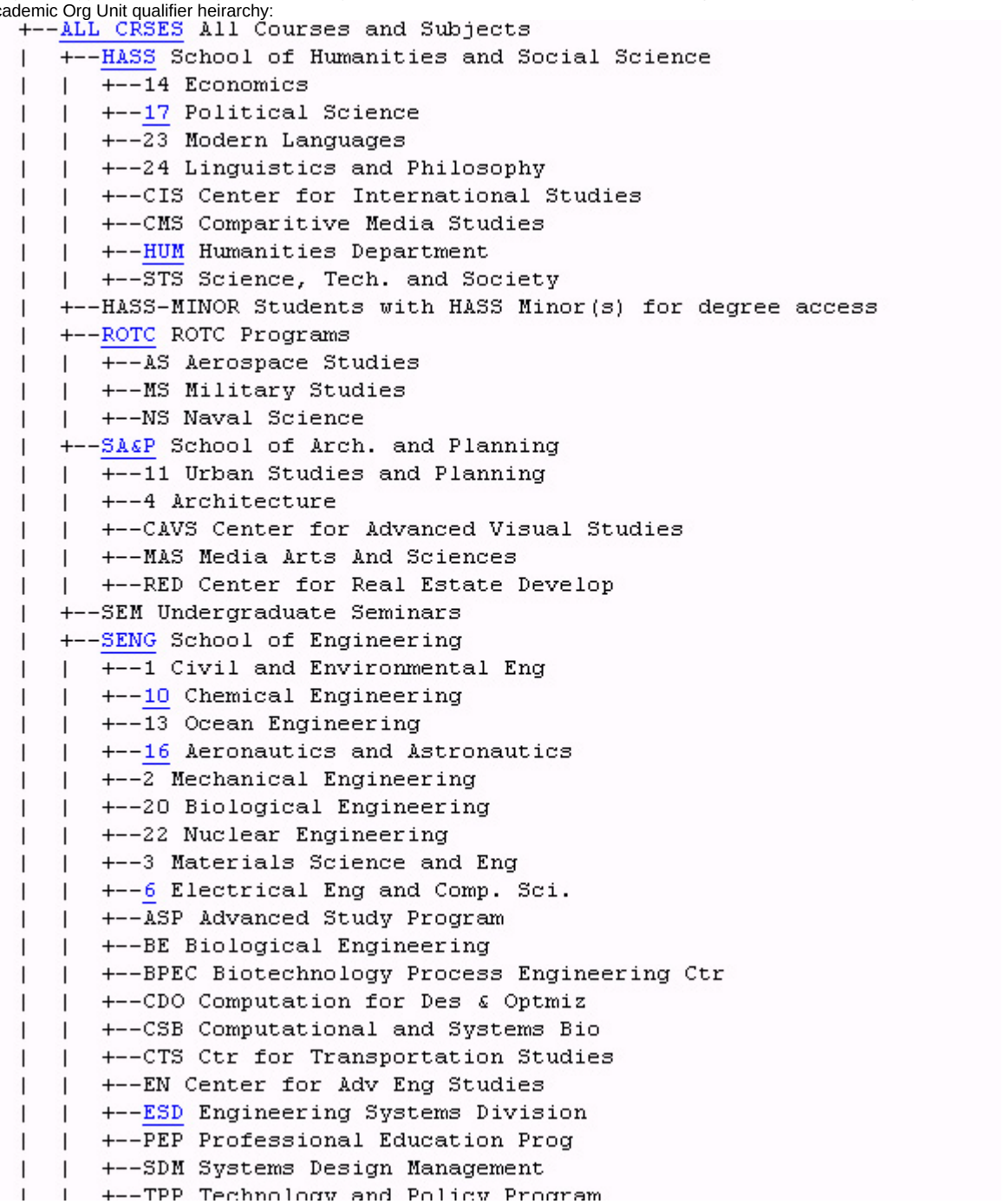
1. Authenticate the user. The authenticated user will be the SUBJECT of the ASPEC.
2. Prompt the user for bill identifier, or a student identifier.
3. Look up the department(s) associated with the specified bill or student. (Depending on the business process, each bill might have a department associated with it. Or alternatively, the student is associated with one or more departments, and each of the student's bill's relationship to a department, is derived from the student's relationship to the deparment.) This result becomes the Qualifier, or scope, that will be applied to the function.
4. For each of these departments, ask the question: Is the user of the application authorized for function VIEW STUDENT BILLS BY DEPT for the specific department derived in step 3? If the answer to the question for any of the pertinent departments is Yes, then let the application user view the bill. Otherwise, deny access

Within perMIT some setup would be required.

First, you would define the following:

- Function name = VIEW STUDENT BILLS BY DEPT
- Qualifier type = Academic org unit

Note that the qualifier type "Academic org unit" does not go down to the student level. The leaves are departments.Here is an example view of the data within the Academic Org Unit qualifier heirarchy:



ASPECs can then be assigned to allow an administrator in a given department to view student bills for any student within that department. The University-billing-adminsitrator(s) would be defined as the people that have been assigned this function with qualifier that exists at the root of the qualifier hierarchy. The University-billing-administrator(s) would also be given the ability to grant this function to others. The University-billing-administrator(s) can then create Local-Billing-Adminsitrators by granting the function to others and specifying the appropriate qualifier in the heirarchy. Local-Billing-Administrators would not be given the ability to grant the function to others.

Example ASPECs:

Subject	Function	Qualifier	Grant?
Parviz	VIEW STUDENT BILLS BY DEPT	All CRSES	Y
Dopirak	VIEW STUDENT BILLS BY DEPT	SENG	N

Presumably, these authorizations would be widely viewable, since names of departmental administrators in various departments are already widely known. Also, in the case of students associated with more than one department, it seems likely that a departmental administrator from one department will at times find it necessary to determine which peers from another department may be reviewing the same bill.

Similarily, we'd also define:

- Function name = VIEW STUDENT BILL DELEGATES BY DEPT
- Qualifier type = Academic org unit

According to the CMU use case only one ASPEC using this function would be defined:

Subject	Function	Qualifier	Grant?
Parviz	VIEW STUDENT BILL DELEGATES BY DEPT	All CRSES	N

Other sites may be more permissive with respect to who can determine who the delegates are.

Next, within perMIT, you would also define:

- Function name = VIEW INDIVIDUAL STUDENT BILL
- Qualifier type = Students/Bills

Presumably these authorizations would be less widely viewable. This could be accomplished by putting this function in a different category than VIEW STUDENT BILLS BY DEPT. perMIT also provides the ability to make the qualifier type sensitive, thus restricting access to view the hierarchy that contains student numbers and bill numbers. In order to meet the use case requirements, the University Billing Administrator could be granted the ability to view this area, but the departmental billing administrators would not.

The Students/Bills hierarchy would start at the root, go down to schools, have departments under each school, students under each department, years under each student, and bill numbers under each year. perMIT does not require qualifiers to fall under a strict hierarchy, and in this case you would take advantage of that flexibility by allowing a student-ID to fall under more than one department in cases where a student has a double major. If bills are department-specific, then you could have objects in the hierarchy that represent student-dept pairs, and individual bills would be attached to student-dept pairs rather than just to student-IDs. At MIT we do something similar with our Environmental Health and Safet Principal Inevestigator/Room-Set (EHS PI/Room-Set) hierarchy in which we have objects that represent PI-department pairs rather than just PI objects.

To look up Authorizations for this Function, an application would specify a target user (who wants to view the bill?) and a bill ID or student ID.

To use these ASPECs the bill viewing application would go through steps similar to those before:

1. Authenticate the user of the application. This user woud become the subject.
2. Prompt the user of the application for a student identifier or bill identifier. This would become the qualfier.
3. Query perMIT, presumably via the web service, if the user can view the bill.

Note that at this point we have talked about populating the ASPECs for the individual students. In other words we haven't talked about creating ASPECs that are self-referrential such as "JOE, (CAN) VIEW INIDIVDUAL STUDENT BILL, (of) JOE". Often applications will contain internal business logic that performs that equivalent, without actually creating the explicit ASPEC within perMIT. Such applications may also have internal business logic which determines that "JOE" can always grant "(CAN) VIEW INIDIVDUAL STUDENT BILL, (of) JOE" to others. If the application was constructed this way, the only ASPECs that contain the function "VIEW INIDIVDUAL STUDENT BILL" within the perMIT database, would be the actual delegations created by the students.

Although such an approach minimizes the number of ASPECs in the database for this application, it also has some potential disadvantages:

- The privilige that the student has on their own bills, would not be listed in the set of privileges associated with the student, when viewed from within perMIT.
- If there is ever a need to deny the student the ability to view his or her own bills, this could not be done from within perMIT. This is because the application business logici is making some internal assumptions and not using perMIT as a policy engine.

Of course, there are several ways that you could generate the self-referrential ASPECs for students. Let's ignore manual data entry.

- You could have the billing system itself generate the individual student ASPECs within perMIT when it generates the bills.
- You could use what perMIT has previously called "implied authorizations". This is a simple rule based approach which is discussed in more detail below.

Using Rules

perMIT also provides a way to use rules, combined with enterprise data to populate the ASPECs. Conceptually this is similar to Grouper's combination of the "systems of record" and the "Grouper Loader". There are several ways that this functionality could be used to meet the CMU Billing Use Case.

It is possible that some sites will have data in a system of record that can be used to detemine exactly who becomes a Local-Billing-Administrator. This would relieve the burden from the University-Billing-Administrator of managing the ASPECs used to control who is a Local-Billing-Administrator. However, let's ignore that for now and instead concentrate on how we would use this feature to populate a self-referrential ASPEC for each student.

Once again, we're going to use the Qualifier type "Student/Bills" which was discussed above. We simply need a rule that take each student, and creates an ASPEC where the student is the Subject, the function is "VIEW INDIVDUAL STUDENT BILL" and selects the node from "Student/Bills" which is identical to the subject.

However, we do this in a slightly non-intuitive manner. First we actually create a different function and qualifier code which is used to capture the business rule. For example, we'll create:

- Function name = CAN VIEW BILLS
- Qualifier type = bill-code

The heavy lifting is actually done by the loader that takes the data from the systems of record and populates a conditional ASPEC that interacts with the aforementioned rule. This first ASPEC will look something like:

Subject	Function	Qualifier
Joe	CAN VIEW BILLS	Bills_8945789023

The number 8945789023 would be some sort of identifying number for Joe - it would be best if the number were not well known. Qualifier codes in hierarchies are all visible, even sensitive ones, but the qualifier names are protected. The qualifier Bills_8945789023 would be a child of one or more academic_department_code qualifiers representing the department(s) where JOEGRAD is enrolled.Next, within perMIT we define a simple transfer rule will be used to populate the individual ASPECs. The rule would like like:

- Rule name: Students can view their own bills
- Condition - Function: IS BILL RECIPIENT
- Condition - Object type: Bill-recipient
- Result - Function: CAN VIEW BILLS

All the rule does is to map the "IS BILL RECIPIENT" function to the "CAN VIEW BILLS" function. However, even though this perMIT-rule doesn't have to do anything complicated, it does document the business rule. In this particular case, this two step process may seem overly complicated, or even byzantine, however, in the case of more complicated rules, this process makes more sense.

Service interface

Any of the privileges can be checked by invoking a simple web service interface.

One of the interfaces takes the following input parameters:

- category
- subject name
- function
- qualifier

The return is TRUE if the subject has been assigned the function and qualifier. The return is FALSE if the subject has not been assigned the function and qualifier.

Note that there also richer interfaces which allow for other input parameters such as a date. Other interface provide the ability to input the category, subject, and function, and get back a list of all of the qualfiers to which the subject will have access.