AuthorisationSystem

DSpace Authorisation System Evolution

Initial text added by RobertTansley – please feel free to butcher as required.

proposal from the Katholieke Universiteit Leuven

I think the real problem with authorisation in DSpace, which makes it hard to implement things like delegated community administration, is that the current system conflates **roles** and **permissions**.

Take the example of a workflow 'metadata editor'. This is manifest as a **role** in the system (the `WORKFLOW_STEP_3` role). However, they also need some associated **permissions**: `ADD` on the relevant collection, as when they commit the item to the archive after their editing, it is placed in the collection properly (as opposed to being in the workspace); and `EDIT` on the submitted item itself, so they can actually edit the metadata.

This makes management difficult. Creating a group and assigning it `WORKFLOW_STEP_3` does not automatically also give them the relevant permissions, which is causing a lot of confusion. The alternative would be to have the code responsible for e.g. committing a workspace item to a collection to understand all the relevant roles in the system to determine who is allowed to do it. That doesn't work either.

So I think we really need to separate permissions and roles; we would define a group of roles, and these would be used to administer e-people. Each role would translate into a number of permissions, which are all the underlying code would need to know about. This should make it much easier for people to create and customise roles without having to mess around with too much code. Also, complicated issues like inheritance, as Ben Bosman explored in his proposal, are simplified.

This would mean you'd just need to administer roles; the necessary permissions would be automatically derived by the system. If we find that a role needs some new permission, we can add that information once, and people don't have to go through adminstrative hoops in their local DSpace instances.

Some More thoughts/ideas around Bosman et al's suggestions

added by JonFerguson

First: nice proposal - good thoughts, well presented!

I would suggest we go a bit further. Part of the issues I've found in managing the system is the confusion between Groups and Roles. Groups should be merely a collection of people and we should be able to give people or groups 'roles' in the system. Roles then get associated with 'actions': people o--o role --o actions (in pseudo UML). At that point we can either manage groups separately or just drop the concept. I think there are benefits to the former (see 'Groups' below).

Secondly, Implicit inheritance certainly does appear the way to go (at least for administration - see 'Inheritance Semantics' below). But I would suggest using a 'composite' pattern rather then explicit inheritance. As far as I can see the only real difference between a 'Community Administrator', a 'Collection Administrator' and a 'General Administrator' is the object they are connected to. I do agree we need one at the local level. We should be able to make use of the actual object structure then to determine usage. That would require authorisation code to ask isCallerInRole("ADMINISTRATOR", object) that code would just find administrators attached to the local object, then climb the tree finding administrator's of parents. RULE: Roles are Hierarchical by object tree.

I think there might be another rule brewing: RULE: He who creates (adds) gets ADMINISTRATION ROLE for added object. If I add a collection to a community I should gain automatic administration rights to the thing I created. By the same token if I add an item to a collection, I get the right to read, write, add (bitstream), remove (bitstream). However I don't really need an extra action called 'delete' to remove the thing I just added. That permission should be on the containing community. It is likely that because I had the right to add a collection to that community I also have the right to remove it.. but even if I don't have That right.. adding a 'delete' action confuses things.

Submitter lists should propagate (see 'Submitter' below). So if you click on the 'Edit Submitters' button I should see any e-people which have been given the 'Submitter' role either at that collection or on its containing Community. However - once I start giving e-people an explicit 'submitter' role on this collection the propagation is removed - this enables things to work when a particular collection has a restricted list (see 'Submitter' below). Would this confuse? Perhaps propagated submitters should be shown in a different colour than added ones and an option to 'copy down' should be given...hmm.

Regardless, note that one of the problems with 'Edit Submitters' (and the same 'submitters:Edit on the collection page) is that these are effectively permission operations and therefore are doing the same as the 'Collection's Authorizations: EDIT' on the collection page. The resultant page shows the Policies for Collection "x".. Which is where we associate Roles, Actions, and Collections (object id), (Only 'Role' is called 'Group' here). As mentioned in the above article there is a Normalisation problem here for workflow - since the steps are mapped to fields in the database table for collection. It doesn't make sense that the Group/Role name should be used to link to the Collection and workflow step (I guess that's discussed elsewhere).

Groups

Managing groups of people is implicit in the concept of 'community': users in that department, team, etc. But the mapping may cross boundaries. For example: there might be a group of 'researchers' that should have READ/ADD access to all the documents under the top level community 'Department'. To manage this the community administrator has created a 'Reader' role that has the following available actions: READ on the Community which implies READ on all the contained Collections. She has also created a 'Submitter' role on each of the contained collections: which gives the capability to ADD items to the collections. The Administrator then associates the 'researchers' group with both these roles. There after every eperson added to that group is automatically given the right to view the Community pages including all sub-communities and collections AND to ADD documents to the contained collections. We don't need groups to do this.. We could just add the appropriate 'roles' to each eperson concerned. This might however be a bit tedious and error prone - especially when permissions change - eg. consider the case when an administrator creates a new collection under the community - all that's required to give add permissions to the 'researchers' group is to add the 'submitter' role associated with that collection to the group rather than all the appropriate eperson's.

Inheritance Semantics

I think this should work differently depending on the particular role in question:

Administrator

Administrator implicit inheritance should work as mentioned the article: Parent administrators must always have control over their children - children do not have control over their parents. The top level administrator needs to be able to read everything for example.

Reader

A Reader is different. It makes sense that it should be possible to keep this role from propagating to particular contained collections. In the 'Groups' example, I suggested that by giving READ on the Community this role would be able to view the all contained communities, collections and items. In fact, we should be able to exclude him from some contained collections. To do this I would suggest that we alter the propagation semantics in this type of case: When a contained collection or community publishes its own 'Reader' role permissions stop propagating at that point. Thus if top level community: 'Organisation' has a contained community 'Engineering' with a collection 'Design Docs' (Organisation --> Engineering --> design docs). Anyone who has the role of Reader will be able to read everything. However, by adding the Reader role to 'Design Docs' only users who have the role of Reader on Design Docs would be able to view Design Docs at all. This is different to the way the Administrator role should propagate since adding the Administrator role to 'Design Docs' would add more administrators to the list - eg. the Administrator role's propagation is Additive while the Reader role is Exclusive.

Submitter

A Submitter is an explicit concept to collections - or is it? Shouldn't there be the possibility of making members of a Community Submitters to every collection in that Community? That could then propagate just like the Reader role. The thing that confused me was that I assumed this should mean having ADD priviledges at the Community Level - which is not right since a Submitter should not be able to ADD Collections and other Communities - just Items. I guess this could be resolved by changing the action ADD to SUBMIT on Collections. Thus although you can have the Submitter role for a Community it only allows you to submit to contained collections. Role Submitter --> associtated with ACTION SUBMIT. But this won't let you add to a Community because Communities don't have the SUBMIT action.

Work flow step Editor

Works a bit like Submitter. But here we're not just concerned about the collection its on but the step number as well. (note I'm not looking at submission as part of this work flow.. but rather just the event that kicks it off). I haven't read the WF proposal so just want to say that this permission should be on the WF step that is connected to the collection not the collection per say.. even though permissions can be inherited.