Further thinking on Collections, Sets & Lists

Discussing the strong basis in User Collections, Admin Sets, Display Sets with proposed revisions.

Stanford would use the term "User List" instead of "Collection". This communicates the context of its generation, its primary audience and the requirement for ordering. I apply that terminology here.

The differences between the three models are:

- **User List**: optional, ordered, user generated, recursively composed, rights independent.
- **Admin**: unordered set, mandatory: each work belongs to exactly one Admin Set. Affects rights and description.
- **Display**: basically the same as User List, plus some hierarchical facet magic.

In terms of data modeling, there is one most important takeaway: only the Admin Set affects the work data itself. It imposes a requirement on the Work model, provides default rights and descriptive metadata. Critically, at least one Admin Set must exist before adding any work! The Admin Set resembles our existing management practice.

I would direct attention to two points where the model introduces problematic complexity:

1. **User Lists** exposing recursion to end users, and
2. the **Display** concept of hierarchy.

I will attempt to dissuade you from each.

**Reasons not to allow User Lists that reference other User Lists:**

- Logic: once two sets include each other they are mathematically the same set. That is to say, by referencing any other user’s List, you are giving them the ability to change what is in your List, published under your account.
- Computation: how many queries and subqueries does it take to enumerate the IDs of works in a given User List? **Unknown!** Is the result cacheable? **Never!** Recall that we (quite sensibly) do not want User Lists to affect the index or the repository data, which blocks several strategies for mitigating the cost of recursive dereferencing.
- User may wish to include the **works** of another’s List, but in a different order. Therefore they must be dealing with their own ordered list rather the sublist.
- User may wish to include the **works** of another’s List, but with exclusions. A recursive model would be responsible for representing an additional "exclusion" model, and solving the problem of recursing exclusions.
- User may wish to include the **works** of another’s List, but with exclusions **that don’t exist yet**. I.E., I want to include another’s existing list **without** exposing my list to updates in the sublist.
- User may wish to include the **works** of another’s List, prepares a class presentation based on them and subsequently finds all such works removed!

In general, the effects of recursive references to end users will be bewildering and unanticipated. Effectively the same recursive composition but with intuitive user-mediated flow would be to allow the users to be notified of additions to sets they have interest in. This is no more than an RSS transform of a query, as implemented in most discovery systems already. Example notification: "2 new works added to @atz’s list ‘Research’" with a View button and an “Add to List” button. In such a case, each User List is just flat IDs, no computational risk or unexpected behavior.

Notably, because of the single-membership and the fact you can discern the Admin Set from the actual works data, if you want to make a User List reference Admin Sets, there is no recursion.

I will not spend long on **Display Lists** (not Sets, because ordered), only because I think of them as a special class of User List and I think we can safely declare that a 12-level faceted hierarchy is out of scope. The example Biology Dept. > Master’s Theses > Year problem is, by my view, unrelated to special set membership and the discussion is probably misapplied to modeling of sets. Are you saying that **nowhere** in metadata can you identify that a work was a thesis, the departmental affiliation or datePublished? Ideally, those are 3 independent facets built on existing data. Certainly, in no case should datePublished be specific to any kind of work!

**Recommendations:**

- Support two main kinds of groups: Admin Sets and User Lists.
- Lists should **not** copy Lists, not reference Lists.
- Determine if User Lists referencing Admin Sets is really desirable and necessary for this phase.