Hierarchy Service

**Proof Of Concept**
- This is a functional proof of concept, please feel free to try it out and also post comments, suggestions, etc.

**Information**
The Hierarchy Service supports general storage of hierarchical nodes within Sakai. It can support multiple hierarchies and has a flexible interface which is meant to be generalized and easy to use. The current implementation is optimized to support many reads of many nodes. Use of this would generally require the developer to write their own service which uses this and the Permissions Token Generator.

**Hierarchy Service**

- **Goals**
  - Generalized storage of hierarchical data in Sakai
  - Flexible and easy to use interface
  - Single point of optimization for hierarchical service
  - Supports high performance permissions handling in conjunction with the Permissions Token Generator
  - Ability to create many hierarchies
  - Support for many children per node
  - Support for multiple parents per node

- **Proof of concept code**
  - Working code here: https://source.sakaiproject.org/contrib/caret/hierarchy/trunk/
  - Example of usage (evaluation): https://source.sakaiproject.org/contrib/evaluation/trunk/
  - Continuum build: http://saffron.caret.cam.ac.uk/continuum/servlet/continuum/target/View.vm/fid/shellProject/id/116

- **Usage of the hierarchy**
  - Take a look at the sample in evaluation ExternalHierarchyLogicImp
  - A developer would create a service class which manages the hierarchy for their class or service
    - Each hierarchy would have a single root node which can only have children
  - This service class would use the HierarchyService to create a hierarchy and manage the nodes within it
    - HierarchyService is built from a set of interfaces (this is part of the extendability of the API)
    - HierarchyNodeReader
    - HierarchyNodeWriter
    - HierarchyTokens
  - There is a utility class to make it easier to deal with the hierarchy nodes
    - HierarchyUtils
  - The data associated with each node would be stored within the service (in a database table probably)
    - This allows the developer to focus on what they want to associate with nodes rather than worrying about how they can make them perform well

- **Current Implementation details**
  - Uses a hibernate backend to store the hierarchy data in a database table
  - Provides a guaranteed single database call to get a node and information about all associated nodes
  - Ensures the integrity of the hierarchical tree using transactions to create, move, delete nodes
  - Splits nodes and node meta data to ensure cheap updates of meta data without locking

**Notes**
- Chuck has a doc about what he has done with site hierarchies
  https://source.sakaiproject.org/svn//reference/trunk/docs/architecture/sakai_pseudo_hierarchy.doc
- Matthew Buckett at Oxford is doing something with hierarchies related to sites
  Oxford Portal Hierarchy

**Use Cases / Requirements**

- **Course Management - Josh Holtzman**
  1. The ability to query for sections down an institutional hierarchy.
     This might look like List<Section> getChildSections(String courseSetEid). Translating to hierarchy speak, it's getLeafNodes(Node). I'm pretty sure this simple use case will be covered
  2. The ability to query for memberships up a hierarchy. This might look like List<Membership> getMembershipsAboveSection(String userEid, String sectionEid), returning all memberships for any course offering or course set (node) above this section (leaf node).
     This is complicated by the fact that a child node can have multiple parents (e.g. a course can be in multiple departments). We may need to model this where a course has a node in multiple places in the CM
hierarchy... I'm not sure if that's possible, or if that negates any performance improvements the hierarchy service is attempting to provide.

- Checklist - Dick Elis
  1. Simple hierarchies unlimited in number and depth but unlikely to be deep - e.g., Grad Tools checklists consist of two hierarchies of three levels (institutional requirements for most graduate students or for music performance graduate students, which are inherited by associated departments and to which departmental requirements are added, and which in turn are inherited by their students, to which faculty committee and the student may add personal as opposed to institutional steps) - Masters checklists would consist of 100 hierarchies of 2 levels (the department and students under the department), personal or project checklists might consist of 1000 hierarchies of 1 level.
  2. Cascading CRUD - administrators may make retroactive changes to checklists that should be carried over to checklists that inherit from this checklist - interactive batch processing where speed is an issue, as is connection time out
  3. Lock a section of the hierarchy for cascading CRUD
  4. View Progress - At any level above, show all leaf nodes and individual checklists below showing completion progress
  5. allowUpdateChildren, allowBasedOnChecklist, getChildChecklists, getSharedChecklists, hasChildChecklist, getTree (String root)
  6. Attached screen shots show how the root (blank checklist), branch (based on checklist) and leaf (use checklist) are created, and sharing of a checklist by owner