Entity Objects as Used in A-Board

# Introduction

In order to best decide how to realign Alignment Board with MongoDB Documents, it is necessary to know exactly how the Entity API is used.

# Class Breakdown

**RenderableBean**: Entity for type name. Uses AlignedItem which is an EntityWrapper

**AlignedItem**: Depends on EntityWrapper (is-a), sets value on entities/EntityConstants.ATTRIBUTE\_COLOR. Rendering characteristics based on EntityConstants.ATTRIBUTE\_RENDER\_METHOD.

**EntityWrapper**: Defines child/parent relationships. Uses Entity for id, type, role, owner, creation date.

**RenderableBeanCollection**: Unit Test that builds RenderableBeans.

**ConfigurableColorMapping**: Must collect entity to get/set its color value, which may be based on its Entity Type.

**RenderablesLoadWorker**: checks RenderableBean’s type to see if it is neuron fragment, for purposes of deciding if it is in or out of display.

**ABContextDataSource**: Looks at EntityWrapper classes Sample, Neuron, VolumeImage, CompartmentSet, Compartment, make MaskChanRenderableData.

**UserSettingSerializer**: uses AlignmentBoard Entity to anchor saved settings.

**LayersPanel**: uses a common root/ALIGNMENT\_BOARDS\_FOLDER folder. Makes rooted entity on this. Looks up the alignment board, using this folder (it should be a child). It makes an Alignment Board Context (EntityWrapper), to get the ancestors of the Alignment Board Context(??). Uses AlignmentBoardContext also to get the “compartment set” child. If that does not exist, it will add one of appropriate characteristics. Now launches the worker. LP uses aligned items, to find things by EntityID. Using ID-traversal. Uses the find-item just mentioned, to invalidate entities, in response to subscribed events. If stuff was invalidated, it recreates its model. This is an external-event response. Uses entity names for building a table (for table cell renderer component). Doing a lot with Entity data in its row model.

**AlignmentBoardCreator**: has a RootedEntity as part of its state. Uses entity type name for compatibility check of incoming entity. Calls EntityWrapperFactory to make the domain objects. Moves up tree looking for NeuronSeparatorPipelineResult (uses these for alignment space and resolutions). Examines AlignmentContext objects, looking for a compatible one. Calls Model Manager to create an alignment board (back through all layers). It finally launches the board by its Entity ID.

**AlignmentBoardPopulator**: This is a Drag-N-Drop target, and a NetBeans ServiceProvider. It uses AlignmentBoardContext, which is-a AlignedItem, which is-a EntityWrapper. Therefore, the AlignmentBoardContext is a candidate domain object. The drop call will hand in a list of entities-to-add. These are added as RootedEntity, to the AlignmentBoardContext.

**ABTargetedSearchDialog**: Constructed with AlignmentBoardContext. Calls SearchWorker with AlignmentBoardContext.

**SearchWorker**: Filters results based on EntityConstants types Sample and Neuron Fragment. Treats Sample and NeuronFragment results differently.

**SampleTreeModel**: Constructed with AlignmentBoardContext. Posts an alignment board change event for updating tree. Works with EntityWrapper and AlignedItem. Works at the EntityWrapper level.

**FragmentSizeSetterAndFilter**: Looks for an aligned item with an entity id. Works at level of EntityWrapper and AlignedItem.

**VolumeWritebackHandler**: writes a metadata file for volumes (separate text). Uses EntityWrapper objects to do so.

# Summary of Use Cases for Entity Above

Here are the ways I see Entity and its satellites being used:

* EntityWrapper, which is really a generalized domain object. At this level, it’s just a holder for entity’s data, which can live locally. EntityWrapper is extended by all the AlignedItem classes, which are specific to AlignmentBoard. This, and AlignedItem are the “existing domain objects”. Much of this code can remain as-is, or at most just will require a re-pointing to a different class name, to avoid confusion with Entity objects.
* Using an entity for its type, ID or name. These *might* be gleaned from target objects.
* Using the entity as an anchor point to save EntityData objects. These values should now be stored in the domain model objects.
* Traversal of Entity/EntityData based parent-child hierarchies (as in LayersPanel). This could be the most complex case to deal with. It will partly depend on how things like Sample, and NeuronSeparatorPipelineResult have been handled in the current model.
* Using entity type name to distinguish among all the many ‘flexibility-inspired’ alternatives. These used to all be Entity objects, so the choice could be made late in flight. Examples include selecting things off of menus (and setting up menus based on Entity type as context). I need to consult with the experts on how this has been changed. See AlignmentBoardPopulator and AlignmentBoardCreator.

# Alignment Board Creator

This is how it works, and what it does.

It has a SimpleWorker with a doStuff and a hadSuccess method. The first of these (background) is trying to establish whether an alignment context has been chosen by the user, and if so, make it available to the hadSuccess method.

Within hadSuccess, if there was nothing chosen, the user is shown a message popup to accept the right one. Next, accept a name for the alignment board, and fire up another SimpleWorker.

This one has a call to ModelMgr.createAlignmentBoard. That produces a RootedEntity (more on ModelMgr below). It wraps an Alignment*Board*Context around this new entity (more on AlignmentBoardContext vs AlignmentContext, later). It will add an ‘entity stack’, if provided, to this context. Neurons, Compartments, and Reference Channels, are never added directly to a board (or context/whatever). Instead, they must be placed under a sample. In this way, a whole sample may be added directly, or just a sample member may be added—specially—so that any required sample parent will be added at the same time, but without some or all of the siblings of the things being explicitly added.

## Alignment Context vs Alignment Board Context

The Alignment Context is a concept applying not only to Alignment Boards, but Samples as well. It has characteristics of the alignment. They include, at time-of-writing, only the Pixel Resolution, Optical Resolution, and a name.

The Alignment Board Context (ABC) is specific to an alignment board. Alignment Boards are loaded into being, from the LayersPanel class, which constructs Alignment Board Contexts, and exposes one through a getter. This is built around the Alignment Board RootedEntity. The ABC can be passed around in an event, on an event bus, to be received by the Alignment Board GUI itself. It is used as a feed for a Data Source, because it can get aligned items relevant to it. It is used in the ABTargeted Search Dialog, because it also has the Alignment Context. It is the root object of the SampleTreeModel, because the alignments are its children. In short, the AlignmentBoardContext is an AlignedItem, with an AlignmentContext.

## ModelMgr and AlignmentBoard

The Model Manager is a general class for fetching data back from the database, for the client. In the older design, it was the only means of doing so. Let’s look at what this class is doing for Alignment Board in the old design.

**createAlignmentBoard** – This call delegates ultimately to the AnnotationDAO, and takes all components of the AlignmentContext, plus ownership. The implementation is below. But it creates the entity, adds the space-name and two resolutions as attributes; it creates the common “owner/all boards” root, and once it is known to exist, it adds the new board to this root.

public Entity createAlignmentBoard(String subjectKey, String alignmentBoardName, String alignmentSpace, String opticalRes, String pixelRes) throws DaoException {  
 if (log.isTraceEnabled()) {  
 log.trace("createAlignmentBoard(subjectKey="+subjectKey+", alignmentBoardName="+alignmentBoardName+", alignmentSpace="+alignmentSpace+", opticalRes="+opticalRes+", pixelRes="+pixelRes+")");   
 }  
   
 Entity board = newEntity(EntityConstants.*TYPE\_ALIGNMENT\_BOARD*, alignmentBoardName, subjectKey);  
 board.setValueByAttributeName(EntityConstants.*ATTRIBUTE\_ALIGNMENT\_SPACE*, alignmentSpace);  
 board.setValueByAttributeName(EntityConstants.*ATTRIBUTE\_OPTICAL\_RESOLUTION*, opticalRes);  
 board.setValueByAttributeName(EntityConstants.*ATTRIBUTE\_PIXEL\_RESOLUTION*, pixelRes);  
 saveOrUpdate(board);  
  
 Entity alignmentBoardFolder = getCommonRootFolderByName(subjectKey, EntityConstants.*NAME\_ALIGNMENT\_BOARDS*, true);  
 if (alignmentBoardFolder.getValueByAttributeName(EntityConstants.*ATTRIBUTE\_IS\_PROTECTED*)==null) {  
 EntityUtils.*addAttributeAsTag*(alignmentBoardFolder, EntityConstants.*ATTRIBUTE\_IS\_PROTECTED*);  
 saveOrUpdate(alignmentBoardFolder);  
 }  
   
 addEntityToParent(alignmentBoardFolder, board, alignmentBoardFolder.getMaxOrderIndex()+1, EntityConstants.*ATTRIBUTE\_ENTITY*);  
   
 return board;  
}

**addAlignedItem** – This call adds things to the alignment board context. That code is below, but it creates an entity for the item, and it sets the visibility to the (default) true value, and adds that back to the parent alignment board. Said TYPE\_ALIGNED\_ITEM entities are what is used to control coloring and visibility in the board.

public EntityData addAlignedItem(Entity parentEntity, Entity child, String alignedItemName, boolean visible) throws DaoException {  
 if (log.isTraceEnabled()) {  
 log.trace("addAlignedItem(parentEntity="+parentEntity+", child="+child+", alignedItemName="+alignedItemName+", visible="+visible+")");  
 }  
   
 Entity alignedItemEntity = newEntity(EntityConstants.*TYPE\_ALIGNED\_ITEM*, alignedItemName, parentEntity.getOwnerKey());  
 alignedItemEntity.setValueByAttributeName(EntityConstants.*ATTRIBUTE\_VISIBILITY*, new Boolean(visible).toString());  
 saveOrUpdate(alignedItemEntity);  
   
 addEntityToParent(alignedItemEntity, child, alignedItemEntity.getMaxOrderIndex()+1, EntityConstants.*ATTRIBUTE\_ENTITY*);  
   
 return addEntityToParent(parentEntity, alignedItemEntity, parentEntity.getMaxOrderIndex()+1, EntityConstants.*ATTRIBUTE\_ITEM*);  
}

## Role of LayersPanel

The LayersPanel is actually a tree on the LHS of the board, and it is part of the board’s panel group. This one is where the user will do things like change the color and visibility.

LayersPanel’s role is to open an alignment board. It is actually rather arbitrary whether this or the main view is used, but the Launcher is calling LP, to open the board. LP will know the EntityID, and use that to lazy-load the entity. It then wraps that in an AlignmentBoardContext, calls a recursive “loadAncestors” (probably a misnomer), and ensures that a CompartmentSet is available. The TreeModel is inflated from the gleaned information.

LayerPanel is what ‘loads contextualized children’ for the whole hierarchy of the alignment board. It does so by wrapping entities with old-style domain objects (EntityWrappers), of various suitable kinds.

# Challenges

How to solve these? I need to find out about how EntityWrapper is treated. I need info on what is going on with the hierarchy of objects being added to the board (Sample and Neuron Fragment, along with the Compartments, and the Reference Channels). I need to know about the original containers of Neuron Fragment and Sample (Neuron Separator Pipeline Result). I need to see how KR and David have stored all these objects in the MongoDB database.