JCR-agnostic content apps

This concept is a work-in-progress specification for decoupling content apps from JCR, due to land in Magnolia 5.3. In particular, it features:

- Highlights on the various spots and components that are currently tied to JCR or to our adapters interfaces
- A list of general issues with extending the behavior and appearance of content apps
- Proposals and patterns for overcoming such issues.

Concept work will be tracked by the following Epic:

MGNLUI-2655 - Getting issue details... STATUS

1. Problems

Displaying non JCR content in the workbench

- It is impossible to **plug an external data source** into the workbench
  - One can perhaps extend our default content presenters
    - *TreePresenter*, *ListPresenter*, *ThumbnailPresenter*, *SearchPresenter*
  - The container implementations are hard-coded in there
    - resp. *HierarchicalJcrContainer*, *FlatJcrContainer*, *ThumbnailContainer*, *SearchJcrContainer*
  - Then one needs to extend *all* of these desired presenters to hook in external content
  - And finally configure them as *presenterClass* on *contentViews*, or using custom *ContentPresenterDefinitions*
  - And yet the Workbench and Browser sub-app will complain/crash because custom container items are not representing JCR items.
- Nor is it possible to **filter JCR content** easily
  - Similarly as above, to implement a simple filtering mechanism for a JCR workspace, one must still extend and reconfigure *all* content presenters.
    - e.g. security-app, forum moderation
- Workbench and Browser APIs are inherently tied to JCR
  - Event the view implementations are depending on JCR/JcrAdapters (e.g. ***TreeViewImpl***)
  - WorkbenchDefinition may be reduced to the raw
- How about **Location and path** in the status bar when dealing with non-JCR items?
  - One needs pluggable mechanisms to map these representations together:
    - content as Vaadin Item
    - content key (itemId) as provided by Vaadin Container
    - content key as a human-readable variant of the itemId (location handling, selection info)

Extending the workbench

- It is almost impossible to replace/**extend small parts** of the workbench
  - e.g. hooking a custom WorkbenchPresenter or WorkbenchView
  - this requires to start from the very top with a custom BrowserSubApp class
  - then get a custom BrowserPresenter injected, which itself would eventually get the custom Workbench injected

Content-app architecture

- ***String*** as one and only ***Vaadin item-id***
- Inconsistent listener/event-handler interfaces that sometimes transport item ids (Strings, not Objects), sometimes - Vaadin items or JCR items.
- On all the layers of content app the JCR data is accessed via utils and/or Vaadin JCR adapters.

2. Proposals

1. Propagate Vaadin selection transparently up till action execution
   - as a Vaadin Item or a Set of Vaadin Items
     - so that executor doesn't care about data source
2. Map Vaadin Items to ItemIds to Human-readable location at Browser level
3. Split *WorkbenchDefinition* from *JcrWorkbenchDefinition*
   - Configure a *containerClass* in code - can be overridden in config
   - Have a *JcrBrowserSubAppDescriptor* that overrides *#getWorkbench()* with JCR specific workbench definition type
   - see Appendix diagram (Workbench/JCR split)
4. Unify all JCR Containers at workbench level
   - They represent the same data set
   - JCR is by nature Hierarchical, Indexed and Ordered
   - Advanced behaviors can surely be delegated to smaller units (search, filters)
   - Thumbnail resolution is not among container duties
5. Research how a *ContentPresenter* can start with just *ContentPresenterDefinition* and *Container* to populate itself
   - avoid to carry *WorkbenchDefinition* along everywhere
   - in a similar fashion as *DetailPresenter#start(EditorDefinition, JcrNodeAdapter, ViewType)*
   - Restructure ContentPresenter and Workbench definitions
     - *editable* and *dragDropHandler* belong to content views - they do not necessarily apply to all
     - *columns* should belong to a *ListPresenterDefinition* interface - they do not apply to thumbnails or any arbitrary other content view
6. Expose preconfigured *presenterClass* for browser / workbench / actionbar
   - so that customization doesn't always have to start at subApp level
7. Filters
   - dynamic filters through the UI, e.g. asset filtering (images, documents, videos)
   - permanent filters by configuration or in code (e.g. forum moderation)
   - define *facets*
     - at workbench level - concretely setting container properties
     - ≠ list view's *ColumnDefinitions*; they would be based on a facet, but describe only visual appearance (visible columns, headers, sizes)
     - a facet describes whether it is: *sortable*, *filterable* , maybe even which UI should be used to filter it (e.g. ranges, text, enums)
   - filters apply to *any* content view

3. Proposed milestones

ℹ We may first reapply the Action bar MVP and dependency refactoring that was meant for 5.1, as a sanity rebase.
ℹ We don't affect JCR adapters

Level 1: Make selection JCR agnostic

1. MGNLUI-2657 - Getting issue details... STATUS

2. generalize JcrAdapters into Vaadin Items
3. generalize String ids into Object itemIds
4. Handle multiple selection nicely (e.g. avoid converting sets into lists, items to itemIds)
5. try not to use Items when itemId is sufficient (events)
6. No big API breaking

Level 2: Redefine Browser / Workbench apis

- MGNLUI-2569 - Getting issue details... STATUS
- Define responsibilities of components
    - from BrowserSubApp down to WorkbenchPresenter
    - we leave ContentPresenters as much unchanged as possible (maybe just passing current containers from above)
- We try to reduce WorkbenchDefinition's omnipresence
    - in particular from Browser, Container; also from ContentPresenters (although unlikely at this step)
- first concrete implementation is JCR
- also focus on a different implementation (FileSystem, Bean) to uncover potential problems
- include proposal #6 (e.g. exposing Workbench presenterClass)
- **Subtasks**
    - Untie JCR containers from WorkbenchDefinition - so that they (or at least AbstractJcrContainer) can be moved to vaadin-integration module
    - Get WorkbenchPresenter from componentProvider, not through @Inject (different presenterClasses per WorkbenchDefinition)
    - Split WorkbenchDefinition from JcrWorkbenchDefinition, WorkbenchPresenter from JcrWorkbenchPresenter

WorkbenchDefinition.java

```
// isn't i18n useless on this definition?
@I18nable
public interface WorkbenchDefinition {
    String getName();

    // JCR dependent; should move to JcrWorkbenchDefinition
    String getWorkspace();
    String getPath();
    List<NodeTypeDefinition> getNodeTypes();
    boolean isIncludeProperties();
    boolean isIncludeSystemNodes();

    // Should be removed or moved (not in level 2)
    boolean isDialogWorkbench();                                // doesn't belong to config
    String getDefaultOrder();                                   // belongs to
ListPresenterDefinition
    boolean isEditable();                                       // belongs (currently) only to
TreePresenterDefinition
    Class<? extends DropConstraint> getDropConstraintClass();  // belongs (currently) only to
TreePresenterDefinition

    List<ContentPresenterDefinition> getContentViews();

    // New config options
    Class<? extends Container> getContainerClass();            // preset in code for JCR if/when
unified container is around
    Class<? extends WorkbenchPresenter> getPresenterClass();   // preset in code for JCR
}
```

Level 3: Dialogs and DetailSubApps

- MGNLUI-2656 - Getting issue details... STATUS
- DetailSubApps as well as Dialogs depend on JCR but not tightly - once the previous steps are completed it should be possible to completely get rid of JCR dependency without even extracting the child classes.
- The similar data management tasks that we have in `BrowserSubApp` occur in `DetailSubApp:` we need to serialize/de-serialize item ids to fragments, query items by item id etc.
    - A reasonable solution would be to share a capable component between sub-apps.

- As long as content apps work over same domain - we could introduce the `DataSourceManager` component shared between sub-apps. It would encapsulate the solution of most of Level1/Level3 problems and could be useful for all other components that would need to connect to data of the app.
- DataSourceManager initial interface could look as follows for starters.

DataSourceManager interface

```
public interface DataSourceManager {
    String serializeItemId(Object itemId);
    Object deserializeItemId(String strPath);
    Item getItemById(Object itemId);
    Object getItemId(Item item);
    boolean itemExists(Object itemId);
}
```

Level 4: Redefine ContentPresenter apis

- MGNLUI-2658 - Getting issue details... STATUS
- make content views less workbench and JCR dependent
  - container properties / operations may belong to workbench
  - VS. list only requires visible columns, width...
  - Ideally: `ContentView start(ContentPresenterDefinition definition, Container containerDataSource);`
- one should think about filters upfront
  - think about facets at workbench level
- **Subtasks**
  - Use an action for inplace-editing
  - Get rid of listener mechanism for getIcon - pass such configuration on presenter startup

Independent enhancements

- Workbench filters (dynamic / permanent)
  - TODO Story in Ideas
- Single JCR Container
  - BL-153 - Getting issue details... STATUS
- Support for federated workbench / actions / search

4. Connected topics

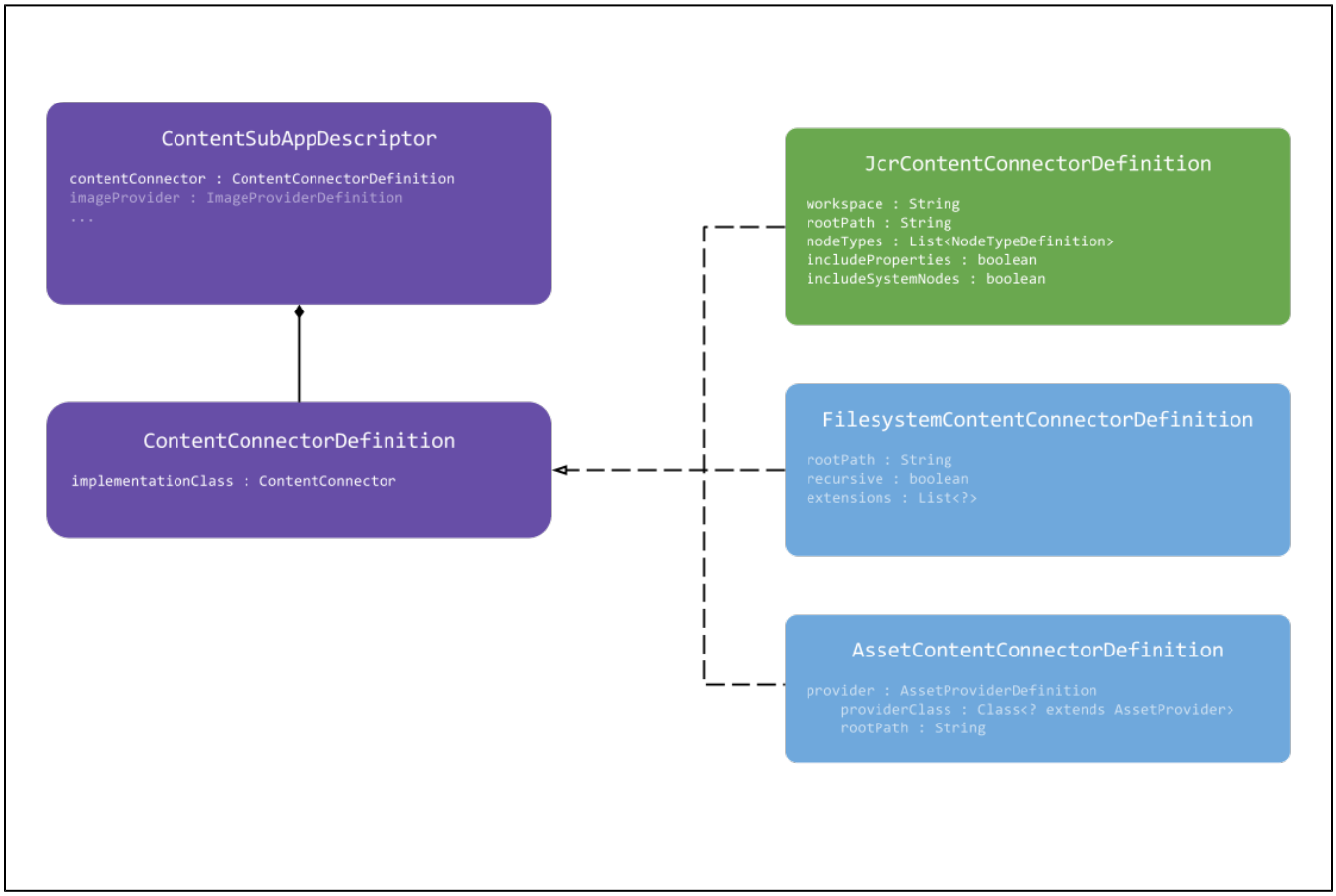
- **Actionbar dependency and MVP**
  - common-widgets shouldn't depend on ui-api
  - Actionbar currently implements the View interface (ui-api) instead of having the ActionbarPresenter produce a ViewImpl
  - Revived refactoring that was postponed for 5.1, in favor of the *i18n* merge
  - MGNLUI-1991 - Getting issue details... STATUS
  - Make sure content views no longer getIcon via listener
- **Multiple actions** deserve a review in the process
  - In particular, we should guarantee distinction between *null* and *root* selection.
  - Implementation needs review
- **Availability & Permissions - independent**
  - Availability is too complex and lacks permission-based assessments
  - ActionExecutor is not the best choice for evaluating action availability
  - See Permissions for UI availability

Appendix

JCR all over the place

- **ActionExecutor**
  - availability depends on JCR Item
- **Actions**
  - OpenCreateDialogAction creates JcrNewNodeAdapter
  - OpenEditDialogAction, SaveDialogAction all deal with JcrAdapter
  - SaveDialogAction relies on adapter applying its own changes
    - Requires a generic hook for saving?
    - JcrSaveAction implicitly configured for all forms?
  - Actions on items from external data sources require their own actions
    - default set of actions with a different implementation **catalogs**?
    - configure save action on openDialog/Form action
- **Adapters**
  - should not perform JCR operations
  - should be stupid **readonly** representations of JCR Item as Vaadin Item
- **Transformers**
  - BasicTransformer (root impl of all) relies on DefaultPropertyUtil, that is implicitely on JCR propertyTypes
- **Containers**
  - As is, all containers should be replaced for all content views
  - Consider getting workbench unified container onboard so that users don't need to configure *n* containers
    - Or chances are that they would be smarter than us and use a single one, configured in multiple locations
- **Components / Presenters**
  - Workbench / BrowserSubApp / ContentPresenters / Events

SubApp / JCR split



Prototyping sessions

See the child page referenced below for the details of prototyping/research made in order to estimate/clarify the effort of making content apps JCR agnostic.

Meeting notes 2014.03.04

✅ Generally accepted:

- ContentConnector gets green light
- Events, JcrItemId get green light
- ContentPresenters are OK with minimal changes
- ImageProvider is OK, the interface should be reviewed and cleared from the unnecessary methods.

⛔ **Action availability is critical**

- No definition should perform business logic
- We need to be very careful about using Voters
- We need to guarantee *permissions* are generally set the same way for apps, app groups, actions, etc.
    - We may well differentiate permission availability from UI state based availability

⚠️ Require consolidation

- DetailSubApp needs more understanding
- Actions with specific set of parameters
- Transformers, Fields
    - we should guarantee detail view works for basic fields (text, checkbox)
- we may try plugin in a SQL database although practically it would be very similar to the filesystem app use case.