Concept - Actions API and configuration

Abstract

There has been some work planned on actions and their configuration for quite a while, but since API stabilization is targeted for alpha3, this is the time to do something about it. The goal is to finalize how actions are configured and accessed from the UI, and what roles and responsibilities they have.

Status

Sections **1**, **2** and **5** are ready for implementation

Sections **3** and **4** need more discussion

Section **6** is currently postponed

Section **7** takes no further action

1. Execution of actions

Problem

- Responsibility for executing actions is spread between workbench action factory and `ActionbarPresenter` (also depicted in appendix A):
  - The `ActionbarPresenter` listens to action "clicks" from the `ActionbarView`
  - It retrieves the corresponding `ActionDefinition` and fires an `ActionbarItemClickedEvent` to the subapp event bus
  - This event is processed e.g. by the `WorkbenchPresenter` which adds params such as workspace and selected item, and calls the `ActionbarPresenter` again to create and execute the action
    - The `ActionbarPresenter` calls its `(Workbench)ActionFactory` for creating the `Action` instance
      - The abstract `FactoryBase` resolves the implementation class and creates an instance of it, with the given parameters
    - The `ActionbarPresenter` performs the magic call "`action.execute();`"
- As a side note, there is even message handling upon execution, in `ActionbarPresenter`

Decision

- We no longer execute actions in `ActionbarPresenter`
- We drop the `ActionFactory` and `FactoryBase` abstraction
- We replace these with an `ActionExecutor` interface
- Execution of actions no longer goes through the event bus
- Instead, it climbs up the hierarchy of presenters, up to the subApp, using listeners
  - The subApp is responsible for:
    - fetching the action definition (action definitions will belong to the subApp descriptor)
    - executing the action (using injected ActionExecutor)

2. Registry & mappings

Problem

- Most `definitionToImplementation` mappings are configured in e.g. `ui-admincentral/workbenchActionFactory`
- We want to simplify how these mappings are configured
- It is currently not easy to use a custom impl for an action definition

Decision

- We drop configuration of these mappings
- We drop the `WorkbenchActionRegistry` and `WorkbenchActionFactory`
- We replace this with an `implementationClass` field in `ActionDefinition`
    - Its default value is hard-coded in the definition classes
    - This makes it much easier to override this field from configuration and use a custom implementation
    - See the sample configured action *editCocktail* in the Content app configuration sandbox

3. Errors & messages

Problem

- It is not clear who should be responsible for sending messages, whether it is the subapp, the action itself, or the action executor in between
- There is a technical constraint as to localizing these messages
- Currently we cannot use the same mechanisms for shell apps, where we also need such functionality

Proposal

- We could use an ActionContext for exposing messaging hooks among others
- Getting the MessageManager via IoC

Decision

- A **temporary quick solution** has been implemented with MGNLUI-743 . It basically delegates to the presenters using ActionExecutor error handling and display of messages in Pulse.
  Far from being ideal, as it involves duplicating the same code in several places.

4. Configuration

Problem

- Actions are configured in `ActionbarDefinition`
    - You either need a workbench definition to have one, either use custom extension of configured definitions to add one
- It is hard to invoke actions in a programmatic way
- It is impossible to invoke actions from other widgets (page editor, inplace-editing, dialogs)

Proposal

- There is a pending proposal for Dialog/Form actions in the Content app configuration sandbox's Dialogs section.

Decision

- Action definitions are configured independently - outside - from workbench and action bar definitions.
- They belong to the subApp
    - If multiple subApps need the same action, each one has to configure it (or use an extend)
- Action bar (as well as further controls such as Context menu):
    - is also at subApp level
    - simply configures its structure (sections/groups/actions), consisting in a text reference to these actions
    - We drop ActionbarItemDefinition and replace it directly with ActionDefinition (in configuration, node name changes from *items* to *actions*).
- We configure UI properties such as label, icon and i18nBasename in the `ActionDefinition`
- **This is the resulting configuration**: Content app configuration sandbox

5. Workbench dependency

Problem

- Workbench `defaultAction` is defined in the action bar definition whereas it has nothing to do with action bar interaction.
    - It references an action by name, which does not guarantee uniqueness (currently multiple actions can have the same name in different groups/sections)
    - One cannot configure different doubleclick/tap actions per item type

Decision

- ⚠️Although this is ready for implementation, it has a dependency to - and should not be implemented before - step 4 (configuration).
    - Uniqueness of action name is within scope of the configuration topic.
- `defaultAction` will be configured under each itemType of the workbench definition, to allow for different default actions per item type (see MGNLUI-389).
    - Double-click/Enter is also used for inplace-editing (if workbench definition is editable), so if both are configured it should log a warning.

6. Context sensitivity

Problem

The action bar can take several states within one sub-app.

- In page editor, some actions are displayed or hidden according to the type of selection (page, area, optional area, component)
- This is currently handled in a dirty way:
    - The action bar provides an API for showing/hiding a whole section
    - Therefore the page editor sub-app defines 7 different sections, which mostly hold similar groups, and sometimes even similar (i.e. duplicated) action definitions.
- Sometimes the **context sensitivity** is only represented by enabling/disabling one or several actions
    - Each sub-app has to reimplement the itemSelected listener to control proper enablement of actions
    - Basic conditional enablement is not ensured, nor is it configurable
    - One cannot easily control which actions are available (visible/enabled) based on parameters out of configuration (roles, instance)
- Conditional enablement of actions has already caused many issues before alpha1

Proposal

- Action bar definition should define two things:
    - the **structure**: sections, groups, and all possible actions in these groups, in one single place. This ensures proper ordering of actions at all time
    - the **states** that describe e.g. the section titles, the set of actions displayed, and potential substates. This mechanism should allow for additive composition of states (think of the use case for an optional AND editable area in page editor)
- Implementing context sensitivity then only means to configure these states properly and toggle between them.
- ⚠️It is likely that context sensitivity is not handled at all at the action bar level, but rather straight on the actions.
    - If so, the action bar definition is reduced to its sole structure.
    - Edge case (page editor): do we provide a way to configure context-sensitive section titles?

Decision

-

7. Architecture & commands

Problem

- Actions and presenters currently hold quite some business logic, and actions clearly lack testing
- Presenters should probably not deal with JCR at all, since they do not ensure re-usability.
    - As a result several presenters / vaadin adapters may implement the same duplicated logic.
- The role of commands vs. actions needs clarification

Answer

- Commands is where back-end code belongs

    - For more information about the "new" Commands, please refer to the Commands page
- Actions are invoked from the UI
    - They should process and update the UI
    - They should delegate back-end code to commands
- Presenters can execute actions (see step 1)

Decision

- ℹ This pattern is not strictly enforced right now, but we should share it loud
    - The goal is that whenever modifying back-end code in presenters/actions, then we should progressively shift to this pattern

Suggestion box

1. Semi-automated enablement of actions.
    a. Enablement of an action can be asserted from two angles:
        - data-centric: the expected data entry is not fulfilled (wrong selection type, empty input)
        - context-centric: there are more business checks that don't depend on the action data (user permissions, workflow)

        For data-centric conditions, the Action class could easily be responsible for telling whether some data is, or is not "actionable".
        Need to bridge the gap with the UI, knowing that instances of Actions only exist upon request for execution.
        Can the same apply to context-centric checks for action visibility?
    b. How about configuring **voters** to assert whether an action is visible/enabled? It might be a bit overdesigned but it would probably fit well there...
2. Integrate with Undo/Redo
    a. undo/redo will be shown as disabled actions in the UI when the undo/redo stack is empty
3. Blocking UIs need to be addressed at some point

Appendices

A. Execution of actions in 5.0-alpha2