How to get by without JCR Item adapters

Related issue

DEV-785 - Jira project doesn't exist or you don't have permission to view it.

Introduction

A `JcrItemAdapter` represents a JCR `Item` (node or property) as a Vaadin 7 data Item. It also implements Vaadin's `Property.ValueChangeListener` in order to inform/change a JCR property when a Vaadin property has changed.

Vaadin Framework 8 introduces a new data model. Framework 8 no longer uses Vaadin's `Item`, `Property` nor `Container`. Domain objects can be directly bound to forms. In Magnolia's case, such domain object, at least at a low level, is certainly a JCR Node.

With Vaadin 8, Magnolia can thus remove the JcrItemAdapter abstraction and use JCR Node, Property and Session API directly when the data source is a JCR repository.

Problem

A JCR Session lives for the time of an HttpRequest whereas a form lifetime typically spans multiple requests. A JCR session used to populate a form would be expired by the time a user hits the save button. How do we solve this problem?

JCR Session management doesn't seem to offer anything out of the box. The only document I found is this Jackrabbit wiki page where they mention *read /write access, transient mods*

> *this is the only case where JCR sessions should be bound (but not stored in) http sessions.*
>
> - *create a JCR session for each http session, but be careful that you don't have them open too long, especially if you expect a lot of http sessions.*

But there seems to be nothing already available, you're on your own implementing this.

How does `JcrItemAdapter` do it?

- In case of a new item, `JCRNewNodeAdapter` keeps a reference to an existing parent node and holds property values internally which will be persisted to JCR later on.
- In case of updating an existing node, `JCRNodeAdapter` keeps a reference to the corresponding existing node and holds the properties to be changed, added or removed.
- Upon saving a form, a call to `JcrItemAdapter#applyChanges` would create the JCR item under the specified parent in case of a new node. Or update an existing one. Such changes would now be in the Session's transient storage until they're saved.

Possible solutions

LazyNodeWrapper

This decorator allows to use a Node even when detached from Session, sort of pojo, and re-acquire a valid session upon need, e.g. getting/setting properties, saving, etc.

A CRUD PoC with Contacts app has been made to try this. Updating or deleting an existing contact seems to work fine (creating a new contact currently issues a binding problem).

> **UI**: Based on Jarda's PoC (see DEV-811) a JcrNodeDataProvider and related classes have been created. To allow binding of Node to a form a basic JcrNodePropertySet.java has been created
>
> https://git.magnolia-cms.com/users/fgrilli/repos/ui/compare/diff?targetBranch=refs%2Fheads%2Fmaster&sourceBranch=refs%2Fheads%2FDEV-785&targetRepoId=547
>
> **Contacts App**
>
> https://git.magnolia-cms.com/users/fgrilli/repos/contacts-app/compare/diff?targetBranch=refs%2Fheads%2Fmaster&sourceBranch=refs%2Fheads%2FDEV-785&targetRepoId=561

One problem emerged with the PoC. Using `Binder.setBean(node)` would bind a bean's property values to the field changes. In `LazyNodeWrapper`'s case that would mean acquiring a JCR session for any value change, even before committing to JCR. This would have likely been a performance issue.

Luckily enough though, Vaadin's Binder comes with a `readBean(bean)` method which only populates the form and does **NOT** bind bean and field value changes. **Vaadin fields can thus become the transient storage themselves**, without the need for any further abstraction on top of them and only upon saving to JCR, a session is acquired and re-attached to the node.

Open questions (for further investigations)

How to deal with the complex fields? Idea is to build them via binders:
the form can be split into several abstraction layers: <data-binding> - <field components> - <layout>
layout can be tabs like we currently have them, or something minimalistic, like laying out the fields horizontally (there you go with the composite field), or with additional controls, that can have some dynamic logic (switchable fields and etc)

Open In Session View

Keep JCR Session with the transient changes in http session (or better as an attribute in session scope of `MgnlContext`) until done. Session is per user, so no concurrency issues would happen.

Multiple sessions (users) may still change the same node but in JackRabbit 2.x, unlike OAK, the state of a session is always updated (the latest state overwrites the previous state), and merge exceptions never happen.
In brief, using an OSIV pattern would not change anything compared to how things work now in terms of data integrity.

FlowScoped PersistenceContext

Used by Spring webflow. This pattern creates a *PersistenceContext* in *flowScope* on flow startup, uses that context for data access during the course of flow execution, and commits changes made to persistent entities at the end. If a save and restart capability is not required, standard HTTP session-based storage of flow state is sufficient.