ReportableWorkflow

Funktionsbeschreibung

Version: 1.0
Date: 2020-10-06

- VERSION HISTORY -

DATE	VERSION	REVISION DESCRIPTION	Author
2020-10-06	1.0	Initial Version	Mario Vaccarisi

Inhalt

Einleitung

Um in Kundenprojekten die Anforderungen an interne Prozesse und die Arbeitsweise bezüglich Freigabe und Löschen von Inhalten in Projekten im CMS, sowohl aus dem SiteArchitect, als auch dem ContentCreator heraus, zu realisieren, bietet FirstSpirit die Möglichkeit der Implementierung von Workflows. Dazu gibt es in FirstSpirit die Möglichkeit Workflows mit Hilfe des WorkflowDesigner im SiteArchitect zu modellieren. Diese bilden allerdings lediglich den Rahmen der Handlungsroutine dar. Die eigentliche Funktionalität wird erst durch die Einbindung von Skripten in einzelnen Aktionen erreicht. Der ReportableWorkflow bildet hierbei ein leicht anzupassendes Set an Aktionen, für einen solchen Workflow und stellt eine Alternative zu den FirstSpirit-BasicWorkflows dar.

Vergleich zu BasicWorkflows

Grundsätzlich ist das Funktionsprinzip des ReportableWorkflow nicht mit den BasicWorkflows vergleichbar, welches sich in vier wesentlichen Aspekten widerspiegelt.

1. erweiterte User-Interaktion
2. projektspezifische Konfiguration
3. einfache Erweiterbarkeit
4. Eventing

Um diese vier Aspekte verstehen zu können, ist die Kenntnis über das Funktionsprinzip entscheidend. Dieses wird im folgend Abschnitt, auch anhand von Beispielen, näher erläutert. Die vier wesentlichen Auf die vier Aspekte wird dann im Anschluss weiter eingegangen.

Funktionsprinzip

Der ReportableWorkflow separiert und abstrahiert alle notwendigen Verarbeitungsschritte anhand der geforderten Aktion und dem/den Eingangselement/-en. Die Aktion beschreibt die Anforderung an ein Release, bzw. Delete von FirstSpirit-Elementen. FirstSpirit-Elemente, auf denen der Workflow gestartet wird stellen dabei die Eingangselemente dar. Dabei ist es unerheblich, ob es sich um ein oder mehrere Elemente handelt.

Die Idee hinter dem ReportableWorkflow ist zum einen das Handler-Prinzip. Ein Handler kümmert sich somit nur um bestimmte Elemente und übernimmt auch nur dedizierte Prüfungen. Er ermittelt aber auch sich ergebende Abhängigkeiten, behandelt diese aber nicht selber, bzw. nicht während der Prüfung des eigentlichen

Elements. Andererseits ermittelt er für dieses Element einen Zustand, welches über die grundsätzliche Möglichkeit der geforderten Aktion auskunft gibt. Die Prüfung aller Elemente einschließlich der während der Prüfung neu ermittelten Elemente ergibt somit für jedes Element ein oder mehrer Zustände, welche die Schlussfolgerung ermöglichen, ob z.B. ein Element freigegeben werden kann. Erst wenn es keine aktionsverhindernde Zustände gibt, wird die eigentliche Aktion tatsächlich durchgeführt. Sollte die geforderte Aktion nicht durchführbar sein, so wird dem Benutzer detailliert Auskunft darüber gegeben, warum die Aktion nicht durchgeführt werden konnte. Diese Interaktion erfolgt durch einen Report im jeweiligen FirstSpirit Client.

Das Handler-Prinzip

Nachfolgend wird das Handler-Prinzip anhand eines Beispiel skizziert und kommentiert. Das Beispiel ist abstrahiert und hat nicht den Anspruch der Vollständigkeit. Es soll lediglich die Funktionsweise demonstrieren. Technische Aspekte werden lediglich angedeutet.

1. Ein Redakteur befindet sich im ContentCreator auf der Startseite eines Projektes und möchte diese nach Änderung erneut freigeben.

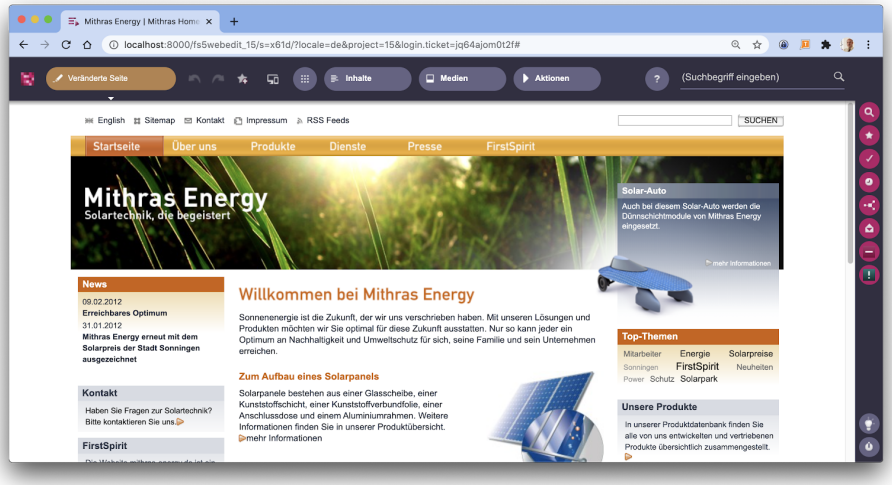

Abb. 1: Screenshot Beispiel-Projekt Mithras Energy

Kommentar:
Technisch betrachtet befindet sich der Redakteur, sofern der im ContentCreator angezeigte Inhalt nicht aus einer Content-Projektion stammt, auf einer Seitenreferenz (PageRef). Diese verweist ihrerseits aber auf eine Seite (Page), welche die eigentlichen Inhalte liefert.

2. Nach dem Start des des Workflows zur erneuten Freigabe nach Änderung und Speicherung, wird durch das entsprechende im Workflow hinterlegte Skirt das Modul ReportableWorkflow aufgerufen und die Prüfung der Freigabefähigkeit gestartet.

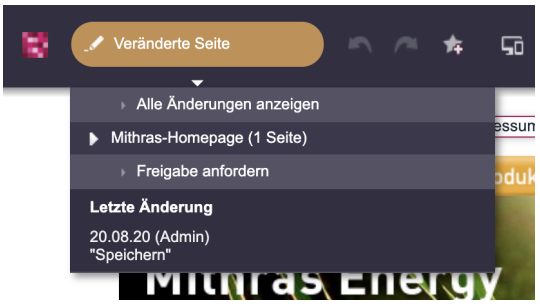

Abb. 2: Screenshot ElementStatusProvider im Beispiel-Projekt Mithras Energy

Kommentar:
Die Modellierung des Workflows ist nicht Bestandteil des ReportableWorkflow und wird hier nicht abgebildet.

3. Innerhalb des Moduls wird ein Controller initialisiert. Er ist dafür zuständig die Eingangs- und Ausgangsliste zu pflegen. Zudem kennt und verwaltet er alle Handler.

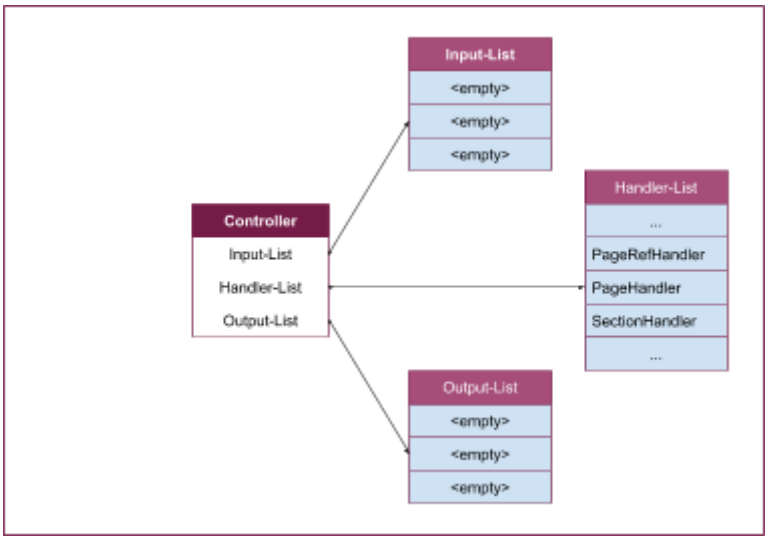

Abb. 3: Schematischer Aufbau ReportableWorkflow

Kommentar:
Zur Veranschaulichung sind hier einmal exemplarisch drei Handler aufgeführt. Das Modul bietet in Summe 17 Default-Handler.

4. Nach der Initialisierung wird das Eingangselement, hier dargestellt durch unsere Startseite, bzw. technisch genauer unsere PageRef, in die Eingangsliste abgelegt.

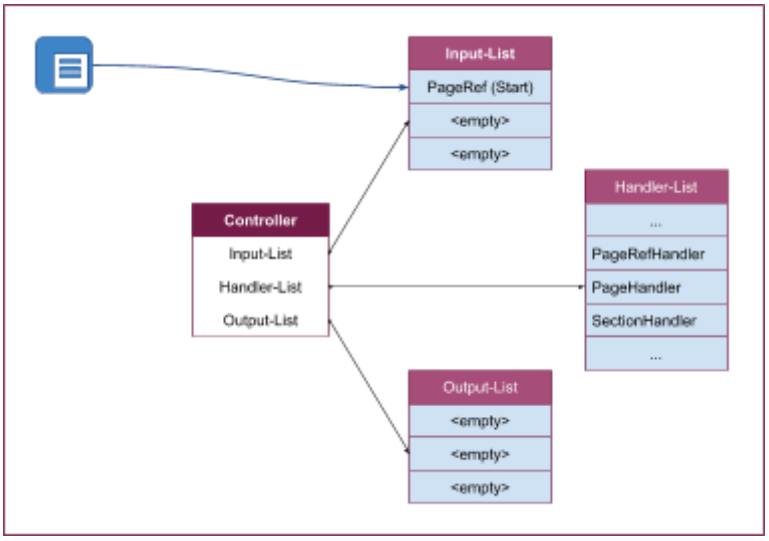

Abb. 4: Schematischer Aufbau ReportableWorkflow: Eingehendes Element

5. Nachdem unsere PageRef sich nun in der Eingangsliste befindet, wird der Controller gestartet. Dieser beginnt nun mit der Verarbeitung der Elemente in der Eingangsliste, bis die Liste keine Elemente mehr enthält. Zur Verarbeitung entnimmt der Controller immer das oberste Element der Liste und übergibt es der Reihe nach jeden bekannten Handler.

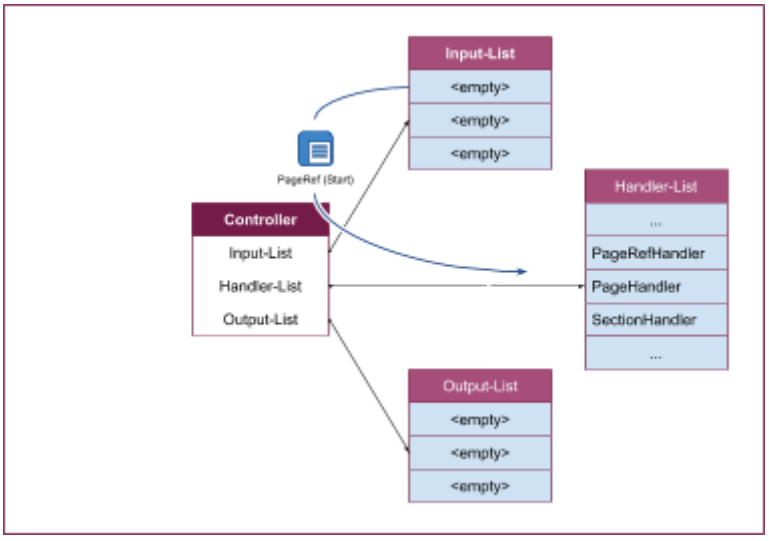

Abb. 5: Schematischer Aufbau ReportableWorkflow: Abarbeiten Eingangsliste

6. Jeder Handler nimmte das Element entgegen und prüft zunächst, ob er für die Verarbeitung eines Solchen Elements zuständig ist. Ist dies der Fall wird er in Abhängigkeit der Aktion tätig. Die notwendigen Tätigkeiten unterscheiden sich nicht nur von der Aktivität, sondern auch von dem Elementtyp. So gibt es bei einer PageRef andere Kriterien, welche z.B. zur Freigabe erfüllt sein müssen, als bei z.B. Medien.

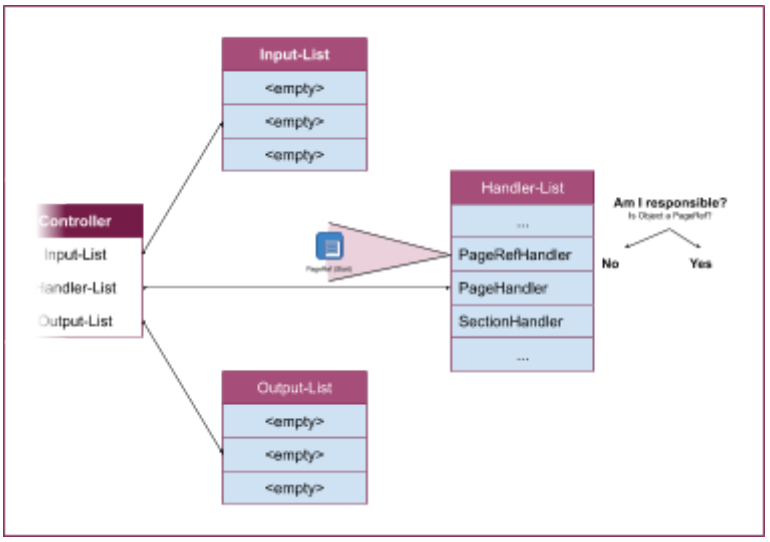

Abb. 6: Schematischer Aufbau ReportableWorkflow: Element-Prüfung in einem Handler

7. Ist ein Handler zuständig, so nimmt er seine Tätigkeit auf. Die besteht im wesentlichen aus zwei Teilen:

    a. spezifischen Prüfungen (z.B. Formulare) und
    b. dem Ermitteln von abhängigen Elementen.

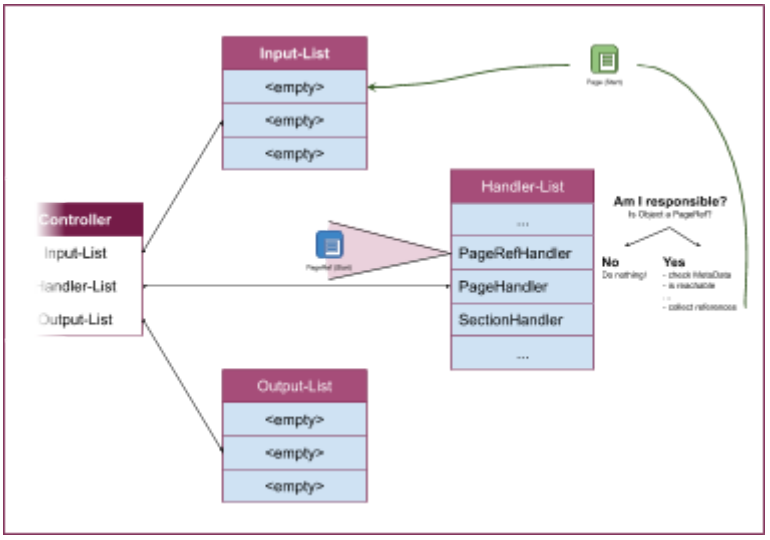

Abb. 7: Schematischer Aufbau ReportableWorkflow: Ermittlung abhängiger Elemente

Kommentar:
Unser PageRefHandler wird tätig, da es sich bei dem vom Controller übergebene Element um eine PageRef handelt. Der Handler prüft nun die direkten Daten der PageRef. Hierbei handelt es sich im wesentlichen um die Metadaten. Bei der Ermittlung von abhängigen Elementen wird u.a. die zugrundeliegende Page ermittelt. Da der PageRefHandler zu diesem Zeitpunkt bereits ein Element in Bearbeitung hat und er grundsätzlich nicht für diesen Elementtyp zuständig ist, wird die gefundene Page zur späteren Verarbeitung in die Eingangsliste gelegt.

8. Aus dem Prüfungsteil der Tätigkeiten fällt ein Ergebnis heraus, welches sich im Zusammenhang mit dem Element zwischengespeichert wird. Die Entscheidung, ob eine Aktion letztenendes möglich ist, wird allerdings noch nicht zu diesem Zeitpunkt gefällt. Sollte eine Prüfung negativ ausfallen, wird zu dem der Grund hierfür ebenfalls im Zusammenhang mit dem Element notiert.

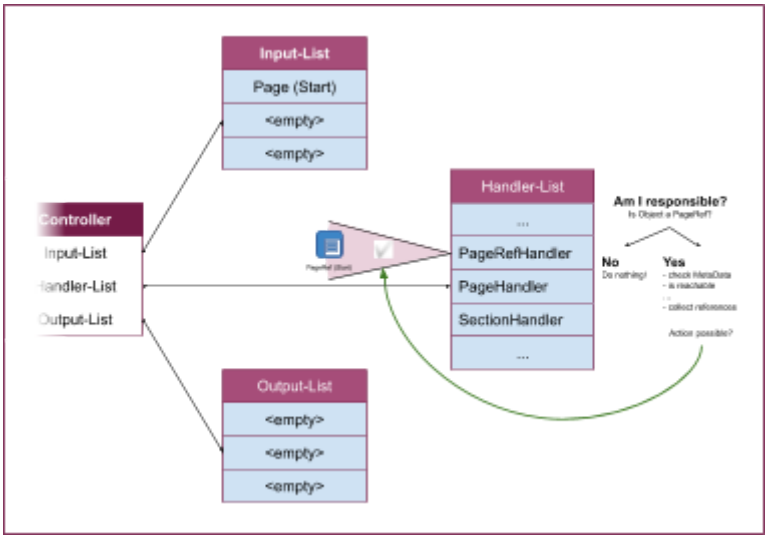

Abb. 8: Schematischer Aufbau ReportableWorkflow: Prüfungsentscheidung

9. Sind alle Handler durchlaufen, wird das Element in die Ausgangsliste abgelegt. Sofern die Eingangsliste noch nicht vollständig geleert ist, beginnt der Zyklus erneut.

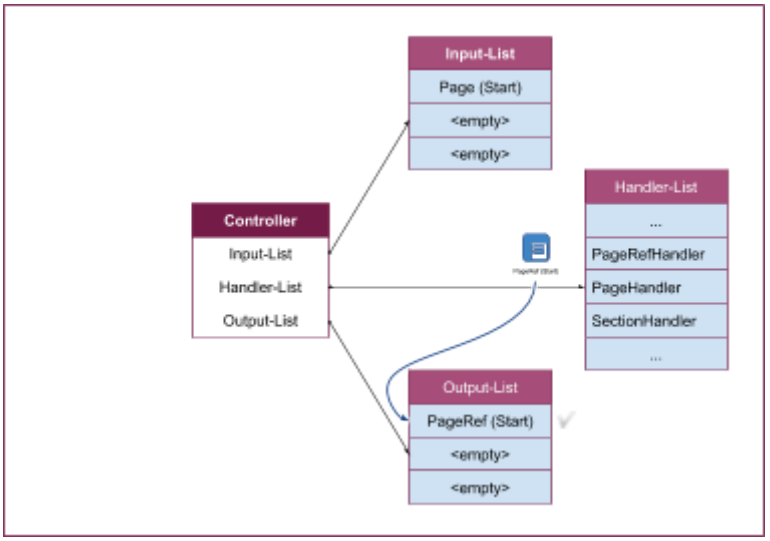

Abb. 9: Schematischer Aufbau ReportableWorkflow: Transfer in Ausgangsliste

<u>Kommentar:</u>
Hierbei ist zu beachten, dass im Laufe des Einsammeln von abhängigen Elementen es natürlich vorkommen kann, dass oftmals gleiche abhängige Elemente ermittelt werden. Diese werden selbstverständlich nicht erneut geprüft.

Ein besonderer Zustand stellen dabei sogn. zirkuläre Abhängigkeiten dar. Sprich: A kann nur freigegeben werden, wenn B bereits freigegeben ist. B kann aber erst freigegeben werden, wenn A bereits freigegeben ist. Dieser Zustand kann nicht automatisch

vom ReportableWorkflow aufgelöst werden. Dieser Zustand wird aber angezeigt und kann vom Redakteur bewußt an gewünschter Stelle unterbrochen werden.

10. Nach der Verarbeitung aller Elemente befinden sich nunmehr keine Elemente mehr in der Eingangsliste. Stattdessen ist nun die Ausgangsliste mit allen während den Iterationen eingesammelten abhängigen Elementen gefüllt. Zudem hat jedes Element mindesten eine Status bezüglich der geforderten Aktion. Ein Element kann u.U. mehrere Staus haben. Da jedes Element grundsätzlich alle aktiven Handler durchläuft, können verschiedene Handler einem Status vergeben. Das Resultat wird dem Redakteur visuell mitgeteilt.

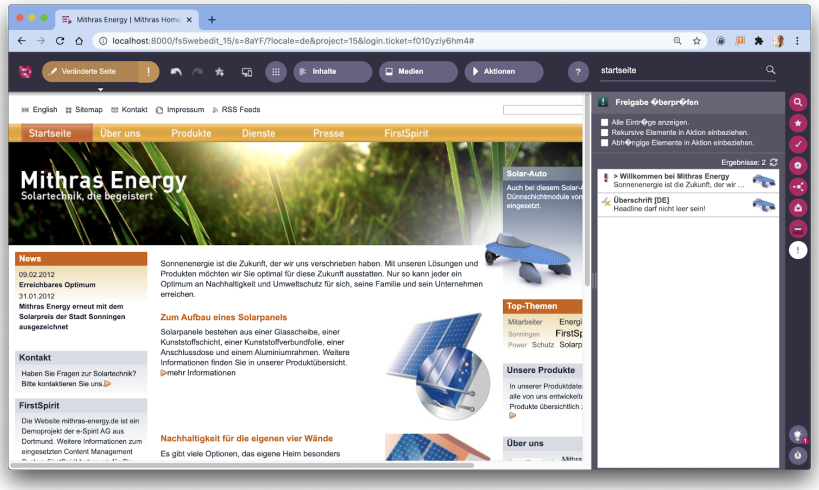

Abb. 10: Ergebnisdarstellung im Beispiel-Projekt Mithras Energy

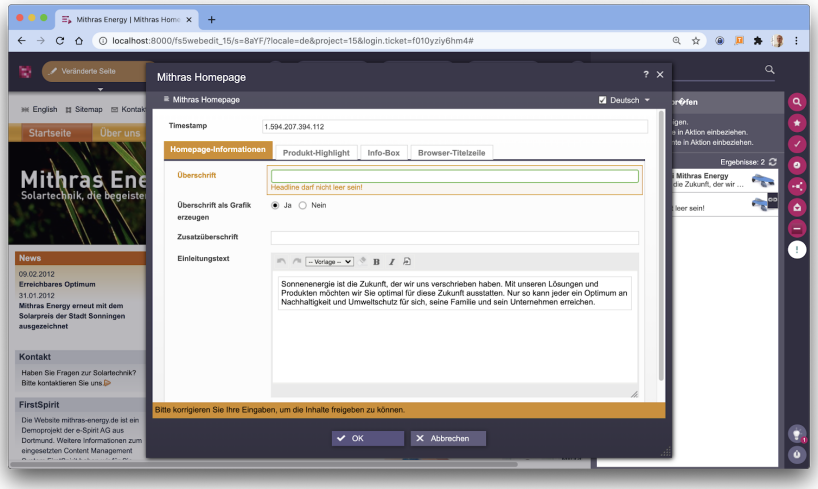

Abb. 11: Durch Klick auf Ergebniseintrag geöffnetes Formular im Beispiel-Projekt Mithras Energy

Kommentar:
Sofern alle Status eines Elements, aller Elemente in der Ausgangsliste, für die geforderte Aktion einen positiven Status haben, kann die Aktion als solches auf dem/den ursprünglichen Element/en durchgeführt werden. Ist dies nicht der Fall, also ist auch nur ein Status eines Elements negativ, so wird die geforderte Aktion nicht durchgeführt. In dies der Fall wird dem Redakteur ein Report angezeigt, welcher zum einen genauere Informationen darüber enthält, warum das Element nicht die für die angeforderte Aktion valide ist. Außerdem kann er durch einen Klick auf den entsprechenden Eintrag direkt zur betreffenden Stelle springen, um ggf. den vorliegenden Grund zu bearbeiten.

Der Controller

Ein Kernelement des ReportableWorkflow ist der Controller. Bei Initialisierung ermittelt er aufgrund der auf dem Server generell zur Verfügung stehenden Handler und der Konfiguration der PorjectApp im jeweiligen Projekt, die für die geforderte Aktion notwendigen Handler. Dies bedeutet, dass im Standard zwar alle DefaultHandler zur Verfügung stehen, diese in der ProjectApp aber individuell de-/aktiviert werden können. Über die DefaultHandler hinaus ermittelt er auch CustomHandler, welche über separate Module auf dem FirstSpirit-Server installiert sind. Diese können ebenfalls über die ProjectApp konfiguriert werden. Im Rahmen des Customizing ist es sogar möglich, einen eigenen Controller als solches zu implementieren und zu verwenden.

Über das Laden der Handler hinaus bietet er auch die Möglichkeit des Eventing. Damit ist den Controller gleichzeitig ein Listener für den eigentlichen Programmablauf.

Erweiterte User-Interaktion

Im Abschnitt Funktionsprinzip unter Punkt 10 wurde bereits die Interaktion mit dem Redakteur durch die Verwendung eines Reports veranschaulicht. Grundsätzlich gibt es jeweils einen Report für die Aktionen Release und Delete. Diese Reports können auch unabhängig vom Workflow durch einen Redakteur aufgerufen werden. Somit ist es möglich auch außerhalb des Workflow für die jeweilige Aktion eine mögliche Freigabe, bzw. Löschen zu prüfen. Die Prüfung wird dabei immer auf dem aktuell angezeigten Element ausgeführt.

Die Prüfung, bzw. das Prüfungsergebnis kann redaktionell beeinflusst werden. Somit kann der Redakteur wählen, ob alle oder nur nicht valide Elemente im Report aufgelistet werden. Außerdem kann die Prüfung, ähnlich der administrativen Freigabe/Löschen im SiteArchitect, dahingehend angepasst werden, dass rekursiv Elemente und/oder abhängige Elemente mit in die Prüfung mit eingeschlossen werden.

Das Modul ReportableWorkflow kann aber auch GUI-less aus einem Server-Auftrag heraus ausgeführt werden. Dabei entfällt das automatische öffnen des Reports im Client.

Projektspezifische Konfiguration

Die Konfiguration des Modul ReportableWorkflow erfolgt auf Projektebene. Dies bedeutet, dass die Implementierung im Gegensatz zu den BasicWorkflows nicht automatisch immer alle Projekte betrifft und in allen Projekten zur Verfügung steht. Zur Einrichtung muss daher einem Projekt die ReportableWorkflow-ProjectApp hinzugefügt werden.

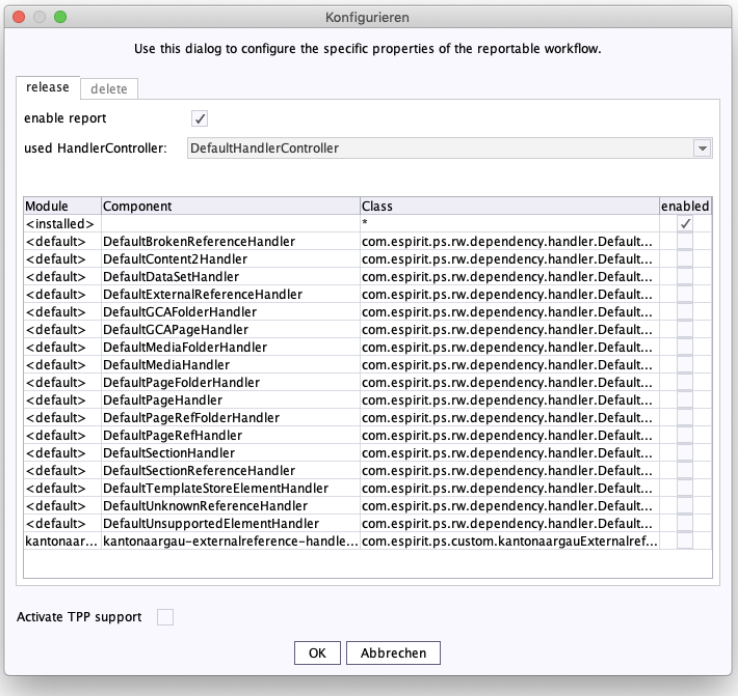

Abb. 12: Konfigurationsdialog innerhalb der ProjectApp

In der Konfiguration kann zum einen ein auf dem FristSpirit-Server installierter Controller ausgewählt werden. Im Standard wird der DefaultHandlerController mitgeliefert. In der Regel ist es nicht notwendig einen eigenen Controller zu implementieren. Eine Ausnahme kann sein, wenn explizit das Eventing des Controller erforderlich ist, da im Standard die Events unberücksichtigt sind.

Des Weiteren können hier die erforderlichen Handler de-/aktiviert werden. Dabei ermittelt die ProjectApp selbstständig alle auf dem Server installierten Handler. CustomHandler befinden sich dabei in einem eigenen Modul und werden nicht im ReportableWorkflow-Modul angepasst.

Die Konfiguration von Controller und Handler ist für Release und Delete separat möglich.

Einfache Erweiterbarkeit

Grundsätzlich erfolgt die Erweiterung des ReportableWorkflow in separaten Kunden-Modulen. Eine kundenspezifische Anpassung im eigentlichen ReportableWorkflow-Modul ist nicht vorgesehen. Dabei ist es zum Zweck des Customizing möglich sowohl eigene Controller, als auch eigene Handler zu implementieren. Diese müssen nicht zusammengefasst in einem Modul implementiert werden. Eine Verteilung auf mehrer Module zum Zwecke der Übersichtlichkeit oder dedizierten Trennung zwischen verschiedenen Projekten in der Entwicklung ist möglich.

Im Rahmen des Customizings gibt es die Möglichkeit entweder einzelne Handler oder gesamte Controller zu implementieren. Beides Implementierungen sind voneinander unabhängig. Um einen eigenen Handler zu implementieren muss lediglich das Interface Handler implementiert werden. Es empfiehlt sich allerdings von der abstrakten Klasse AbstractDefaultHandler zu erben und nur die notwendigen Methoden zu implementieren, bzw. gezielt Methoden zu überschreiben. Gleiches gilt im Übrigen auch für die Implementierung eines eigenen Controllers. Dazu ist das Interface HandlerController zu implementieren. Auch hier gibt es die Möglichkeit auf eine bereits in Teilen vorgefertigte abstrakte Klasse zurück zu greifen: AbstractHandlerController.

Damit eigene Implementierungen von Controllern oder Handlern durch das Modul ReportableWorkflow gefunden und verwendet werden können, sind diese im Rahmen der Modulentwicklung als public-Komponente in der module.xml zu definieren.

Eventing

Während die Handler im wesentlichen für die Prüfungen und das ermitteln von abhängigen Elementen verantwortlich sind, ist der Controller primär für das Eventing, aber auch das programmatische Hinzufügen oder Entfernen von Handlern zuständig. Bei dem Hauptaspekt, dem Eventing, gibt es grundsätzlich die Möglichkeit auf die folgenden Events zu reagieren:

- preIteration()
  Zeitpunkt zu Beginn einer Iteration. Das zu prüfende Element ist bereits bekannt.

- preExecution()
  Zeitpunkt bevor das zu prüfende Element einem Handler übergeben wird.

- postExecution()
  Zeitpunkt nachdem das zu prüfende Element von einem Handler zurückgegeben wurde.

- postIteration()
  Zeitpunkt nachdem das zu prüfende Element alle vorgesehenen Handler durchlaufen hat.

Hinweis:
- *Diese Liste enthält nur Events, welche zwingend bei der Ableitung des AbstractHandlerController zu implementieren sind.*

Der hier verwendet Begriff Iteration bezieht sich auf das programmatische Aufrufen aller aktivierten Handler mit Übergabe des zu prüfenden Elements.

Installation

Die Installation des ReportableWorkflow erfolgt gemäß der allgemeinen Installation von FirstSpirit-Modulen. Dazu ist das entsprechende FSM-File im ServerManager unter *Server* > *Eigenschaften* > *Module* zu installieren. Gegebenenfalls muss nach der Installation das Modul noch um die Option "Alle Rechte" erweitert werden. Nach der Installation der Module muss der FirstSpirit-Server zwingend neu gestartet werden. Nach der Installation und dem neustart steht das Modul grundsätzlich auf dem gesamten FirstSpirit-Server zur Verfügung. Um den ReportableWorkflow nun nutzen zu können, muss nun noch die Verwendung im gewünschten Projekt konfiguriert werden.

Konfiguration im Projekt

Die ProjectApp

Um den ReportableWorkflow innerhalb eines Projektes zur Verfügung zu stellen, muss die ProjectApp in dem gewünschten Projekt hinzugefügt werden. Wechseln Sie dazu im ServerManager in das gewünschte Projekt und fügen anschließend unter *Projekt-Komponenten* mit Klick auf *Hinzufügen* die "*Reportable-Workflow ProjectApp*" hinzu. Diese kann anschließend konfiguriert werden.

Im Konfigurationsdialog *(siehe Abb. 12: Konfigurationsdialog innerhalb der ProjectApp)* kann dann anschließend getrennt für die jeweilige Aktion (Release/Delete) der zu verwendende Controller eingestellt und die einzelnen Handler (de-)aktiviert werden. Mit öffnen des Dialoges wird automatisch nach allen verfügbaren Controllern, bzw. Handlern gesucht, unabhängig davon, mit welchem Modul sie am FirstSpirit-Server installiert wurden.

Zu beachten ist die Option "Activate TPP support", wenn es sich bei dem Projekt um ein solches handelt!

Die WebApp

Damit der ReportableWorkflow auch im ContentCreator zur Verfügung steht und ordnungsgemäß funktioniert, muss die entsprechende WebApp installiert werden. Das Hinzufügen erfolgt im ServerManager im Projekt über

die Option *Web-Komponenten,* unter dem Reiter *ContentCreator*. Mithilfe des Hinzufügen-Button öffnet sich ein Dialog aus dem, die "*Reportable Workflow - WebApp*"-Komponente auszuwählen ist. Anschließend muss das Hinzufügen der WebApp mit *Aktualisieren* abgeschlossen werden.

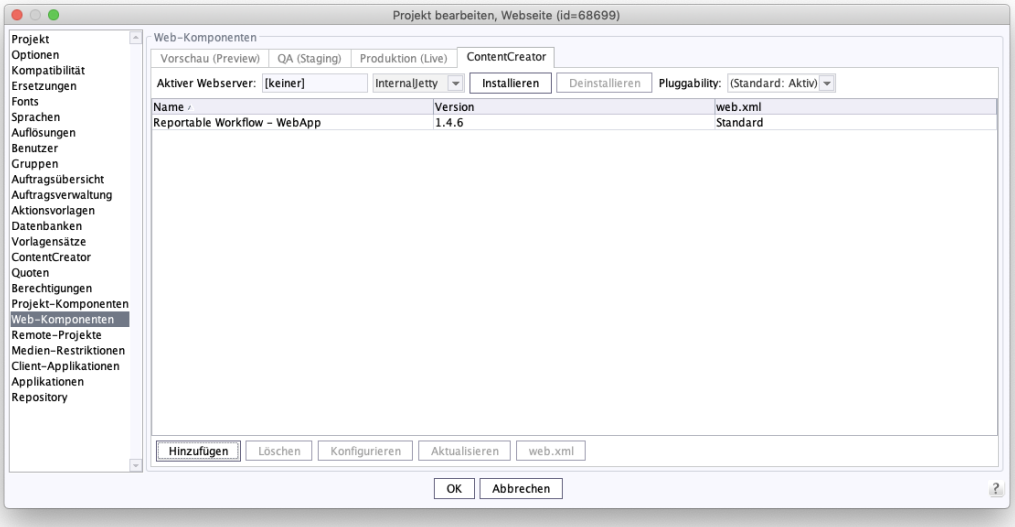

Abb. 13: Dialog Web-Komponenten im Projekt nach Hinzufügen der WebApp

Die Verwendung in Workflows

Der ReportableWorkflow ist kein proaktives Modul. Er bietet Funktionalitäten an, welche durch API-seitige Aufrufe von entsprechenden Executables getriggert werden. Um diese Aufrufe aus einem der zur Verfügung stehenden Clients (SiteArchitect/ContentCreator) auszuführen, sind entsprechende FirstSpirit-Skripte notwendig. Innerhalb dieser Skripte müssen die entsprechenden Executables angesprochen werden. Jede Executable repräsentiert dabei eine Mögliche Kombination von geforderter Aktion und Prüfungseinstellung.

Folgende Executables sind aufrufbar:

- Release
  - `reportable_workflow_validate_release_element_executable`
    Prüft, ob ein Element freigebbar ist.
  - `reportable_workflow_validate_recursive_release_element_executable`
    Prüft, ob ein rekursiv Element freigebbar ist (also: inkl. Kinder).

- `reportable_workflow_validate_depended_release_element_executable`
  Prüft, ob ein Element inkl. abhängiger Elemente freigebbar ist.
- `reportable_workflow_validate_depended_recursive_release_element_executable`
  Prüft, ob ein Element rekursiv, inkl. abhängiger Elemente freigebbar ist.
- `reportable_workflow_release_element_executable`
  Gibt ein Element frei.
- `reportable_workflow_recursive_release_element_executable`
  Gibt ein Element rekursiv frei (also: inkl. Kinder).
- reportable_workflow_depended_release_element_executable
  Gibt ein Element inkl. abhängiger Elemente frei.
- `reportable_workflow_release_depended_recursive_element_executable`
  Gibt ein Element rekursiv, inkl. abhängiger Elemente frei.

- Delete
  - `reportable_workflow_validate_delete_element_executable`
    Prüft, ob ein Element löschbar ist.
  - `reportable_workflow_delete_element_executable`
    Löscht ein Element.

Diese Executables können nun direkt in Workflow-Skripte zur Anwendung kommen. Dabei sind lediglich die Referenznamen er weiterführenden Transitionen aus einer Aktion heraus zu beachten. Damit der Workflow durch die Executables entsprechend automatisch in den nächsten Zustand überführt werden kann, müssen die Transitionen auf ***_ok***, bzw. ***_nok*** enden. Der Suffix `_ok` bezeichnet dabei eine erfolgreiche Aktion/Validierung. Der Suffix `_nok` entsprechend das Fehlschlagen.

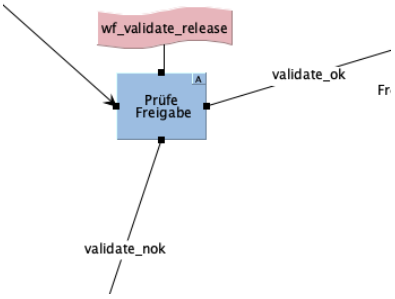

Abb. 14: Beispiel einer Script-Aktion in einem Validierungs-Workflow