Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3462

Visuelle Modellierung von Screenflows

Timotheus Giuliani

Studiengang:	Softwaretechnik
Prüfer:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuer:	Dipl.-Inf. Dipl.-Wirt. Ing.(FH) Karolina Vukojevic Dr. Andreas Nauerz (IBM)
begonnen am:	28. Januar 2013
beendet am:	25. Juli 2013
CR-Klassifikation:	H.4.1, H.5.2, H.5.3

Kurzfassung

Automatisierte Prozesse erfordern häufig Interaktionen mit Menschen. Für diese Human Tasks werden in der Regel Benutzungsschnittstellen benötigt. Für die Umsetzung und Bereitstellung von Benutzungsschnittstellen eignen sich Portale sehr gut. Im Portal werden die einzelnen Human Tasks durch die Nutzung von Task Listen einer potenziellen Gruppe an Nutzern zur Verfügung gestellt. Über die Task Liste wird in der Regel ein Human Task einer entsprechenden Portalseite (Screen) zugeordnet.

Allerdings hat sich herausgestellt, dass eine einfache 1:1 Abbildung zwischen Human Tasks und Portalseiten nicht immer ausreichend ist. Oft ergibt sich die Notwendigkeit, eine von einem einzelnen Nutzer schnell zu prozessierende Abfolge von Human Tasks über mehrere Screens abzuarbeiten. Es ist unnötig und unpraktikabel jeden dieser Screens auf einen Human Task abzubilden, vor allem wenn die Prozessierung kurzlebig und kein Wechsel zwischen Nutzern notwendig ist.

Als Lösung wurden Screenflows eingeführt, die es erlauben, eine Abfolge von Screens im Portal deklarativ zu modellieren. Im Zusammenspiel mit einem Workflow kann nun die Kontrolle für einen einzelnen Human Task an das Portal übergeben werden. Das Portal stellt dann eine Abfolge von Screens, entsprechend des modellierten Screenflows, zur Verfügung. Anschließend gibt es die Kontrolle an das Workflowsystem zurück. Diese Lösung erlaubt eine Abbildung eines Human Tasks auf einen Screenflow.

Die Modellierung der Screenflows geschieht derzeit noch über komplexe XML-Beschreibungen, welche für technisch nicht versierten Nutzer unverständlich sind.

In dieser Arbeit wird der Entwurf und die Entwicklung eines Modellierungswerkzeugs für die visuelle Modellierung von Screenflows beschrieben. Es werden Konzepte vorgestellt, welche für die Umsetzung einer geeigneten Lösung benötigt werden. Diese Konzepte werden anschließend prototypisch in einem webbasierten Modellierungswerkzeug umgesetzt. Abschießend werden die erarbeiteten Konzepte auf Modellierungswerkzeuge für Scientific Workflows übertragen.

Inhaltsverzeichnis

1. Einleitung

1.1. Einführung und Motivation

In den vergangenen Jahren haben sowohl *Workflow-* als auch *Portaltechnologien* zunehmend an Bedeutung gewonnen. Workflowtechnologien werden unter anderem im Unternehmensumfeld eingesetzt, um wiederkehrende Prozesse zu modellieren, auszuführen und zu analysieren. Diese automatisierten Prozesse werden als *Workflows* (deutsch: Arbeitsabläufe) bezeichnet. Durch die Ausdehnung dieser Technologie auf Prozesse in wissenschaftlichen Bereichen entstand der Begriff *Scientific Workflows* [IEDM07]. Die Portaltechnologie wiederum bietet ihren Nutzern einen zentralen und persönlichen Zugang (Single Point of Access) zu Diensten und Informationen [GK]. *Portale* integrieren dafür die heterogenen Systeme von Informationslandschaften, aggregieren Informationen und stellen verschiedene Dienste und Anwendungen über eine einheitliche Nutzungsschnittstelle zur Verfügung. Des Weiteren können Portale eine Arbeitsumgebung durch zusätzliche Dienste und unterstützende Werkzeuge erweitern. Beispiele hierfür sind Werkzeuge für die Zusammenarbeit, Inhalts- und Dokumentenverwaltung oder die Suche von Inhalten.

Die Zusammenführung beider Technologien liegt für Workflows nahe, bei denen Interaktionen mit Benutzern erforderlich sind. Sogenannte *Human Tasks* benötigen in der Regel eine Benutzungsschnittstelle und Portale eigenen sich hervorragend für die Umsetzung von Benutzungsschnittstellen. Daher werden Human Tasks häufig auf Portalseiten abgebildet. Oft ergibt sich, auch innerhalb des Portals, die Notwendigkeit eine von einem einzelnen Nutzer schnell zu prozessierende Abfolge von Schritten über mehrere sogenannte *Screens* (Views) abzuarbeiten. Ein Screen ist ein möglicher Zustand des Inhalts einer dynamischen Portalseite. In gewissen Situationen ist es unnötig und unpraktikabel jeden dieser Screens durch einen eigenen Human Task zu beschreiben, da die Prozesse äußert kurzlebig sind oder ein Wechsel zwischen unterschiedlichen Nutzern nicht notwendig ist. In solchen Situationen gab es bisher zwei Möglichkeiten.
Erstens, komplexe Portlets zu entwickeln, welche den Prozess in ihrer Programmlogik abbilden. Das hat jedoch den Nachteil, dass solche Portlets sehr unflexibel sind und kaum wiederverwendet werden. Zusätzlich erfordert jede Änderung am Prozess eine Änderung am Programmcode des Portlets.
Zweitens, mehrere simple Portlets zu entwickeln, welche jeweils einzelne Schritte des Prozess abbilden. Bei diesem Ansatz fehlt jedoch eine Nutzerführung, sodass der Benutzer selbst zu entscheiden hat, in welcher Reihenfolge die Portlets zu verwenden sind.

Das Buchen einer Reise auf einem öffentlichen Reiseportal ist ein mögliches Beispiel für einen solchen Prozess. In der Regel legt der Kunde verschiedene Details zu seiner Reise fest, bevor diese dann tatsächlich gebucht wird. Meist wird der Kunde dabei durch eine Art Dialog geführt, in dem Details wie zum Beispiel das Reiseziel, das Reiseantrittsdatum, die Hotelklasse und dergleichen abgefragt werden.

Der Lösungsansatz im *IBM WebSphere Portal Server*[1] für dieses Problem ist die Einführung von sogenannten *Screenflows*, welche es erlauben, eine Abfolge von Screens im Portal deklarativ zu modellieren. Das hat den Vorteil, dass der Screenflow flexibel ist und schnell geändert werden kann. Die verwendeten Portlets können spezifisch für eine Aufgabe entwickelt werden, wodurch sie wiederverwendet werden können. Der Benutzer wird geführt, indem er von Screen zu Screen geleitet wird. Im Zusammenspiel mit einem Screenflow kann ein Workflow nun die Kontrolle für einen einzelnen Human Task an das Portal übergeben. Das Portal stellt dann eine Abfolge von Screens, entsprechend des modellierten Screenflows, zur Verfügung. Nachdem der Screenflow beendet ist, gibt der Screenflow die Kontrolle zurück an das Workflowsystem. Die Lösung ermöglicht also die Abbildung eines Human Tasks auf einen Screenflow.

Für herkömmliche Workflows existieren diverse Modellierungswerkzeuge, die es ermöglichen, den Workflow grafisch zu modellieren. Die Modellierung der Screenflows geschieht derzeit jedoch noch über komplexe XML Beschreibungen, welche für den nicht technisch versierten Nutzer unverständlich sind.

1.2. Ziele dieser Arbeit

Diese Arbeit entstand im Rahmen einer Diplomarbeit in Kooperation zwischen dem Institut für die Architektur von Anwendungssystemen, im Bereich SimTech, der Universität Stuttgart und dem Bereich WebSphere Development & Services, IBM Collaboration Solutions, in der Abteilung XWebX Development der IBM Software Group.

Das Ziel dieser Arbeit ist die Entwicklung und Implementierung eines grafischen Modellierungswerkzeugs, mit dessen Hilfe Screenflows visuell modelliert werden können. Im Fokus steht dabei die Entwicklung eines einfach und intuitiv zu bedienenden und zugleich leistungsfähigen Werkzeuges. Dadurch sollen auch Modellierer, die keine Computerexperten sind, in die Lage versetzt werden, Screenflows zu modellieren, an denen ein Endnutzer später entlang geführt werden soll. Ferner soll erörtert werden, inwiefern sich die gewonnen Erkenntnisse auf Modellierungswerkzeuge für Scientific Workflows übertragen lassen.

[1] Für weitere Informationen zum IBM WebSphere Portal Server siehe http://www-03.ibm.com/software/products/us/en/portalserver.

1.3. Gliederung

Nachdem die Motivation und die Aufgabenstellung für diese Arbeit vorgestellt wurden, soll die folgende Gliederung einen Überblick über die weiteren Kapitel dieser Arbeit geben.
In Kapitel 2 werden die für diese Arbeit elementaren Technologien beschrieben. Der erste Teil dieses Kapitels handelt von Prozessen und Workflows. Der zweite Teil des Kapitels beschreibt die Portaltechnologie mit ihren einzelnen Bestandteilen.
In Kapitel 3 wird beschrieben, warum Workflows oft nicht ohne Interaktionen mit Menschen auskommen. Anschließend wird die heutige Kommunikation zwischen Workflow- und Portalsystemen beschrieben. Danach wird das Konzept der Screenflows vorgestellt. Als Letztes wird in diesem Kapitel beschrieben, wie Screenflows in Workflows verwendet werden können und wie der Datenaustausch dabei stattfindet.
In Kapitel 4 werden zum Thema Screenflow verwandte Arbeiten vorgestellt. Der erste Teil des Kapitels handelt von konzeptionellen Arbeiten. Der zweite Teil beschreibt anschließend Arbeiten, die auch konkrete Implementierungen umfassen.
In Kapitel 5 wird der IBM UX Screenflow Manager vorgestellt, auf dem die Konzepte der Arbeit aufbauen. In diesem Kapitel werden die grundlegenden Konzepte und Funktionsweisen des Screenflow Managers aufgezeigt. Anschließend werden alle Komponenten, die für eine Screenflow Definition benötigt werden, aufgeführt.
Kapitel 6 bildet den Kern dieser Arbeit. Im ersten Abschnitt des Kapitels wird die Ausgangssituation geschildert. Anschließend werden die Anforderungen für das grafische Modellierungswerkzeug aufgestellt. Danach wird der Lösungsansatz beschrieben, indem auf die einzelnen Konzepte zur Lösung der Problematik eingegangen wird.
In Kapitel 7 wird die Umsetzung der in Kapitel 6 vorgestellten Konzepte beschrieben. Das Kapitel besteht aus zwei Teilen, der Beschreibung der Clientseite und der Serverseite.
In Kapitel 8 werden die Konzepte aus Kapitel 6 auf Modellierungswerkzeuge für Scientific Workflows übertragen. Zu Beginn des Kapitels werden Screenflows und Scientific Workflows miteinander verglichen. Anschließend werden die erarbeiteten Konzepte auf den wissenschaftlichen Bereich übertragen.
In Kapitel 9 werden die Ergebnisse dieser Arbeit zusammengetragen und reflektiert. Abschließend werden Anknüpfungspunkte für weiterführende Arbeiten empfohlen.

2. Grundlagen

Die für diese Arbeit wichtigsten Technologien sind Workflows und Portale. Screenflows können sehr gut mit Workflows kombiniert werden. Screenflows eignen sich für Aktivitäten, die menschliche Handlungen erfordern und dadurch eine Benutzungsschnittstelle benötigen. Die Screenflows können einen Benutzer durch eine komplexe Benutzungsschnittstelle führen. Die in dieser Arbeit vorgestellten Screenflows nutzen bei ihrer Umsetzung Portaltechnologien. Dieses Kapitel beschreibt daher die Grundlagen der Workflow- und Portaltechnologien. Weitere Informationen sind in der angeführten Referenzliteratur zu finden.

2.1. Workflows

Ein *Prozess* besteht aus einer Reihe von Aktivitäten, die für die Erfüllung einer bestimmten Aufgabe ausgeführt werden müssen. Wenn der Ablauf des Prozesses mit sämtlichen Rahmenbedingungen ausdrücklich vorgegeben ist, existiert für den Prozess ein *Prozessmodell* [PP07].

In einem Unternehmen ist jedes Produkt das Ergebnis einer Ausführung von einer Reihe von Aktivitäten. *Geschäftsprozesse* sind das wichtigste Instrument zur Organisation dieser Aktivitäten. Gleichzeitig helfen sie ein besseres Verständnis über die Zusammenhänge der Aktivitäten zu erlangen. In [Wes07] definiert Weske einen Geschäftsprozess wie folgt:

Definition 1 *„Ein Geschäftsprozess besteht aus einer Menge von Aktivitäten, die in Abstimmung auf eine organisatorische und technische Umgebung ausgeführt werden. Diese Aktivitäten erfüllen gemeinsam das Unternehmensziel. Jeder Geschäftsprozess wird von einer einzigen Organisation ausgeführt, aber möglicherweise interagiert er mit Geschäftsprozessen, die von anderen Organisationen ausgeführt werden."*

Großmann und Koschek definieren einen Geschäftsprozess in [GK] folgendermaßen:

Definition 2 *„Ein Geschäftsprozess ist eine Verfahrensanweisung zur Bearbeitung eines Geschäftsvorfalls. Er setzt sich zusammen aus einer Folge von geordneten, fachlich zusammenhängenden Aktivitäten.*
...
Geschäftsprozesse haben einen definierten Anfang, ausgelöst durch ein Ereignis, sowie ein festgelegtes Ende. Zudem ist das Ergebnis des Geschäftsprozesses beschrieben."

Ein Geschäftsprozess besteht also aus einer Folge von Aktivitäten die ausgeführt werden, um ein Unternehmensziel zu erreichen.

Um Prozesse wiederholt ausführen zu können, werden Prozessmodelle definiert, welche als eine Art Vorlage für einen konkreten Prozess dienen. Leymann und Roller beschreiben das Geschäftsprozessmodell in [LR00] wie folgt:

„Das Prozessmodell beschreibt die Struktur eines Geschäftsprozess in der realen Welt. Es definiert alle möglichen Pfade durch den Geschäftsprozess, inklusive der Regeln, die definieren welcher Pfad gewählt werden soll und alle Aktivitäten, die ausgeführt werden müssen."

Weske definiert in [Wes07] ein Geschäftsprozessmodell folgendermaßen:

Definition 3 *„Ein Geschäftsprozessmodell besteht aus einer Menge von Aktivitätsmodellen und Ausführungsregeln zwischen ihnen.*
... "

Das *Aktivitätsmodell* ist eine allgemeine Beschreibung einer konkreten Aktivität, wie es das Prozessmodell für einen Prozess ist. Zusammengefasst beschreibt das Prozessmodell eine Menge von Aktivitäten, die in einem Geschäftsprozess ausgeführt werden sowie das Regelwerk für deren Ausführung.

Dient ein Geschäftsprozessmodell als Vorlage für einen Geschäftsprozess, wird der ausgeführte Prozess als *Prozessinstanz* bezeichnet. Eine Prozessinstanz besteht aus einer Menge von Werten, die den Ablauf innerhalb des Prozesspfads entscheiden. Eine Prozessinstanz enthält eine Menge von Aktivitätsinstanzen [LR00]. Jedes Geschäftsprozessmodell ist eine Vorlage für eine Menge von Prozessinstanzen und jedes Aktivitätsmodell eine Vorlage für eine Menge von Aktivitätsinstanzen.

Die Aktivitäten eines Geschäftsprozesses können zu unterschiedlichen Graden automatisiert sein. Dies reicht von Aktivitäten, die komplett ohne Computerunterstützung ausgeführt werden, über Aktivitäten, die teilweise mit der Hilfe von Computern durchgeführt werden, bis hin zu Aktivitäten, die vollautomatisch und autonom von Computersystemen ausgeführt werden.

2.1.1. Business Workflows

Die Teile des Prozessmodells, die von einem Computer ausgeführt werden können, werden als *Workflow Modell* (Arbeitsablaufmodell) bezeichnet. Aus einem Workflow Modell können wie bei einem Geschäftsprozessmodell Instanzen erzeugt werden. Eine Instanz von einem Workflow Modell wird als *Workflow* (Arbeitsablauf) bezeichnet. Abhängig vom Automatisierungsgrad des Geschäftsprozesses, umfasst das Workflow Modell nur einen Teil oder das gesamte Prozessmodell [Wes07]. Abbildung 2.1 veranschaulicht den Zusammenhang zwischen den Modellen und ihren Instanzen.

Weske gibt in [Wes07] für einen Workflow die folgende Definition an:

Definition 4 *„Ein Workflow ist die Automatisierung eines gesamten oder Teilgeschäftsprozesses, durch den Dokumente, Informationen oder Aufgaben von einem Teilnehmer zu einem anderen anhand einer Menge von Verfahrensregeln weitergegeben werden."*

Ein Workflow ist also ein durch einen Computer automatisierter (Teil-) Prozess.

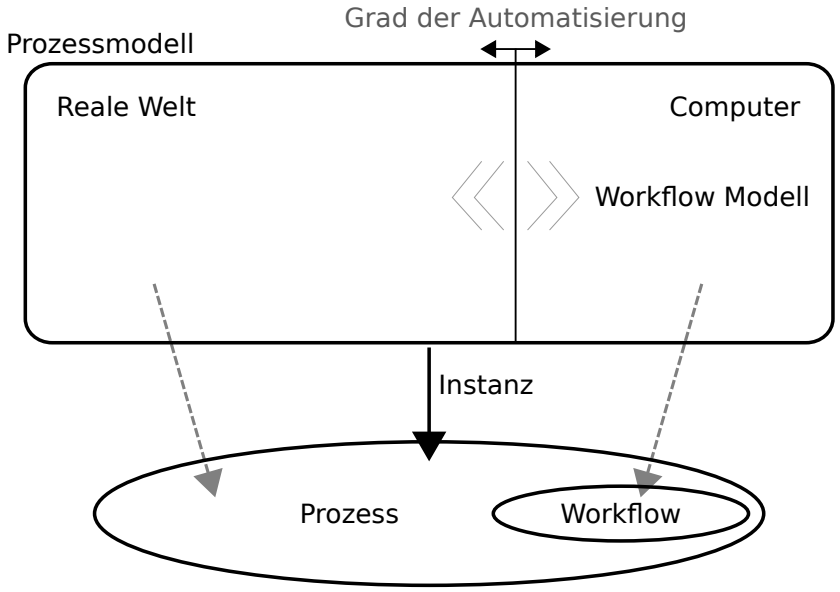

Abbildung 2.1.: Zusammenhang von Prozessmodell, Workflow Modell, Prozess und Workflow [Weso7].

Um einen Workflow auszuführen, muss dieser definiert werden. Dafür existieren unterschiedliche Sprachen und Notationen. Ein Beispiel für eine solche Sprache ist die *Business Process Execution Language for Web Services* (BPEL) [AAA+07]. Dabei handelt es sich um eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, die durch die Verwendung von *Web Services*[1] die Aktivitäten des Prozesses abbildet.

Ein Workflow wird entweder durch eine dafür extra entwickelte Anwendung oder durch eine generische Workflow Engine ausgeführt. Um den gesamten Workflow Prozess zu strukturieren und zu verwalten, werden *Workflow Management Systeme* eingesetzt. Weske definiert in [Weso7] ein Workflow Management System wie folgt:

Definition 5 *„Ein Workflow Management System ist ein Softwaresystem, das durch die Nutzung von Software Workflows definiert, erstellt und die Ausführung verwaltet. Diese führt es auf einer oder mehrerer Workflow Engines aus, die dazu in der Lange sind, die Prozessdefinition zu interpretieren. Es interagiert mit den Workflowteilnehmern und ruft benötigte IT-Werkzeuge und Anwendungen auf."*

[1] **Web Service:** ist ein Softwaresystem das für Interaktionen über ein Netzwerk von Maschine zu Maschine entworfen ist. Es besitzt ein Schnittstelle, die in einem maschinenlesbaren Format beschrieben ist (WSDL). Andere Systeme interagieren mit dem Web Service in einer vorgeschrieben Art und Weise [WCL+05].

Ein Workflow Management System unterstützt somit alle Personen, die an einem Workflow beteiligten sind. Von der Entwicklung des Workflows bis hin zu dessen Ausführung.

2.1.2. Interaktion von Workflows mit Menschen

Trotz der heutigen Möglichkeiten einen Prozess zu automatisieren, besteht in vielen Situationen die Notwendigkeit menschliche Nutzer in den Workflow mit einzubeziehen. Zum Beispiel um zu entscheiden, wie sich ein automatisiertes System in einem Fehlerfall verhalten soll [KDS+12]. In [ABD+07] werden solche Prozesse als *Human Centric Workflows* bezeichnet. Eine Aktivität in einem Workflow, die ein Mensch ausgeführt, wird als *Human Task* bezeichnet. Ein Human Task erfordert die Beurteilungen, Fähigkeiten, Entscheidungen oder das Urteilsvermögen eines am Prozess beteiligten Menschen. In der Regel wird die Aufgabe durch eine menschliche Handlung erfüllt. Der Mensch entscheidet, wann die Aufgabe als abgeschlossen gilt. Erst wenn die bearbeitete Aufgabe beendet ist, wird mit der nächsten Aktivität im Prozess fortgefahren [ABD+07]. Das Einbinden von Menschen in den Workflow und die Zuordnung von Aufgaben an sie wird als *Human Task Management* bezeichnet. Sogenannte *Worklist* oder *Tasklist Anwendungen* listen dem Benutzer die Aufgaben für die Ausführung meist in einer grafischen Oberfläche auf [SDK10].
Die Organization for the Advancement of Structured Information Standards (OASIS)[2] [OAS] hat mit der Spezifikation *WS-HumanTask* [AAD+07a] eine abstrakte Beschreibung für die Interaktion von Workflows mit Menschen entwickelt. Der Standard wurde zum Beispiel in *BPEL4People* [AAD+07b] für den BPEL Standard umgesetzt.

2.1.3. Scientific Workflows

Workflows können in unterschiedlichen Bereichen eingesetzt werden. Neben den Business Workflows, die im Unternehmensumfeld verwendet werden, wird die Workflow Technologie zunehmend auch in wissenschaftlichen Bereichen eingesetzt. Diese Workflows werden dann als *Scientific Workflows* bezeichnet.

Durch den Einsatz von Business Workflow Technologien im wissenschaftlichen Bereich, können die Wissenschaftler von den erprobten und bewährten Konzepten der Business Workflows profitieren. In [GSK+11] beschreiben Leymann et al. die Vorteile der Verwendung von Workflows im wissenschaftlichen Bereich. Zum Beispiel können die Wissenschaftler ihre Ergebnisse und Daten über Services mit anderen Wissenschaftlern teilen. So können sie dann mit Hilfe von Workflows die Ergebnisse (gemeinsam) analysieren. Workflows sind in der Lage mit großen Mengen an Daten zu arbeiten. Gerade im wissenschaftlichen Bereichen fallen oft große Datenmengen an, zum Beispiel durch die Messungen von Sensoren. Ein weiterer,

[2] **Die Organization for the Advancement of Structured Information Standards (OASIS):** OASIS ist ein internationales, non-profit Konsortium, das sich mit der Standardisierung von IT Sicherheit, Cloud Computing, SOA, Web Services, und anderen Technologien beschäftigt [OAS].

wichtiger Punkt ist, dass Workflows in verteilten und heterogenen Umgebungen ausgeführt werden können. Solche Umgebungen sind im wissenschaftlichen Bereich die Regel und können bei einer Zusammenarbeit von verschiedenen Institutionen kaum verhindert werden. Des Weiteren erlaubt die Automatisierung von Schritten beim Entwurfs und der Ausführung der Workflows dem Wissenschaftler, sich auf seine Forschungen zu konzentrieren. Zusätzlich ermöglichen Workflows unterschiedliche Verfahren zu Fehlerbehandlung.

Die Anforderungen an die Workflows unterscheiden sich in den Einsatzbereichen eines Unternehmensumfeldes und eines wissenschaftlichen Bereiches. Daher müssen die Business Workflows zusätzlich an die Bedürfnisse der Wissenschaftler angepasst werden.
Wissenschaftler sind meist keine Computerexperten und benötigen daher bedienerfreundliche Anwendungen. Während Business Workflows Management Systeme möglichst allgemein gehalten werden, um unabhängig vom Geschäftsmodell und der Infrastruktur zu sein, sind Scientific Workflows Management Systeme oft sehr spezifisch und auf einen Anwendungsbereich angepasst.
Der *Lebenszyklus* von Geschäftsprozessen im Business Process Management[3] besteht aus mehreren getrennten und wiederholbaren Phasen. Der Lebenszyklus besteht aus den Phasen Modellierung, Deployment (Installation und Konfiguration), Ausführung, Überwachung und Analyse von Workflows. Jede dieser Phasen wird von einer spezifischen Rolle bearbeitet. Dazu existieren im Workflow Management System meist spezielle Werkzeuge. In der Business Welt wird üblicherweise jede Rolle von einer anderen Person eingenommen. Bei Scientific Workflows ist der Kreis der beteiligten Personen viel kleiner. Meist wird der gesamte Lebenszyklus von einem Wissenschaftler bearbeitet. Daher müssen die Werkzeuge für die Bearbeitung der Phasen einfach und möglichst miteinander integriert sein. Abbildung 2.2 stellt die Lebenszyklen von Business Workflows und Scientific Workflow gegenüber.

Ein Wissenschaftler hat in der Regel ein exploratives Vorgehen. Er führt einen Prozess aus und beobachtet die Resultate. Unter Umständen unterbricht er die Ausführung und ändert das Modell und führt das Modell erneut aus. Daher existiert im Lebenszyklus eines wissenschaftlichen Prozesses eine Rücksprungmöglichkeit von der Ausführungs- zur Modellierungsphase [SK]. Außerdem ist das Deployment sehr technisch. Die Komplexität sollte möglichst vor dem Wissenschaftler verborgen werden. Daher sind im Lebenszyklus eines Scientific Workflows meist die Phasen Modellierung und Deployment zusammengefasst, sodass der Wissenschaftler sich nicht um das Deployment sorgen muss.

Das Deployment von Workflows ist ein sehr technischer Schritt. Dabei wird das Prozessmodell in eine ausführbare Repräsentation überführt. Wissenschaftler sind aber keine Computerexperten und sind deshalb nicht in der Lage mit der Komplexität des Deployment umzugehen. Daher werden Scientific Workflows meist ohne eine Deployment Phase ausgeführt. Stattdessen führen die Wissenschaftler die Workflows oft direkt nach der Modellierung aus. Das Deployment wird im Hintergrund automatisch ausgeführt. Bei diesem Vorgehen

[3] **Business Process Management** (BPM): Zu Business Process Management zählt das Abfragen, Analysieren, Beobachten und Reparieren von Prozessen sowie Verwalten des Verlaufs ausgeführter Prozesse und deren Ressourcen [LRoo].

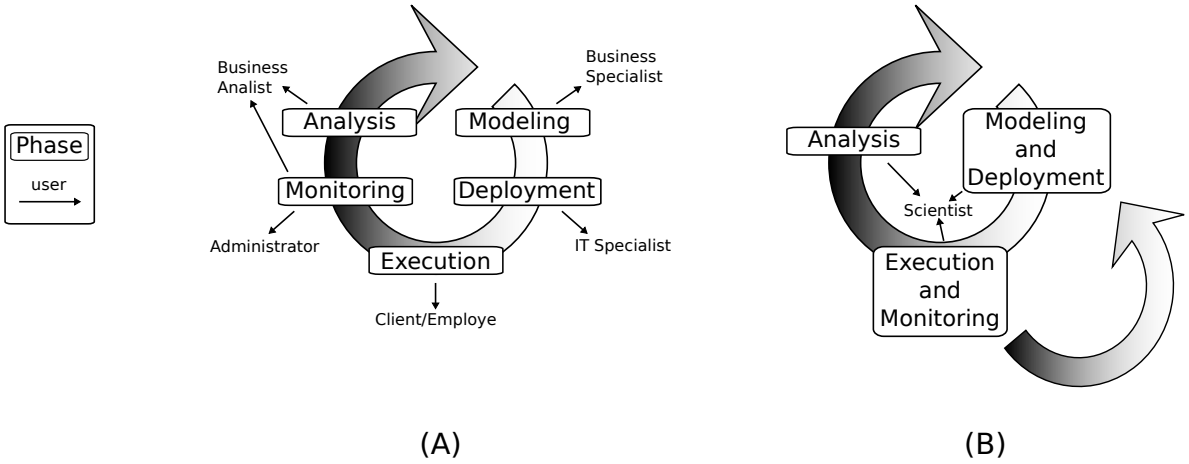

Abbildung 2.2.: (A) Lebenszyklus Business Process Management; (B) Lebenszyklus Scientific Workflow [SK].

nähern sich Wissenschaftler häufig durch Ausprobieren an eine ausführbare Lösung an [SK]. Auch wenn Business Workflows mittlerweile die Nutzung von Grids durch die Open Grid Services Architecture (OGSA) [IUH+05] und das Web Services Ressourcen-Framework (WSRF) [Ban05] erlauben, haben Scientific Workflows meist zusätzliche Bedürfnisse an die Arbeit mit Grids. Beispiele hierfür sind die Suche nach freien Ressourcen oder die Planung von Ausführungen.
Beim Business Workflow Management dient das Monitoring der Überwachung des gesamten Systemzustands. Im wissenschaftlichen Workflow Management ist es dagegen wichtig, dass der Wissenschaftler den Fortschritt seiner Berechnungen beobachten kann [SDK10].

Ein Scientific Workflow kann wie folgt definiert werden:

Definition 6 *Ein Scientific Workflow ist ein Workflow zur automatisierten Ausführung von Prozessen im wissenschaftlichen Bereich. Diese Workflows sind auf die Anforderungen der Wissenschaftler spezialisiert.*

2.2. Portale

Portale haben in den vergangen Jahren einen bedeutenden Stellenwert als Technologie im Bereich der Informationsbereitstellung erlangt. Sie entstanden Ende der 90er Jahre aus der Not heraus das das World Wide Web rasant und unkontrolliert wuchs. Nutzer des Internets benötigten Hilfe diese Flut an Informationen zu filtern, um für sie relevante Informationen zu erhalten. Einige Online-Dienst Anbieter erkannten diesen Problem und stellen auf ihren Web Seiten gezielte Informationen und Dienste bereit. Diese Seiten konnten die Nutzern dann als Ausgangspunkt bei ihrer Suche nutzen. Eine kontextbezogene Vorauswahl sollte die Informationsüberflutung der Nutzer verhindern. [GK] Großmann und Koschek definieren ein Portal wie folgt:

Definition 7 *„Ein Portal ist ein zentraler und persönlicher Einstieg (Single Point of Access) in die Informationswelt des Internet oder Intranet, von dem aus Verbindungen zu den relevanten Informationen und Diensten hergestellt werden können."*

Ein Portal ist eine Webanwendung, die als Präsentationsschicht für Informationssysteme dient. Portale werden dazu verwendet, Inhalte aus unterschiedlichen Anwendungen und Informationsquellen auf einer Seite zusammenzuführen. Zusätzlich bieten sie häufig eine personalisierte Auswahl und Darstellung der Informationen sowie eine vereinheitlichte Anmeldung und Authentifizierung bei den beteiligten Systemen an [Kuso5].

Portale können anhand von diversen Kriterien klassifiziert werden. In [GK] wird eine Klassifikation anhand des Fokus (horizontal und vertikal) und des Nutzerkreises durchgeführt. Ein *horizontales Portal* dient als Plattform für verschiedene Anwendungen. Es besitzt ein breites Informationsangebot. Ein horizontales Portal richtet sich an keine spezifische Zielgruppe. Ein Beispiel für ein horizontales Portal ist Google[4]. Ein *vertikales Portal* verfügt über eine spezielle Auswahl von Anwendungen oder Funktionen. Der Funktionsumfang eines vertikalen Portals ist spezialisiert auf die Anforderungen der Zielgruppe, für die es bereit gestellt wird. Ein Beispiel für ein vertikales Portal ist ein Portal für die Abwicklung von Projekten in einem Unternehmen. Ein *offenes Portal* ist für jeden Benutzer zugänglich. Auf ein *geschlossenes Portal* hat nur eine definierte Benutzergruppe Zugriff. Sowohl das offene Portal als auch das geschlossene Portal können über das Internet oder ein Intranet verfügbar sein.

Anhand dieser Kriterien leiten Großmann und Koschek vier Klassen von Portalen ab. Abbildung 2.3 zeigt eine Matrix mit diesen vier Klassen.

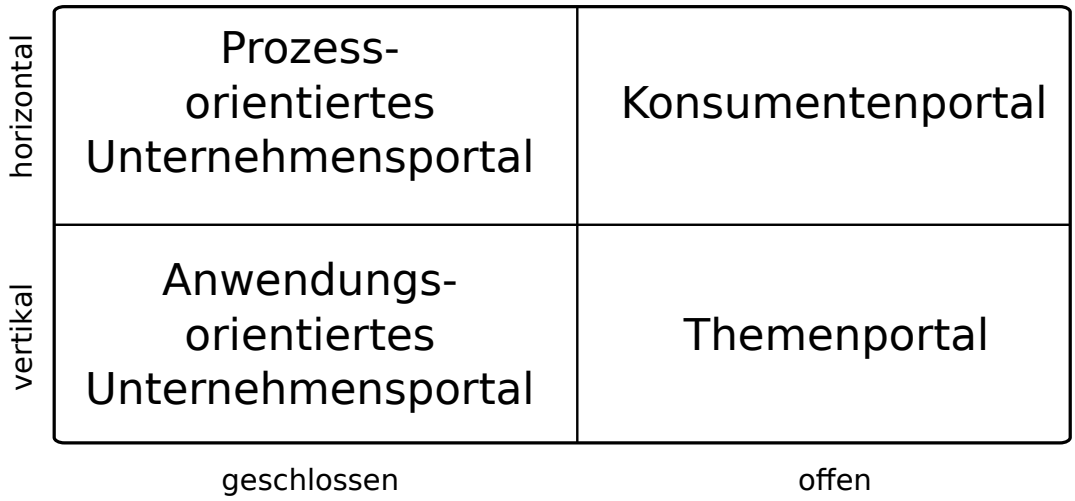

Abbildung 2.3.: Klassifikation von Portalen nach Fokus(horizontal, vertikal) und Nutzerkreis(offen, geschlossen) [GK].

[4]**Google:** http://www.google.com

„Ein *prozessorientiertes Unternehmensportal* stellt einer geschlossenen Benutzergruppe die (automatisierten) Geschäftsprozesse des Unternehmens in einer einheitlichen Ablaufumgebung zur Verfügung“ [GK]. Das *anwendungsorientiertes Unternehmensportal* integriert die Anwendungen und Datenbestände eines Unternehmens über die Benutzungsoberfläche des Portals [GK]. Ein *Konsumentenportal* ist öffentlich zugänglich und stellt Informationen und Dienste zur Verfügung. Es richtet sich aber an keine spezielle Zielgruppe. Das *Themenportal* ist wie das Konsumentenportal öffentlich zugänglich. Allerdings sind die Anwendungen und Dienste, die in dieser Klasse von Portalen angeboten werden, auf einen bestimmten Nutzerkreis ausgelegt [Weg02].

Portale stellen heutzutage eine breite Palette an Diensten bereit.
Anpassung an den Nutzer: dabei erkennt das Portal einen Benutzer und stellt spezifische Inhalte für diesen bereit. Die *Inhaltszusammenführung* dient dazu, Inhalte aus unterschiedlichen Quellen im Portal einheitlich zusammenzustellen. Die Inhalte werden dann meist in verschiedenen Ausgabeformaten angeboten. Durch *Single Sign On* muss der Benutzer am Portal nur einmal authentifiziert werden, um alle darin enthalten Anwendungen nutzen zu können. *Unterstützung verschiedener Geräte:* das Portal kann Inhalte in Abhängigkeit des verwendeten Endgerätes über verschiedene Kommunikationskanäle bereitstellen. *Portal Administration:* die Inhalte des Portals können über einen Administrationsbereich angepasst werden. Der Bereich spannt sich von der Erstellung von Benutzergruppen bis hin zum Look and Feel der Inhalte. *Portal Benutzerverwaltung:* über die der Zugang zu dem Portal verwaltet werden kann. Abhängig vom Portalsystem kann ein Portal so nur bestimmten Nutzern zugänglich gemacht werden [Weg02].

2.2.1. Portalspezifische Architektur

Ein Portal kann von unterschiedlichen Blickpunkten betrachtet werden: aus der Sicht eines *Portalnutzers*, der auf Inhalte zugreift, die ihm im Portal zur Verfügung gestellt werden, aus der Sicht des *Content Providers*, der Inhalte für einen Portalnutzer bereitstellt sowie aus der Sicht eines *Portlet Entwicklers* der die Anwendungen entwickelt, die für die Bereitstellung der Inhalte im Portal benötigt werden [Pro11]. Im Folgenden wird näher auf die Sicht des Entwicklers eingegangen.

Für die Komposition von Inhalten verwendet ein Portal eine Reihe von Komponenten. Die Inhalte eines Portals werden in Portalseiten bereitgestellt. Abbildung 2.4 beschreibt den Aufbau einer Portalseite. Die Inhalte der Portalseiten von Portlets generiert. Nauerz definiert in [Dr.12] wie folgt:

Definition 8 *„Ein Portlet ist eine Komponente (eine Anwendung) die Inhalte darstellt und Zugriff auf Dienste und Informationen liefert. Portlet Anwendungen sind Bündel von zusammengehörigen Portlets und Ressourcen, welche zusammengepackt sind. Alle Portlets die zusammen gepackt sind, teilen einen gemeinsamen Kontext, der Ressourcen wie Bildern, Konfigurationsdateien und Klassen beinhaltet.“*

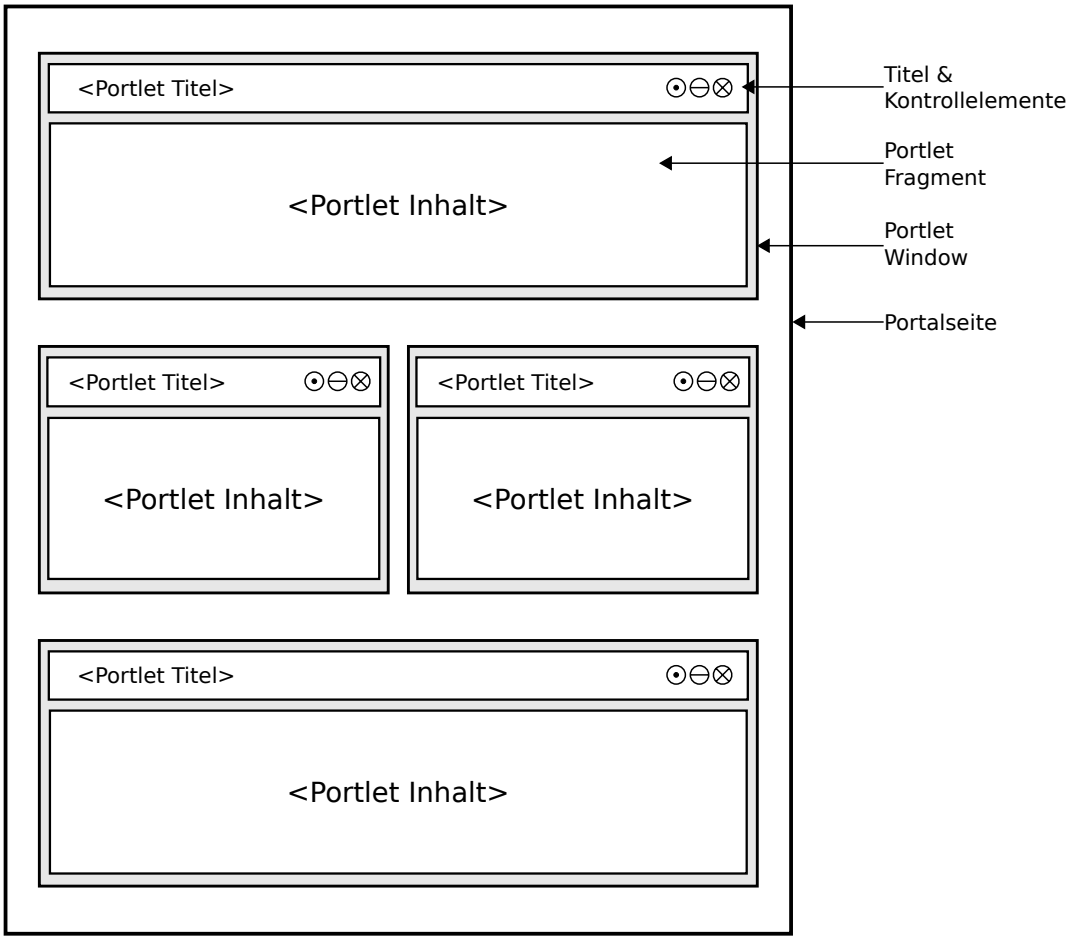

Abbildung 2.4.: Aufbau einer Portalseite [Hepo8].

Ein Portlet erzeugt keine vollständige Antwort sondern nur Markup Fragmente aus zum Beispiel HTML oder XML. Diese Fragmente werden häufig von der Portal Engine um einen Titel und Steuerelemente erweitert. Das Ergebnis wird als *Portlet Window* bezeichnet. Die Inhalte des Portlet Windows können sich abhängig von dem Client Request ändern. Portlet Windows werden in Portalseiten eingebettet. Die Portalseite dient zur Aggregation der Portlet Inhalte [Hepo8]. Nauerz definiert eine Portalseite in [Dr.12] folgendermaßen:

Definition 9 *„Eine (Portal)Seite stellt Inhalte dar. Eine Seite kann aus einem oder mehreren Portlets bestehen.*
...
In vielen Fällen definiert der Portal Administrator den Aufbau der Seite.
...“

Die fertige Portalseite wird nach der Erstellung als Antwort an den Client zurück gesendet. Abbildung 2.5 veranschaulicht diesen Prozess.

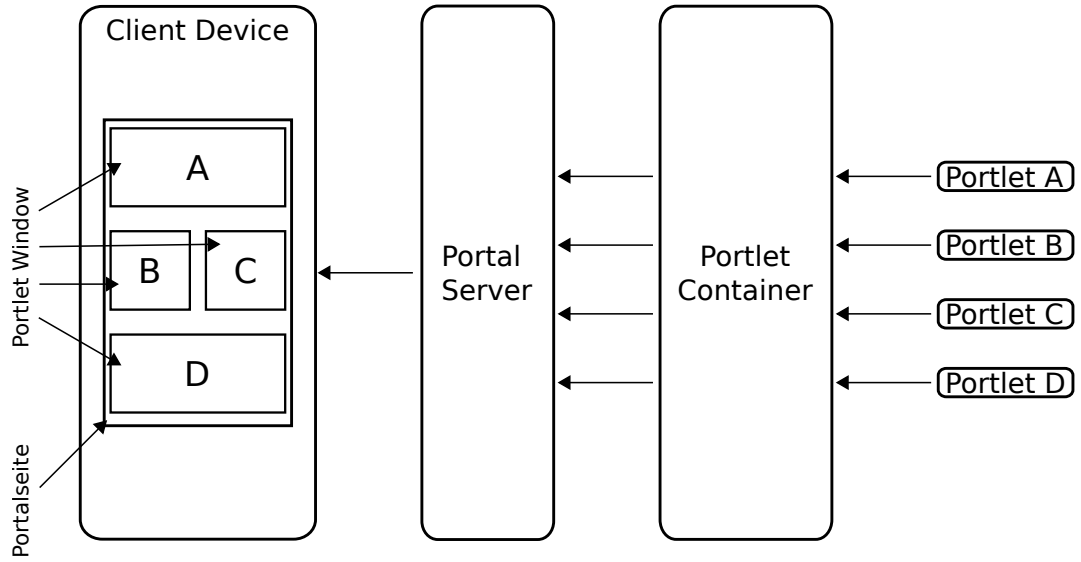

Abbildung 2.5.: Aggregation der Portlet Inhalte [Hep08].

Ein Portal besteht in der Regel aus mehreren Portalseiten, durch die ein Benutzer navigieren kann. Damit eine Navigation zwischen den Portalseiten möglich ist, müssen die Portalseiten miteinander verbunden werden. Nauerz hat dafür in [Dr.12] folgende Definition:

Definition 10 *„ Ein Navigationsmodell repräsentiert die Beziehung zwischen einzelnen Portalseiten und definiert somit die in der Regel hierarchische, Navigationsstruktur des gesamten Web Portals.“*

2.2.2. Portlets und Portlet Container

Portale werden oft als JEE[5] Anwendungen entwickelt. In JEE werden die Komponenten in speziellen *Containern* innerhalb des Application Servern bereitgestellt. Ein Container ist eine spezielle Laufzeitumgebung für die Komponenten, in der spezielle Dienste bereitstellt sind. Portlets werden zum Beispiel im Portlet Container ausgeführt. Der Portlet Container ist eine Erweiterung der Servlet Containers Spezifikation [Mor09].

Definition 11 *Ein Portlet Container führt Portlets aus und stellt ihnen die benötigte Laufzeitumgebung zur Verfügung. Ein Portlet Container verwaltet die Instanzen und den Lebenszyklus von Portlets.*

Die *Portlet API* beschreibt die Schnittstelle zwischen einem Portlet und dem Portlet Container. Neben der Portlet API muss das Portal dem Portlet weitere Dienste bereitstellen. Beispiel

[5]**JEE** (Java Plattform, Enterprise Edition): Ist eine Spezifikation für eine Middelwarearchitektur [DS]. Weitere Informationen können unter `http://www.oracle.com/technetwork/java/javaee` gefunden werden.

hierfür sind Dienste für das Persistieren von Daten oder das Abfragen von Benutzerinformationen. Diese Dienste stellt das Portal über das *Portal Service Interface* zur Verfügung. Abbildung 2.6 veranschaulicht den Aufbau einer Portal Server Anwendung.

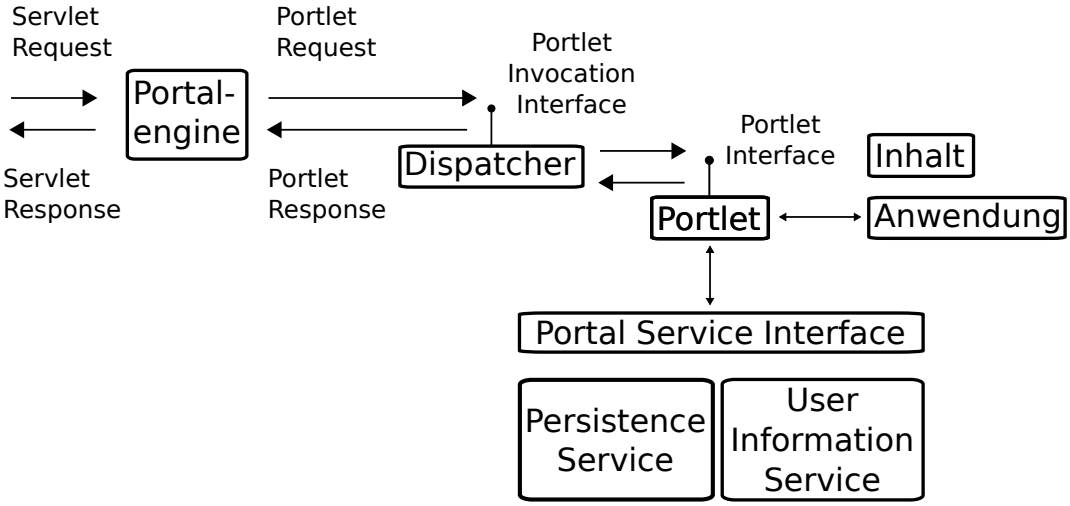

Abbildung 2.6.: Komponenten Portal Anwendung [Weg02].

Eine Portal Anwendung ist als *Servlet* implementiert. Ein Servlet ist eine Anwendung die Anfragen von einem Client entgegennehmen und beantworten kann. Die Portal Engine nimmt die Servlet Anfragen vom Servlet Container entgegen. Die Anfragen transformiert sie im Anschluss in einen *Portlet Request*. Ein Portlet Request beinhaltet zusätzliche Informationen, die für die weitere Verarbeitung von den Portlets benötigt werden. Der Portlet Request wird dann über den *Dispatcher* (deutsch: Disponent) an die entsprechenden Portlets verteilt. Die Portlets werden im Portlet Container ausgeführt und über die Portlet API angesprochen. Die Portlets können während sie ausgeführt werden über das Portal Service Interface auf Dienste des Portals zugreifen. Nachdem alle Portlets ausgeführt wurden, fügt die Portal Engine die Antworten aller Portlets zusammen und sendet sie an den Client zurück [Weg02].

2.2.3. Portlet API

Die *Portlet API* ist durch den *Java Specification Request*[6] 168 und 286 spezifiziert. Die Spezifikation [Hep08] dient als Standard für die Implementierung des Java Portlet Containers und Portlets. Durch diesen Standard können Portlets herstellerunabhängig und portierbar entwickelt werden [Weg02]. Dies ermöglicht eine Interoperabilität von Portlets auf unterschiedlichen Portal Servern.

[6]**Java Specification Requests** (JSR): ist die Beschreibung von geplanten und fertigen Spezifikationen für die Java Plattform. Die Spezifikationsanfragen werden vom Java Community Process (JCP) `http://jsp.org`, einer Gemeinschaft für die Entwicklung von Java Standards, verwaltet.

Die Portlet API beschreibt die *Request* und *Response* Objekte, welche die Anfrage an das Portlet und dessen Antwort beinhalten. Des Weiteren beschreibt die Portlet API den *Portlet Lebenszyklus*. Der Lebenszyklus ist durch die *Portlet* Schnittstelle abgebildet. Jedes Portlet muss diese Schnittstelle implementieren. Der Portal Container führt die Portlets anhand der Phasen des Lebenszyklus aus. Der Portlet Lebenszyklus beginnt mit der Initialisierung des Portlets. Anschließend verarbeitet das Portlet eingehende Anfragen (Requsts). Wenn das Portlet nicht mehr benötigt wird, wird es beendet und entfernt [Kus05].

Abbildung 2.7 veranschaulicht die Phasen des Portlet Lebenszyklus

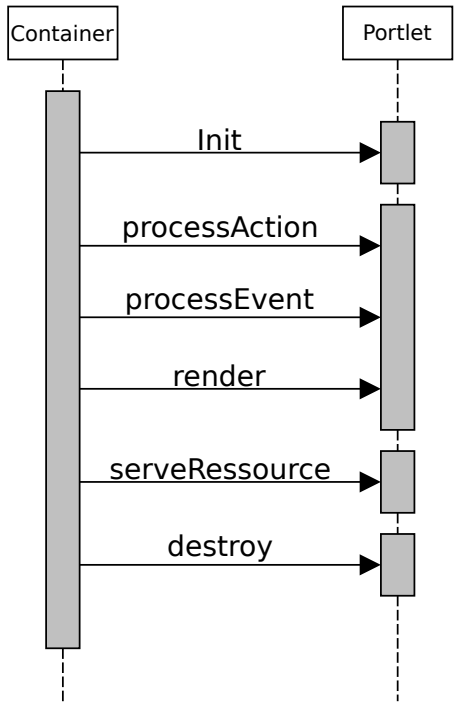

Abbildung 2.7.: Lebenszyklus eines Portlets [Hepo8].

Die *ini* Methode dient für mögliche Instanzierungen innerhalb des Portlets. Sie wird bei der Instantiierung des Portlets vom Portlet Container aufgerufen. Die *processAction* und die *render* Methode dienen für die Verarbeitung von Nutzeranfragen. Während *processAction* für Anfragen vorgesehen ist, die den Zustand des Portlets verändern, dient die *render* Methode der Erzeugung des Inhaltsfragments. Die *processEvent* Methode dient der Verarbeitung von Ereignissen, die zwischen Portlets gesendet werden können. Diese Methode ist im JSR 286 zur Portlet Spezifikation hinzugekommen. Wird die *processAction* Methode aufgerufen, führt der Portal Container auch die *processEvent* und *render* Methoden aller Portlets auf der Portalseite aus. So ist es möglich, dass Portlets sich untereinander Ereignisse senden.
Die *serveResource* Methode liefert weitere Ressourcen wie z.B. Bilder, die im Inhaltsfragment benötigt werden. Um Ressourcen auszuliefern, kann die Methode zu jedem Zeitpunkt

ausgeführt werden (solange das Portlet existiert), ohne das Portlet neu rendern zu müssen. Daher kann diese Methode auch für asynchrone Aufrufe an das Portlet genutzt werden. Wenn das Portlet nicht mehr benötigt wird, wird vom Portal Container die *destory* Methode aufgerufen. In ihr können Aufräumarbeiten implementiert werden [Hepo8].

2.3. Zusammenfassung

In diesem Kapitel wurden die für diese Arbeit wichtigsten Technologien, Workflows und Portale, vorgestellt. Ein Prozess ist eine Ausführung von Aktivitäten, um eine definiertes Ziel zu erreichen. Sind die Aktivitäten und Rahmenbedienungen für die Erreichung des Ziels definiert, folgt der Prozess einem Prozessmodell. Prozesse müssen nicht automatisiert sein. Wird ein Prozess teilweise oder ganz durch Computerunterstützung ausgeführt, wird dieser als Workflow bezeichnet. Prozesse die zur Erreichung von Unternehmenszielen eingesetzt werden, werden als Geschäftsprozesse Prozesse bezeichnet. Workflows im Unternehmensumfeld werden als Business Workflows bezeichnet. Auch im wissenschaftlichen Bereich werden Prozesse eingesetzt. Die automatisierten Prozesse werden in diesem Bereich als Scientific Workflows bezeichnet. Zwischen Business Workflows und Scientific Workflows existieren Anforderungen, die sich überschneiden, zum Beispiel das Arbeiten mit großen Datenmengen. Dennoch existieren auch Anforderungen, bei denen sich Business Workflows und Scientific Workflows unterscheiden. Ein Beispiel hierfür ist der Prozess Lebenszyklus. Während an einem Business Workflow eine ganze Reihe an Personen arbeiten (Workflow Modellierer, IT Experte, Benutzer, Analyst), arbeitet an einem Scientific Workflow meist nur ein Wissenschaftler.
Sowohl in Geschäfts- als auch in wissenschaftlichen Prozessen sind Interaktionen mit einem Benutzer erforderlich, zum Beispiel im Falle eines Fehlers. Aktivitäten, die aus solchen Interaktionen bestehen, werden als Human Tasks bezeichnet.

Portale sind Webanwendungen, die für die Integration von Informationssystemen entwickelt wurden. Sie dienen als zentraler Zugangspunkt und sind an den Benutzer anpassbar. Portale können anhand verschiedener Kriterien klassifiziert werden. Zum Beispiel anhand des Fokus (horizontal, vertikal). Ein horizontales Portal besitzt ein breites Informationsangebot, ein vertikales Portal ein spezifisches Informationsangebot. Eine anderes Möglichkeit ist die Klassifizierung nach der Benutzergruppe (offen, geschlossen). Ein Portal kann öffentlich zugänglich sein, dann handelt es sich um ein offenes Portal. Wenn das Portal nur für eine ausgewählten Benutzergruppe zugänglich ist, handelt es sich um ein geschlossenes Portal. Ein Portal besteht in der Regel aus mehrere Portalseiten. Eine Portalseite enthält ein oder mehrere Portlet Windows. Ein Portlet Window enthält die Inhalte (Fragmente), die von einem Portlet generiert werden. Ein Portlet ist eine Anwendung, die Inhalte darstellt und Zugriff auf Dienste und Informationen liefert. Portlets werden in einem Portal Container ausgeführt. Die Schnittstelle zwischen Portlet und Portlet Container ist standardisiert.

3. Screenflows

Wie im Kapitel Grundlagen beschrieben, sind auch in automatisierten Prozessen häufig noch (immer) menschliche Handlungen erforderlich. Diese werden als Human Tasks bezeichnet. Beispiele für Human Tasks sind unter anderem die Eingabe von Daten, die für den Prozess benötigt werden oder die Entscheidung, wie sich ein automatisiertes System im Fehlerfall verhalten soll. Für die Interaktion mit einem Benutzer, sind Benutzungsschnittstellen erforderlich. Im Kapitel Grundlagen wurde beschrieben, dass ein Portal als Präsentationsschicht für Informationssysteme dient. Portale sind daher ein geeignetstes Mittel für die Umsetzung von Benutzungsschnittstellen.

3.1. Benutzungsschnittstellen mit Portalen

Der Einsatz von Portalen für die Umsetzung von Benutzungsschnittstellen hat zahlreiche Vorteile. Benutzungsschnittstellen die auf Basis der Portaltechnologie erstellt werden, sind plattformunabhängig. Sie können auf allen Systemen verwendet werden, die einen Webbrowser besitzen. Eine Voraussetzung hierfür ist natürlich, dass die Inhalte der verwendeten Portlets auf HTML aufgebaut sind.
Portale ermöglichen es, komplexe Benutzungsschnittstellen zu erstellen. Je nach Bedarf können Entwickler unterschiedliche Technologien bei der Umsetzung verwenden. So lassen sich von einfachen HTML Formularen bis hin zur komplexen Webanwendungen mit einem Portlet realisieren.
Über die im Portal vorhanden Werkzuge können Benutzungsschnittstellen und Benutzer verwaltet werden. Durch eine Benutzerverwaltung innerhalb des Portal Servers kann die Authentifizierung und die Autorisierung der Benutzer geregelt werden. Ein Administrator ist für die Vergabe von Rollen und einzelnen Rechten verantwortlich. So kann der Zugriff auf die einzelnen Benutzungsschnittstellen durch die dafür vorgesehenen Nutzergruppen gesteuert werden.
Über das Portal kann auch ein einheitliches Erscheinungsbild der Benutzungsschnittstellen hergestellt werden. Das sogenannte Look and Feel kann vom Portal Administrator für das gesamte Portal eingestellt werden. Änderungen des Look and Feels können meist mit wenig Aufwand vorgenommen werden.
Wie in Kapitel 2 beschrieben, dient ein Portal als zentraler Zugangspunkt und häufig zur Integration von Anwendungen. Die Benutzer können so das Portal als zentrale Arbeitsumgebung nutzen, in der alle benötigten Anwendungen über Portlets bereitgestellt werden. Die verfügbaren Anwendungen können so bei Bedarf mit dem Workflowsystem verbunden werden [SLWM07].

3.2. Heutige Kommunikation zwischen Workflow- und Portalsystemen

Ein häufig eingesetztes Konzept für die Interaktion zwischen Workflowsystem und Benutzern, ist die sogenannte *Task List*. In einer Task List werden die Human Tasks aufgelistet, die von einer potenziellen Gruppe von Benutzern abgearbeitet werden können. Wählt einer der Benutzer den Human Task zur Bearbeitung aus, wird ihm die dafür vorgesehene Benutzungsschnittstelle angezeigt.

In Portalen kommt dieses Konzept häufig zum Einsatz. In Abbildung 3.1 wird das Zusammenspiel zwischen einer Workflow Engine, Task List Portlet und Task Page veranschaulicht.

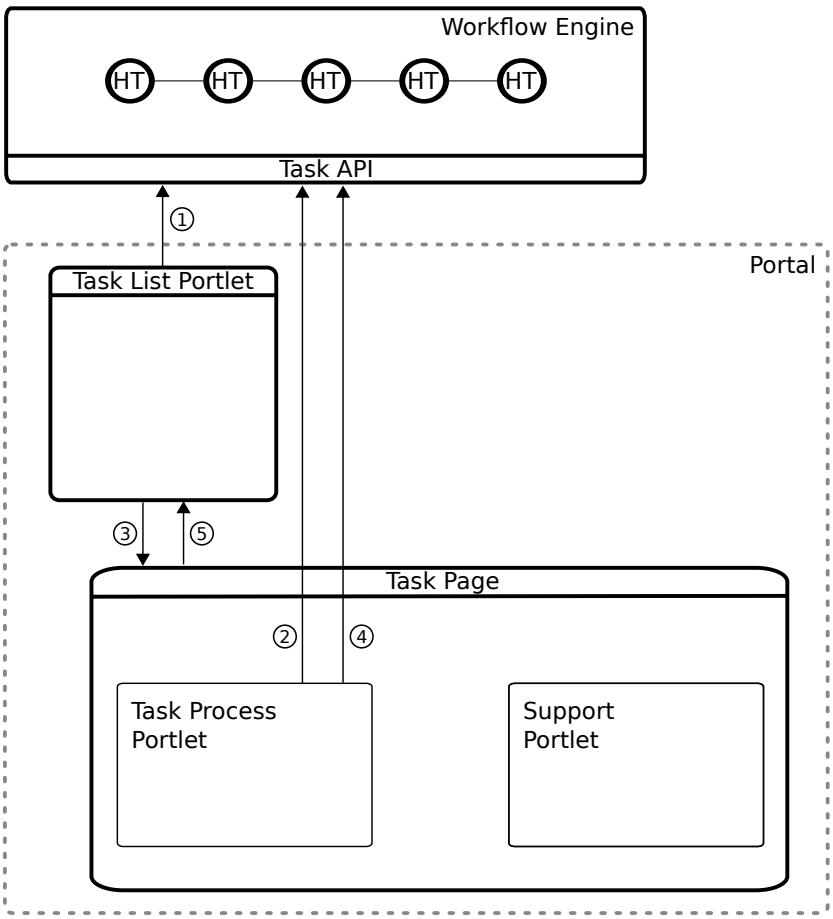

Abbildung 3.1.: Interaktion zwischen Workflow Engine und Task List Portlet.

Um die (Human) Tasks (deutsch Aufgaben) für einen Benutzer im Task List Portlet anzuzeigen, ruft das Task List Portlet alle vorhanden Human Tasks (HT) für den aktuellen Benutzer von der Task API des Workflowsystems ab (Polling) (1). Nachdem ein Task vom Benutzer angenommen (claim) wurde, wird der Taskname zu einem Link. Durch das Flogen (anklicken) des Links wird das für den Task definierte Portlet (Task Process Portlet)

instanziiert. Während diesem Aufruf übergibt das Task List Portlet dem Portlet die ID des Human Task. Bei seiner Initiierung kann das Portlet alle benötigten Daten von der Task API, des Workflowsystems abfragen (2). Nachdem die Initiierung abgeschlossen ist, wird der Benutzer auf eine Portalseite (Task Page) für die Bearbeitung des Tasks weitergeleitet (3). Die Portalseite enthält das initiierte Portlet, das für die Bearbeitung des konkreten Tasks vorgesehen ist. Zusätzlich kann sie noch weitere Portlets (Support Portlet) enthalten, die den Benutzer bei der Bearbeitung des Tasks unterstützen. Beispiele hierfür sind ein Kalender Portlet oder ein Portlet, das dem Benutzer das Abfragen von Kundendaten ermöglicht.
Nachdem die Benutzungsschnittstelle vollständig geladen ist, kann der Benutzer mir der Bearbeitung des Tasks beginnen. Wenn der Benutzer die Bearbeitung des Tasks abgeschlossen hat, werden die Ergebnisdaten über die Task API an den Prozess übergeben (4). Der Prozess kann danach mit der Verarbeitung fortfahren. Nachdem die Ergebnisdaten übertragen wurden, wird der Benutzer zurück zur Task List geleitet (5). Dort kann er dann den nächsten Task für die Bearbeitung auswählen [SLWM07]. Durch die Verwendung einer Task List für die Interaktion von Menschen mit einem Workflow werden Human Tasks auf Task Pages abgebildet.

Abbildung 3.2 veranschaulicht einen beispielhaften Ablauf einer Abarbeitung von drei Human Tasks mit Hilfe einer Task List. Erst greift die Task List aus die Task API zu. Anschließend werden nacheinander die Portlets A, B und C für die Bearbeitung der Human Tasks instanziiert.

3.3. Screenflows

Häufig wollen Modellierer von Geschäftsprozessen nicht jeden Human Task separat im Prozessmodell modellieren. Insbesondere dann nicht, wenn sie wissen, dass mehrere aufeinander folgende Tasks vom selben Benutzer ausgeführt werden. In so einem Fall ist es überflüssig, den Benutzer für jeden seiner Tasks erneut auf die Task Page zu leiten. Zusätzlich ist es für den Benutzer unkomfortabel, jeden dieser Tasks explizit aus der Task List auszuwählen (häufige Weiterleitungen zur Task List).

Um Prozesse mit Portlets abzubilden, existieren zwei Möglichkeiten. Erstens, das Schreiben eines komplexen Portlets, welches die gesamte Prozesslogik enthält. Bei diesem Ansatz kann der Nutzer durch den gesamten Prozess innerhalb des Portlets geführt werden. Nachteilig ist jedoch, dass bei jeder Änderung des Prozesses der Quellcode des Portlets angepasst werden muss. Zusätzlich ist das entwickelte Portlet durch den definierten Prozess sehr spezifisch und kann so selten wiederverwendet werden.
Zweitens, das Schreiben von mehreren einfachen, feingranularen Portlets, die gemeinsam den Prozess abbilden. Die einzelnen Portlets eignen sich so hervorragend für die Wiederverwendung. Allerdings fehlt bei diesem Ansatz eine Führung des Benutzers durch den Prozess. Es fehlt eine zentrale Komponente, die den Ablauf definiert. Die Reihenfolge in der die Portlets ausgeführt werden müssen, bleibt dem Benutzer überlassen.

Der *Screenflow Manager* vereint die Vorteile beider Ansätze. Er erlaubt es, einfache und feingranulare Portlets zu verwenden, die dann deklarativ als *Screenflow* miteinander verbunden

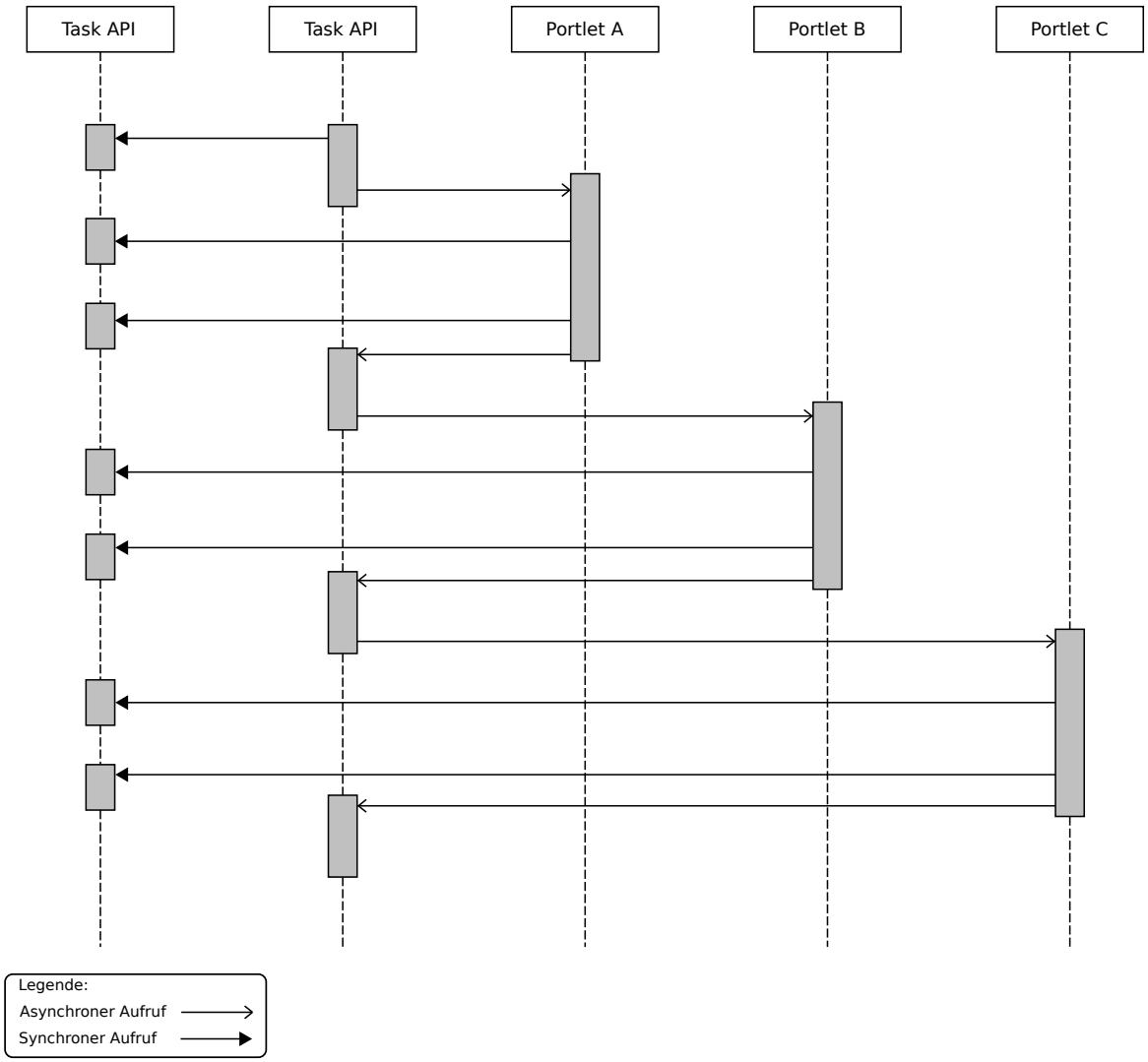

Abbildung 3.2.: Sequenzdiagramm der Ausführung von drei Human Tasks mit einer Task List.

werden. Ein Screenflow besteht aus einer Abfolge von Portalseiten, welche die miteinander verbundenen Portlets enthalten. Der Screenflow wird über die *Screenflow Definition* beschrieben. Sie definiert alle beteiligten Portlets und beschreibt alle Transitionen zwischen ihnen. In Abbildung 3.3 wird das Zusammenspiel dieser Komponenten veranschaulicht. Weitere Details können im Kapitel Screenflow Manager nachgelesen werden.

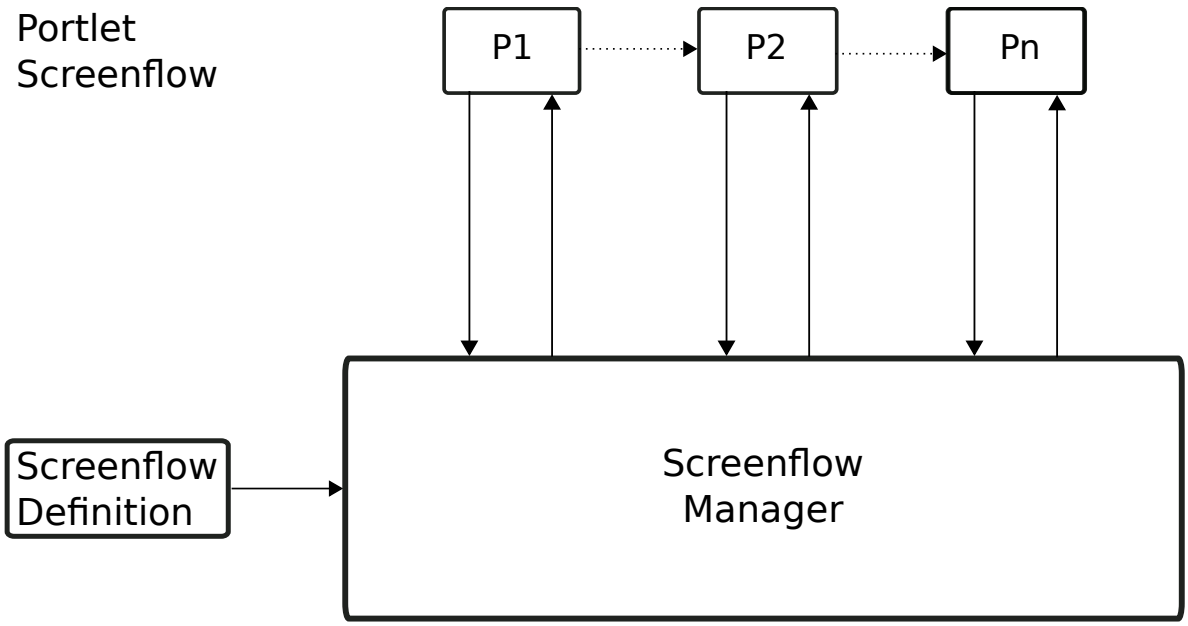

Abbildung 3.3.: Komponenten eines Screenflows.

3.4. Screenflows als Teil von Workflows

In Erweiterung dessen, was im vorhergehenden Abschnitt beschrieben ist, dass durch den Einsatz der Portal Technologie Human Tasks auf Task Pages abgebildet werden können, existiert mit dem Screenflow Manager die Möglichkeit, Human Tasks auf ganze Sequenzen von Task Pages (Screenflows) abzubilden.

Eine Möglichkeit Screenflows in den Workflow mit einzubinden, ist die Verwendung einer modifizierten Task List. Diese stellt statt einzelner Human Tasks, eine Liste von Screenflows bereit. Der Screenflow substituiert dann mehrere aufeinanderfolgende Human Tasks, die vom gleichen Benutzer abzuarbeiten sind. Durch das Klicken auf einen Listeneintrag wird von dem modifizierten Task List Portlet ein spezielles Startereignis gesendet. Dadurch wird der Screenflow Manager benachrichtigt, eine Instanz des definierten den Screenflows zu starten. Das Startereignis beinhaltet die ID des Human Tasks. Diese speichert der Screenflow Manager im *Dialog Context*[1].

Wie bei einem herkömmlichen Task List Portlet tauschen die am Screenflow beteiligten Portlets Daten mit dem Workflowsystem aus. Dabei sind unterschiedliche Ansätze möglich. Ein Ansatz ist, dass das erste Portlet alle benötigten Daten aus dem Human Task lädt und im Dialog Context abspeichert. Alle weiteren Portlets greifen dann auf den Dialog Context zu, um mit den Daten zu arbeiten. Das letzte Portlet gibt dann alle Ergebnisse an das Workflowsystem zurück und schließt den Human Task ab.

[1] **Dialog Context:** Der Dialog Context ist ein temporärer Speicher für die Screenflows. Weitere Informationen zum Dialog Context können im Kapitel Screenflow Manager nachgelesen werden.

Eine weitere Möglichkeit ist, statt das erste und das letzte Portlet für die Kommunikation mit dem Workflow System zu nutzen, *Event Mapper Klassen*[2] einzusetzen. Diese können nach dem Auftreten des Startereignisses die Daten für den Human Task aus dem Workflowsystem lesen und im Dialog Context ablegen. Die am Screenflow beteiligten Portlets greifen dann nur auf den Dialog Context zu. Nachdem der Screenflow abgeschlossen ist und das Endereignis vom letzten Portlet ausgelöst wird, gibt eine weitere Event Mapper Klasse die Daten an das Workflowsystem zurück und beendet den Human Task.
Noch ein weiterer Ansatz wäre, dass jedes Portlet selbst für das Lesen und Schreiben der Daten mit dem Workflowsystem verantwortlich ist. Dabei würde dann erst das letzte Portlet den Human Task als abgeschlossen markieren. Dieser Ansatz wird in Abbildung 3.4 veranschaulicht. Es ist auch eine Mischung aus den zuvor vorgestellten Ansätzen möglich. Dabei kommuniziert nur eine Teilmenge der am Screenflow beteiligten Portlets mit dem Workflowsystem. Portlets oder Event Mapper, die mit dem Workflowsystem kommunizieren, können auf den Dialog Context zugreifen, um die ID des Human Tasks zu erhalten und andere Daten miteinander auszutauschen.

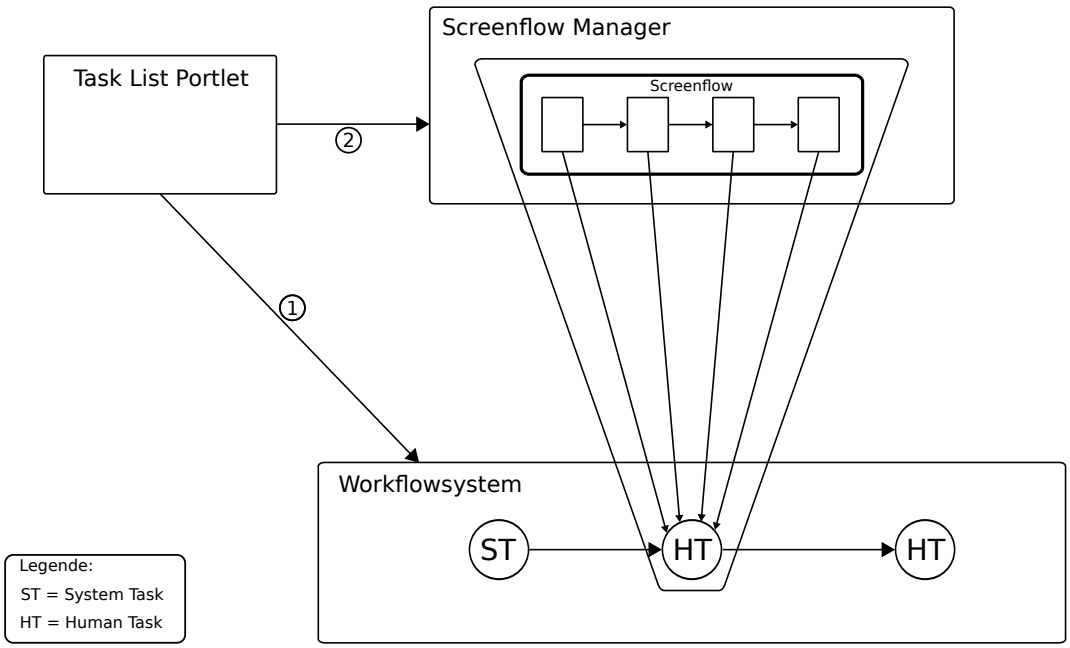

Abbildung 3.4.: Interaktion zwischen Workflow Engine und Screenflow.

[2]**Event Mapper Klasse:** Durch eine Event Mapper Klasse kann Code für die Transformation von Ereignissen ausgeführt werden. In diesem Kontext wird der Code für das Auslesen und Schreiben der Daten mit Workflowsystem verwendet werden. Weitere Informationen zu den Event Mappern können im Kapitel Screenflow Manager gefunden werden.

3.5. Zusammenfassung

Human Tasks sind Aktivitäten in einem Workflow, die menschliche Handlungen erfordern. Für die Interaktion mit Menschen werden Benutzungsschnittstellen benötigt. Portale eignen sich sehr gut für die Umsetzung von Benutzungsschnittstellen. Ein sehr verbreitetes Konzept für die Bearbeitung von Human Tasks basiert auf der Nutzung von Task Listen. Diese listen dem Benutzer alle Human Tasks auf, die von ihm bearbeitet werden sollen. Es existieren jedoch Situationen, in denen ein Prozessmodellierer nicht für jede Aktivität, die vom selben Benutzer ausgeführt wird einen Human Task im Prozessmodell definieren will. Hier wird dann der Teilprozess auf der Ebene der Benutzungsschnittstelle abgebildet. Um die Portlets, die für die Abarbeitung solcher Human Tasks verwendet werden, dennoch wiederverwendbar zu machen und eine Führung des Benutzers zu ermöglichen, wurde das Konzept des Screenflow Managers entwickelt. Der Screenflow Manager kann einen Benutzer durch eine definierte Abfolge von Portalseiten und Portlets leiten. Eine solche Abfolge wird als Screenflow bezeichnet. Ein Screenflows kann dazu eingesetzt werden, um einen Human Task auf einer Sequenz von Portalseiten abzubilden.

4. Verwandte Arbeiten

Nachdem im vorhergehenden Kapitel das Konzept der Screenflows vorgestellt wurde, handelt dieses Kapitel von verwandten Arbeiten aus diesem Bereich. Der Fokus dabei liegt auf webbasierten Screenflow- oder Dialogkontrollsystemen. Die auswählten Arbeiten haben einen theoretischen oder praktischen Bezug zu diesem Thema.

4.1. Theoretische Arbeiten

Bei den theoretischen Arbeiten handelt es sich um Arbeiten, welche lediglich Konzepte ohne konkrete Implementierung beschreiben. Arbeiten welche auch konkrete Implementierungen umfassen, werden anschließend im Abschnitt 4.2 beleuchtet.

Modellierung webbasierter Dialog Flows für eine automatische Dialogkontrolle

In [BG04] stellen Book und Gruhn ein auf MVC[1] basiertes Konzept für die Implementierung einer Dialogflusskontrolle vor. In webbasierten Anwendungen ist die Dialogkontrolllogik meist in der Präsentations- oder der Anwendungslogik enthalten. Dies erschwert die Kontrolle bei komplexen Dialogen und verhindert eine Wiederverwendung der plattformunabhängigen Anwendungslogik, da verschiedene Geräte meist unterschiedliche Anzeigemechanismen verwenden. Um diese Probleme zu lösen und eine ständige Reimplementierung der Kontrolllogik zu vermeiden, wird in dem Paper eine Lösung namens *Dialog Control Framework* vorgestellt. Das Framework kann unabhängig von der Präsentations- und der Businessschicht mehrere Dialog Flows (Screenflows) verwalten. Es verwendet für die Dialogspezifikation eine für diesen Zweck entwickelte Dialogflussnotation. Mit deren Hilfe kann ein *Dialog Graph* erstellt werden. Dieser wird dann in ein maschinenlesbares Format übertragen. Dieses Modell kann kann dann in ein objektorientiertes *Dialogflussmodell* transformiert werden, welches dann dem Framework zur Laufzeit zur Verfügung steht. Durch diese Vorgehensweise wird im Framework die Dialogkontrolle automatisiert, sodass sich die Entwickler auf Aufgaben wie die Entwicklung der Anwendungslogik, die Gestaltung der Präsentationsschicht und die Flow Definition konzentrieren können.

[1] **MVC:** Steht für Model, View und Controller. Dabei handelt es sich um ein Entwurfsmuster, welches die Benutzungsschnittstelle von der Anwendungslogik entkoppelt, wodurch die Wiederverwendbarkeit und Flexibilität erhöht wird [GHJV96].

Das in dem Paper vorgestellte Dialog Control Framework soll zwar eine grafische Notation für die Dialogdefinition verwenden, es wird jedoch nicht näher spezifiziert, welche Werkzeuge für die Modellierung existieren und wie ein Dialog Graph in ein maschinenlesbares Format transformiert wird. Es ist zwar generell eine unabhängige Definition von Screenflows möglich, es fehlen aber die grafischen Modellierungswerkzeuge, die vor allem von technisch nicht versierten Nutzern benötigt werden.

Java BluePrints

Die *Java BluePrints* [Ora] sind eine Sammlung von Werken, die eine Übersicht über die Hauptmerkmale der JEE Plattform geben und dabei dem Leser das JEE Programmiermodell näher bringen. In [Ktoo] stellen Kassem et al. eine Beispielanwendung namens *Java Pet Store* zur Veranschaulichung der JEE Technologie vor.
Der Java Pet Store ist eine Onlineshop Anwendung, welche verschiedene Bereiche im E-Business abdeckt. Der Shop besteht aus einer webbasierten Benutzungsschnittstelle, über die Kunden ihre Einkäufe im Shop tätigen können. Einer Schnittstelle für eine Administrationsanwendung, welche es den Shop Administratoren erlaubt die Bestellungen der Kunden zu verwalten und das Inventar im Shop zu pflegen. Und schließlich einer Business-to-Business Schnittstelle, über die der Shop mit seinen Lieferanten kommunizieren kann.
Ein besonderes Interesse im Zusammenhang mit dieser Arbeit besteht am webbasierten Teil der Shop Anwendung. Genauer, an der Architektur der Dialogkontrolle. Beim Design der Anwendung wurde darauf Wert gelegt, die einzelnen Screens voneinander zu entkoppeln, sodass der Shop leicht erweitert oder verändert werden kann. Dazu wurde eine Komponente namens *ScreenflowManager* eingeführt. Der ScreenflowManager ist als Zustandsautomat[2] (endlicher Automat) aufgebaut, der anhand der momentan angezeigten Seite des Shops und in Abhängigkeit von der Nutzerinteraktion die Folgeseite ermittelt. Die Übergänge sind in der ScreenflowManager Klasse fest kodiert. Das macht die Anwendung unflexibel.

Der in dieser Arbeit verwendete Screenflow Manager erlaubt im Gegensatz zum Screenflow Manager des Pet Shops, eine deklarative Beschreibung der Abläufe in einer plattformunabhängigen Beschreibungssprache. Dies hat den Vorteil, dass die Anwendung für eine Änderung eines Ablaufs nicht neu kompiliert werden muss.

4.2. Praktische Arbeiten

Bei den praktischen Arbeiten handelt es sich um Frameworks, welche die Entwickler bei der Erstellung von Webanwendungen in Bezug auf Screenflows unterstützen. Die Frameworks werden in Open Source Projekten entwickelt.

[2] **Zustandsautomat:** Ein Zustandsautomat (endlicher Automat) ist ein Modell für ein System, das aus einer endlichen Anzahl an Zuständen beseht. Ein Zustand enthält die Informationen, über die bisherigen Eingaben und über die Eingaben die notwendig sind um einen Zustandswechsel in einen Folgezustand zu erreichen [JRJ02].

Spring Web Flow

Spring Web Flow[3] ist ein Bestandteil des quelloffenen, Java basierten Spring Frameworks[4]. Ein Web Flow dient der Steuerung von Abläufen und der Kontrolle des Flusses durch die grafische Benutzungsschnittstelle einer Webapplikation. Für die Trennung zwischen der Präsentations- und der Anwendungslogik baut Spring Web Flow auf das Spring Web MVC Framework auf.

Ein Web Flow kapselt eine Reihe von Schritten, die in verschiedenen Szenarien ausgeführt werden können. Web Flows sind in sich abgeschlossen definiert, wodurch die Wiederverwendung von Teilen der Anwendung innerhalb der Anwendung und in anderen Projekten möglich wird. In Spring Web Flow besteht ein Web Flow aus einer Menge von Zuständen. Dabei repräsentiert ein Zustand einen Punkt im Fluss, an dem eine Aktion ausgeführt wird. Jeder Zustand hat eine oder mehrere Transitionen, die einen Übergang in einen Folgezustand darstellen. Eine Transition wird durch ein Ereignis ausgelöst. Ein Übergang in einen Folgezustand endet meist in einer neuen Ansicht für den Benutzer, über die er mit dem System interagieren kann. Während der Interaktion werden Ereignisse generiert. Diese generierten Ereignisse können dann zu neuen Übergängen führen, wodurch eine Art Navigation durch die Benutzungsschnittstellen entsteht.

Ein Web Flow kann mittels einer XML-basierten Sprache, der Flow Definition Language, beschrieben werden. Die wichtigsten Sprachelemente sind der *View-State*, die *Transition* und der *End-State*. Das View-State Element repräsentiert einen Schritt im Web Flow und wird mit einem Template verknüpft, welches für die Darstellung der Ansicht zuständig ist. Mittels dem Transition Element lassen sich Ereignisse, die in der Regel im View-State generiert werden, mit Zustandsübergängen verbinden. Durch den resultierenden Übergangsgraphen entsteht ein Navigationspfad zwischen den verschieden View-States. Durch das End-State Element können Endzustände im Fluss definiert werden [DVG+].

Spring bietet für die eigene Entwicklungsumgebung Spring IDE ein Werkzeug namens Spring IDE Web Flow Editor an, welches es ermöglicht, Spring Web Flows während der Designzeit grafisch zu visualisieren und zu modellieren.

Die Konzepte des Spring Web Flows haben große Ähnlichkeit zu den in dieser Arbeit verwendeten Screenflows. Im Gegensatz zur Spring Web Flow Editor Implementierung ermöglicht das in dieser Arbeit entstandene Modellierungswerkzeug eine Modellierung der Screenflows zur Laufzeit.

[3] Weitere Informationen zu Spring Web Flow können unter http://www.springsource.org/spring-web-flow gefunden werden.
[4] Die Webseite des Spring Framework ist unter http://www.springsource.org zu finden.

RIFE

RIFE[5] ist ein in Java implementiertes Framework für die Entwicklung von Webanwendungen mit einer breiten Palette an mitgelieferten Werkzeugen. Ziel des Projektes ist die Unterstützung der Entwickler von Java basierten Webanwendungen. Dabei wurde besonders darauf Wert gelegt, dass alle Arbeitsschritte während des Entwicklungszyklus voneinander getrennt werden können, damit verantwortlichen Personen sich optimal auf ihre Aufgaben konzentrieren können.

Von den zahlreichen Funktionen die RIFE bietet, ist im Zusammenhang mit dieser Arbeit besonders das Konzept des Anwendungsflusses von Interesse. RIFE trennt dabei Anwendungslogik von der Ablaufsteuerung. Durch die Verwendung von *Flow und Data Links* kann ein Ablauf und Kontrollfluss zwischen den einzelnen Seiten einer Webanwendung hergestellt werden. Um Elemente in RIFE miteinander zu verbinden, gibt es die sogenannten *Flow Links*. Damit Elemente verbunden werden können, müssen sie einen Eingang oder einen Ausgang bereitstellen. Der Ausgang eines Elements ist dann der Startpunkt eines Flow Links und der Eingang eines weiteren Elements der Endpunkt. Dabei können die Elemente auch reflexiv sein, also einen Flow Link auf sich selbst haben. Um auch Daten zwischen den Elementen austauschen zu können, müssen sogenannte *Data Links* erstellt werden. Hierfür ist es erforderlich, dass in der Elementdefinition Dateneingänge und Datenausgänge festgelegt werden. Der Data Link verbindet dann einen Datenausgang eines Elements mit dem Dateneingang eines Zielelementes [Como6].

Durch die Flow und Data Links stellt RIFE den Entwicklern ein ähnlich mächtiges Werkzeug zur Entwicklung von Screen Flows zur Seite wie den Screenflow Manager aus dieser Arbeit. Das Framework bietet für die Flow Definition jedoch kein grafisches Modellierungswerkzeug. Zusätzlich sind auf der RIFE Projektseite schon seit längerer Zeit keine Aktivitäten mehr festzustellen, sodass die Zukunft des Projekts fraglich ist.

Apache Cocoon

Apache Cocoon[6] ist ein Framework, das auf Basis von XML Dokumenten die Erstellung von Webseiten und Publikationslösungen[7] ermöglicht. Das Cocoon Projekt basiert selbst auf einer Reihe anderer Apache Projekte. Für die grundlegende Architektur von Cocoon wurde zum Beispiel das Apache Projekt Avalon[8] herangezogen, welches die Entwicklung von komponentenorientierter Software unterstützt. Cocoon ist in Java programmiert und kann lokal oder als Servlet ausgeführt werden. Innerhalb eines Servlet Containers kann Cocoon auf ankommende Requests reagieren. Cocoon ermöglicht es mittels der Basiskomponenten

[5] Weitere Informationen zu RIFE können unter `http://rifers.org` gefunden werden.
[6] Weitere Informationen zu Apache Cocoon können unter `http://cocoon.apache.org` gefunden werden.
[7] **Publikationslösung:** Herausgabe von Inhalten in unterschiedlichen Formaten für verschiedene Medienkanäle und Endgeräte.
[8] Informationen zu Apache Avalon können unter `http://avalon.apache.org` gefunden werden.

auch ohne Java Programmierung Applikationen zu erstellen. Cocoon richtet sich an Entwickler, die umfangreiche Webanwendungen entwickeln wollen, bei denen Layout und Logik voneinander getrennt sind und welche in verschiedenen Formaten zur Verfügung gestellt werden sollen.

Das Kernkonzept in Cocoon ist eine *Pipeline*, die vom Request bis zur Auslieferung der Daten durchlaufen wird. Eine auf Cocoon beruhende Anwendung besteht in der Regel aus einer Menge dieser Pipelines, die jeweils eine Anfrage verarbeiten und als Ergebnis ein Dokument, meist eine HTML-Datei, als Antwort zurückliefern. Ein XML-Datenstrom durchläuft innerhalb einer Pipeline die drei Phasen[9] Generierung, Transformation und Serialisierung. Der Pipeline Ansatz erlaubt es, verschiedene Komponenten auf eine einfache Weise zu verbinden. Die verschieden Pipelines werden in der *Sitemap*, einer zentralen Konfigurationsdatei von Cocoon definiert. Jeder Ablauf einer einzelnen Pipeline ist durch die Definition innerhalb der Sitemap bestimmt. Eine Pipeline basiert auf einer Kette von verschiedenen Komponenten, die entweder aus dem Basisumfang von Cocoon oder aus Eigenentwicklungen stammen können. Eigene Komponenten können durch eine entsprechende Konfigurierung in die Sitemap aufgenommen werden [Hee07].

Für die Dialogkontrolle verwendet Cocoon *Flowscripts* (JavaScript) und *Javaflows* (Java), dessen Ausführung mit Hilfe des Continuations[10] Konzept persistiert wird. Die Flowscripts bzw. die Javaflows haben Zugriff auf die Request Parameter und die Anwendungslogik. Sie nehmen den Request entgegen, starten die nötigen Funktionen in der Anwendungslogik und entscheiden am Ende, welche Seite als nächstes an den Client (Browser) zurückgeliefert wird. Durch den Einsatz von Continuations muss der Übergang zwischen den einzelnen Webseiten einer Anwendung innerhalb der Flowscripts und Javaflows nicht als Zustandsautomat modelliert werden [Apa12].

Die Verwendung des Continuations Konzept vereinfacht die Implementierung des Flow Managers und erleichtert die Lesbarkeit des Codes. Jedoch macht die Definition des Flows

[9] **Cocoon Pipeline:** In der ersten der Pipeline steht der Generator, der Daten einliest und diese bei Bedarf auch in XML Daten transformiert. Anschließend schickt der Generator die einzelnen XML-Tags als SAX Events (SAX = Simple API for XML) durch die Pipeline. Meist folgt in der zweiten Phase der Pipeline auf den Generator, ein Transformator. Ein Transformator wird durch SAX-Ereignisse ausgelöst und ist dann in der Lage, auf diese Events zu reagieren und mit den übergebenen XML-Daten zu arbeiten. Ein Transformator kann XML-Tags in den Datenstrom einfügen beziehungsweise entfernen oder den Inhalt auf eine andere Art und Weise manipulieren. Eine Pipeline kann mehrere Transformatoren nutzen. Ein Transformator ist auch in der Lage, Tags für nachfolgenden Transformatoren zu generieren. In der letzten der drei Phasen agiert der Serialisierer. Der Serialisierer sorgt dafür, dass die XML-Daten in ein Format umgewandelt werden, welches der Empfänger benötigt. Durch die Verwendung von XSLT-Stylesheets lassen sich eine Vielzahl von Formaten wie zum Beispiel HTML, WML oder PDF erzeugen [Hee07].

[10] **Continuation:** Eine Continuation (Fortsetzung) ist eine Funktion, die das Ergebnis der Ausführung des restlichen Programms von einem Zustand aus beschreibt [Loc12]. Das Programm kann an einer definierten Codepassage unterbrochen werden. Dabei wird der aktuelle Zustand des Speichers gesichert. Zu einem späteren Zeitpunkt kann das Programm dann an dieser Codepassage mit dem vorherigen Kontext fortgesetzt werden. Das Continuations Konzept stammt ursprünglich aus der funktionalen Programmierung. Es ermöglicht eine Art prozeduralen Ablauf des Programm. Des Weiteren bietet das Konzept eine gute Basis für die Erstellung einer Flusskontrolle, da mit den vorhanden Informationen leicht ein Folgezustand ermittelt oder ein ursprünglicher Zustand wiederhergestellt werden kann.

innerhalb der Flowscripts oder Javaflows die Entwicklung unflexibel. Die Vermischung der Kontrolllogik des Programmablauf mit der Kontrolllogik der Screenflows führt zu einer höheren Kopplung und der Screenflow ist für den Entwickler in der Regel nicht direkt ersichtlich. Wie schon die meisten vorhergehenden Arbeiten bietet Cocoon auch keine unterstützenden Werkzeuge für die Modellierung des Screenflows an.

4.3. Zusammenfassung

Nachdem zu dieser Arbeit verwandte Arbeiten vorgestellt wurden, kann zusammengefasst gesagt werden, dass alle vorgestellten Arbeiten die Umsetzung eines Screenflows ermöglichen. Alle haben dabei das Ziel, die Wiederverwendbarkeit von Komponenten zu erhöhen, mehr Flexibilität zu bieten und den Entwicklern die Arbeit mit ihren rollenspezifischen Aufgaben zu erleichtern. Bezüglich ihrer Umsetzung bestehen jedoch gewisse Unterschiede, die in Tabelle 4.1 noch einmal aufgelistet sind. Die Kriterien sind dabei, ob der Flow deklarativ oder imperativ definiert wird, auf welcher Basis der Screenflow Manager implementiert wurde und letztlich, ob im Kontext des Ablaufs Daten zwischen den Screens ausgetauscht werden können.

Arbeit	Flow Definition	Screenflow Management	Kontextdaten
Dialog Control Framework	deklarativ	Endlicher Automat	keine
Java BluePrints	imperativ	Endlicher Automat	keine
Spring Web Flow	deklarativ	Endlicher Automat	keine
RIFE	deklarativ	Endlicher Automat	Data Links
Apache Cocoon	imperativ	Continuation	Continuation

Tabelle 4.1.: Unterschiedliche Umsetzung von Screenflows in verwandten Arbeiten.

Spring Web Flow bietet als einzige Lösung ein grafisches Modellierungswerkzeug für die Screenflow Definition. Keine der in diesem Kapitel vorgestellten Arbeiten bietet ein grafisches Modellierungswerkzeug, das eine Modellierung während der Laufzeit erlaubt.

5. Screenflow Manager

Ausgeführt werden die in Kapitel 3 vorgestellten Screenflows von einem *Screenflow Manager*. Der im folgenden vorgestellte IBM UX Screenflow Manager ist eine Erweiterung für den IBM WebSphere Portal Server. Als Quelle diente die technischen Dokumentation des IBM UX Screenflow Manager [DL13].

5.1. Terminologie

Bei einem Screenflow Manager handelt es sich um eine Technologie, mit deren Hilfe Benutzungsschnittstellenartefakte miteinander verbunden werden können, um Endbenutzer durch eine Folge von Screens zu führen. Eine solche Abfolge von Screens wird als *Screenflow* oder *Dialog* bezeichnet. Ein Screenflow kann aus einem oder vielen Schritten bestehen. Ein einzelner Schritt in einem Screenflow wird als *Subdialog* bezeichnet. Im Portalumfeld repräsentiert ein Subdialog eine Portalseite oder ein Portlet. Ein Portlet kann wiederum eine Reihe von Portlet Windows darstellen. Abbildung 5.1 stellt den Zusammenhang zwischen einem Dialogschrit (Subdialog), einer Portalseite und einem Portlet dar.

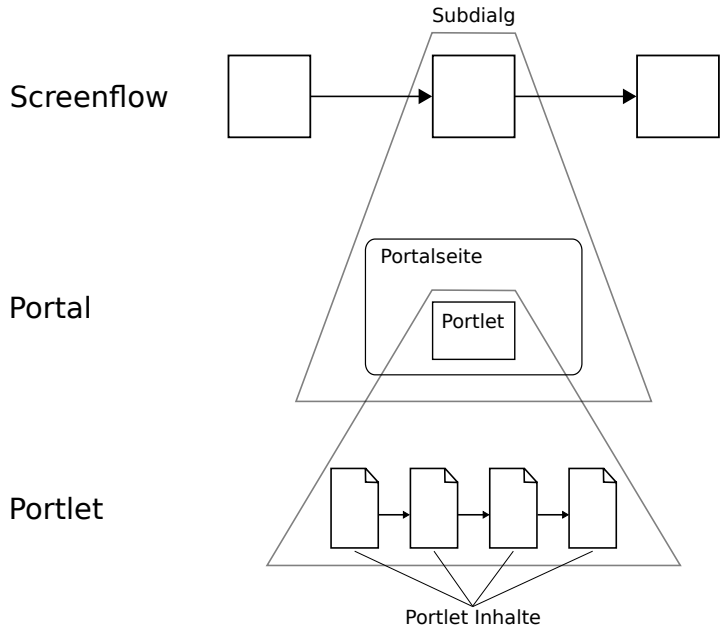

Abbildung 5.1.: Beispiel für einen Subdialog in einem Screenflow.

Über Transitionen werden die Übergänge zwischen den einzelnen Subdialogen definiert. Der Screenflow Manager kann dadurch entscheiden, wie er den Benutzer von einem Subdialog zum nächsten leitet bzw. wie er von einem Schritt im Screenflow zum nächsten routen muss. Der aktuell aktive Schritt im Screenflow legt fest, welchen Subdialog der Benutzer zu diesem Zeitpunkt angezeigt bekommt.

5.2. Kernkomponenten

Der Kern des IBM UX Screenflow Manager besteht aus drei Komponenten. Deren Zusammenspiel wird in Abbildung 5.2 veranschaulicht.

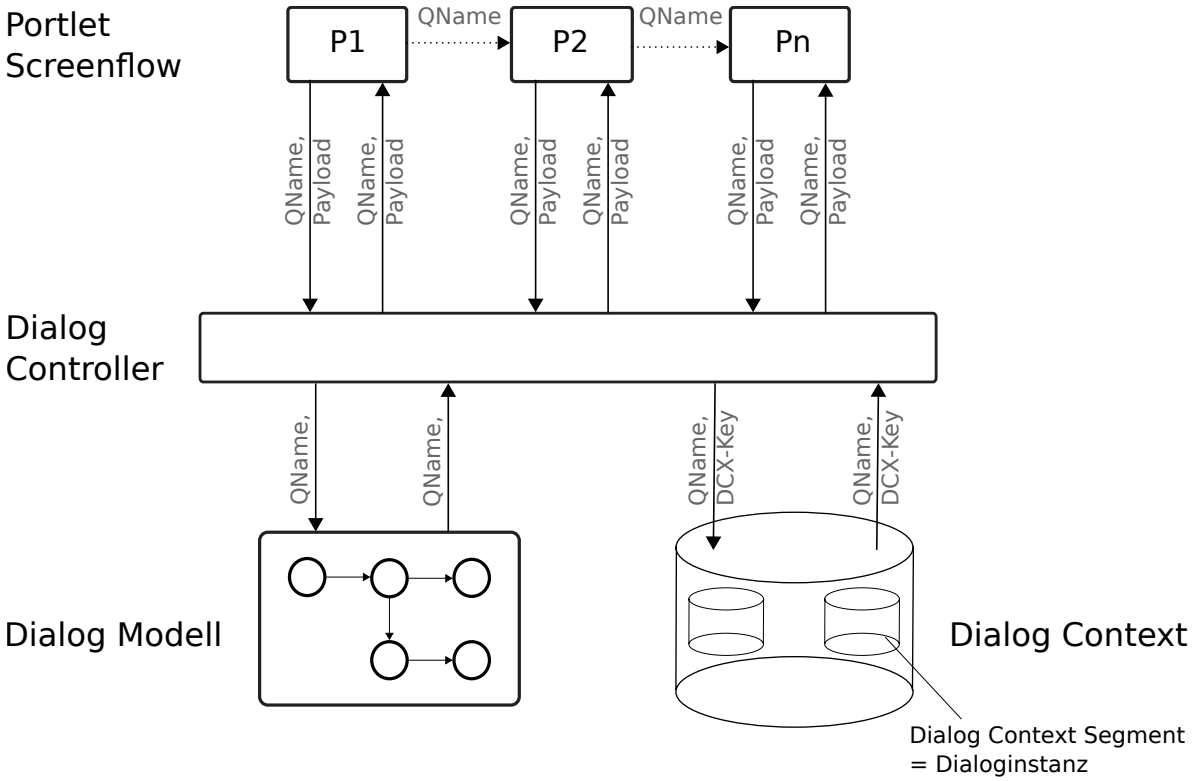

Abbildung 5.2.: Kernkomponenten des IBM UX Screenflow Manager.

Der *Dialog Controller (DC)* arbeitet als generische Komponente, die beliebige Ereignisse innerhalb eines Zustandsübergang senden und empfangen kann. Normalerweise tauschen Portlets JSR 286 Ereignisse untereinander aus. Der Dialog Controller kann diese Ereignisse abfangen und dient dann als Vermittler zwischen der Quelle und dem Ziel. Bei einem Zustandswechsel empfängt der Dialog Controller das Ereignis, welches von der Quelle gesendet wird. Damit kann der Dialog Contoller das Dialog Modell nach dem nächsten Schritt im Screenflow fragen. Im Anschluss kann der Controller Daten im Dialog Context ablegen und anfordern. Als Letztes sendet er dann das vom Dialog Modell festgelegte Ereignis mit den aus dem Dialog Context geladenen Daten an das Ziel.

Das *Dialog Modell (DM)* verhält sich wie ein endlicher Zustandsautomat. Es hat die Informationen über die Transitionen, die in Abhängigkeit des aktiven Schritts im Screenflow und Ereignis möglich sind. Es beschreibt welche Portlets als Quelle (Source) und welche Portlets oder Portalseiten als Ziel (Target) definiert sind und was für Ereignisse sie miteinander austauschen. Die Portalseiten und Portlets werden anhand ihres eindeutigen Namens, den Unique Name, identifiziert. Weitere Details sind im Abschnitt Dialogdefinition beschrieben.

Der *Dialog Context (DCX)* dient dem Screenflow Manager als flüchtiger Speicher. Er speichert Kontextinformationen, die von einem Subdialog an einen nachfolgenden Subdialog weitergegeben werden. Alle Daten von einem Subdialog, die der Dialog Controller im Dialog Context ablegt, sind im Anschluss für alle nachfolgenden Subdialoge verfügbar. Für jede Dialoginstanz wird ein sogenanntes Dialog Context Segment angelegt, in dem die Daten aus der konkreten Dialoginstanz abgelegt werden. Das Dialog Context Segment existiert nur solange wie die Dialoginstanz existiert (oder die Session[1]). Der Dialog Context ist ein assoziatives Datenfeld[2] , auf dessen Inhalte über den sogenannten *DCX-Key* zugegriffen werden kann. Weitere Details sind im Abschnitt erweitertes Laden und Speichern zu finden.

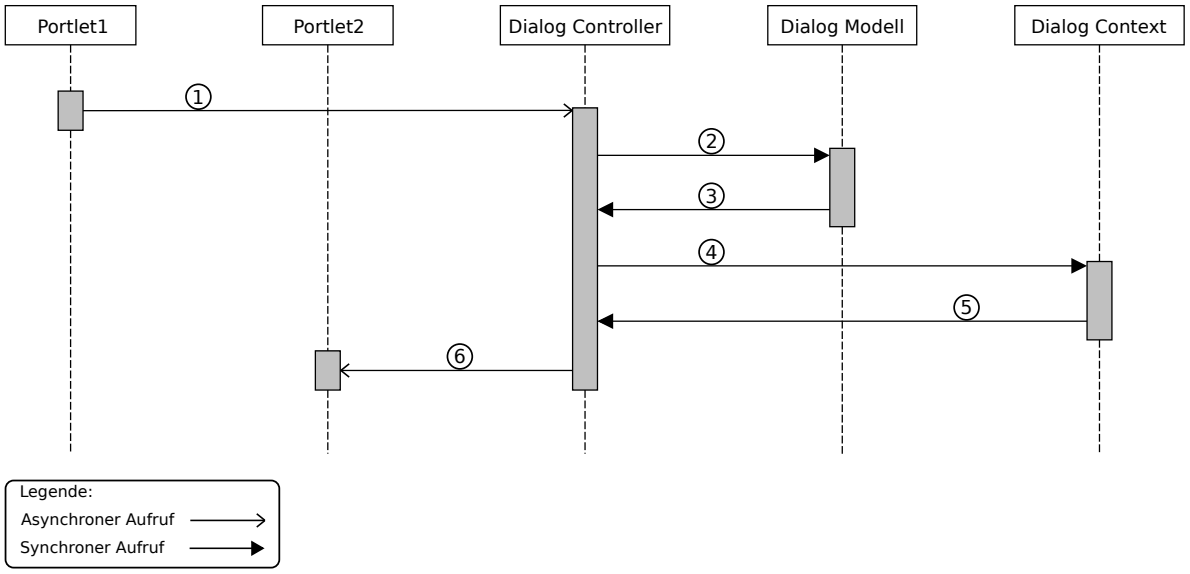

Abbildung 5.3.: Sequenzdiagramm eines Zustandsübergangs im Screenflow Manager.

[1] **Session:** HTTP ist als zustandsloses Protokoll implementiert, was bedeutet, dass sich der Server nach dem Verbindungsabbau im selben Zustand befindet wie vor dem Verbindungsaufbau. Das hat zur Folge, dass alle Informationen über die Anfrage für den Server verloren gehen. Ein bewährtes Mittel ist die Verwendung einer Session (deutsch Sitzung). Dabei wird ein Cookie auf der Clientseite abgelegt, in dem die Zustandsinformationen enthalten sind. Das Cookie wird dann bei jeder Anfrage an den Server mitgesendet [Hei02].

[2] **Assoziatives Datenfeld:** Ein assoziatives Datenfeld ist eine Datenstruktur bei der jedem gespeicherten Wert ein eindeutiger Schlüssel zugewiesen wird. Über den Schlüssel kann in der Datenstruktur sehr effizient nach Werten gesucht werden [MS].

Abbildung 5.3 veranschaulicht den Ablauf eines Zustandsübergangs innerhalb des Screenflow Managers. Portlet1 sendet das Ereignis e1 an das Portlet2 und der Dialog Controller fängt dieses Ereignis e1 ab (1). Anschließend fragt der Dialog Controller das Dialog Modell, ob eine Transition existiert, dessen Quelle ein Ereignis e1 sendet (2). Existiert eine entsprechende Transition mit einer entsprechenden Quelle im Dialog Modell, erhält der Dialog Controller das Ziel und das Ereignis, an das er weiterleiten soll (3). In diesem Fall das Portlet2 und das Ereignis e2. Danach legt der Dialog Controller die Nutzdaten von Ereignis e1 im Dialog Context unter dem DCX-Key ke1 ab (4). Anschließend lädt der Dialog Controller die Nutzdaten, die mittels Ereignis e2 übertragen werden sollen, mit dem DCX-Key ke2 aus dem Dialog Context (5). Die Nutzdaten können aus einem beliebigen vorhergehenden Schritt stammen. Letztlich leitet der Dialog Controller das Ereignis e2 an Portlet2 weiter (6).

Weitere Komponenten

Die Möglichkeiten einer parallelen Verarbeitung von Screenflows durch den Benutzer sind beschränkt. In einem einzelnen Browser Tab kann nur eine Instanz eines Screenflows auf einmal ausgeführt werden. Um mehrere Instanzen parallel zu auszuführen, müssen diese in unterschiedlichen Browser Tabs gestartet werden. Es gilt die Regel: ein Screenflow pro Browser Tab. Als Alternative dazu erlaubt der IBM UX Screenflow Manager einen Screenflow zu unterbrechen und zu einem späteren Zeitpunkt weiter zu bearbeiten. Alle unterbrochen Screenflows können im *Dialog Stack (DS)* eingesehen werden. Von dort aus lassen sich die Screenflows an der zuvor unterbrochen Position fortsetzen. Der Screenflow Manager unterbricht einen laufenden Screenflow automatisch, wenn im selben Browsers Tab ein weiterer Screenflow gestartet wird.

Der *Dialog State Display* ermöglicht es dem Benutzer während der Ausführung eines Screenflows seine aktuelle Position im Dialog anzuzeigen. So kann er ablesen, wie viele Dialogschritte er schon abgearbeitet hat und welche Schritte noch vor ihm liegen. Das Dialog State Display visualisiert dem Benutzer die Struktur des Screenflows in einem Graphen und hebt dabei die aktuelle Position im Dialog hervor. Des Weiteren kann der Benutzer über das Dialog State Display zwischen den einzelnen Dialogschritten springen. Zusätzlich kann der Benutzer einen gerade ausgeführten Screenflow über das Dialog State Display abbrechen oder wie zuvor erwähnt unterbrechen.

Benutzungsschnittstellenartefakte

Der Screenflow Manager verwendet JSR 286 Ereignisse, um einen ausgelösten Zustandsübergang zu erkennen und um Daten zwischen den Subdialogen auszutauschen. Alle Benutzungsschnittstellenartefakte, die in den Screenflow integriert werden sollen, müssen JSR 286 Ereignisse senden oder empfangen können. Das bedeutet, dass die Portlets so entwickelt werden müssen, dass sie die Java Portlet Spezifikation 2.0 [Hepo8] erfüllen. Für

Formulare und Widgets bedeutet dies, dass ein Portlet als Wrapper-Klasse[3] verwendet werden muss.

Damit ein Portlet Ereignisse senden und empfangen kann, müssen die Ereignisse in der portlet.xml[4] definiert werden. Des Weiteren müssen die Methoden, um eingehende Ereignisse zu verarbeiten oder Ereignisse zu senden, im Quelltext des Portlet implementiert werden.

5.3. Dialogdefinition

Der folgende Abschnitt beschreibt die wichtigsten Elemente der Dialogdefinition, die für die Modellierung eines Screenflows notwendig sind. Die Dialogdefinition dient dem Dialog Modell für die Erstellung des endlichen Zustandsautomaten. Eine komplette Dialogdefinition für einen Screenflow, der einen Reisebuchungsprozess beschreibt, befindet sich im Anhang unter A.1.

Dialog

Innerhalb eines *dialog* Elements sind alle Artefakte eines Dialogs enthalten. Dazu gehören alle Ressourcen und alle Transitionen die festlegen, wie der Benutzer durch den Dialog geleitet wird (siehe Abschnitt Transition Endpoints und Abschnitt Transitionen). Eine Dialogdefinition muss einen eindeutigen Namen besitzen. Im Beispiel der Auflistung 5.1 werden zwei Portlets (Zeile 3-8) und zwei Transitionen (Zeile 9-14) im Dialog festgelegt.

[3] **Wrapper-Klasse:** Ein Adapter, auch Wrapper genannt, dient dazu, eine Schnittstelle in eine andere zu übersetzen. Durch den Adapter können Klassen trotz inkompatibler Schnittstellen miteinander kommunizieren [GHJV96].

[4] **portlet.xml:** Die Datei portlet.xml ist einer von zwei Deployment Descriptoren einer Portletanwendung. Darin werden Ressourcen für das Portlet spezifiziert [Hepo8].

Listing 5.1 Dialogdefinition: Ausschnitt einer Definition eines Dialogs mit zwei Portlets und zwei Transitionen [DL13].

```
1  <dialog-set>
2    <dialog name="dialog1">
3      <transition-endpoint name="portlet1">
4           ...
5      </transition-endpoint>
6      <transition-endpoint name="portlet2">
7           ...
8      </transition-endpoint>
9      <transition>
10          ...
11     </transition>
12     <transition>
13          ...
14     </transition>
15   </dialog>
16 </dialog-set>
```

Transition Endpoints

Alle Ressourcen eines Screenflows, die als Quelle oder Ziel an einer Transition beteiligt sind, werden als Transition Endpoints bezeichnet. Über das *transition-endpoint* Element können Portalseiten und Portlets als Ressourcen eingebunden werden. Diese können, wie bereits zuvor beschrieben, auch Formulare und Widgets enthalten. Jede Ressource muss dabei mit einem eindeutigen Namen identifiziert werden. Die Auflistung in 5.2 zeigt die Definition eines Transition Endpoints, der ein Portlet referenziert (Zeile 2-10).

Listing 5.2 Dialogdefinition: Ausschnitt einer Definition eines *transition-endpoint* Elements [DL13].

```
1  <dialog name="dialog1">
2    <transition-endpoint name="portlet1">
3      <localedata locale="en">
4        <title>Subdialog 1</title>
5        <description>This is a subdialog</description>
6      </localedata>
7      <resource uniquename="uniquename.portlet1"/>
8      <invocation type="static"/>
9      ...
10   </transition-endpoint>
11 ...
```

Titel und Beschreibung

Portlets und Portalseiten können normalerweise nur einen *Titel* und eine *Beschreibung* pro Sprache besitzen. Wenn Ressourcen mehrfach versendet werden, kann es dadurch

zu dem Problem kommen, dass der Titel und die Beschreibung nicht in den Kontext passen. Angenommen ein Kalenderportlet soll für eine Reisebuchungsanwendung sowohl für die Auswahl des Abflugdatum in einem Dialogschritt als auch für die Auswahl des Rückflugdatum in einem späteren Dialogschritt verwendet werden. Hier stellt sich die Frage, wie der Titel und die Beschreibung für das Portlet zu wählen sind, damit es für den Benutzer klar ist, wann er das Abflugdatum und wann das Rückflugdatum auswählen muss.

Zur Lösung dieses Problem können im *transition-endpoint* Element für jede Ressource ein Titel und eine Beschreibung pro Sprache definiert werden. Der definierte Titel und die Beschreibung ersetzen dann im zugehörigen Dialogschritt und im Dialog State Display den Titel und die Beschreibung der Ressource. Es ist absolut valide zwei Transition Endpoints zu definieren, die zwar dieselbe Ressource referenzieren, aber unterschiedliche Titel und Beschreibungen erhalten. Sind kein Title oder keine Beschreibung im Transition Endpoint für die entsprechende Sprache definiert, verwendet der Screenflow Manager den ursprünglichen Title oder Beschreibung der referenzierten Ressource. In Auflistung 5.3 werden kontextabhängig der Title und die Beschreibung für ein Kalenderportlet für die Sprachen Deutsch und Englisch definiert (Zeile 3-10 und Zeile 15-22).

Listing 5.3 Dialogdefinition: Ausschnitt einer Definition von Title und Beschreibung eines Portlets in unterschiedlichen Kontexten [DL13].

```
1  <dialog name="dialog1">
2    <transition-endpoint name="calendar.leave">
3      <localedata locale="en">
4        <title>Date to leave</title>
5        <description>Please specify the date to leave</description>
6      </localedata>
7      <localedata locale="de">
8        <title>Abreisedatum</title>
9        <description>Bitte geben Sie Ihr Abreisedatum an</description>
10     </localedata>
11     <resource uniquename="uniquename.calendar"/>
12     <invocation type="static"/>
13   </transition-endpoint>
14   <transition-endpoint name="calendar.return">
15     <localedata locale="en">
16       <title>Date to return</title>
17       <description>Please specify the date to return</description>
18     </localedata>
19     <localedata locale="de">
20       <title>Rckreisedatum</title>
21       <description>Bitte geben Sie Ihr Rckreisedatum an</description>
22     </localedata>
23     <resource uniquename="uniquename.calendar"/>
24     <invocation type="static"/>
25   </transition-endpoint>
26   ...
```

Transitionen

Eine *Transition* beschreibt einen Übergang von einem Dialogschritt zum nächsten. Abbildung 5.4 veranschaulicht alle Bestandteile einer Transition von einem Portlets A zu Portlet B. Eine Transition besteht aus zwei Teilen, einer Quelle (Source) und einem Ziel (Target). Quellen und Ziele beinhalten Tupel aus Referenzen zu Transition Endpoints und Ereignissen. Die Quelle ist der Ausgangspunkt einer Transition.

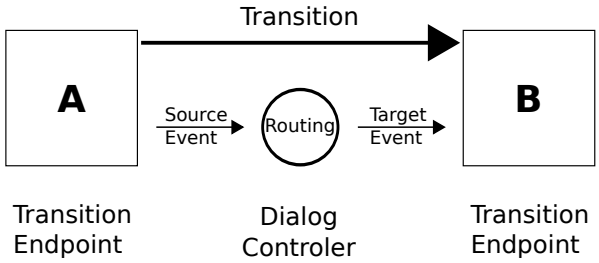

Abbildung 5.4.: Beispielhafte Darstellung einer Transition im Screenflow zwischen zwei Portlets A und B.

In einer Quelle darf nur eine Referenz auf einen Transition Endpoint existieren, der auf ein Portlet zeigen muss. Die Restriktion ergibt sich daraus, dass Portalseiten keine Ereignisse senden können. Die Referenz des Quell Transition Endpoint darf nur mit einem Ereignis verbunden werden. Um deterministisch zu sein, darf innerhalb aller Transitionen eine Kombination aus Transition Endpoint und Ereignis nur einmal für eine Quelle verwendet werden. Wenn das definierte Ereignis von dem Transition Endpoint gesendet wird, wird die Transition aktiv. Das Ziel repräsentiert den Folgezustand. Im Ziel dürfen mehrere Transition Endpoints referenziert werden, die sowohl auf Portalseiten als auch auf Portlets referenzieren können. Folgende Kombinationen von Transition Endpoints sind bei der Definition des Ziels möglich:

- Ein einzelnes Portlet
- Eine einzelne Portalseite
- Mehrere Portlets
- Mehrere Portlets und eine Portalseite

Ist eine Portalseite als Ziel Transition Endpoint definiert, wird das ankommende Ereignis an alle Portlets verteilt, die auf der Portalseite vorhanden sind. Im Falle dass mehrere Portlets oder mehrere Portlets und eine Portalseite als Ziel referenziert werden, muss dafür Sorge getragen werden, dass alle Portlets auf der entsprechenden Portalseite enthalten sind. Zu jedem Ziel Transition Endpoint können jeweils ein oder mehrere Ereignisse definiert werden.

Das Beispiel in Auflistung 5.4 demonstriert die Definition zweier Transitionen. Die erste Transition hat ein Portlet (Zeile 8-12) als Ziel und die zweite Transition eine Portalseite (Zeile 20-24).

Listing 5.4 Dialogdefinition: Ausschnitt einer Definition von zwei Transitionen mit Portlet bzw. Portalseite als Ziel [DL13].

```
1  ...
2  <transition>
3    <source>
4      <transition-endpoint nameref="portlet1">
5        <event qname="e1"/>
6      </transition-endpoint>
7    </source>
8    <target>
9      <transition-endpoint nameref="portlet2">
10       <event qname="e2"/>
11     </transition-endpoint>
12   </target>
13 </transition>
14 <transition>
15   <source>
16     <transition-endpoint nameref="portlet1">
17       <event qname="e3"/>
18     </transition-endpoint>
19   </source>
20   <target>
21     <transition-endpoint nameref="page2">
22       <event qname="e4"/>
23     </transition-endpoint>
24   </target>
25 </transition>
26 ...
```

Nicht jede Transition in der Dialogdefinition kann eine neue Dialoginstanz starten oder einen laufenden Dialog beenden. Das hat den Grund, dass sonst nicht entscheidbar ist, wie in gewissen Situationen weiter vorzugehen ist. Wenn zum Beispiel jede Transition einen Dialog starten kann, ist es nicht immer entscheidbar, wann eine Transition als Übergang in den nächsten Dialogschritt und wann sie die Instantiierung einer neuen Dialoginstanz startet. Wenn keine Dialoginstanz existiert, ist dies zwar noch entscheidbar, nicht aber wenn bereits die Dialoginstanz ausgeführt wird. Daher muss für Start- und Endtransitionen das *transition* Element mit einem speziellen Attribut erweitert werden. Für eine Starttransition ist es das Attribut *type* mit dem Wert *start* und bei einer Endtransition das Attribut *type* mit dem Wert *end* vorgesehen. Jeder Dialog muss mindestens eine Start- und mindestens eine Endtransition besitzen. Auflistung 5.5 zeigt einen Ausschnitt der Definition einer Start- und einer Endtransition (Zeile 2 und 5).

Die Sonderformen der Start- und Endtransitionen mit ihren zugehörigen speziellen Ereignissen sind nicht Teil dieser Arbeit und werden daher nicht weiter ausgeführt. Ein solches spezielles Ereignis kann zum Beispiel dafür definiert werden, um zu bestimmen, wie es nach dem Ende eines Dialoges weitergehen soll.

Listing 5.5 Dialogdefinition: Ausschnitt der Definition einer Start- und Endtransition [DL13].

```
1 ...
2 <transition type="start">
3   ...
4 </transition>
5 <transition type="end">
6   ...
7 </transition>
8 ...
```

5.4. Erweitertes Laden und Speichern

Der IBM UX Screenflow Manager unterstützt zwei Mechanismen, um zueinander inkompatible Portlets miteinander zu kommunizieren, ohne dass deren Quellcode geändert werden muss. Sind lediglich die Namen der auszutauschenden Ereignisse unterschiedlich, kann dies durch die Verwendung der sogenannten DCX-Keys korrigiert werden. Müssen dagegen die Daten der Ereignisse transformiert werden, sind dazu Event Mapper nötig, die im nachfolgenden Abschnitt 5.5 vorgestellt werden.
Mit Hilfe des DCX-Keys kann bestimmt werden, wie die Daten im Dialog Context abgelegt werden sollen, nachdem ein Portlet ein Ereignis sendet und wie Daten aus dem Dialog Context gelesen werden, um sie an das nächste Ziel zu senden. Die Kontrolle dafür übernimmt der Dialog Controller. Wird kein DCX-Key festgelegt, wird der QName des Ereignisses stattdessen verwendet. Alle Daten, die aus einem Subdialog im Dialog Context abgelegt werden, sind für die nachfolgen Subdialoge zugreifbar.

Angenommen zwei Portlets sind Teil eines Screenflows. Portlet1 sendet das Ereignis mit dem QName ID und Portlet2 erwartet ein Ereignis mit dem QName UserID. Ohne Hilfsmittel können die beiden Portlets nicht miteinander kommunizieren, da die QNames nicht übereinstimmen. Durch die Verwendung des DCX-Key ID kann Portlet2 dazu veranlasst werden, die Daten von dem Ereignis mit dem QName ID aus dem Dialog Context zu lesen. Alternativ kann durch den Einsatz von DCX-Key UserID Portlet1 dazu veranlasst werden, die Daten unter dem DCX-Key UserID im Dialog Context abzuspeichern. Die Auflistung 5.6 zeigt einen Ausschnitt für den Fall, das Portlet1 die Daten des Ereignis mit dem QName ID unter dem DCX-Key userID im Dialog Context ablegt (Zeile 5). Dadurch kann Portlet2 die Daten lesen (Zeile 10).

5.5. Event Mapper

In komplexeren Situationen ist es oft nicht ausreichend, den Ereignisnamen zwischen zwei Ereignissen anzupassen. Stattdessen ist es erforderlich, die Daten zwischen den inkompatiblen Ereignissen zu transformieren. Für solche Situationen können dem Screenflow Manager sogenannte Event Mapper zur Verfügung gestellt werden.

Listing 5.6 Dialogdefinition: Ausschnitt der Definition einer Transition in der die Daten des gesendeten Ereignis unter einem anderen DCX-Key abgelegt werden [DL13].

```
1  ...
2  <transition>
3    <source>
4      <transition-endpoint nameref="portlet1">
5        <event qname="ID" dcx-key="userID"/>
6      </transition-endpoint>
7    </source>
8    <target>
9      <transition-endpoint nameref="portlet2">
10       <event qname="userID"/>
11     </transition-endpoint>
12   </target>
13 </transition>
14 ...
```

Es stehen zwei Typen von Event Mapper zur Verfügung:
Der *PayloadToContextMapper:* er kann die Daten eines Ereignisses bearbeiten, die eine Quelle gesendet hat, bevor diese im Dialog Context abgelegt werden. Dadurch lassen sich sowohl der Name des DCX-Key unter dem die Daten abgelegt werden als auch die Daten selbst, für alle Transition Endpoints, die anschließend auf diese Daten zugreifen, verändern.
Der *ContextToPayloadMapper:* er kann die Daten eines Ereignisses bearbeiten, die an ein Ziel weitergeleitet werden, nachdem diese aus dem Dialog Context geladen werden. Dadurch bleiben die Daten innerhalb des Dialog Context in ihrem Ursprungszustand und werden nur für den entsprechenden Ziel Transition Endpoint verändert. Auch bei diesem Event Mapper lassen sich DCX-Key und die Daten selbst ändern.

Mapper haben vollen Zugriff auf das Dialog Context Segment der aktuell bearbeiteten Dialoginstanz sowie auf die gerade gesendeten Daten. So sind auch sehr komplexe und dialogübergreifende Transformationen möglich. Über den DCX-Key kann gesteuert werden, auf welche Daten im Dialog Context zugegriffen werden soll. Die Mapper müssen in Java implementiert und vor ihrer Verwendung auf dem Portal Server installiert werden. Der Ausschnitt aus Auflistung 5.7 zeigt die Definition einer Transition, in der ein ContextToPayloadMapper verwendet wird (Zeile 10). Der Mapper erhält als Eingabe die Daten des DCX-Key ID.

5.6. Dynamische Ressource Endpoints

Der IBM UX Screenflow Manager unterstützt zusätzlich zu den statischen Ressourcen auch das Instantiieren von dynamischen Ressourcen. Dafür verwendet er eine Funktion namens *Dynamic UI Management* [IBM]. Durch das Dynamic UI Management lassen sich flüchtige Kopien von Portalseiten und Portlets zur Laufzeit erstellen. Der Vorteil einer solchen dynamischen Kopie ist, dass von einem Portlet oder einer Portalseite mehrere Instanzen parallel betrieben werden können (in unterschiedlichen Browser Tabs oder über den Dialog

Listing 5.7 Dialogdefinition: Ausschnitt der Definition einer Transition die einen ContextToPayloadMapper verwendet [DL13].

```
1  ...
2  <transition>
3    <source>
4      <transition-endpoint nameref="portlet1">
5        <event qname="ID"/>
6      </transition-endpoint>
7    </source>
8    <target>
9      <transition-endpoint nameref="portlet2">
10       <event qname="userID" dcx-key="ID" mapper-class="myPackage.myMapper"/>
11     </transition-endpoint>
12   </target>
13 </transition>
14 ...
```

Stack), ohne dass die Inhalte aus den unterschiedlichen Instanzen gegenseitig überschrieben werden.

Die dynamische Kopie einer Portalseite ist in der Regel das Abbild einer *Templateseite*. Die Templateseite wird auch als Base Page bezeichnet und die dynamische Kopie kann als eine Art Schnappschuss der Base Page gesehen werden. Die dynamische Kopie enthält alle Portlets, die sich auf der Base Page befinden, einschließlich der gesamten Einstellungen. Wird eine dynamische Kopie einer Portalseite angelegt, muss diese unter einem sogenannten Extension Node eingefügt werden. Der Extension Node ist ein spezielles Konstrukt, um eine dynamische Kopie in das Navigationsmodell (siehe Kapitel 2) der Webanwendung aufzunehmen.

Die dynamische Kopie eines Portlets ist die Kopie der Portletdefinition. Dynamische Portlets können nur zu einer Dynamischen Page hinzugefügt werden. Das bedeutet: wenn eine dynamische Kopie eines Portlets erzeugt wird, muss vorher eine leere dynamische Kopie einer Portalseite erzeugt werden, auf der das Portlet eingebettet werden kann.

Während der Verarbeitung eines Dialoges können einzelne Subdialoge entweder statisch oder dynamisch sein. Wird eine Transition aktiv, die auf einen Transition Endpoint zeigt, der eine dynamische Kopie ist, muss der Screenflow Manager die Kopie erst instantiieren und unter dem entsprechenden Extension Node einhängen, bevor er den Benutzer auf die Seite weiterleiten kann. Dynamische Ressourcen werden automatisch entfernt, wenn sie nicht mehr benötigt werden.

Ob eine Ressource dynamisch gestartet werden soll, kann über die Dialogdefinition gesteuert werden. Hier kann auch festgelegt werden unter welchem Extention Node die dynamische Kopie hinzugefügt werden soll. Das Beispiel in der Auflistung 5.8 zeigt die Definition eines statischen Portlets (Zeile 4), eines dynamischen Portlets (Zeile 8) und einer dynamischen Portalseite (Zeile 12) als Ressource Endpoint, die unter dem Extension Node extensionNode1 eingehängt werden.

Listing 5.8 Dialogdefinition: Ausschnitt der Definition eines statischen und zwei dynamischen Ressource Endpoints.

```
1  ...
2  <transition-endpoint name="portlet1">
3    <resource uniquename="uniquename.portlet1"/>
4    <invocation type="static"/>
5  </transition-endpoint>
6  <transition-endpoint name="portlet2">
7    <resource uniquename="uniquename.portlet2"/>
8    <invocation type="dynamic" extension-node="extensionNode1"/>
9  </transition-endpoint>
10 <transition-endpoint name="page1">
11   <resource uniquename="uniquename.page1"/>
12   <invocation type="dynamic" extension-node="extensionNode1"/>
13 </transition-endpoint>
14 ...
```

5.7. Entwicklung von Screenflows

Für die Entwicklung eines Screenflows sind die folgenden Schritte nötig. Erstens, die Entwicklung der benötigten Benutzungsschnittstellenartefakte, in der Regel Portlets. Diese senden und empfangen JSR 286 Ereignisse, aus der Java Portlet Spezifikation 2.0. Alle Artefakte zusammen ergeben den späteren Screenflow. Zweitens, die Erstellung einer Dialogdefinition, in der alle Verbindungen zwischen Benutzungsschnittstellenartefakten festgelegt werden. Die Definition kann sowohl Artefakte aus Schritt 1 als auch bereits vorhandene Artefakte enthalten. Drittens, die Installation der neuen Benutzungsschnittstellenartefakte und der Dialogdefinition.

5.8. Akteure und Rollen

Bei der Entwicklung von Screenflows lassen sich die Aufgaben auf drei Arten von Akteuren oder Rollen aufteilen.

- Dem Entwickler von Portlets und Portalseiten: er erstellt die Benutzungsschnittstellenartefakte. Dabei kann es sich um einen Mitarbeiter oder um Drittanbieter handeln.
- Dem Dialogmodellierer: er erstellt den Screenflow, indem er die Benutzungsschnittstellenartefakte zusammenfügt. Er hat das Wissen über den zu modellierenden Prozess und benötigt für seine Arbeit keine Programmierkenntnisse.
- Dem Administrator: er administriert den Portal Server, installiert neue Benutzungsschnittstellenartefakte und neue Dialogdefinitionen.

5.9. Zusammenfassung

Der IBM UX Screenflow Manager erlaubt es, einen Benutzer durch einen Dialog (Screenflow), bestehend aus Portlets und Portalseiten, zu führen. Die Kernkomponenten des Screenflow Managers sind der Dialog Controller, das Dialog Modell und der Dialog Context. Der Dialog Controller leitet den Benutzer durch den Dialog anhand des Dialog Modells. Daten die Dialoge miteinander austauschen, werden im Dialog Context temporär abgelegt.
Das Dialog Modell wird in der Dialogdefinition festgelegt. Die wichtigsten Komponenten der Dialogdefinition sind Transition Endpoints und Transitionen. Ein Transition Endpoint repräsentiert eine Portalseite oder ein Portlet. Eine Transition definiert in einem Dialog einen Übergang von einem Transition Endpoint zu einem anderen.
Die zwischen den Dialogschritten ausgetauschten Ereignisse müssen häufig angepasst werden. Je nach Situation eignet sich dafür ein geänderter DCX-Key oder ein Event Mapper. Neben statischen Ressourcen kann der Screenflow Manager auch dynamische Ressourcen referenzieren. Diese existieren nur temporär, solange sie benötigt werden.

6. Konzept

Nachdem im letzten Kapitel die Konzepte und Funktionsweise des Screenflow Manager vorgestellt wurden, widmet sich dieses Kapitel der Entwicklung eines grafisches Modellierungswerkzeug für Screenflows.

6.1. Ausgangssituation

Bisher müssen Modellierer einen Screenflow für den IBM UX Screenflow Manager in einer XML-basierten Auszeichnungssprache definieren. Für die Definition eines Screenflows muss der Modellierer die eindeutigen Namen der Portlets und Portalseiten kennen, welche Teil des Screenflows sein sollen. Dies sind jedoch Daten, die vom Portal Server intern verwaltet werden. Für einen Portalnutzer sind diese Daten nicht direkt zugänglich. Bei einem großen Screenflow kann es für einen Modellierer schwierig werden, aus der Definition den exakten Ablauf des Screenflows abzulesen.

Dieses Kapitel stellt daher Konzepte für die Entwicklung eines Modellierungswerkzeugs vor, welches auch einen technisch nicht versierten Modellierer in die Lage versetzt, einen Screenflow grafisch zu modellieren. Das Modellierungswerkzeug setzt auf dem IBM UX Screenflow Manager auf.

Die genauen Anforderungen an das Modellierungswerkzeug werden im nachfolgenden Abschnitt festgelegt.

6.2. Anforderungen

Im folgenden Abschnitt werden die Anforderungen aufgelistet, die für die Erstellung eines grafischen Modellierungswerkzeugs für Screenflows notwendig sind. Die Anforderungen basieren auf dem in Kapitel Screenflow Manager vorgestellten Screenflow Manager.

Funktionale Anforderungen

Die folgenden funktionalen Anforderungen lassen sich direkt aus der Funktionalität des Screenflow Managers ableiten.

Dialogdefinition erstellen Zu den elementarsten Bestandteilen eines Screenflows gehört die Dialogdefinition. Sie bildet den Rahmen für alle Elemente eines Screenflows. Wenn der Modellierer einen Screenflow erstellen will, muss er in die Lage versetzt werden, eine neue Dialogdefinition zu erstellen.

Dialogdefinition konfigurieren Um eine Dialogdefinition exakt identifizieren zu können, muss sie einen eindeutigen Namen erhalten. Zusätzlich kann der Modellierer einer Dialogdefinition einen Titel und eine Beschreibung mitgeben. So kann zum Beispiel eine detaillierte Beschreibung zum Screenflow angegeben werden. Um diese Eigenschaften bearbeiten zu können, ist es notwendig, eine Dialogdefinition konfigurieren zu können.

Dialogartefakte hinzufügen Wie bereit beschrieben besteht ein Screenflow aus einer Reihe von Screens. Das Modellierungswerkzeug muss es dem Modellierer ermöglichen, Portalseiten und Portlets auszuwählen, um diese dem Screenflow als Transition Endpoint hinzuzufügen. Die hinzugefügten Transition Endpoints müssen für den Modellierer visuell dargestellt werden.

Dialogartefakte konfigurieren Jeder Transition Endpoint kann mit zusätzlichen Eigenschaften ausgestattet werden. Dies erfordert, dass der Modellierer ein Dialogartefakt konfigurieren kann. Er muss für den Transition Endpoint für alle unterstützen Sprachen einen Titel und eine Beschreibung eingeben können, die im entsprechenden Dialogschritt den Originalen-Titel und -Beschreibung des Transition Endpoints substituieren. Des Weiteren muss der Modellierer die Möglichkeit besitzen festzulegen, ob der Transition Endpoint statisch oder als dynamische Kopie geladen werden soll. Im Falle dass der Modellierer eine dynamische Kopie verwendet, muss er für den Transition Endpoint einen Extension Node angeben können.

Transition definieren Um einen Übergang im Screenflow von einem Subdialog zum nächsten zu modellieren, werden Transitionen eingesetzt. Eine Transition beschreibt den Übergang von einem Quell Transition Endpoint zu einem Ziel Transition Endpoint. Das Modellierungswerkzeug muss dem Modellierer ein Mittel bereitstellen, mit dem er Transitionen erstellen kann.

Transitionen konfigurieren Der Screenflow Manager benötigt für jede Transition die Angabe eines Ereignisses, welches die Transition aktiviert, wenn es vom Quell Transition Endpoint gesendet wird und die Definition des Ereignisses, das der Ziel Transition Endpoint erwartet. Der Modellierer muss mit dem Modellierungswerkzeug diese Ereignisse für jede Transition vergeben können. In Erweiterung dazu muss der Modellierer die DCX-Keys für die Ereignisse, die von dem Quell und Ziel Transition Endpoint der Transition gesendet werden, anpassen können. Dafür benötigt er im Modellierungswerkzeug eine Eingabemöglichkeit der DCX-Keys. Auch die Zuweisung eines PayloadToContextMappers und eines

ContentToPayloadMappers, für die Transformation der Ereignisse einer Transition muss das Modellierungswerkzeug ermöglichen. Letztlich muss der Modellierer noch bei jeder Transition markieren können, ob es sich hierbei um eine Start- oder Endtransition handelt, die den Screenflow startet oder beendet.

Zusätzlich haben sich die folgenden funktionalen Anforderungen aus Gesprächen mit den Entwicklern des Screenflow Managers und aus Designmeetings entwickelt.

Dialogdefinition speichern Nachdem der Modellierer den Modellierungsprozess abgeschlossen hat, muss er in der Lage sein, den modellierten Dialog zu speichern. Das Modellierungswerkzeug muss die Eingabe eines eindeutigen Namens sowie die Eingabe eines Titels und einer Beschreibung für den Dialog unterstützen.

Liste der vorhanden Dialogdefinitionen anzeigen Das Modellierungswerkzeug muss dem Modellierer alle vorhanden Dialogdefinitionen auflisten können. Von dieser Liste aus soll der Modellierer eine Reihe von Operationen auf die Dialogdefinitionen anwenden können. Die möglichen Operationen werden in den nachfolgenden Punkten beschrieben.

Dialogdefinition anzeigen Die aus der vorhergehenden Anforderung aufgelisteten Dialogdefinitionen müssen von Modellierer eingesehen werden können. Dafür muss das Modellierungswerkzeug eine vorhandene Dialogdefinition öffnen können, damit sie vom Modellierer betrachtet werden kann.

Dialogdefinition bearbeiten Neben dem Öffnen muss das Modellierungswerkzeug auch das Bearbeiten von Dialogdefinitionen erlauben. Eine geöffnete Dialogdefinition muss vom Modellierer verändert und unter dem gleichen oder einem neuen Namen abgespeichert werden können.

Dialogdefinition kopieren Der Modellierer muss aus einer Vorlage, zum Beispiel durch das Klonen einer vorhanden Dialogdefinition, eine neue Dialogdefinition erstellen können. Auf die neu erstellten Dialogdefinitionen sollen dann die selben Operationen anwendbar sein wie auf die bereits vorhanden. Das Modellierungswerkzeug muss daher das Kopieren von Dialogdefinitionen beherrschen.

Dialogdefinition löschen Dialogdefinitionen die nicht weiter benötigt werden, müssen aus dem System gelöscht werden können. Das Modellierungswerkzeug sollte daher die Möglichkeit bieten eine oder mehrere ausgewählte Dialogdefinitionen zu löschen.

Dialogdefinition exportieren Um Dialogdefinitionen außerhalb des Portal Servers zu speichern und um die Dialogdefinitionen zwischen unterschiedlichen Servern austauschen zu können, besteht die Anforderung an das Modellierungswerkzeug, eine Funktion für das Exportieren von Dialogdefinition bereitzustellen. Mittels dieser Funktion soll der Modellierer beliebige Dialogdefinitionen exportieren können.

Dialogdefinition importieren Im Zuge der vorhergehenden Anforderung, muss das Modellierungswerkzeug eine exportierte Dialogdefinition auch in ein System landen zu können. Der Modellierer muss in die Lage versetzt werden, den Pfad zu einer exportierten Dialogdefinition angeben zu können, um diese zu den bestehenden Dialogdefinitionen in dem Portal Server zu laden.

Nichtfunktionale Anforderungen

Neben den funktionalen Anforderungen existieren auch nichtfunktionale Anforderungen, die sich aus der Aufgabenstellung aus Kapitel 1 und Gesprächen mit den Entwicklern des Screenflow Managers ergeben. Das Modellierungswerkzeug soll für den Modellierer einfach und intuitiv zu bedienen sein. Zusätzlich soll das Modellierungswerkzeug für neue Funktionen einfach zu erweitern sein. Des Weiteren soll die grafische Oberfläche flüssig bedient werden können.

Anwendungsfälle

Aus den funktionalen Anforderungen lassen sich Anwendungsfälle formulieren. Sie beschreiben aus der Sicht eines Außenstehenden, welche Akteure beteiligt sind, wie sie mit dem System interagieren und was das System dabei leisten soll. Jeder Anwendungsfall beschreibt das Systemverhalten für eine spezielle Situation. Anwendungsfälle können dabei helfen, einen Prototypen zu entwickeln, Testfälle für das System zu spezifizieren, die Entwicklung zu planen und die Benutzerdokumentation zu erstellen [PP07]. Im Anhang A.2 befinden sich die Anwendungsfälle für die oben beschrieben Anforderungen.

6.3. Lösungsansatz

Wie die Anforderungen aus dem letzten Abschnitt einzeln umgesetzt werden können, wird im Folgenden beschrieben. Das Vorgehen dabei ist Top Down vom Generellen ins Detail.

6.3.1. Technische Integration

Dieser Absatz beschreibt, wie das Modellierungswerkzeug in die vorhandene Architektur integriert werden kann. Da der gesamte Screenflow Manager auf die Verwendung im Portal ausgelegt ist, liegt es nahe, auch das Modellierungswerkzeug als Portlet umzusetzen. Auf diese Weise kann der Screenflow Manager, der bereits das Dialog State Display Portlet und das Dialog Stack Portlet (siehe Kapitel 5) mitbringt, einfach um den grafischen Modellierer erweitert werden. Abbildung 6.1 veranschaulicht die Integration des Modellierungswerkzeugs in den Portalkontext.

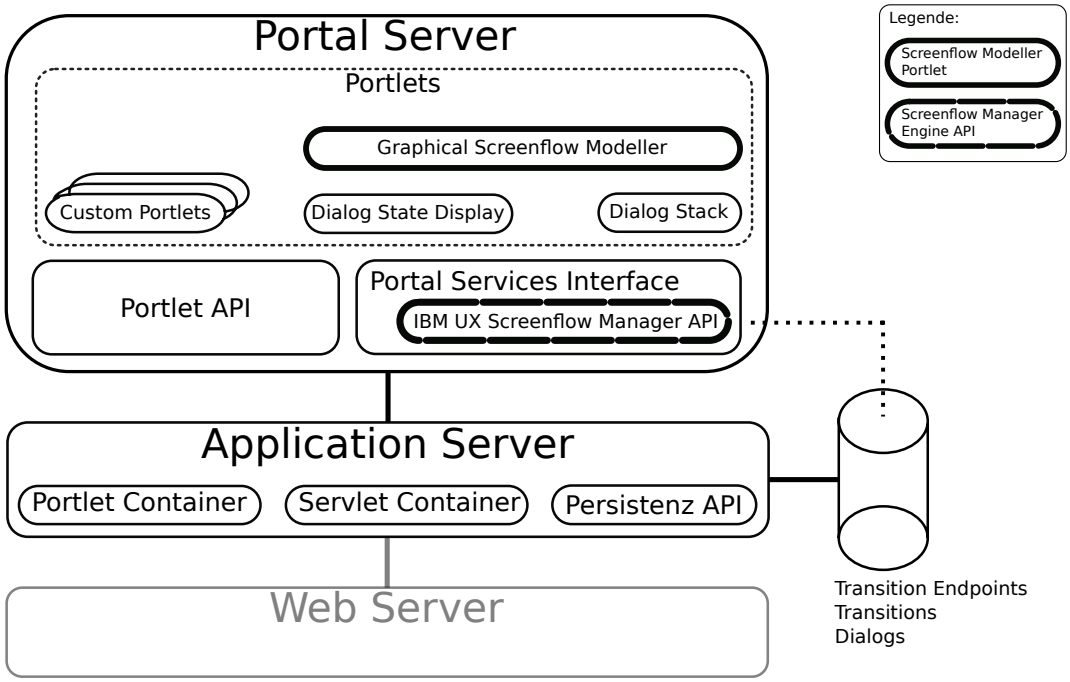

Abbildung 6.1.: Integration des Modellierungswerkzeugs in die Portal Architektur.

Als Portlet hat das Modellierungswerkzeug, das in der Grafik als *Graphical Screenflow Modeller* bezeichnet ist, direkten Zugriff auf die API des Screenflow Managers. Zusätzlich kann es alle benötigen Daten, von den am Screenflow beteiligten Portlets und Portalseiten, direkt über die IBM Service API vom Portal Server abfragen. Als Portlet ist das Modellierungswerkzeug plattformunabhängig und kann auf jedem System verwendet werden, das einen grafischen Browser bietet. Zusätzlich bietet dies für den Modellierer den Vorteil, dass das Werkzeug im Portal integriert ist und er so nur eine Anwendung (den Browser) benötigt,

um den Screenflow zu entwickeln und zu testen. Da alle Komponenten für den zu modellierenden Screenflow bereit im Portal zur Verfügung stehen, ermöglicht dieser Ansatz eine Modellierung zur Laufzeit.

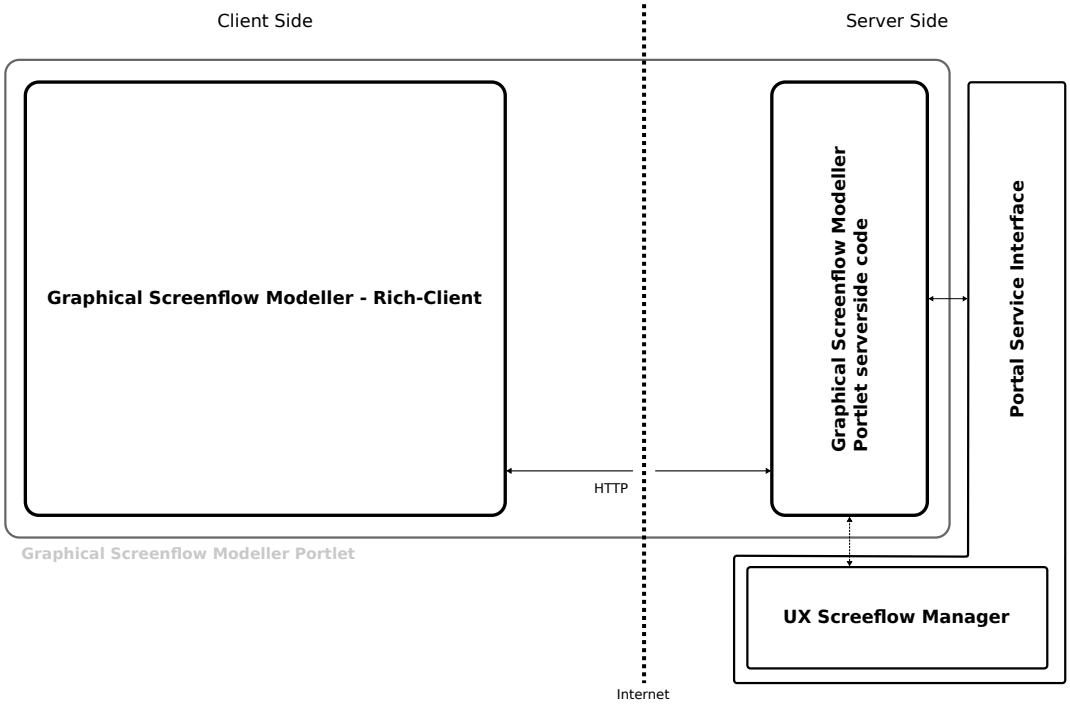

Abbildung 6.2.: Client-Server Modell des Modellierungswerkzeugs (Portlet).

Abbildung 6.2 zeigt die mehrschichtige Architektur der Web-Anwendung, die sich aus der Implementierung als Portlet ergibt. Die Präsentationsschicht des Portlets beschränkt sich auf die Auslieferung des Rich-Clients an die Clientseite. Der Rich-Client kommuniziert dann über HTTP-Aufrufe von der Clientseite aus mit der Anwendungsschicht des Portlets. Die Anwendungsschicht verarbeitet die Aufrufe und involviert bei Bedarf die entsprechenden Dienste, um auf die Screenflow Manager API zuzugreifen oder um Details über die Portlets und Portalseiten des Screenflows abzufragen. Die entsprechenden Daten liefert die Anwendungsschicht dann an die Clientseite zurück, wo sie vom Modellierungswerkzeug weiter verarbeitet werden. Die Logik für die Darstellung des Screenflows befindet sich im Rich-Client und benötigt keine Kommunikation mit der Anwendungslogik im Portlet. So kommt die Anwendung mit wenig Kommunikation aus.

Die Entscheidung das Modellierungswerkzeug als Portlet zu entwickeln, lässt sich wie folgt begründen. Der Screenflow Manager kann so um die Möglichkeit einer grafischen Modellierung erweitert werden. Als Portlet lässt sich die Anwendung einfach in die bestehende Architektur integrieren. Die Benutzungsschnittstelle des Modellierungswerkzeug kann ohne Aufwand ins Portal eingebettet werden. Eine alternative Umsetzung zum Beispiel als externe Anwendung kam daher nicht in Betracht.

6.3.2. Visuelle Integration

Dieser Abschnitt beschreibt, wie das Modellierungswerkzeug visuell in das Portal integriert wird. Für die Platzierung der Benutzungsschnittstelle werden zwei Möglichkeiten in Betracht gezogen. Das Ziel bei der Auswahl war eine für den Modellierer optimale Lösung zu finden, sodass er den Modellierungsprozess am einfachsten ausführen kann.

Platzierung in der Werkzeugleiste Der IBM WebSphere Portal Server besitzt eine Werkzeugleiste, welche bei Bedarf vom oberen Rand des Bildschirms herunter geklappt werden kann. Die erste mögliche Lösung ist, das Modellierungswerkzeug in die Portal Werkzeugleiste zu integrieren. Die Werkzeugleiste behält alle Inhalte auch bei einem Seitenwechsel im Portal. Das Ziel bei der Integration des Modellierungswerkzeug in die Werkzeugleiste ist, dass der Modellierer wie gewohnt im Portal umhernavigieren kann, um sich Artefakte auszuwählen, die er dem Screenflow hinzufügen möchte. Die gewünschten Artefakte kann er dann per Drag and Drop in das Modellierungswerkzeug ziehen, wo sie dann als Modell dargestellt werden. Einzelne Portlets können direkt von der entsprechenden Portalseite in das Modellierungswerkzeug gezogen werden. Eine Portalseite kann der Modellierer aus der Navigationsleiste in die Fläche des Modellierungswerkzeug ziehen. Dabei werden dann alle Portlets, die sich auf der gewählten Portalseite befinden, dem grafischen Modellierer hinzugefügt. Da ein Portlet immer in einer Portalseite enthalten sein muss, wird implizit auch die zugehörige Portalseite referenziert.

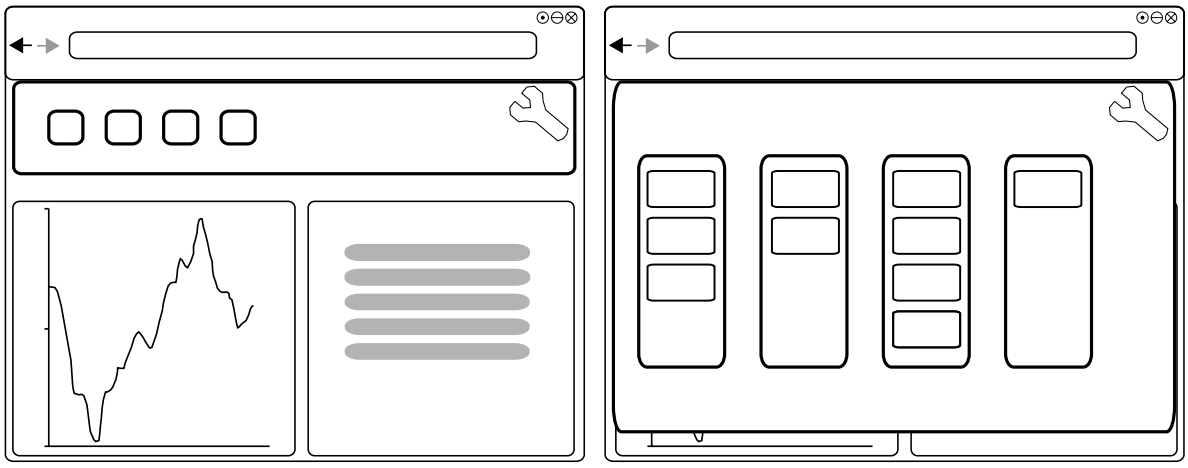

Abbildung 6.3.: Modellierungswerkzeug in der Werkzeugleiste. Links einfacher Modus, rechts erweiterter Modus.

Da der Platz auf dem Bildschirm begrenzt ist, erhält das Portlet in der Werkzeugleiste zwei Modi. Einen einfachen und einen erweiterten Modus, siehe Abbildung 6.3 links der einfache Modus und rechts der erweiterte Modus. Im einfachen Modus wird das Portlet in seiner Höhe beschränkt und das Modell der Artefakte komprimiert angezeigt. Dadurch sind nur wenig Bildschirminhalte von der Werkzeugleiste verdeckt. Dies verhilft dem Modellierer zu mehr Übersicht und erleichtert das Drag and Drop mit den Artefakten. Nachdem die Auswahl der

Artefakte abgeschlossen ist, kann das Portlet für den restlichen Modellierungsprozess im erweiterten Modus angezeigt werden. Dabei ist das Portlet dann in voller Größe dargestellt und das Modell des Screenflows expandiert.

Platzierung als eigenständiges Portlet Die andere mögliche Lösung ist das Modellierungswerkzeug als eigenständiges Portlet in einer Portalseite zu integrieren. Das Werkzeug könnte zum Beispiel im Administrationsbereich eingebunden werden. Da der Modellierer bei einem eigenständigen Portlet nicht umhernavigieren kann um Dialogartefakte auszuwählen, muss das Portlet dafür eine Liste aller Ressourcen zur Verfügung stellen. Von dort aus kann der Modellierer die gewünschten Artefakte in die Modellierungsfläche ziehen, wo er weiter mit ihnen arbeiten kann. Abbildung 6.4 veranschaulicht den Ansatz in einer eigenen Portalseite. Links im Bild ist die Liste der vorhanden Ressourcen angedeutet, der rechte Teil des Bildes ist für die Modellierungsfläche für die Screenflows vorgesehen.

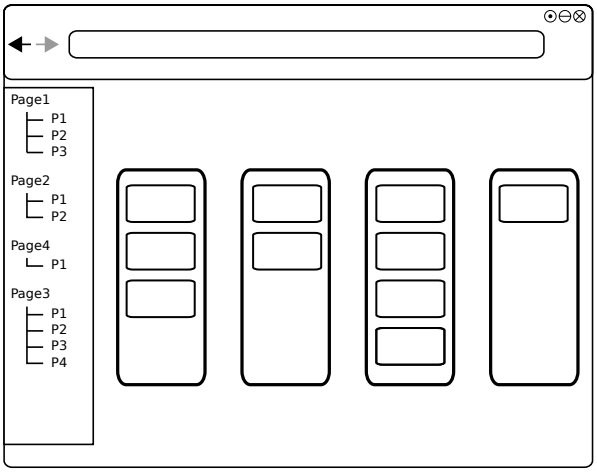

Abbildung 6.4.: Modellierungswerkzeug in eigener Portalseite.

Auswahl Letztlich wurde entschieden das Portlet in die Werkzeugleiste des Portals zu integrieren. Das Modellierungswerkzeug als eigenständiges Portlet in eine Portalseite einzubetten hat zwar den Vorteil, dass Modellierer den Screenflow von einem zentralen Punkt aus entwickeln kann, ohne dass er wie beim alternativen Ansatz umhernavigieren muss, um die Artefakte für den Screenflow zusammenzusuchen. Gerade aber das Navigieren zu den entsprechenden Portalseiten war bei der Entscheidung ausschlaggebend, da der Modellierer so die potenziellen Artefakte direkt sehen kann. Ähnlich wie bei einem Onlineshop die Artikel in den Einkaufswagen gelegt werden, wählt der Modellierer seine gewünschten Artefakte aus und zieht sie in die Werkzeugleiste. Dieses ist ein sehr intuitives Vorgehen. Werden die Ressourcen in einer Liste bereitgestellt, muss der Modellierer alle Namen der Artefakte kennen, die er dem Dialog hinzufügen möchte. Außerdem haben große Listen das Problem, dass sie schnell unübersichtlich werden können.

6.3.3. Grafische Darstellung von Screenflows

Dieser Abschnitt beschreibt, wie ein Screenflow im Modellierungswerkzeug grafisch repräsentiert wird. Dafür wurden mehrere mögliche Diagramme entwickelt. Diagramme nutzen die Aussagekraft von Bildern und sind daher ein gutes Mittel zur Veranschaulichung von Abläufen. Sie sind in der Regel schnell und einfach zu verstehen und können auch von Personen gelesen werden, die nicht alle Details kennen [USA05].
Gerade Graphen werden häufig dazu eingesetzt, Abläufe von Prozessen visuell darzustellen. Beispiele hierfür sind unter anderen BPMN [OMG11] im Bereich der Geschäftsprozesse oder den UML Zustands- und Flussdiagrammen [OMG05], die in der Programmentwicklung das Verhalten von Systemen veranschaulichen.

Auch das grafische Modellierungswerkzeug soll für die Modellierung der Screenflows einen Graphen verwenden. Es existieren viele Möglichkeiten, wie ein Prozessgraph für einen Screenflow gestaltet werden kann. Im Folgenden werden vier mögliche Ansätze betrachtet. Die zugehörigen Beispielgraphen stellen das folgende Szenario dar. Es besteht aus zwei Portalseiten, die jeweils 3 Portlets enthalten. Portalseite PS_1 beinhaltet die Portlets A, B und C während Portalseite PS_2 die Portlets D, E und F enthält. A sendet ein Ereignis an B. B sendet ein Ereignis an C und ein weiteres an PS_2, was einem Broadcast an D, E und F entspricht. C und F senden beide jeweils ein Ereignis an E. Abbildung 6.5 veranschaulicht die Transitionen des Beispielszenarios. Für eine bessere Übersieht sind die Transitionen mit Nummern beschriftet.

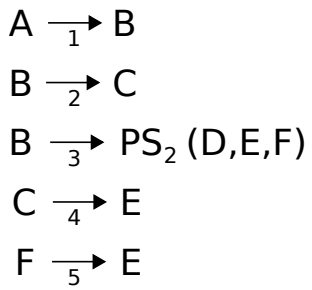

Abbildung 6.5.: Transitionen des Beispielszenarios für die grafische Darstellung von Screenflows.

Verschachtelter Graph Dieser Ansatz verwendet zwei Knotentypen, einen für die Darstellung der Portalseiten (große Knoten) und einen für die Repräsentation von Portlets (kleine Knoten). Ein Knoten für eine Portalseite beinhaltet alle Portlets (kleine Knoten), die sich auf der repräsentierten Portalseite befinden. Jeder dieser Knoten enthält einen Teil des gesamten Graphen. Zur Verbindung der Knoten sind drei Arten von Kanten erlaubt. Erstens, Kanten zwischen den Portlets innerhalb einer Portalseiten. Zweitens, Kanten zwischen den Portlets aus unterschiedlichen Portalseiten. Und drittens, Kanten von einem Portlet zu einer Portalseiten, was einen Broadcast an alle Portlets innerhalb der Portalseite repräsentiert.
Abbildung 6.6 stellt das obige Szenario mit den zwei Portalseiten und den sechs Portlets als verschachtelten Graphen dar.

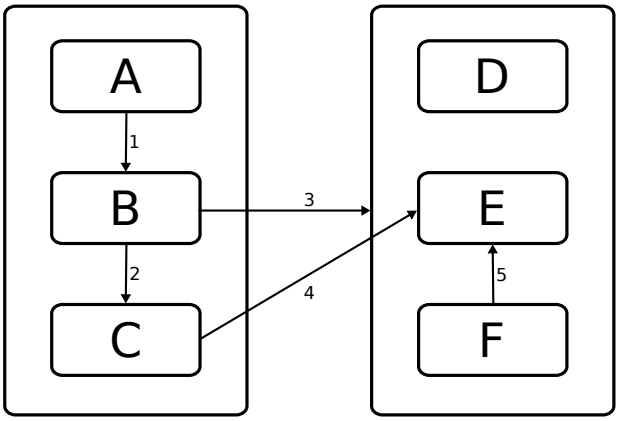

Abbildung 6.6.: Beispielszenario mit verschachteltem Graphen.

Dieser Ansatz hat den Vorteil, das die Knoten einen ähnlichen Aufbau besitzen wie Portalseiten mit ihren Portlets. So erhält der Modellierer eine gewisse Vertrautheit zu dem Modell. Der Modellierer kann auf einen Blick sehen, wie die Zugehörigkeit der Portlets zu ihrer Portalseite ist. Bei einem Broadcast ist nur eine Kante notwendig, anstatt dass für jedes beteiligte Portlet eine Kante gezogen werden muss. Bei jedem Kantenübergang kann sofort abgelesen werden auf welcher Portalseite sich der Endbenutzer dann befinden wird.

Freier Graph Dieser Ansatz, verzichtet auf die explizite Darstellung von Portalseiten. Eine Transition auf eine Portalseite, also der Broadcast an alle darin enthalten Portlets, wird durch entsprechende Kanten an alle beteiligten Knoten dargestellt. Durch den Verzicht auf die Darstellung der Portalseiten, kommt dieser Ansatz mit nur einer Knotenart aus. In Abbildung 6.7 wird das vorher beschriebene Szenario als freier Graph abgebildet.

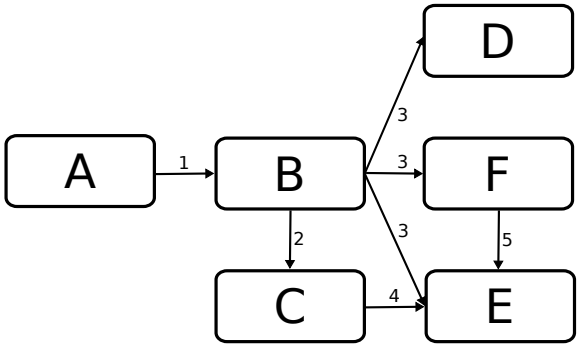

Abbildung 6.7.: Beispielszenario mit freiem Graphen.

Von Vorteil ist, dass der Graph beliebig angeordnet werden kann. Nachteilig ist jedoch, dass ohne eine Repräsentation der Portalseiten für einen Broadcast jedes Portlet explizit mit einer Kante verbunden werden muss. Darunter kann auch schnell die Lesbarkeit leiden. Für den

Modellierer ist auch nicht ersichtlich, welche Portlets sich gemeinsam auf einer Portalseite befinden.

Freier Graph mit zwei Knotenarten Dieser Ansatz ist eine Kombination aus Ansatz 1 und 2. Hierbei handelt es sich zwar um einen freien Graphen, es existieren jedoch zwei Arten von Knoten. Ähnlich wie beim Ansatz mit dem verschachtelten Graphen existiert eine Art von Knoten für die Portlets und eine für die Portalseiten. Der Unterschied ist jedoch, dass die Knoten für die Portalseiten keine Untergraphen enthalten. Dennoch repräsentiert eine Kante zu einem solchen Knoten einen Broadcast an alle Portlets, die auf der repräsentierten Portalseite enthalten sind. Sendet ein Portlet aus der betreffenden Portalseite nach diesem Broadcast ein Ereignis, wird die Kante abgehend vom Knoten der Portalseite dargestellt (Kante 5). Ansonsten wird das Portlet als eigenständiger Knoten dargestellt.
Grafik 6.8 veranschaulicht das obige Szenario mit dem freien Graphen und seinen zwei Knotentypen.

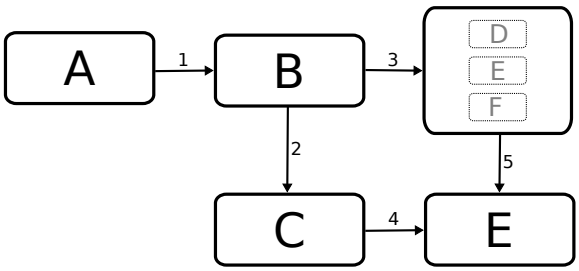

Abbildung 6.8.: Beispielszenario mit freiem Graphen mit zwei Knotentypen.

Dieser Ansatz hat gegenüber dem vorhergehenden Graphen den Vorteil, dass ein Broadcast sofort abgelesen werden kann. Nachteilig ist jedoch, dass bei Kanten die von einem Knoten wegführen, der sich innerhalb einer Portalseite befindet, der Ursprung nicht sofort ersichtlich ist (Kante 5, $F \rightarrow E$).

Freier Graph mit Multi-Knotenmenge Dieser Ansatz ist eine Erweiterung des zuvor vorgestellten freien Graphen. Auch bei diesem Ansatz wird auf die Darstellung von Portalseiten komplett verzichtet. In Erweiterung zum Graphen des zweiten Ansatzes können bei diesem Graphen die Knoten der Portlets jedoch mehrfach vorkommen. Zusätzlich besteht die Regel, dass jeder Knoten maximal eine eingehende und maximal eine ausgehende Kante besitzen darf. In Abbildung 6.9 wird das Beispielszenario durch einen Graphen mit Multi-Knotenmenge dargestellt.

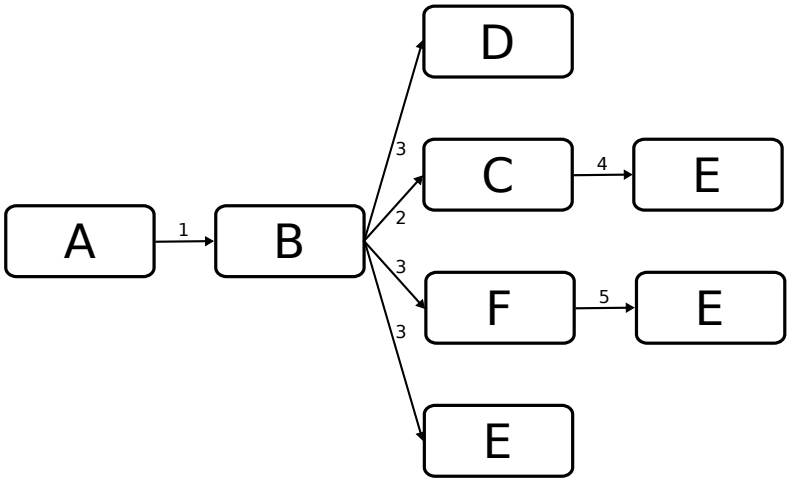

Abbildung 6.9.: Beispielszenario mit freiem Graphen mit Multi-Knotenmenge.

Vorteilhaft bei diesem Graphen ist, dass er eine Leserichtung besitzt. Er kann von links nach rechts gelesen werden. Alle möglichen Pfade in einem Screenflow können sofort abgelesen werden. Das hat jedoch gleichzeitig den Nachteil, dass die Graphen schnell sehr groß werden. Auch Broadcasts können nur schwer vom Graphen abgelesen werden. Des Weiteren ermöglicht der Graph keine Modellierung von Zyklen, die für Rücksprünge auf einen vorherigen Subdialog erforderlich sein können.

Auswahl Für die Darstellung der Screenflows wurde der verschachtelte Graph gewählt. Dieser Ansatz eignet sich am besten für diesen Zweck. Die Graphen aus dem zweiten und vierten Ansatz haben den Nachteil, dass die Erstellung von Broadcasts für den Modellierer unkomfortabel sind, da er jeden beteiligen Knoten explizit verbinden muss. Zusätzlich sind Broadcasts bei diesen beiden Ansätzen praktisch nicht von anderen Ereignissen zu unterscheiden. Eine ungünstige Positionierung der Knoten kann dieses Problem noch verstärken. Der Ansatz mit dem freien Graphen mit Multi-Knotenmenge kann zusätzlich keine Zyklen darstellen, die eventuell bei Rücksprüngen nötig sind. Ansatz drei, der freie Graph mit zwei Knotentypen, steht dem verschachtelten Graphen bei der Darstellung von Broadcasts in nichts nach. Dieser Ansatz weist jedoch Schwächen bei der Darstellung von Transitionen auf. Transitionen von Knoten, die zuvor an einem Broadcast beteiligt waren, können der Quelle nicht mehr zugeordnet werden ($F \rightarrow E$). Da diese Transition am Knoten der Portalseite dargestellt wird, ist es nicht ersichtlich von welchem Knoten die Transition ausging.
Aus der Sicht eines Modellierers eignet sich daher besonders der verschachtelte Graph für die Darstellung der Screenflows.

6.3.4. Modellierungsprozess

Dieser Abschnitt beschreibt, wie sich der Modellierungsprozess gestaltetet. Dafür werden die einzelnen Aktivitäten, die ein Modellierer im Modellierungswerkzeug durchführen muss, aufgelistet.

Artefakte hinzufügen Während dieser Aktivität fügt der Modellierer dem Graphen die gewünschten Artefakte hinzu. Dafür navigiert er zur gewünschten Portalseite und zieht das entsprechende Artefakt per Drag and Drop in das Modellierungswerkzeug in der Werkzeugleiste. Das Modellierungswerkzeug fügt daraufhin das Artefakt umgehend dem Graphen hinzu. Damit der Modellierer nicht durch die Werkzeugleiste bei der Auswahl gehindert wird, ist sie, wie bereit im Abschnitt Visuelle Integration beschrieben, komprimiert dargestellt.

Transitionen hinzufügen Während dieser Aktivität verbindet der Modellierer die Artefakte miteinander. Dazu zieht er im Graphen, von einem Knoten zum anderen, eine Kante. Der Knoten von dem die Aktivität aus beginnt, wird als Quelle der Transition verwendet. Der Knoten an dem die Aktivität endet als das Ziel. Das Verbinden der Artefakte mit einer Kante definiert lediglich welche Transition Endpoints an der Transition beteiligt sind und wer von ihnen die Quelle oder das Ziel ist. Zusätzlich müssen noch weitere Eigenschaften wie zum Beispiel das auszutauschende Ereignis festgelegt werden. Dies geschieht in der nächsten Aktivität.

Ereignisse der Transitionen festlegen Während dieser Aktivität spezifiziert der Modellierer die Eigenschaften einer Transition. Dazu muss die entsprechende Transition ausgewählt sein. Eine Transition kann der Modellierer mit der Maus auswählen oder über die vorhanden Transitionen iterieren. Im angezeigten Dialog kann der Modellierer auswählen, welches von den potenziell möglichen Ereignisse von der Quelle für diese Transition gesendet werden soll und welches der potenziell möglichen Ereignisse das Ziel entgegen nehmen soll. Zusätzlich kann der Modellierer für das Ereignis von der Quelle und dem Ziel einen Aliasnamen vergeben, unter dem die Nutzdaten des Ereignis im Dialog Context abgelegt oder geladen werden soll. Um gegebenenfalls ein Ereignis zu transformieren, kann der Modellierer aus den vorhanden Event Mappern (siehe Kapitel 5) einen PayloadToContextMapper für die Quelle und einen ContextToPayloadMapper für das Ziel auswählen. Schließlich kann der Modellierer noch auswählen, ob es sich um eine Start- oder Endtransition handelt. Bei dieser Auswahl handelt es sich um ein exklusives Oder. Beide Optionen können nicht gleichzeitig gewählt werden. Wird keine der beiden Optionen gewählt, wird die Transition als gewöhnliche Transition behandelt. Nur die Transitionen, für die Eigenschaften spezifiziert sind, werden vom Modellierungswerkzeug in die Dialogdefinition aufgenommen. Das bloße Verbinden von Transition Endpoints reicht dafür nicht aus.

Eigenschaften für den Screenflow festlegen Während dieser Aktivität vergibt der Modellierer globale Eigenschaften für den Screenflow. Im angezeigten Dialog kann er einen Standard Extension Node definieren, der dann als Voreinstellung für alle dynamischen Kopien (siehe Kapitel 5) gesetzt ist.

Eigenschaften für die Portalseiten festlegen Während dieser Aktivität legt der Modellierer erweiterte Eigenschaften für eine Portalseite fest. Dazu muss die entsprechende Portalseite ausgewählt sein. Eine Portalseite kann der Modellierer mit der Maus auswählen oder über die vorhanden Portalseite iterieren. Der Modellierer kann dann im angezeigten Dialog für die Portalseite, für jede unterstütze Sprache einen Titel und eine Beschreibung festlegen. Diese substituieren dann den originalen Titel und Beschreibung der Portalseite im entsprechenden Dialogschritt. Des Weiteren legt der Modellierer hier fest, ob die Portalseite als dynamische Kopie oder statisch geladen werden soll. Dazu muss er einen Haken unter dynamische Kopie setzen. Im Falle, dass er die dynamische Kopie wählt, kann hier auch den zugehörigen Extension Node angeben. Wenn der Modellierer bei den Eigenschaften des Screenflow bereits einen Standard Extension Node vergeben hat, ist dieser hier voreingestellt.

Eigenschaften für die Portlets festlegen Während dieser Aktivität legt der Modellierer zusätzliche Eigenschaften für die einzelnen Portlets fest. Das entsprechende Portlet muss dafür ausgewählt sein. Ein Portlet kann der Modellierer mit der Maus auswählen oder über die vorhanden Portlets iterieren. Der Modellierer kann dann im angezeigten Dialog für das Portlet für jede unterstütze Sprache einen Titel und eine Beschreibung festlegen. Diese substituieren dann den originalen Titel und Beschreibung des Portlets im entsprechenden Dialogschritt. Im angezeigten Dialog kann der Modellierer wählen, ob das Portlet als dynamische Kopie in den Screenflow eingebunden werden soll. Dazu muss er den Haken für die dynamische Kopie setzen. Im Fall, dass der Modellierer diese Option aktiviert, muss er eine dynamische Kopie einer Portalseite (Template) angeben, in der das Portlet eingebettet wird. Zusätzlich kann er einen Extension Node spezifizieren. Hat der Modellierer einen Extension Node bereits in den Eigenschaften des Screenflow festgelegt, ist dieser hier voreingestellt.

Testen Während dieser Aktivität kann der Modellierer den Screenflow testen. Diese Aktivität ist jedoch nicht Bestandteil dieser Arbeit. Es ist jedoch vorstellbar, dass bei dieser Aktivität der Ablauf des Screenflows vom Modellierer simuliert wird.

Dialogdefinition speichern Während dieser Aktivität speichert der Modellierer den zuvor definierten Screenflow im System. Dazu muss er einen eindeutigen Namen für den Screenflow festlegen. Alternativ kann der Modellierer auch den Screenflow als XML-Datei exportieren, um ihn lokal abzulegen oder um ihn an jemanden weiterzugeben. Der exportierte Screenflow kann dann zum Beispiel auf einem anderen System geöffnet werden.

Geführter Modellierungsprozess

Um den Modellierer optimal bei der Ausführung dieser Aktivitäten zu unterstützen, wird er im Modellierungswerkzeug schrittweise durch einen vordefinierten Modellierungsprozess geführt. Dem Modellierer werden dabei immer nur die Werkzeuge eingeblendet, welche für die aktuelle Aktivität im entsprechenden Prozessschritt notwendig sind. Ist eine Aktivität abgeschlossen, kann er zum nächsten Prozessschritt wechseln. Durch dieses Vorgehen kann der Modellierer den Screenflow Schritt für Schritt entwickeln. In welchem Prozessschritt er sich gerade befindet, wird ihm über eine Navigationsleiste angezeigt, die sich im oberen Bereich des Modellierungswerkzeug befindet und die den aktiven Prozessschritt hervorhebt. Abbildung 6.13 veranschaulicht diese Navigationsleiste. Trotz dieses vordefinierten Prozesses kann der Modellierer bei Bedarf zwischen den einzelnen Aktivitäten springen, indem er in der Navigationsleiste die entsprechende Aktivität auswählt. Des Weiteren bietet dieser Ansatz die Möglichkeit, die Eingaben des Modellierers in jedem Prozessschritt zu validieren. Einen Wechsel in den nächsten Prozessschritt könnte so lange verhindert werden, bis alle Eingaben vorhanden sind.

Eine Alternative ist im Modellierungswerkzeug auf eine Benutzerführung zu verzichten. In diesem Fall stehen dem Modellierer zu jedem Zeitpunkt alle Werkzeuge zur Verfügung. Der Modellierer muss dann selbst entscheiden, welche Aktivitäten durchzuführen sind und in welcher Reihenfolge er diese dann durchführt. Dies erfordert jedoch eine gewisse Erfahrungen vom ihm. Ein vordefinierter Prozess eignet sich für alle Arten von Anwender. Technisch weniger versierte können sich Schritt für Schritt durch den Modellierungsprozess führen lassen, während fortgeschrittene Benutzer nicht benötigte Schritte überspringen.

6.3.5. Grafische Umsetzung der Benutzungsschnittstelle

Die im vorhergehenden Absatz beschrieben Aktivitäten erfordern jeweils eine Interaktion mit der Benutzungsschnittstelle. Die dazu notwendigen Dialoge und Anzeigen werden in diesem Abschnitt kurz vorgestellt.

Dialog für die Festlegung der Ereignisse einer Transition Dieser Dialog dient dem Modellierer zur Konfiguration der Ereignisse einer Transition. Abbildung 6.13 zeigt eine Skizze des Dialogs. Über den Dialog wird der Quelle und dem Ziel ein Ereignis zugewiesen. Für die Quelle werden alle Ereignisse in einer Auswahl bereit gestellt, die der Transition Endpoint senden kann. Das selbe gilt für das Ziel, wo alle Ereignisse aufgelistet werden, die der Transition Endpoint empfangen kann. Zusätzlich kann der Modellierer einen Aliasnamen vergeben, unter dem die Nutzdaten des Ereignisses gespeichert oder geladen werden sollen. Zusätzlich bietet der Dialog eine Auswahl von allen vorhanden Event Mapper für die Ereignistransformation. Über zwei Kontrollkästchen kann der Modellierer festlegen, ob die Transition eine Start- oder Endtransition ist. Von den Kästchen kann maximal nur eines aktiviert werden. Zwei Buttons im unteren Bereich des Dialogs erlauben dem Modellierer über die vorhandenen Transitionen zu iterieren.

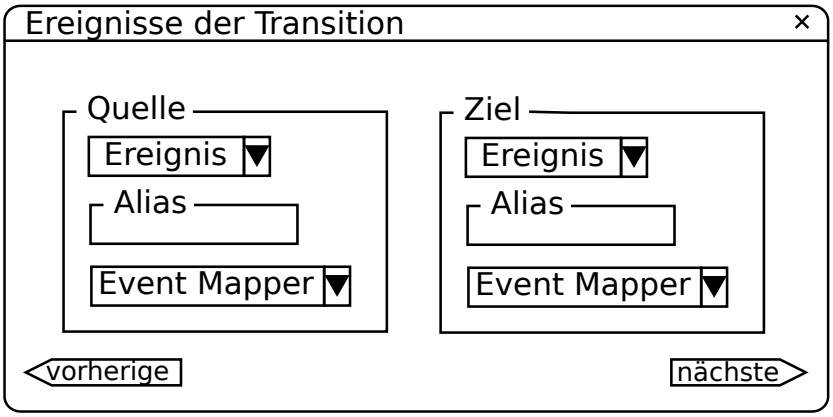

Abbildung 6.10.: Dialog zur Definition der Eigenschaften einer Transition.

Dialog für globale Eigenschaften des Screenflows Über diesen Dialog werden Eigenschaften festgelegt, die den gesamten Screenflow betreffen. Der Modellierer kann im Dialog einen Extension Node festlegen, der dann bei allen Transition Endpoints voreingestellt wird, die als dynamische Kopie in den Screenflow eingebunden sind. Ein voreingestellter Extension Node kann bei jedem Transition Endpoint angepasst werden, siehe Dialog für die Eigenschaften einer Portalseite und Dialog für die Eigenschaften eines Portlet. Abbildung 6.11 skizziert diesen Dialog.

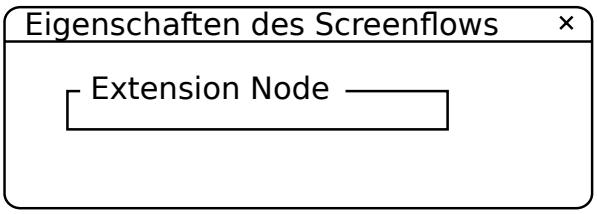

Abbildung 6.11.: Dialog zur Definition der globalen Eigenschaften des Screenflows.

Dialog für die Eigenschaften einer Portalseite Durch diesen Dialog werden die Eigenschaften einer Portalseite definiert. In Abbildung 6.12 ist ein möglicher Dialog abgebildet. Der Dialog teilt sich in zwei Rubriken auf, die durch Tabs getrennt sind. In der ersten Rubrik kann der Modellierer für die Portalseite einen Title und eine Beschreibung für jede unterstützte Sprache vergeben. Die Eingabefelder sind untereinander aufgelistet, um eine schnelle und einfache Eingabe der Daten zu ermöglichen. In der zweiten Rubrik kann der Modellierer über ein Kontrollkästchen wählen, ob die Portalseite als dynamische Kopie in den Screenflow eingebunden werden soll. Ist das Kontrollkästchen aktiviert, kann der Modellierer ein Template und ein Extension Node festlegen. Wenn in den globalen Eigenschaften ein Extension Node gesetzt ist, ist dieser in diesem Dialog im Extension Node Eingabefeld bereits eingetragen. Der voreingestellte Extension Node kann jedoch geändert werden. Zwei Buttons im unteren Bereich des Dialogs erlauben dem Modellierer über die vorhandenen Portalseiten zu iterieren.

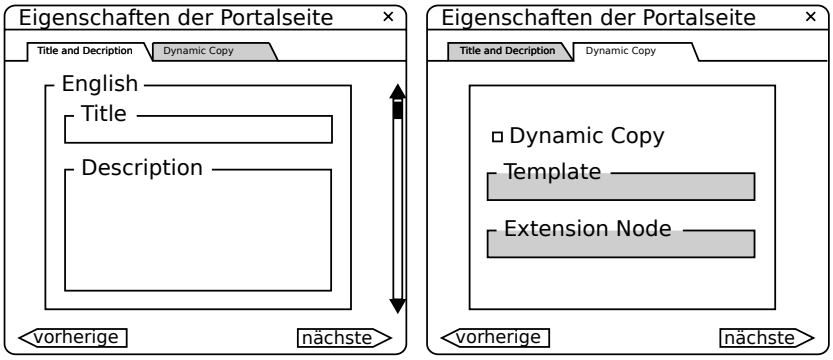

Abbildung 6.12.: Dialog zur Definition der Eigenschaften einer Portalseite.

Dialog für die Eigenschaften eines Portlet In diesem Dialog werden die Eigenschaften eines Portlets konfiguriert. Der Dialog ist identisch zum Dialog für die Konfiguration der Eigenschaften einer Portalseite. Wie der vorherige Dialog für die Eigenschaften einer Portalseite besteht auch dieser Dialog aus zwei Rubriken, die durch Tabs getrennt sind. In der ersten Rubrik kann der Modellierer für das Portlet einen Title und eine Beschreibung für jede unterstützte Sprache vergeben. Die Eingabefelder werden wie im vorherigen Dialog untereinander angeordnet. In der zweiten Rubrik besitzt der Dialog ein Kontrollkästchen, über das der Modellierer festlegt, ob das Portlet als dynamische Kopie in den Screenflow eingebunden wird. Nach Aktivierung dieser Option kann der Modellierer der dynamischen Kopie ein Template und einen Extension Node zuweisen. Wenn in den globalen Eigenschaften ein Extension Node gesetzt ist, ist dieser in diesem Dialog im Extension Node Eingabefeld bereits eingetragen. Der voreingestellte Extension Node kann jedoch geändert werden. Zwei Buttons im unteren Bereich des Dialogs erlauben dem Modellierer über die vorhandenen Portlets zu iterieren.

Navigationsleiste für den Modellierungsprozess Der Modellierungsprozess ist in mehrere Aktivitäten aufgeteilt. Jede dieser Aktivitäten entspricht einem Schritt im Modellierungsprozess. Eine Navigationsleiste, wie in Abbildung 6.13 dargestellt, soll dem Modellierer visualisieren, in welchem Prozessschritt des Modellierungsprozess er sich gerade befindet. Zusätzlich kann der Modellierer über die Navigationsleiste zwischen verschieden Aktivitäten hin und her wechseln.

Abbildung 6.13.: Navigationsleiste für den Modellierungsprozess.

Entwurf der Benutzungsschnittstelle

Für den Entwurf der Benutzungsschnittstelle wurden Mockups[1] gezeichnet. Diese Mockups wurden in Designmeetings validiert und schrittweise weiterentwickelt. Die endgültige Fassung der Mockups ist im Anhang A.3 zu finden.

6.4. Zusammenfassung

Aufgrund des zugrundeliegenden Screenflow Managers bestehen eine Reihe von Anforderungen, die ein grafisches Modellierungswerkzeug erfüllen muss, um einen Screenflow zu definieren. Zusätzlich bestehen noch Anforderungen, die von den Entwicklern des Screenflow Managers an das Modellierungswerkzeug gestellt wurden. Der präsentierte Lösungsansatz stellt eine mögliche Lösung für ein grafisches Modellierungswerkzeug für Screenflows vor. Das Modellierungswerkzeug wird als Portlet implementiert, um optimal in das Gesamtkonzept des Screenflow Managers integriert zu werden. Innerhalb des Portals wird das Modellierungswerkzeug in die Werkzeugleiste eingebettet, um dem Modellierer ein möglichst intuitives Arbeiten bei der Auswahl der am Screenflow beteiligten Artefakte zu ermöglichen. Das Portlet hat zwei Modi. Einen vereinfachten Modus für die Auswahl der Artefakte und einen erweiterten Modus für den restlichen Modellierungsprozess. Neben der Auswahl der Artefakte besteht der Modellierungsprozess aus einer Reihe weiterer Aktivitäten, die notwendig sind, um alle Informationen für eine Dialogdefinition zusammenzutragen. Um dem Modellierer das Vorgehen zu vereinfachen, sind die Aktivitäten in einem vordefinierten Modellierungsprozess festgelegt, durch den der Modellierer geführt wird. Für jede Aktivität im entsprechenden Prozessschritt werden nur die notwendigen Werkzeuge bereitgestellt. In welchem Prozessschritt des Modellierungsprozess er sich befindet, wird ihm über eine extra Navigationsleiste angezeigt, die den aktuellen Schritt optisch hervorhebt. Die feste Reihenfolge kann der Modellierer umgehen, indem er in der Navigationsleiste zu einer anderen Aktivität wechselt.

[1] **Mockup:** Als Mockup wird ein maßstabstreues Modell bezeichnet. Gerade bei der Entwicklung von grafischen Oberflächen werden Mockups in der Entwurfsphase verwendet.

7. Implementierung

Nachdem nun die Konzepte für das Modellierungswerkzeug vorgestellt wurden, beschreibt dieses Kapitel die konkrete Implementierung. Die Implementierung des Portlets teilt sich in zwei Bereiche auf: Erstens die *Clientseite*, die in JavaScript, HTML und CSS implementiert ist und im Browser des Nutzers ausgeführt wird. Sie wird durch das Aufrufen des Portlets in der Werkzeugleiste an den Browser übertragen. Zweitens die *Serverseite*, die in Java implementiert ist und innerhalb des Portal Servers ausgeführt wird. Clientseite und Serverseite kommunizieren über HTTP-Aufrufe miteinander.
Abbildung 7.1 stellt die Architektur mit den Komponenten des Portlets auf der Server- und Clientseite und ihren Beziehungen dar. Die einzelnen Komponenten werden in den folgenden Abschnitten detailliert beschrieben.

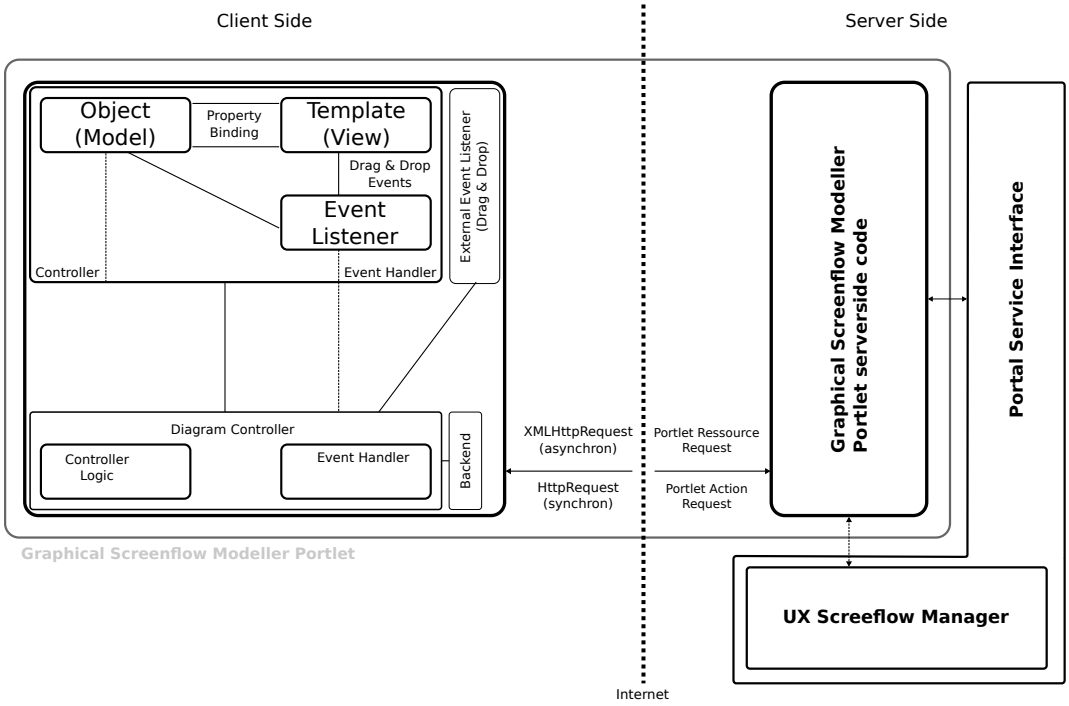

Abbildung 7.1.: Detailliertes Client-Server Modell des Modellierungswerkzeug (Portlet).

7.1. Clientseite

Die Clientseite besteht, wie zuvor erwähnt, aus JavaScript, HTML und CSS. Um die Entwicklung zu beschleunigen und um eine möglichst browserunabhängige Anwendung zu erhalten, wurden die folgenden JavaScript Frameworks bei der Implementierung verwendet.

7.1.1. JavaScript Frameworks

Für die Implementierung der Clientseite des Modellierungswerkzeugs wurden die JavaScript Frameworks *Dojo Toolkit*[1] und *IBM ILOG Dojo Diagrammer*[2] verwendet. Abbildung 7.2 veranschaulicht den Aufbau und die Abhängigkeiten der Frameworks.

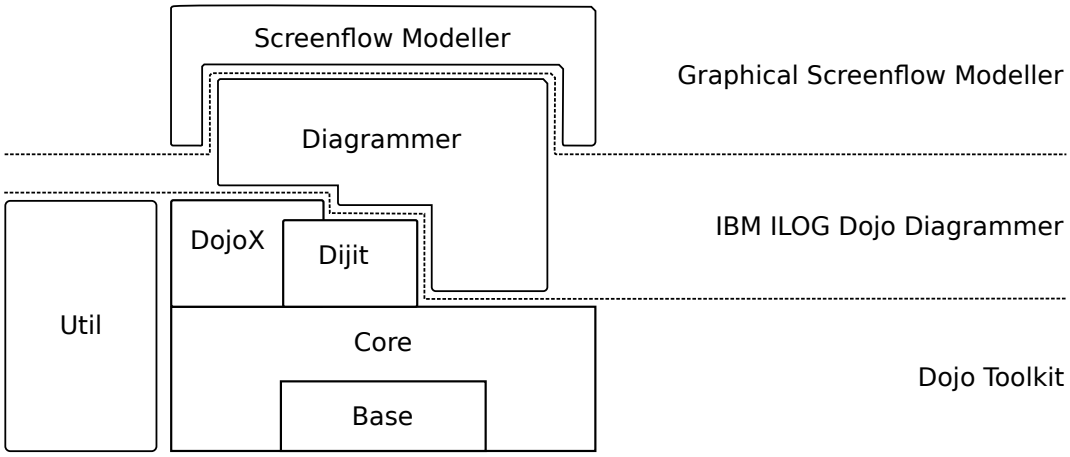

Abbildung 7.2.: Abhängigkeiten der JavaScript Frameworks [Mato8].

Als Basis für die Implementierung wird das Dojo Toolkit eingesetzt. Auf dem Dojo Toolkit baut der IBM ILOG Dojo Diagrammer auf. Das Modellierungswerkzeug wiederum nutzt Module und Klassen aus beide Frameworks für die Umsetzung.

Dojo Toolkit

Das Dojo Toolkit ist ein JavaScript Framework, das eine browserunabhängige Entwicklung von JavaScript Anwendungen ermöglicht. Dojo ist sehr modular aufgebaut. Der Kern von Dojo ist nur wenige Kilobyte groß. Weitere benötigte Module können während der Laufzeit

[1] Weitere Informationen zum Dojo Toolkit können unter http://dojotoolkit.org gefunden werden.
[2] Weitere Informationen zum IBM ILOG Dojo Diagrammer können unter http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.web2mobile.ilog.dojo.diagrammer.help/Content/Visualization/Documentation/Dojo/Dojo_Diagrammer/_pubskel/ps_dojo_diagrammer_ic2.html gefunden werden.

dynamisch nachgeladen werden. Abbildung 7.2 veranschaulicht die Beziehungen der elementaren Komponenten von Dojo. In der Implementierung wird das Dojo Toolkit in der Version 1.8 verwendet.

Base Der Kernel von Dojo ist ein Bestandteil der Base. Er bildet die Basis des gesamten Toolkits. Zum Kernel gehört unter anderem das Packaging System, das es erlaubt, Module während der Laufzeit nachzuladen. Des Weiteren bietet die Base Werkzeuge, die unter anderem für die Unterstützung von *Ajax*[3] und diversen Operationen auf dem DOM[4] verwendet werden können. Elemente, die direkt zur Base gehören, liegen direkt im *dojo* Namespace [Mato8].

Core Zu Core zählen Elemente, die zwar häufig genutzt werden, aber nicht allgemein genug sind, um zu Base zu gehören. Beispiele hierfür sind Funktionen für das Parsen von Widgets, die erweiterten Animationseffekte, das Drag and Drop oder die Internationalisierung (i18n). Elemente die zu Core zählen, befinden sich in untergeordneten Namespaces von Base wie zum Beispiel *dojo.fx*, *dojo.io* oder *dojo.data* [Mato8].

Dijit Das Wort Dijit beinhaltet die beiden Wörter Dojo und Widget. Dijits sind fertige Komponenten, die aus JavaScript, HTML und CSS bestehen und direkt in die Webanwendung eingefügt werden können. Dijits setzen direkt auf der Funktionalität von Core auf. Die vorhanden Dijits lassen sich grob in drei Klassen einteilen. Allzweck Widgets, zu denen beispielsweise Fortschrittsbalken zählen, Layout Widgets für die Anordnung von Elementen wie zum Beispiel Tab Container und Oberflächen Widgets mit Elementen wie Schaltflächen oder ähnlichem. Dijits befinden sich im Namensraum *dijit* [Mato8].

DojoX DojoX steht für Dojo Extensions und enthält eine Sammlung von Dojo Modulen, die in keine der zuvor erwähnten Komponenten passen. Ein Beispiel für ein solches Modul ist das DojoX GFX Modul, das eine API für die Erstellung von 2D Vektorgraphiken bereitstellt. Die in DojoX enthalten Module gewährleisten nicht dieselbe Stabilität wie Module aus Base oder Core. Aus diesem Grund enthält jedes Teilprojekt eine Beschreibung über seinen aktuellen Zustand und mögliche Ausnahmen und Probleme. Module aus DojoX sind dem Namespace *dojox* untergeordnet [Mato8].

[3] **Asynchronous JavaScript and XML** (Ajax): ist eine Lösung für die asynchrone Kommunikation zwischen einem Browser und einem Server. Dadurch können Seiteninhalte dynamisch (nach-)geladen werden, ohne die gesamte Seite neu laden zu müssen [Anto8].

[4] **DOM:** Document Object Model: Das Document Object Model ist eine plattform- and sprachneutrale Schnittstelle, um dynamisch auf den Inhalt, die Struktur und das Aussehen von Dokumenten zuzugreifen und diese zu manipulieren [W3C].

Util Util enthält Werkzeuge für das Arbeiten mit JavaScript Code. Unter anderem enthält Util Werkzeuge für die Codekomprimierung und das Erstellen von benutzerdefinierten Dojo Versionen. Ein weiterer Bestandteil von Util ist ein Testframework für automatisierte Unit Tests zur Qualitätssicherung [Mato8].

IBM ILOG Dojo Diagrammer

Der *IBM ILOG Dojo Diagrammer* ist ein von der IBM entwickeltes JavaScript Framework für die Erstellung und Darstellung von Graphen. Der IBM ILOG Dojo Diagrammer baut auf diversen Dojo Modulen auf. Dazu zählt unter anderem das *Dojox GFX* Modul, das ein API für die Erstellung von 2D Vektorgraphiken zur Verfügung stellt und *Dojo Data* für die Nutzung eines Data Stores, in dem Graphen vordefiniert werden können. Abbildung 7.2 veranschaulicht den Aufbau auf dem Dojo Toolkit. Der ILOG Dojo Diagrammer stellt für die Darstellung der Graphen ein Widget bereit, das sogenannte Diagramm (englisch diagram). Für den Aufbau des Graphen stellt der Diagrammer unterschiedliche Graphelemente zur Verfügung, deren Erscheinungsbild über sogenannte Templates festgelegt ist. Diese Templates können bei Bedarf angepasst oder ausgetauscht werden. Für eine optimierte Darstellung des Graphen und dessen Elemente, stellt das Framework zahlreiche Algorithmen zur Verfügung, sodass die Graphen automatisiert angeordnet werden können. Der IBM ILOG Dojo Diagrammer wird in mehreren webbasierten IBM Produkten eingesetzt und mit dem *Feature Pack for Web 2.0 and Mobile* für den *IBM WebSphere Application Server* ausgeliefert.

7.1.2. Architektur der Clientseite

Die Clientseite ist nach dem Entwurfsmuster MVC implementiert. Die Basis ist das *Diagramm* aus dem IBM ILOG Dojo Diagrammer mit seinen Funktionen. Das Framework stellt ein Datenmodell (Model) für die Graphelemente bereit, dessen Objekte mit einer grafischen Repräsentation verbunden sind (View). Der *DiagramController* (Controller) koordiniert die Erstellung der Graphelemente. In Abbildung 7.3 werden die Klassen anhand ihrer Rolle der MVC-Architektur dargestellt.

Darstellung

Für die Darstellung verwendet das Modellierungswerkzeug die Klasse *DiagramEditor* als Container für den Graphen. Sie ist eine Spezifizierung der *Diagram* Klasse des IBM ILOG Dojo Diagrammer. Der *DiagramEditor* enthält bereits Funktionen für das Bearbeiten von Graphen, wie zum Beispiel das Verbinden von Knoten. Der *DiagramEditor* wurde in der Implementierung so erweitert, dass die Werkzeuge für die Aktivitäten im Modellierungsprozess, aktiviert und deaktiviert werden können. Zusätzlich wurden gewisse Schnittstellen nach außen zugänglich gemacht, damit der *DiagramEditor* über den *DiagramController* gesteuert werden kann.

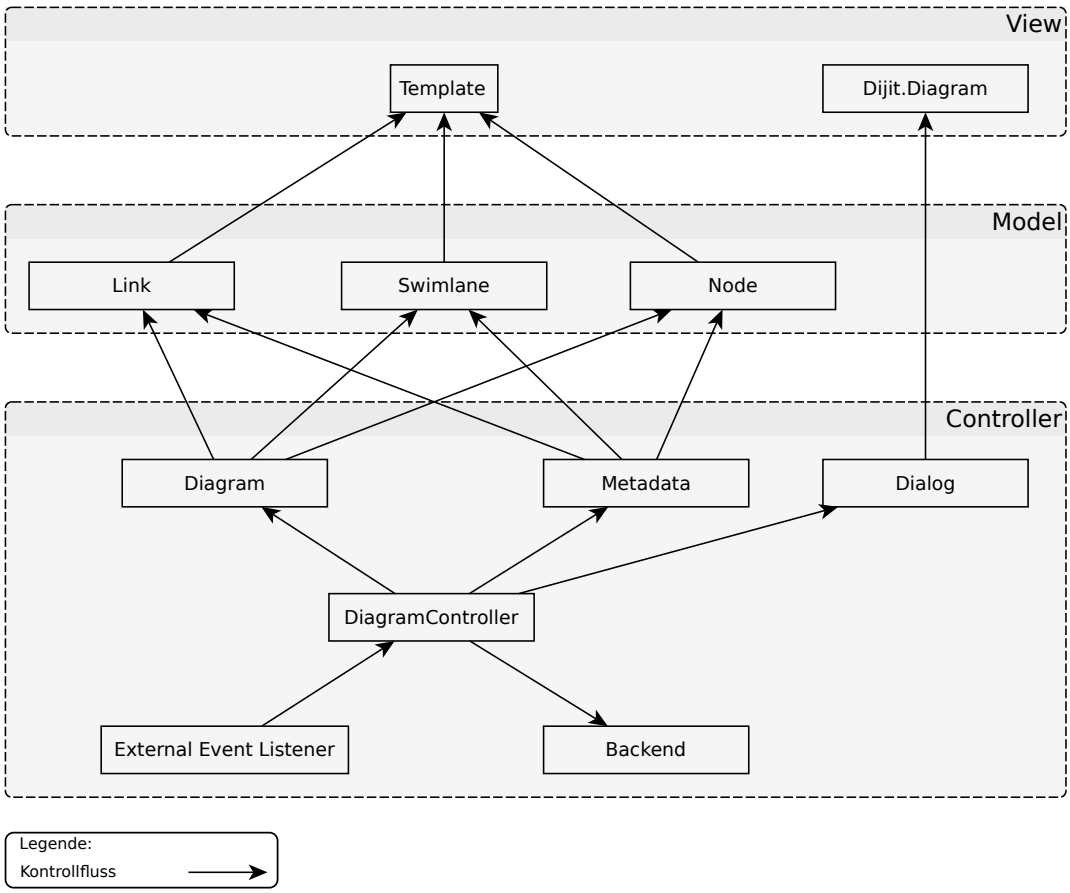

Abbildung 7.3.: Klassendiagramm der Clientseite, geordnet anhand der Rollen in der MVC-Architektur.

Für die Gestaltung des Graphen werden die Klassen *Swimlane*, *Node* und *Link* verwendet, die alle von der Klasse *GraphElement* abstammen. Abbildung 7.4 veranschaulicht die Vererbungshierarchie der Graphelemente und ihre Abhängigkeit zur *Diagram* Klasse.

Die *Swimlane* Klasse repräsentiert eine Portalseite. Diese Klasse wurde von der Basisklasse abgewandelt, um das Erscheinungsbild und gewisse Funktionen auf die Bedürfnisse des Modellierungswerkzeugs anzupassen. Beispielsweise wurde in der grafischen Repräsentation die Schaltfläche für das Minimieren der Swimlane entfernt. Des Weiteren wurde eine neue Schaltfläche hinzugefügt, über die alle Portlets einer Portalseite eingeblendet werden können. Eine Swimlane ist ein Teilgraph und beinhaltet Knoten, die aus Objekten der *Node* Klasse bestehen. Diese symbolisieren die Portlets, welche in der Portalseite enthalten sind, die durch die Swimlane dargestellt wird. Die *Link* Klasse repräsentiert die Transitionen im Graphen. Sie verbinden die einzelnen Graphelemente miteinander.

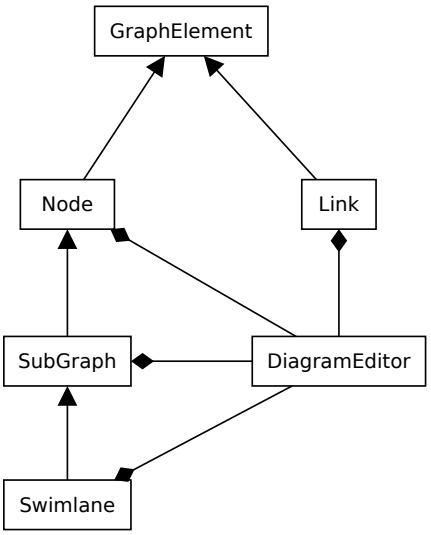

Abbildung 7.4.: Klassendiagramm der Graphelemente (Hierarchie).

DiagramController

Die *DiagramController* Klasse implementiert die Anwendungslogik des Modellierungswerkzeugs auf der Clientseite. Der Controller ist unter anderem für das Hinzufügen von Graphelementen zuständig. Wenn der *Externe Event Listner* ein Ereignis empfängt, das per Drag and Drop ein Portlet oder eine Portalseite dem Modellierungswerkzeug hinzugefügt wurde, wird der *DiagramController* benachrichtigt. Weitere Details werden im folgenden Abschnitt Externer Event Listener beschrieben. Zusätzlich ist der *DiagramController* für das Umschalten zwischen den Schritten im Modellierungsprozess zuständig. Dazu versetzt der *DiagramController* das Diagramm in einen Ausgangszustand, von dem aus er dann die entsprechenden Werkzeuge aktiviert, die im korrespondierenden Modellierungsprozessschritt benötigt werden. Für jeden Schritt im Modellierungsprozess stellt der *DiagramController* eine Methode bereit, um in den entsprechenden Schritt zu wechseln. Ein Beispiel hierfür ist die Methode *setAddArtifactsState*, durch deren Aufruf das Diagramm in den vereinfachten Modus übergeht und der Benutzer dem Graphen Artefakte hinzufügen kann.

Externer Event Listener

Dieser *Event Listener* reagiert auf die Drag and Drop Ereignisse, die vom Portal gesendet werden, wenn ein Portlet oder eine Portalseite in das Modellierungswerkzeug gezogen wird. Innerhalb des Portals wird das HTML5 Attribut *dragable* eingesetzt, um bei Portlets und Portalseiten das Drag and Drop zu aktivieren. Die Ereignisse, die bei einer Drag and Drop Aktion dann gesendet werden, sind HTML5 Drag and Drop Ereignisse, die sich von den Drag and Drop Ereignissen im ILOG Dojo Diagrammer unterscheiden. Der *externe Event Listener* empfängt diese Ereignisse und benachrichtigt den *DiagramController*. Dabei muss

er den *Portal Drag and Drop Contract* einhalten. Das bedeutet: er reagiert auf die Ereignis und kann die enthaltenen Daten interpretieren, wie es durch den Contract festlegt ist. Der vollständige Ablauf wird in Abbildung 7.5 dargestellt.

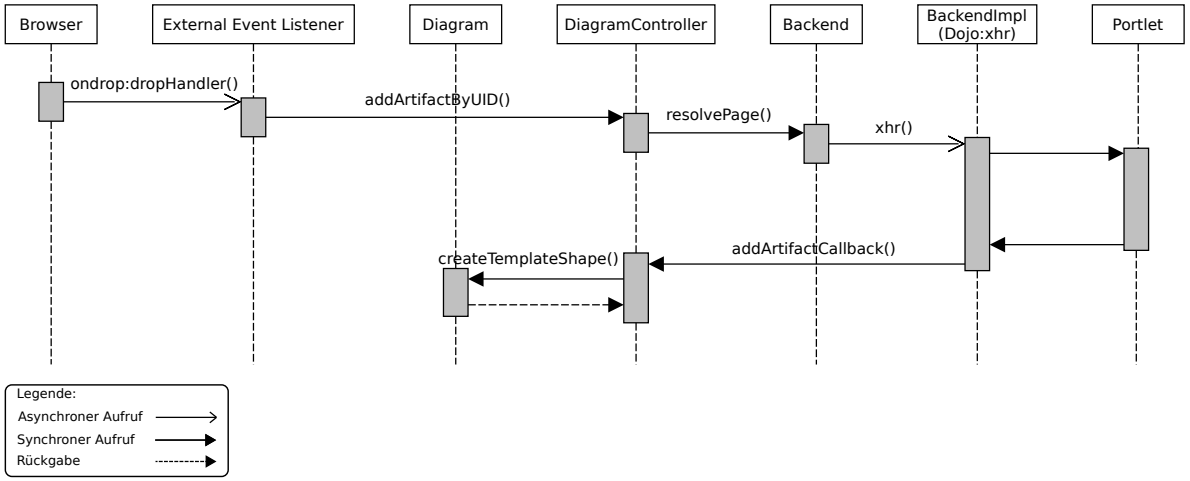

Abbildung 7.5.: Verarbeitung eines externen Drag and Drop Ereignisses im Modellierungswerkzeug (Portalseite oder Portlet hinzufügen).

Nachdem das Element, das hinzugefügt werden soll, in dem Modellierungswerkzeug abgesetzt (Drop) wurde, sendet der Browser ein *ondrop* Ereignis. Der Externe Event Listener empfängt das Ereignis und ruft die dafür definierte *dropHandler* Methode auf. Die *dropHandler* Methode liest die Unique Names aus dem Ereignis aus und gibt diese an die *AddArtifactByUID* Methode des *DiagramControllers* weiter. Der *DiagramController* ruft in der Backend Schnittstelle die Methode *resolvePage* auf, um die zugehörige Portalseite mit allen darin eingebetteten Portlets zu erhalten. Der Aufruf ans Backend geschieht über die Backend Implementierung *PortletBackend*. In diesem Fall wird dazu das Dojo Modul *xhr* verwendet, das einen *XMLHttpRequest*[5] an das Portlet im Portal Server sendet. Der Aufruf des *xhr* geschieht asynchron, damit, während die Abfrage auf der Serverseite ausgeführt wird, das Modellierungswerkzeug nicht blockiert und der Modellierer währenddessen weiter arbeiten kann. Nach der Bearbeitung im Portal Server wird das Ergebnis im JSON[6] Format zurückgesendet. Das *xhr* Modul ruft dann das *AddArtefactCallback* im *DiagramController* auf. Das Callback startet daraufhin die createTemplateShape Methode des ILOG Dojo Diagrammer Diagramms, wodurch die entsprechenden Knoten im Graphen erstellt werden.

Damit die Anwendung nicht an diese Implementierung des Event Listeners gebunden ist, ist dieser als eigenständige Komponente implementiert. Durch diese Trennung kann die Implementierung einfach durch eine andere ersetzt werden.

[5]**XMLHttpRequest**: Bei dem XMLHttpRequest handelt es sich um eine API für den Datenaustausch zwischen einem Client und einem Server. Das W3C arbeitet derzeit an einem Standard [ASH12].

[6]**JavaScript Object Notation** (JSON): JSON ist ein sprachunabhängiges Dateiformat zur Beschreibung von Daten. Weitere Informationen können unter `http://json.org` gefunden werden.

Backend

Der Backend Namespace beinhaltet vier Klassen.

Die *_AbstractBackend* definiert alle Methoden die eine Backend Implementierung zur Verfügung stellen muss. Alle Klassen, die diese Schnittstelle bereitstellen soll, erben von dieser abstrakten Klasse. Da Dojo kein Mittel für das definieren von abstrakten Klassen bereitstellt, wird dieses Verhalten durch den folgenden Ansatz simuliert: Methoden die als abstrakt definiert werden sollen, beinhalten im Methodenrumpf lediglich die Ausgabe einer Fehlermeldung, zum Beispiel „Fehler! Methode <*METHODEN NAME*> wurde nicht implementiert!“. Klassen die von dieser Klasse erben, müssen daher die entsprechenden Methoden überschreiben, um die Ausgabe dieser Fehlermeldung zu unterbinden.
Um das Modellierungswerkzeug bei einem Aufruf einer Methode nicht zu blockieren, finden die Aufrufe asynchron statt. Dazu muss der aufgerufen Methode ein Callback übergeben werden, welches nach der Ausführung dieser Methode aufgerufen werden kann.

Die Klasse *PortletBackend* ist eine mögliche Implementierung für die Kommunikation mit dem Portlet auf dem Portal Server. Dazu nutzt die PortletBackend Klasse das Dojo *xhr* Modul. Darüber können *XMLHttpRequest* an das Portlet gesendet werden, um Methoden des Portlets auf der Serverseite aufzurufen. Das PortletBackend erbt von der _AbstractBackend Klasse.

Die Klasse *DummyBackend* ist ein Testtreiber um das Systemverhalten zu testen, ohne dass eine Verbindung zum Portal Server erforderlich ist. In diesem Fall sind die Rückgabewerte der Methoden statisch definiert. Das DummyBackend erbt von der _AbstractBackend Klasse.

Die *Backend* Klasse ist für die Abstraktion der konkreten Implementierung zuständig. Sie wird bei allen Klassen für die Aufrufe des Backend verwendet und agiert als Vermittler zwischen der aufrufenden Komponente und der eigentlichen Implementierung. Welche Implementierung verwendet werden soll, wird in dieser Klasse festgelegt. Für eine Umstellung vom DummyBackend zum PortletBackend muss lediglich eine Zeile im Quelltext der *Backend* Klasse geändert werden.

Dialoge

Für die Darstellung der Dialoge wird das Dojo Modul *Dijit.Dialog* verwendet. Dieses Modul stellt ein Widget bereit, das einen leeren Dialog implementiert. Dieser Dialog kann dann mittels weiterer Widgets und HTML-Elementen erweitert werden. Für die Gestaltung, Initialisierung und Steuerung des Widgets, ist für jeden benötigten Dialog eine eigene *Dialog Klasse* implementiert, zum Beispiel für die Definition der Ereignisse einer Transition (Event Mapping Dialog). Dabei handelt es sich um eine Controller Klasse für den Dialog. Diese Klasse enthält eine Referenz auf den eigentlichen Dialog (*Dijit.Dialog*). Der *DiagramController* instantiiert bei Bedarf diese Klasse und registriert deren Event Handler im Diagramm. Damit kann auf die entsprechenden Ereignisse wie zum Beispiel das Klicken auf eine Transition reagiert werden. Das Diagramm ruft dann beim Eintreten eines solchen Ereignisses die registrierte Methode in der Dialogklasse auf. In Abbildung 7.6 ist das Klassendiagramm

der Dialogklassen abgebildet. Zur Veranschaulichung sind in der Abbildung die Beziehungen des Dialogs für die Festlegung der Ereignisse einer Transition hervorgehoben (Dicke Linien).

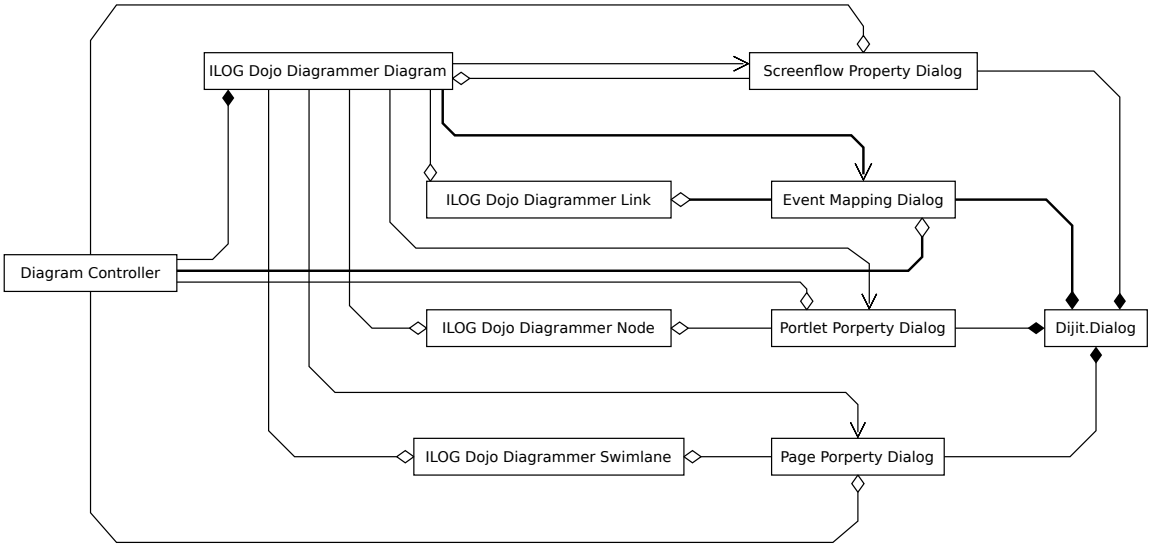

Abbildung 7.6.: Klassendiagramm der Dialogklassen mit ihren Abhängigkeiten.

Einstellungen, die im Dialog definiert werden, werden über die Metadaten Implementierung direkt im Datenmodell des Diagramms und der Graphelemente abgelegt. Auf die Implementierung für die Metadaten wird im folgenden Abschnitt genauer eingegangen. Abbildung 7.7 zeigt den fertigen Dialog für die Definition von Ereignissen.

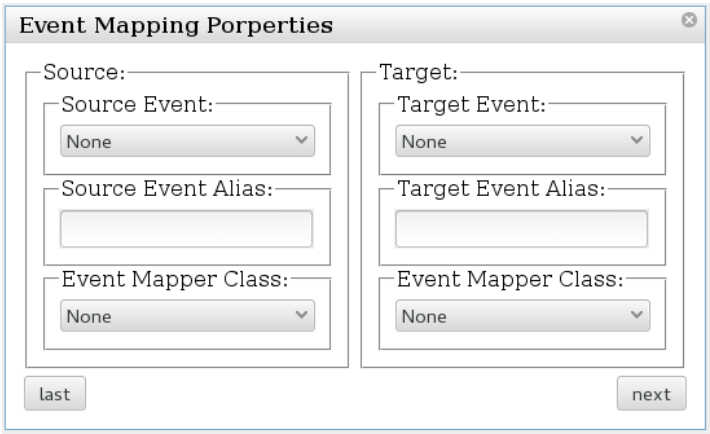

Abbildung 7.7.: Bildschirmfoto des konkret implementierten Dialogs für das Festlegen der Eigenschaften einer Transition.

Erweiterung des Datenmodells

Für das Speichern der für den Screenflow relevanten Daten, wird das Datenmodell der grafischen Elemente (*Diagram*, *Swimlane*, *Node* und *Link*) erweitert. Dazu ist das *Metadata Modul* zuständig. Das Modul bietet eine einheitliche API für das Laden und Speichern von Daten in das Datenmodell der grafischen Elemente. Das Modul erstellt innerhalb des Datenmodells ein Objekt namens _metaData, in dem dann alle Daten als Schlüssel-Wert-Paare abgelegt werden. Durch die API ist der Zugriff auf diese Daten einheitlich implementiert. Im folgenden wird auf die Erweiterung des Serialisierers und Deserialisierers eingegangen, die notwendig ist, damit auch die Metadaten gesichert werden können.

Serialisierung und Deserialisierung

Das Diagram des IBM ILOG Dojo Diagrammer verfügt bereits über eine *Serialisierungs-* und *Deserialisierungsfunktionalität*. Diese wird unter anderem dafür eingesetzt, den Zustand eines Objektes für das Rückgängigmachen von Änderungen (undo) zu speichern. Die *DiagramSerializer* Klasse serialisiert und deserialisiert nur die Daten des Datenmodells, die für die Wiederherstellung des Graphen und der Graphelemente unbedingt nötig sind. Daher wurde eine neue Klasse vom *DiagramSerializer* abgeleitet und erweitert. Dadurch können auch die Metadaten aus dem Datenmodell serialisiert und deserialisiert werden. Der Zugriff auf die Metadaten geschieht mit der Hilfe des *Metadata Moduls*. Zusätzlich wurde das Diagramm erweitert, sodass auf den Serialisierer und Deserialisierer auch über den Dialog Controller zugegriffen werden kann.

7.2. Serverseite

Die Serverseite hat zwei Zuständigkeitsbereiche. Sie ist für die Bearbeitung der Anfragen zuständig, die von der Clientseite gesendet werden. Die Anfragen werden über einen asynchronen URL Request an das Portlet gesendet. Zu den Anfragen gehören Abfragen über die Details von Portalseiten und Portlets, aber auch das Persistieren des Screenflows im Portal Server. Des Weiteren wird durch die Serverseite der Zustand des Modellierungswerkzeugs (Serialisierung des Graphen) über die Session verwaltet. Zusätzlich wird durch die Serverseite der aktuelle Prozessschritt des Modellierungsprozess gesetzt. Dazu wird über ein Action-Request der gewünschte Prozessschritt als Parameter an die Serverseite übermittelt.

Aus zeitlichen Gründen war es nicht möglich die Serverseite des Modellierungswerkzeug zu implementieren. Im Folgenden werden daher nur die möglichen Konzepte einer Implementierung beschrieben.

7.2.1. Architektur der Serverseite

Auf der Serverseite ist die Architektur in zwei Schichten eingeteilt. Eine Schicht für die Anwendungslogik, die auf der Serverseite für die Bearbeitung von Anfragen aus der Clientseite zuständig ist und auf die Dienste und Schnittstellen des Portal Servers zugreift (Java Klasse) sowie eine Präsentationsschicht die für die Auslieferung des Rich-Clients an den Client verantwortlich ist (JSP).

Zustandsübergänge

Ein Zustandswechsel im Modellierungsprozess wird durch einen Action-Request an das Portlet eingeleitet. Bei einem Action-Request handelt es sich um einen Aufruf einer speziellen URL, die im Portlet als Action-URL definiert ist (siehe Kapitel Grundlagen). Dieser URL wird ein Parameter mitgegeben, der den gewünschten Folgezustand des Modellierungsprozess enthält. Durch den Aufruf der Action URL wird im Portlet die Methode *processAction* aufgerufen. In dieser Methode kann über das *ActionRequest* Objekt auf die übergeben Parameter zugegriffen werden. Die *processAction* Methode hat lediglich die Aufgabe den Prozessschritt auszulesen und diesen an die Java Server Page (JSP) weiterzugeben, aus der die HTML Repräsentation des Portlets generiert wird. Bei der Interpretation der JSP wird der Parameter (Zustand des Modellierungsprozess) im JavaScript Code dem *DiagramController* als Parameter hinzugefügt. Durch die Interpretation des JavaScripts im Browser wechselt das Modellierungswerkzeug in den angeforderten Schritt des Modellierungsprozess. Der aktuelle Prozessschritt ist durch dieses Vorgehen immer in der aktuellen URL codiert. URLs ohne Parameter führen zum ersten Schritt im Modellierungsprozess. Abbildung 7.8 zeigt den Ablauf eines Wechsels in einen anderen Modellierungsprozessschritts.

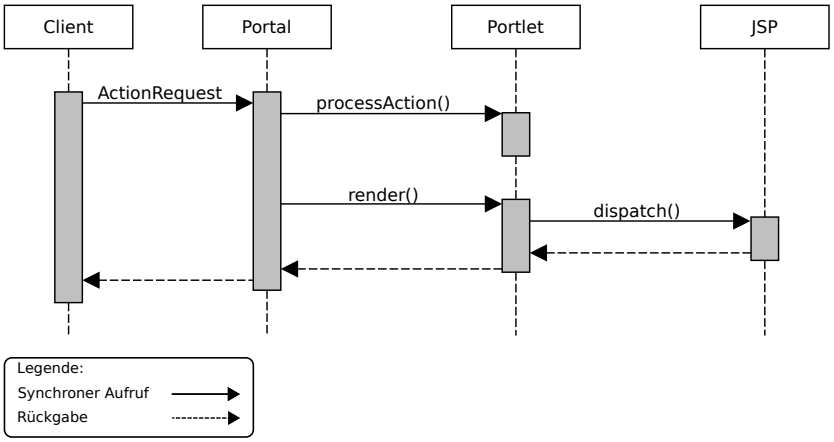

Abbildung 7.8.: Sequenzdiagramm eines Schrittwechsels im Modellierungsprozess.

Verarbeitung der Client Anfragen

Über die *serverResource* Methode bietet das Portlet einen Mechanismus, Anfragen an das Portlet zu senden, ohne dass vom Portal die Seite neu geladen wird. So lassen sich Ressourcen bereit stellen oder asynchrone Clientanfragen über *Ajax* beantworten. Abbildung 7.5 veranschaulicht den Ablauf eines asynchronen Aufrufs an das Portlet. Dieser Mechanismus wird ausgenutzt, um Abfragen von der Clientseite des Modellierungswerkzeugs an die Serverseite zu senden. Über die *serverResource* Methode können die Parameter aus der aufgerufenen URL ausgewertet werden. Die Parameter werden dann dazu verwendet, Methodenaufrufe im Portal Server auszuführen, zum Beispiel um die Unique Names der Portlets zu erhalten, die in einer Portalseite enthalten sind. Das Ergebnis der Ausführung wird dann an die Clientseite zurückgesendet.

7.3. Zusammenfassung

Das Modellierungswerkzeug ist als Portlet implementiert. Die Anwendung basiert auf einer Client-Server Architektur.
Die Clientseite: die aus JavaScript, HTML und CSS besteht und die Serverseite, die in Java implementiert ist. Für die Implementierung der Clientseite werden die JavaScript Frameworks Dojo Toolkit und der IBM ILOG Dojo Diagrammer verwendet. Die Clientseite folgt dem Entwurfsmuster MVC. Das Model und der Großteil der View sind durch die IBM ILOG Dojo Diagrammer Komponenten umgesetzt. Des Weiteren werden Dojo Dialoge in der View verwendet. Zusätzlich sind der Controller und sonstige Hilfsklassen auf Basis vom Dojo Toolkit entwickelt.
Die Serverseite liefert zu Beginn der Session das Modellierungswerkzeug (Clientseite) an den Client aus. Anschießend dient das Portlet der Bearbeitung von Anfragen, die von der Clientseite an den Portal Server gesendet werden. Auch die Verwaltung des aktuellen Schritts im Modellierungsprozess ist im Portlet auf der Serverseite implementiert.

8. Übertragung der Konzepte

Im folgenden Kapitel wird der zweite Teil der Aufgabenstellung aus dem Einleitungskapitel behandelt. Dazu werden die in Kapitel 6 erarbeiten Konzepte für Modellierungswerkzeuge von Screenflows auf Modellierungswerkzeuge von Scientific Workflows übertragen.

8.1. Screenflows und Scientific Workflows

Screenflows und Scientific Workflows können nicht direkt miteinander verglichen werden, da sie für unterschiedliche Zwecke entwickelt wurden. Screenflows wurden für das Routing von Benutzern durch Screens entwickelt. Im Gegensatz dazu dienen Scientific Workflows der Ausführung von Aufgaben. Während Screenflows ausschließlich der Interaktion mit dem Benutzer dienen, haben Scientific Workflows in der Regel genau das gegenteilige Ziel. Sie sind meist auf eine massive Datenverarbeitung ausgelegt. Interaktionen mit dem Benutzer sind während der Ausführung nur für Ausnahmesituationen vorgesehen. Während bei Scientific Workflows der Wissenschaftler meist an allen Phasen des Lebenszyklus eines Scientific Workflows arbeitet, ist bei Screenflows der Modellierer lediglich für die Modellierung des Screenflows zuständig. Ausgeführt wird ein Screenflows im Normalfall vom Endnutzer. Screenflows sind für eine kurze Ausführung ausgelegt. Ein Screenflow muss innerhalb der Gültigkeitsdauer einer Browser Session ausgeführt werden. Längere Prozesse müssen durch eine Workflow Engine ausgeführt werden. Scientific Workflows dagegen können kurze oder langläufige Prozesse sein. Bei Screenflows wird der Zustand des Dialog Modells nicht persistiert. Das bedeutet, dass der Screenflow erneut ausgeführt werden muss, wenn der Browser während der Ausführung geschlossen wird. Dabei können Eingaben verloren gehen. Aktivitäten die bis zu diesem Zeitpunkt ausgeführt wurden, werden nicht rückgängig gemacht. Im Gegensatz dazu können Scientific Workflows Techniken wie zum Beispiel Transaktionen einsetzen, um immer einen konsistenten Zustand zu gewährleisten. Der Screenflow Manager erlaubt keine Deklaration von Bedingungen in seinem Modell. Das Routing anhand von definierten Bedingungen muss über die verwendeten Portlets implementiert werden. Eine Workflow Engine ist dazu jedoch in der Lage. Screenflows unterstützen innerhalb eines Browser Tabs keine Parallelität. Ein Mensch kann zu einem Zeitpunkt auch nur mit einem Screenflow aktiv interagieren. Im Gegensatz dazu ist bei den Scientific Workflows für eine massive Datenverarbeitung Parallelität sogar sehr wichtig.

Neben den Unterschieden zwischen Screenflows und Scientific Workflows können auch Gemeinsamkeiten identifiziert werden. Zum Beispiel ist sowohl der Modellierer der Screenflows als auch der Wissenschaftler, der Scientific Workflows modelliert, in der Regel kein Computerexperte.

Trotz der teilweise großen Unterschiede können Teile der erarbeiten Konzepte, die für ein Modellierungswerkzeug für Screenflows entwickelt wurden, auf Modellierungswerkzeuge für Scientific Workflows übertragen werden. Dies wird im folgenden Abschnitt beschrieben.

8.2. Übertragung der erarbeiten Konzepte auf Modellierungswerkzeuge für Scientific Workflows

Eines der grundlegenden Konzepte für das Modellierungswerkzeug von Screenflows ist die Implementierung als Portlet. Dieses Konzept eignet sich auch gut für die Entwicklung von Modellierungswerkzeugen für Scientific Workflows. Da ein Wissenschaftler in der Regel alle Phasen des Business Process Lifecycles bearbeitet, könnten die Werkzeuge für die einzelnen Phasen als Portlets implementiert werden. Die einzelnen Portlets könnten dann über das Portal zu einer zusammenhängenden Oberfläche integriert werden. Zusätzlich könnten Portlets für spezifische Anforderungen entwickelt werden, die dann den Wissenschaftlern nach dem Baukastenprinzip zur Verfügung stehen. Der Wissenschaftler kann so die Benutzungsschnittstelle nach den eigenen Bedürfnissen erweitern. Es wäre zum Beispiel vorstellbar, ein Portlet für die Visualisierung von Simulationsdaten bereitzustellen. Durch Technologien wie JavaScript oder WebGL[1] könnten sehr komplexe Modelle und Darstellungen im Browser visualisiert werden. So könnte ein generisches Scientific Workflow Management System, basierend auf einem Portal, aufgebaut werden. Die grafische Oberfläche kann dabei an die einzelnen Bedürfnisse der Wissenschaftler angepasst werden.

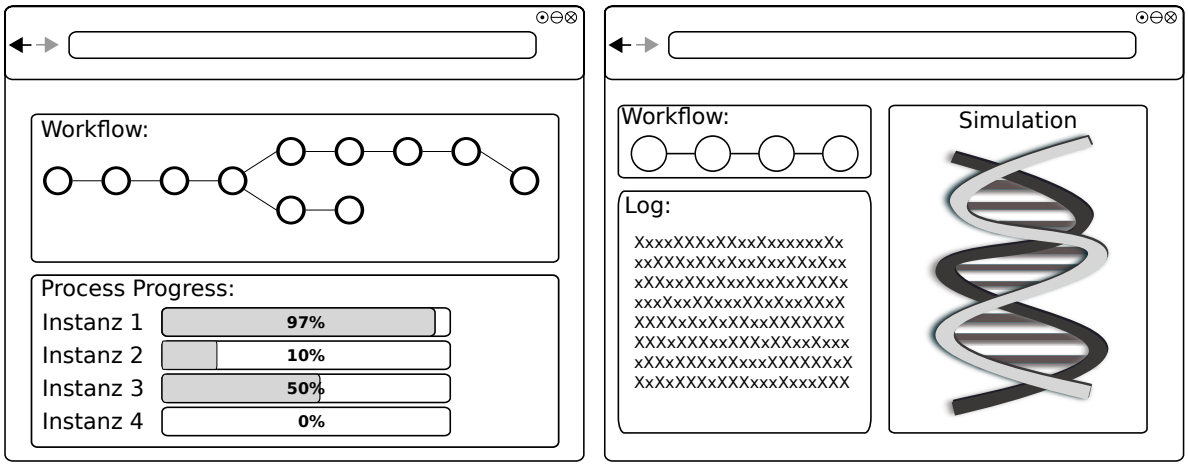

Abbildung 8.1.: Beispielhafte Darstellung eines Scientific Portals, das für die unterschiedlichen Bedürfnisse von Wissenschaftlern angepasst wird.

[1] **WebGL:** Eine OpenGL Implementierung für Webanwendungen. OpenGL ist ein Standard zur Entwicklung von 2D und 3D Anwendungen. Weitere Informationen zu OpenGL können unter http://www.opengl.org gefunden werden.

Abbildung 8.1 veranschaulicht beispielhaft den Aufbau eines solchen Portals. Jedes Fenster stellt dabei ein, individuell für einen Wissenschaftler, speziell angepasstes (Scientific) Portal dar. Das linke Beispiel besteht aus einem Portlet für die grafische Modellierung und Ausführung eines Scientific Workflows sowie einem Portlet zur Bobachtung des Prozessfortschritts. Das rechte Beispiel beseht aus drei Portlets: Einem für die Ausführung eines Scientific Workflows, einem für die Auflistung der Log-Daten und einem für die Visualisierung der Simulationsdaten.

Durch die Verwendung der Portal Technologie kann eine N-Tier Architektur verwendet werden, deren Komplexität für den Wissenschaftler verborgen bleibt. Abbildung 8.2 veranschaulicht beispielhaft den Aufbau einer solchen Anwendung.

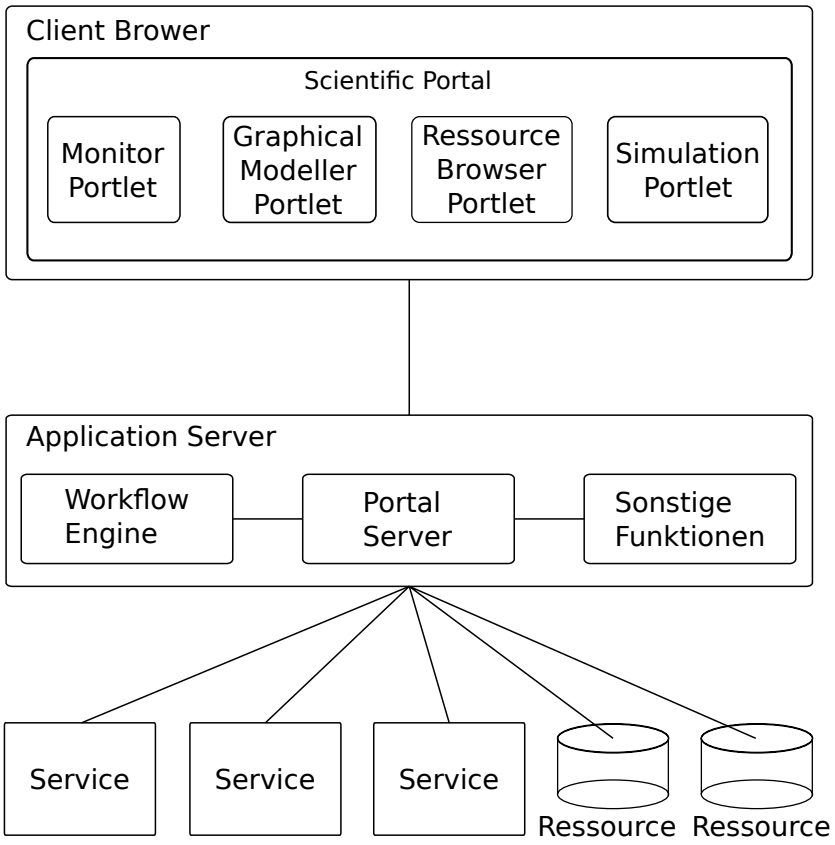

Abbildung 8.2.: Beispielhafte Darstellung der Komponenten einer N-Tier Architektur, mit dem Portal als Benutzungsschnittstelle.

Das Portal dient als Präsentationsschicht. Die Benutzungsschnittstelle ist so plattformunabhängig und es kann ortsungebunden darauf zugegriffen werden. Dies ist besonders bei den heterogenen Systemen, wie sie im wissenschaftlichen Bereich vorkommen, von Vorteil. Auf dem Application Server wird in diesem Beispiel das Portal und die Workflow Engine ausgeführt. Der Application Server bildet die Anwendungsschicht. Die Workflow Engine könnte aber auch auf einem anderen System ausgeführt werden. Der Application Server

dient neben der Ausführung des Portal Servers auch der Integration der Portalanwendung mit allen anderen Ressourcen und Diensten.

Das Einbetten des Modellierungswerkzeugs in die Werkzeugleiste des Portals, wie es für das Modellierungswerkzeug für die Screenflows vorgeschlagen wurde, eignet sich nicht für Scientific Workflows. Die Artefakte des Prozessmodells eines Scientific Workflows sind keine Elemente der grafischen Benutzungsschnittstelle. Der Modellierer muss daher nicht im Portal navigieren, um dem Prozessmodell die Komponenten hinzuzufügen. In die Werkzeugleiste des Portals könnten jedoch Werkzeuge aufgenommen werden, die oft von Wissenschaftlern benötigt werden. Vorstellbar wären auch Portlets für das Anzeigen von Statusinformationen oder zur Beobachtung des Prozessvorschritts.

Das Konzept für die grafische Darstellung von Screenflows kann nicht direkt für eine grafische Repräsentation von Scientific Workflows verwendet werden. Scientific Workflows sind viel umfangreicher und komplexer als Screenflows. Dennoch kann von dem Konzept übernommen werden, dass eine grafische Repräsentation die Komplexität einer Prozessmodellierung reduzieren kann. Wissenschaftler sind keine Computerexperten, daher ist eine Modellierung des Workflows in einer textbasierten Sprache, wie zum Beispiel XML, in der Regel zu komplex. Durch die Verwendung einer geeigneten grafischen Notation könnte diese Komplexität vor dem Modellierer verborgen werden.
Bei der Entwicklung einer grafischen Notation sind zwei Ansätze möglich.
Erstens, das Entwickeln einer unabhängigen Notation, die mit Hilfe einer Transformation in eine Prozessausführungssprache wie zum Beispiel BPEL transformiert wird. Die aus der Transformation resultierende Prozessausführungssprache kann dann von einer Workflow Engine ausgeführt werden. Die Notation wird mit Hilfe von Transformationsregeln in die spezifische Prozessausführungssprache überführt. Abbildung 8.3 veranschaulicht dieses Prinzip. In der Abbildung soll eine Transformation von einer grafischen Notation nach BPEL dargestellt werden. Dieser Ansatz hat den Vorteil, dass die Notation unabhängig von

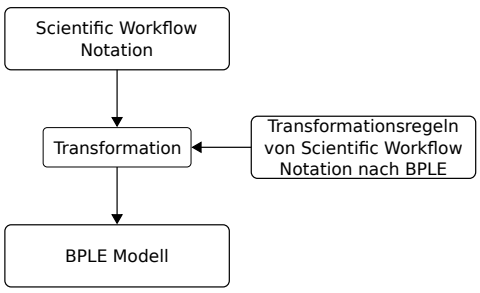

Abbildung 8.3.: Prinzip einer Transformation von einem grafisch repräsentierten Scientific Workflow in ein BPEL Modell.

der eingesetzten Workflow Engine ist. Zusätzlich können so verschiedene Notationen für unterschiedliche Anforderungen entwickelt werden. Für eine neue Notation müssen dann lediglich die Transformationsregeln angepasst werden. Das unterliegende System müsste aber nicht geändert werden. Nachteilig ist, dass die Entwicklung eines solchen Ansatzes komplex ist und dass bei jeder Änderung des Modells eine Transformation vor der Ausführung vorgenommen werden muss.

Der zweite Ansatz ist die Entwicklung einer Notation die von einer Workflow Engine interpretiert werden kann. Das würde bedeuten, dass auch die verwendete Workflow Engine erweitert oder selbst implementiert werden müsste. In diesem Fall wäre die Notation eng an die Workflow Engine gekoppelt.

Obwohl sich die Modellierungsprozesse von Screenflows und Scientific Workflows unterscheiden, kann das Konzept eines vordefinierten Modellierungsprozess, durch den der Modellierer geführt wird, auf Scientific Workflows übertragen werden. Eine Benutzerführung bietet sich immer an, wenn ein unerfahrener Benutzer ein System bedienen soll. Dies verbessert die Bedienbarkeit der Anwendung. Zusätzlich kann in jedem Prozessschritt überprüft werden, ob die benötigten Daten vorliegen und korrekt sind. Wie bei dem Modellierungswerkzeug für Screenflows können dem Wissenschaftler immer nur die Werkzeuge zur Verfügung gestellt werden, die er für die aktuelle Prozessphase benötigt.

8.3. Zusammenfassung

Screenflows und Scientific Workflows können nicht direkt miteinander verglichen werden. Sie wurden für unterschiedliche Aufgaben entwickelt. Dennoch können Teile der Konzepte, die in dieser Arbeit für Modellierungswerkzeug für Screenflows entwickelt wurden, auf Modellierungswerkzeuge für Scientific Workflows übertragen werden.
Die Portal Technologie eignet sich sehr gut für Benutzungsoberflächen von Scientific Workflows. Die Werkzeuge für die Bearbeitung des Prozesslebenszyklus können als Portlet implementiert und an einer zentralen Stelle miteinander integriert werden. Zusätzlich können noch weitere benötigte Werkzeuge so zur Verfügung gestellt werden. Des Weiteren kann die Oberfläche des Scientific Portals an die Bedürfnisse des Wissenschaftlers angepasst werden.
Für Scientific Workflows eignet sich das Konzept, das Modellierungswerkzeug in die Werkzeugleiste zu integrieren, nicht. Dies liegt daran, dass Wissenschaftler nicht im Portal umher navigieren müssen, um Artefakte des Portals dem Workflow Modell hinzuzufügen.
Durch die Verwendung einer grafischen Notation kann die Komplexität des unterliegenden Workflow Systems vor dem technisch weniger versierten Wissenschaftler verborgen werden. Durch geeignete Transformationstechniken kann die Notation in eine beliebige Prozessausführungssprache umgewandelt werden. Dadurch ist die Notation von der verwendeten Workflow Engine entkoppelt.
Die Verwendung eines vordefinierten Modellierungsprozesses, für Scientific Workflows, ermöglicht auch technisch nicht versierten Wissenschaftlern das Modellieren von Workflows.

9. Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Modellierung und Entwicklung eines grafischen Modellierungswerkzeugs für Screenflows. Des Weiteren sollten die erarbeiteten Konzepte auf Modellierungswerkzeuge von Scientific Workflows übertragen werden.

Zunächst wurde mit der Recherche und Einarbeitung der wichtigsten Technologien begonnen. In diesem Zuge wurde auch eine Recherche nach verwandten Arbeiten betrieben. Nach der Klärung der Rahmenbesinnungen wurde die Anforderungen an das Modellierungswerkzeug aufgestellt. Diese wurden hauptsächlich von der Screenflow Definition und in Gesprächen mit den Entwicklern erarbeitet. Der darauf folgende Hauptteil konzentrierte sich auf die Entwicklung der Konzepte für das Modellierungswerkzeug. Dabei wurden Entwürfe für die grafische Benutzungsschnittstelle (Mockups) und die Architektur des Modellierungswerkzeugs entwickelt.

Zu den wichtigsten Konzepten, die erarbeite wurden, zählt die Entwicklung des Modellierungswerkzeugs als Portlet, sowie das Einbetten des entwickelten Portlets in die Werkzeugleiste des Portals. Dies ermöglicht eine Modellierung der Screenflows zur Laufzeit. Des Weiteren die Definition eines Modellierungsprozesses, an dem der Modellierer der Screenflows während der Modellierung geführt wird und letztlich eine geeignete grafische Repräsentation für die Screenflows.

Die entwickelten Konzepte galt es anschließend im praktischen Teil der Arbeit prototypisch umzusetzen. Dafür wurde eine Webanwendung entwickelt. Für deren Umsetzung wurden JavaScript Frameworks eingesetzt, mit dessen Hilfe die Entwicklung der Clientseite stattgefunden hat. Vor der Verwendung der Frameworks, musste deren Eignung überprüft werden. Nach der Implementierung wurde als letzter Teil dieser Arbeit erörtert, inwieweit die erarbeiteten Konzepte auf Modellierungswerkzeuge für Scientific Workflows übertragen werden können.

Das Ergebnis dieser Arbeit sind Konzepte und Entwürfe für die Umsetzung eines Modellierungswerkzeugs für die visuelle Modellierung von Screenflows. Zusätzlich wurde prototypisch ein großer Teil der webbasierten Clientseite des Modellierungswerkzeugs umgesetzt.

Ausblick

Das Konzept des Screenflow Managers innerhalb des Portals ist eine vielversprechende Technologie. Screenflows können auch eingesetzt werden, um Abläufe im Portal ohne ein Workflowsystem zu modellieren. Sie sind leichtgewichtiger als Workflows und werden direkt im Portal Server ausgeführt. Es ist keine extra Installation eines Workflowsystems für deren Ausführung notwendig. Durch Umsetzung der erarbeiteten Konzepte und einer Fertigstellung des Modellierungswerkzeugs, würde eine Lösung existieren, um die sehr flexiblen Screenflows schnell und einfach an dynamische Prozesse von Kunden anzupassen.

Anknüpfungspunkte an die Arbeit

Die erarbeiteten Konzepte ermöglichen eine rudimentäre Umsetzung des Modellierungswerkzeugs. Im Rahmen der Arbeit wurden nur die für die Modellierung notwendigsten Funktionen betrachtet. Eine Erweiterung der Konzepte ist daher durchaus vorstellbar. Im Folgenden werden mögliche Anknüpfungspunkte an diese Arbeit vorgestellt.

Für den Modellierungsprozess wurde eine Phase *Test Screenflow* vorgestellt. Diese war jedoch nicht Teil dieser Arbeit. Für einen Modellierer kann der Test oder die Simulation eines Screenflows jedoch sehr von Nutzen sein. Für eine Umsetzung ergeben sich folgende Fragestellungen. Wie sollte ein Test oder eine Simulation von einem Screenflow gestaltet werden? Wie kann der Test oder die Simulation ausgeführt werden, da in der Regel Eingaben erforderlich sind und Systeme im Hintergrund diese Eingaben verarbeiten? Wie lässt sich ein Test oder eine Simulation eines Screenflows in das Modellierungswerkzeug integrieren?

Ein anderer Anknüpfungspunkt basiert darauf, dass die Screenflows derzeit über keine Fehlerbehandlung verfügen. Es wäre jedoch wünschenswert, im Fehlerfall einen Benutzer auf eine definierte Fehlerseite zu leiten. Eine Arbeit könnte sich mit der Entwicklung und der Umsetzung einer Fehlerbehandlung im Screenflow Manager und dem Modellierungswerkzeug befassen.

Auch die Einführung von Validierungsregeln innerhalb des Screenflows wäre denkbar. Mit Hilfe der Regeln könnte zum Beispiel überprüft werden, ob der Screenflow in einem definierten Dialogschritt alle benötigten Daten im Dialog Context vorliegen hat. Nur in diesem Fall würde der Screenflow weiter prozessiert werden. In der Arbeit könnte die Erweiterung des Dialog Modells, des Screenflow Managers und die entsprechende Adaptierung des Modellierungswerkzeugs behandelt werden.

Auch das Routing der Screenflows bietet Erweiterungsmöglichkeiten. Derzeit wird das Routing im Screenflow Manager anhand des aktuellen Dialogschritts und einem eingetretenen Ereignis entschieden. Dieses Konzept könnte erweitert werden, sodass auch die transportierten Daten in den Ereignissen beim Routing miteinbezogen werden. Alternativ könnte auch eine Schnittstelle für ein externes regelbasiertes Entscheidungssystem implementiert werden. Im Zusammenhang mit dieser Aufgabenstellung müsste der Screenflow Manager und das Modellierungswerkzeug um die entsprechende Funktionalität erweitert werden.

Ein weiterer möglicher Anknüpfungspunkt ist der Export von Screenflows. Eine Screenflow Definition kann mit den vorgestellten Konzepten als XML-Datei exportiert werden, um sie in einem anderen System zu importieren. Damit der Screenflow auf diesem System ausgeführt werden kann, müssen alle Artefakte des Screenflows auf dem System vorhanden sein. Eine weiterführende Arbeit könnte sich mit der Entwicklung eines Konzepts beschäftigen, das es einem Screenflow Modellierer erlaubt, einen Screenflows mit allen zugehörigen Komponenten zu exportieren. Der exportierte Screenflow sollte dann anschießend auf einem anderen System ausführbar sein, auch wenn Teile des Screenflows vor dem Import nicht auf dem System installiert waren.

A. Anhang

A.1. Dialogdefinition

Die folgende Dialogdefinition beschreibt einen Screenflow für einen Reisebuchungsprozess. Der Screenflow beginnt mit einem Portlet für die Auswahl des Reiseziels. Im darauffolgenden Schritt kann der Benutzer ein Hotel buchen. Danach kann der Benutzer einen Mietwagen auswählen. Zum Schluss erhält der Benutzer eine Zusammenfassung der Buchung.

Listing A.1 Dialogdefinition für einen Reisebuchungsprozess.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" type=""
     xsi:noNamespaceSchemaLocation="PCM_1.0.xsd">
  <portal action="update">
    <dialog-set>
      <dialog name="Travel Booking Dialog">
        <transition-endpoint name="portlet.pcm.demo.travelRequest">
          <localedata locale="en">
            <title>Travel Request</title>
            <description>Travel Request</description>
          </localedata>
          <resource uniquename="portlet.pcm.demo.travelRequest" />
          <invocation type="static" />
        </transition-endpoint>
        <transition-endpoint name="portlet.pcm.demo.flightBooking">
          <localedata locale="en">
            <title>Flight Booking</title>
            <description>Flight Booking</description>
          </localedata>
          <resource uniquename="portlet.pcm.demo.flightBooking" />
          <invocation type="static" />
        </transition-endpoint>
        <transition-endpoint name="portlet.pcm.demo.hotelBooking">
          <localedata locale="en">
            <title>Hotel Booking</title>
            <description>Hotel Booking</description>
          </localedata>
          <resource uniquename="portlet.pcm.demo.hotelBooking" />
          <invocation type="static" />
        </transition-endpoint>
        <transition-endpoint name="portlet.pcm.demo.carBooking">
          <localedata locale="en">
            <title>Car Booking</title>
```

```
33            <description>Car Booking</description>
34          </localedata>
35          <resource uniquename="portlet.pcm.demo.carBooking" />
36          <invocation type="static" />
37        </transition-endpoint>
38        <transition-endpoint name="portlet.pcm.demo.travelSummary">
39          <localedata locale="en">
40            <title>Travel Summary</title>
41            <description>Travel Summary</description>
42          </localedata>
43          <resource uniquename="portlet.pcm.demo.travelSummary" />
44          <invocation type="static" />
45        </transition-endpoint>
46
47        <transition type="start">
48          <source>
49            <transition-endpoint nameref="portlet.pcm.demo.travelRequest">
50              <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}travelRequest"
                    dcx-key="travelRequest" />
51            </transition-endpoint>
52          </source>
53          <target>
54            <transition-endpoint nameref="portlet.pcm.demo.flightBooking">
55              <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}travelRequest"
                    dcx-key="travelRequest" />
56            </transition-endpoint>
57          </target>
58        </transition>
59        <transition>
60          <source>
61            <transition-endpoint nameref="portlet.pcm.demo.flightBooking">
62              <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}flightBooking"
                    dcx-key="flightBooking" />
63            </transition-endpoint>
64          </source>
65          <target>
66            <transition-endpoint nameref="portlet.pcm.demo.hotelBooking">
67              <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}travelRequest"
                    dcx-key="travelRequest" />
68            </transition-endpoint>
69          </target>
70        </transition>
71        <transition>
72          <source>
73            <transition-endpoint nameref="portlet.pcm.demo.hotelBooking">
74              <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}hotelBooking"
                    dcx-key="hotelBooking" />
75            </transition-endpoint>
76          </source>
77          <target>
78            <transition-endpoint nameref="portlet.pcm.demo.carBooking">
79              <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}travelRequest"
                    dcx-key="travelRequest" />
80            </transition-endpoint>
81          </target>
```

```
82          </transition>
83          <transition>
84            <source>
85              <transition-endpoint nameref="portlet.pcm.demo.carBooking">
86                <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}carBooking"
                      dcx-key="carBooking" />
87              </transition-endpoint>
88            </source>
89            <target>
90              <transition-endpoint nameref="portlet.pcm.demo.travelSummary">
91                <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}travelSummary"
                      mapper-class="com.ibm.wps.portlet.mapper.TravelSummaryMapper" />
92              </transition-endpoint>
93            </target>
94          </transition>
95          <transition type="end">
96            <source>
97              <transition-endpoint nameref="portlet.pcm.demo.travelSummary">
98                <event qname="{http://portal.ibm.com/dialogmanager/pcm/demo}done" />
99              </transition-endpoint>
100           </source>
101           <target>
102             <transition-endpoint nameref="portlet.pcm.demo.travelRequest">
103               <event
                      qname="{http://www.ibm.com/xmlns/prod/websphere/portal/v6.1.0/portal-pcm}EndDialog"
                      />
104             </transition-endpoint>
105           </target>
106         </transition>
107       </dialog>
108     </dialog-set>
109   </portal>
110 </request>
```

A.2. Anwendungsfälle

Im Folgenden werden die Anwendungsfälle aufgestellt, welche sich aus den Anforderungen aus dem Kapitel 6. Konzept ergeben.

Name:	Dialogdefinition erstellen
Ziel:	Der Modellierer kann eine neue Dialogdefinition erstellen.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer bekommt die Liste der vorhanden Dialogdefinitionen angezeigt.
Ablauf:	Der Modellierer öffnet das Modellierungswerkzeug.
Nachbedingung:	Alle Aktivitäten im Modellierungswerkzeug werden in der Dialogdefinition gespeichert.

Tabelle A.1.: Anwendungsfall: Dialogdefinition erstellen

Name:	Dialogdefinition konfigurieren
Ziel:	Der Modellierer kann die Dialogdefinitionen konfigurieren.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat den Dialog für die Konfiguration der Dialogdefinition geöffnet.
Ablauf:	1. Der Modellierer vergibt einen eindeutigen Namen für die Dialogdefinition. 2. Der Modellierer legt einen Titel und eine Beschreibung fest.
Nachbedingung:	Nachdem der Modellierer den Dialog verlässt, sind die Änderungen in der Dialogdefinition übernommen.

Tabelle A.2.: Anwendungsfall: Dialogdefinition konfigurieren

Name:	Artefakte hinzufügen
Ziel:	Der Modellierer kann dem Screenflow Dialogartefakte hinzufügen.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat die Portalseite, die dem Screenflow hinzugefügt werden soll, geöffnet.
Ablauf:	Der Modellierer zieht die Portalseite oder Portlets, die sich auf ihr befinden, in das Modellierungswerkzeug.
Nachbedingung:	Der hinzugefügte Transition Endpoint wird im Modellierungswerkzeug visuell dargestellt.

Tabelle A.3.: Anwendungsfall: Artefakt hinzufügen

Name:	Dialogartefakt konfigurieren
Ziel:	Der Modellierer kann die Dialogartefakte konfigurieren.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat den Dialog für die Konfiguration eines Artefakts geöffnet.
Ablauf:	1. Der Modellierer legt einen Titel und eine Beschreibung in unterschiedlichen Sprachen fest. 2. Der Modellierer wählt, ob das Artefakt als dynamische Kopie instantiiert werden soll. 2.1 Der Modellierer gibt einen Extension Node an.
Nachbedingung:	Nachdem der Modellierer den Dialog verlässt, sind die Änderungen in der Dialogdefinition übernommen.

Tabelle A.4.: Anwendungsfall: Dialogartefakt konfigurieren

Name:	Transition definieren
Ziel:	Der Modellierer kann Transitionen zwischen Dialogartefakten definieren.
Akteure:	Modellierer
Vorbedingung:	Im Modellierungswerkzeug befindet sich mindestens ein Dialogartefakt.
Ablauf:	Der Modellierer zieht eine Verbindung von einem Dialogartefakt zu einem anderen oder zu demselben.
Nachbedingung:	Nachdem der Modellierer (zwei) Dialogartefakte miteinander verbunden hat, existiert die Transition (ohne definierte Ereignisse) in der Dialogdefinition.

Tabelle A.5.: Anwendungsfall: Transition definieren

Name:	Transition konfigurieren
Ziel:	Der Modellierer kann Transitionen konfigurieren.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat eine Transition geöffnet und der Dialog für das Konfigurieren der Transition wird ihm angezeigt.
Ablauf:	1. Der Modellierer wählt für den Quell Transition Endpoint ein Ereignis. 2. Der Modellierer wählt für den Ziel Transition Endpoint ein Ereignis. 3. Optional wählt der Modellierer für den Quell Transition Endpoint einen DCX-Key. 4. Optional wählt der Modellierer für den Ziel Transition Endpoint einen DCX-Key. 5. Optional wählt der Modellierer einen PayloadToContextMappers. 6. Optional wählt der Modellierer einen ContentToPayloadMapper. 7. Optional markiert der Modellierer die Transition als Starttransition oder Endtransition.
Nachbedingung:	Nachdem der Modellierer den Dialog verlässt, sind die Änderungen in der Dialogdefinition übernommen.

Tabelle A.6.: Anwendungsfall: Transition konfigurieren

Name:	Liste der Dialogdefinitionen anzeigen
Ziel:	Der Modellierer kann eine Liste der vorhanden Dialogdefinitionen anzeigen.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat das Modellierungswerkzeug geöffnet.
Ablauf:	Der Modellierer wählt den Button Dialogdefinitionen anzeigen.
Nachbedingung:	Der Modellierer bekommt die Liste der vorhanden Dialogdefinitionen angezeigt.

Tabelle A.7.: Anwendungsfall: Liste der Dialogdefinitionen anzeigen

Name:	Dialogdefinition anzeigen
Ziel:	Der Modellierer kann eine Dialogdefinition anzeigen.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat das Modellierungswerkzeug geöffnet und bekommt die Liste der Dialogdefinitionen angezeigt.
Ablauf:	Der Modellierer wählt eine Dialogdefinition aus der Liste.
Nachbedingung:	Die auswählte Dialogdefinition wird angezeigt.

Tabelle A.8.: Anwendungsfall: Dialogdefinition anzeigen

Name:	Dialogdefinition bearbeiten
Ziel:	Der Modellierer kann eine Dialogdefinition bearbeiten.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat das Modellierungswerkzeug geöffnet und bekommt die ausgewählte Dialogdefinition angezeigt.
Ablauf:	Der Modellierer bearbeitet den Graphen oder konfiguriert Artefakte.
Nachbedingung:	Die Änderungen werden in der Dialogdefinition gespeichert.

Tabelle A.9.: Anwendungsfall: Dialogdefinition bearbeiten

Name:	Dialogdefinition kopieren
Ziel:	Der Modellierer kann eine Dialogdefinition aus einer Vorlage (Template) erstellen.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat die Liste der Dialogdefinitionen geöffnet.
Ablauf:	1. Der Modellierer wählt eine Dialogdefinition aus der Liste. 2. Die Dialogdefinition wird geöffnet. 3. Anschießend kann er die Dialogdefinition unter einem neuen Namen speichern.
Nachbedingung:	Die neue Dialogdefinition ist gespeichert.

Tabelle A.10.: Anwendungsfall: Dialogdefinition kopieren

Name:	Dialogdefinition speichern
Ziel:	Der Modellierer kann eine oder mehrere Dialogdefinitionen speichern.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat die Dialogdefinition geöffnet.
Ablauf:	Der Modellierer klickt den Speichern-Button.
Nachbedingung:	Die Dialogdefinition ist gespeichert und kann im Modellierungswerkzeug angezeigt werden.

Tabelle A.11.: Anwendungsfall: Dialogdefinition speichern

Name:	Dialogdefinition exportieren
Ziel:	Der Modellierer kann eine Dialogdefinition exportieren.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat die Dialogdefinition geöffnet.
Ablauf:	Der Modellierer klickt den Export-Button.
Nachbedingung:	Die Dialogdefinition wird im Modellierungswerkzeug angezeigt.

Tabelle A.12.: Anwendungsfall: Dialogdefinition exportieren

Name:	Dialogdefinition importieren
Ziel:	Der Modellierer kann eine Dialogdefinition importieren.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat das Modellierungswerkzeug geöffnet.
Ablauf:	Der Modellierer klickt den Import-Button.
Nachbedingung:	Die Dialogdefinition wird im Modellierungswerkzeug angezeigt.

Tabelle A.13.: Anwendungsfall: Dialogdefinition importieren

Name:	Dialogdefinition löschen
Ziel:	Der Modellierer kann eine oder mehrere Dialogdefinitionen löschen.
Akteure:	Modellierer
Vorbedingung:	Der Modellierer hat die Liste der Dialogdefinitionen geöffnet.
Ablauf:	1. Der Modellierer wählt eine Dialogdefinition aus der Liste. 2. Der Modellierer klickt den Löschen-Button.
Nachbedingung:	Die gewählte Dialogdefinition wurde gelöscht.

Tabelle A.14.: Anwendungsfall: Dialogdefinition löschen

A.3. Mockups

Die folgenden Mockups wurden mit dem Wireframing Tool Balsamiq[1] erstellt. Die Mockups wurden während des Entwicklungsprozesses in mehreren Iterationen entwickelt. Die folgenden Mockups zeigen den Stand aus der letzten Iteration.
Die Mockups führen den Betrachter durch den Modellierungsprozess eines Screenflows. Dabei veranschaulichen die Mockups die geplanten Funktionen des Modellierungswerkzeugs. Zu Beginn werden im Modellierungswerkzeug ein Portlet und eine Portalseite hinzugefügt. Anschließend werden Transitionen definiert und Ereignisse dafür festgelegt. Danach folgen die Konfigurationsdialoge für den Screenflow, die Portalseiten und die Portlets. Daraufhin folgt der Test des Screenflows, der jedoch nicht näher spezifiziert ist. Anschließend wird der Screen zum Speichern oder Exportieren des Screenflows angezeigt.

[1] Weite Informationen zu Balsamiq können unter http://balsamiq.com gefunden werden.

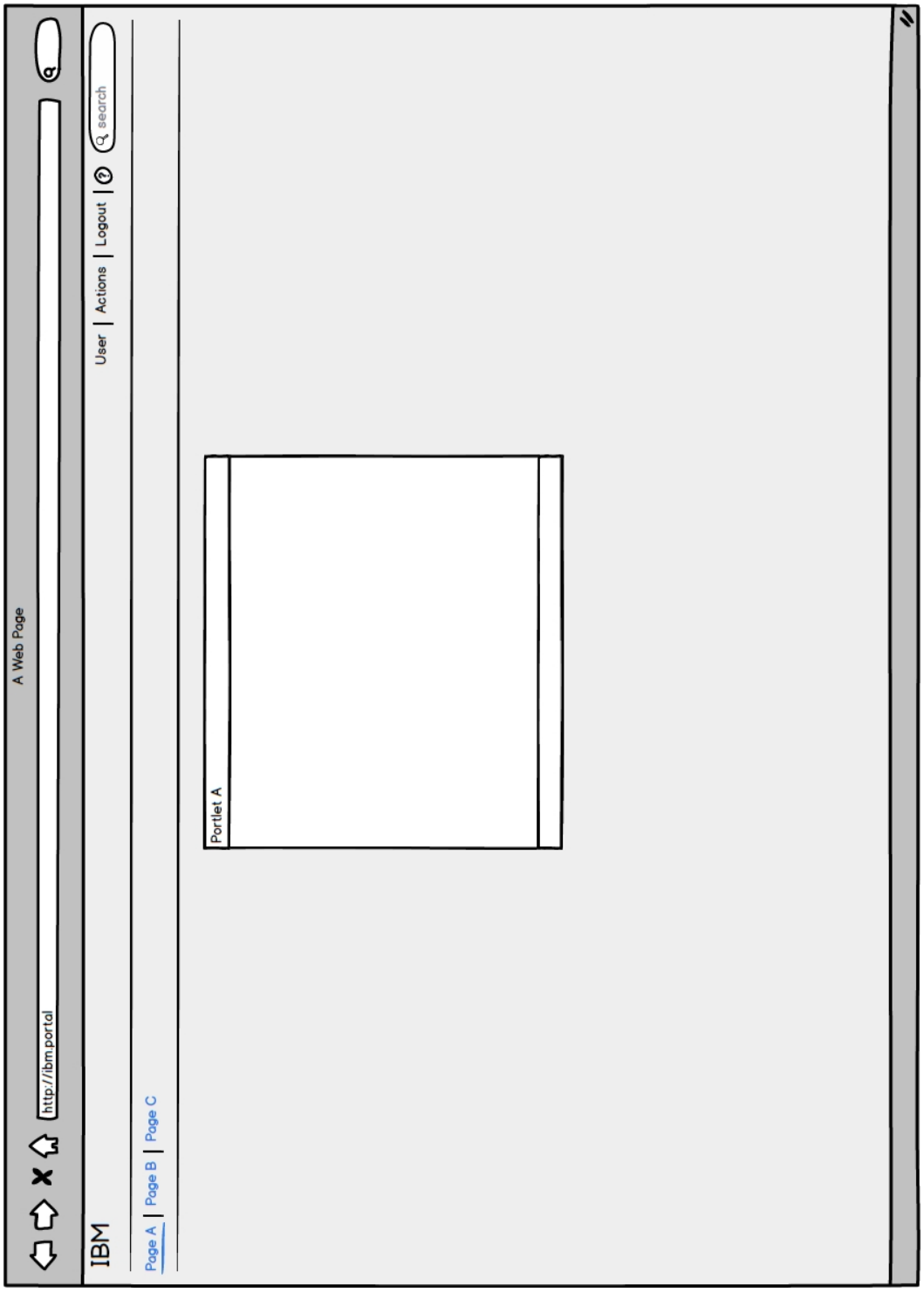

Abbildung A.1.: Dieses Mockup veranschaulicht den Inhalt der Portalseite A. Sie beinhaltet das Portlet A.

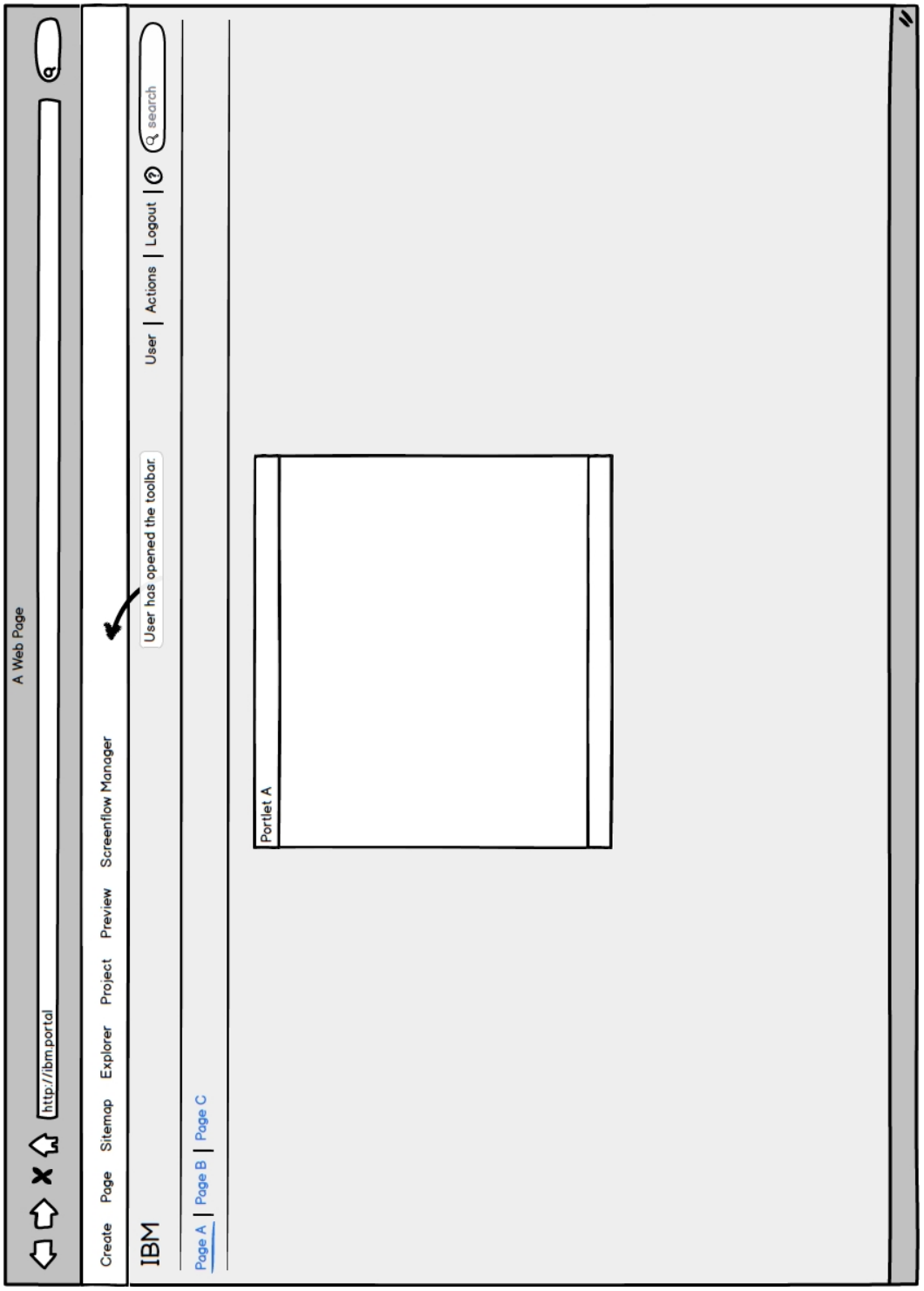

Abbildung A.2.: Dieses Mockup veranschaulicht den Inhalt der Portalseite A. Zusätzlich ist die Werkzeugleiste des Portals heruntergeklappt.

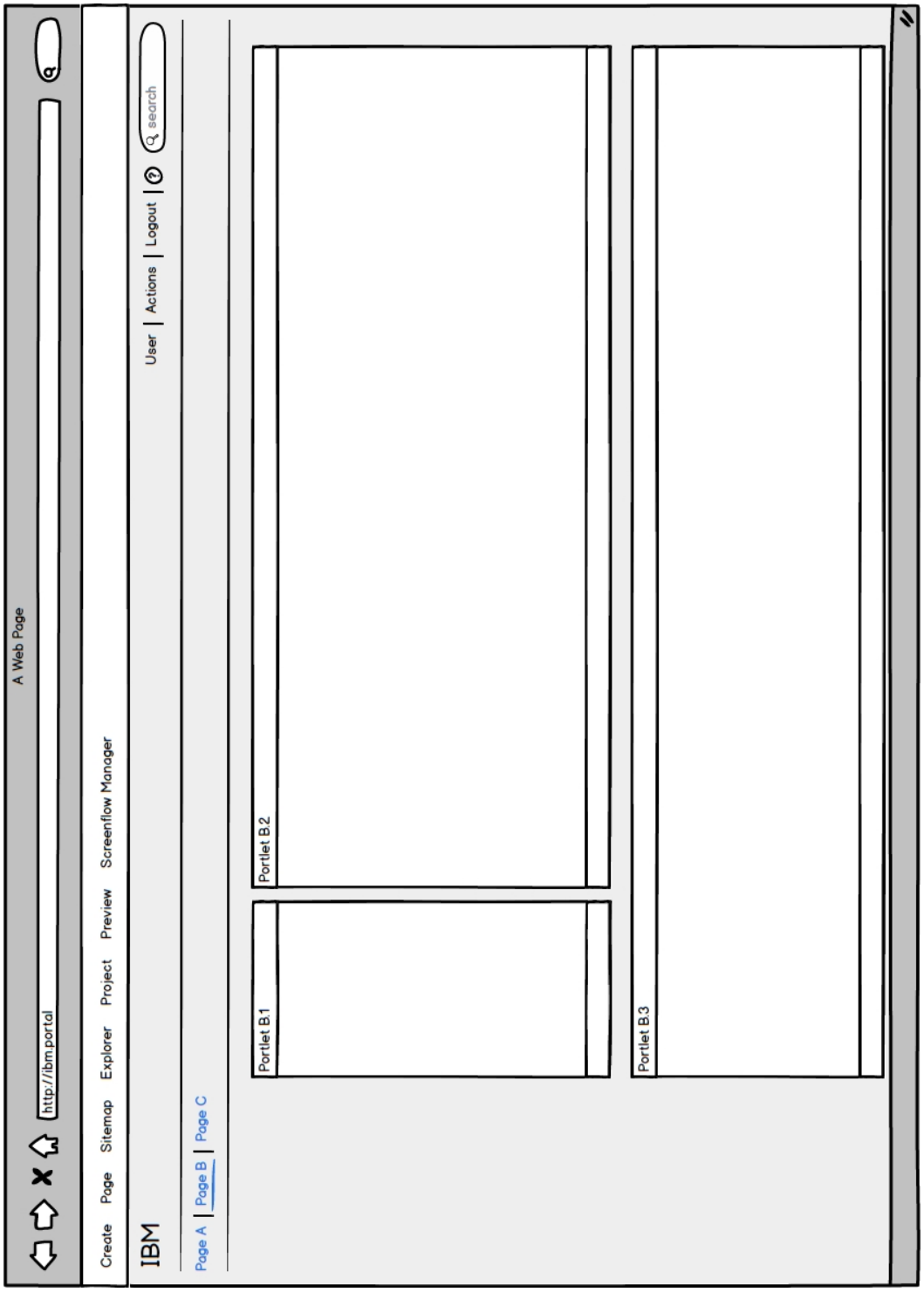

Abbildung A.3.: Dieses Mockup veranschaulicht den Inhalt der Portalseite B. Sie beinhaltet die Portlet B.1, B.2 und B.3.

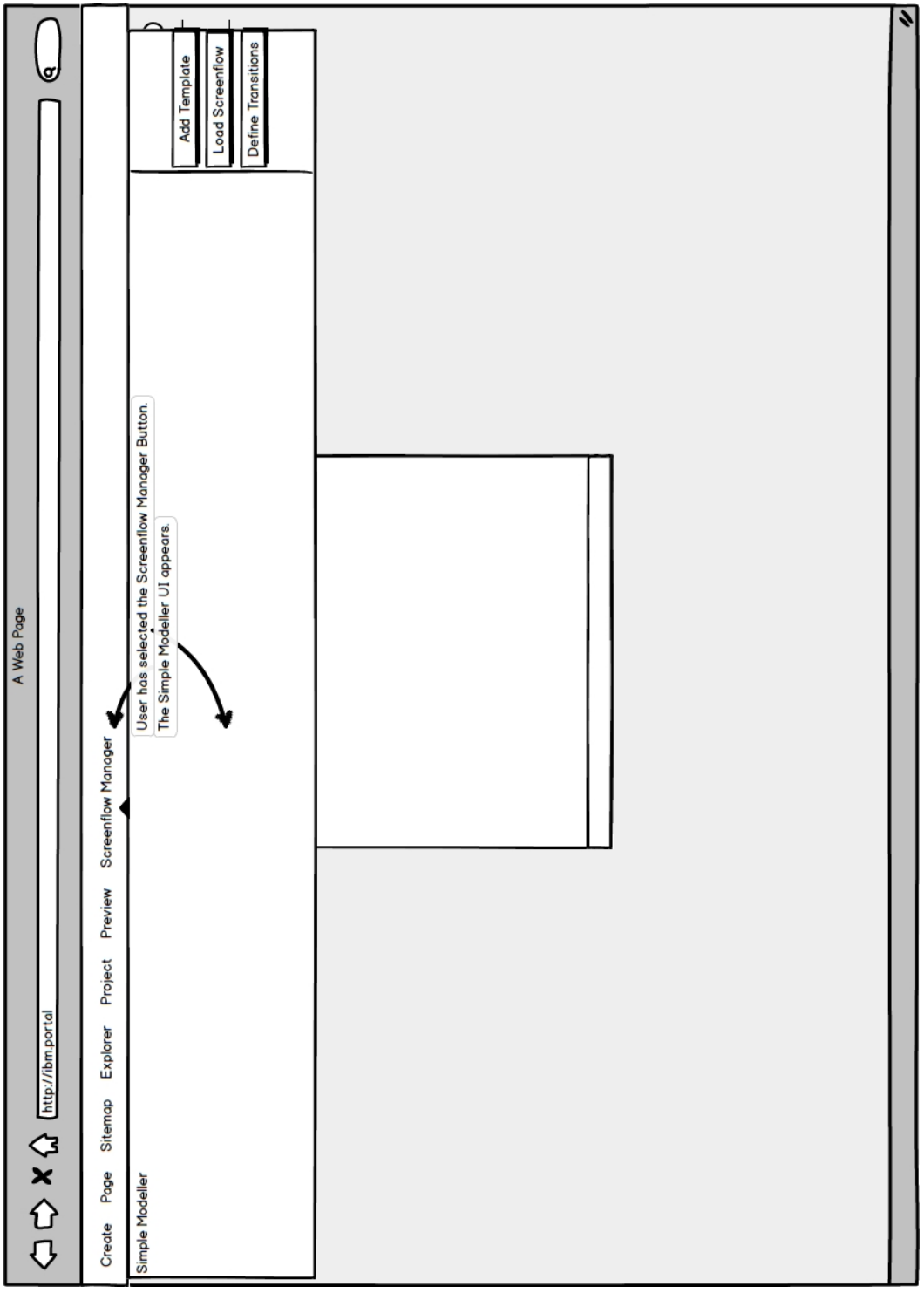

Abbildung A.4.: Dieses Mockup veranschaulicht das Modellierungswerkzeug im einfachen Modus. Der einfache Modus dient dazu, Dialogartefakte hinzuzufügen.

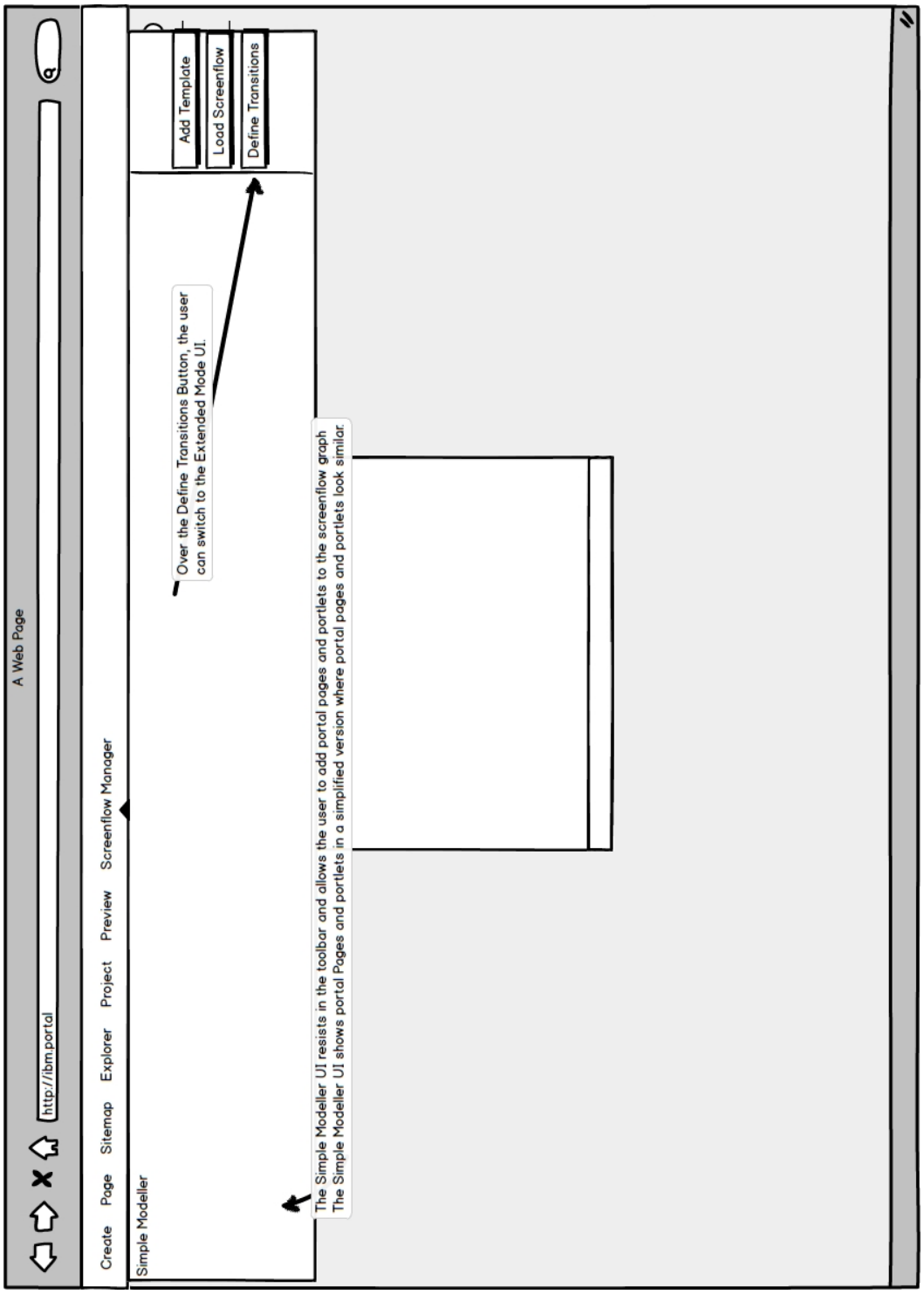

Abbildung A.5.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus und zeit, wie der Benutzer in den erweiterten Modus wechseln kann.

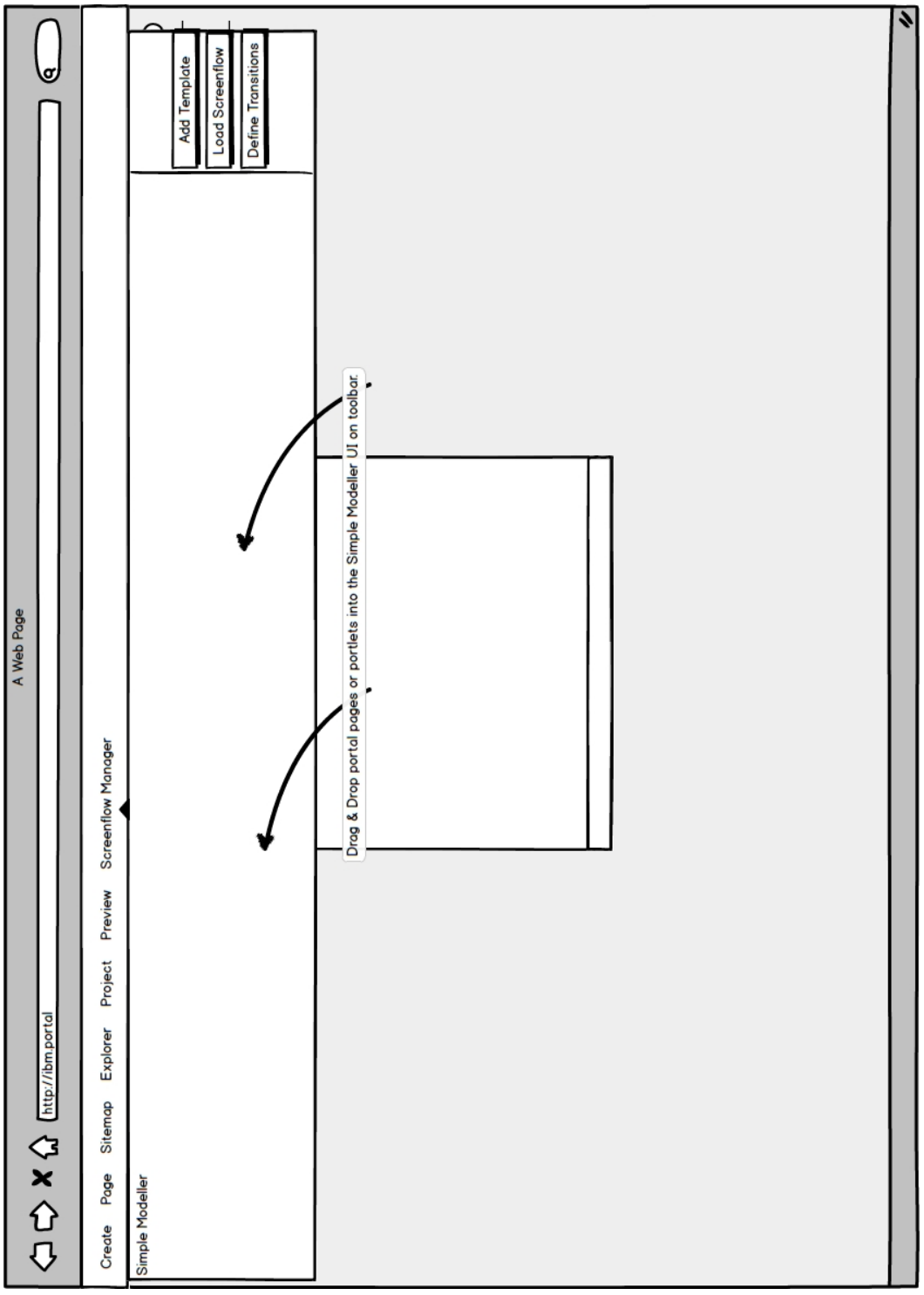

Abbildung A.6.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus und beschreibt, dass Portlets und Portalseiten per Drag and Drop in das Modellierungswerkzeug gezogen werden können.

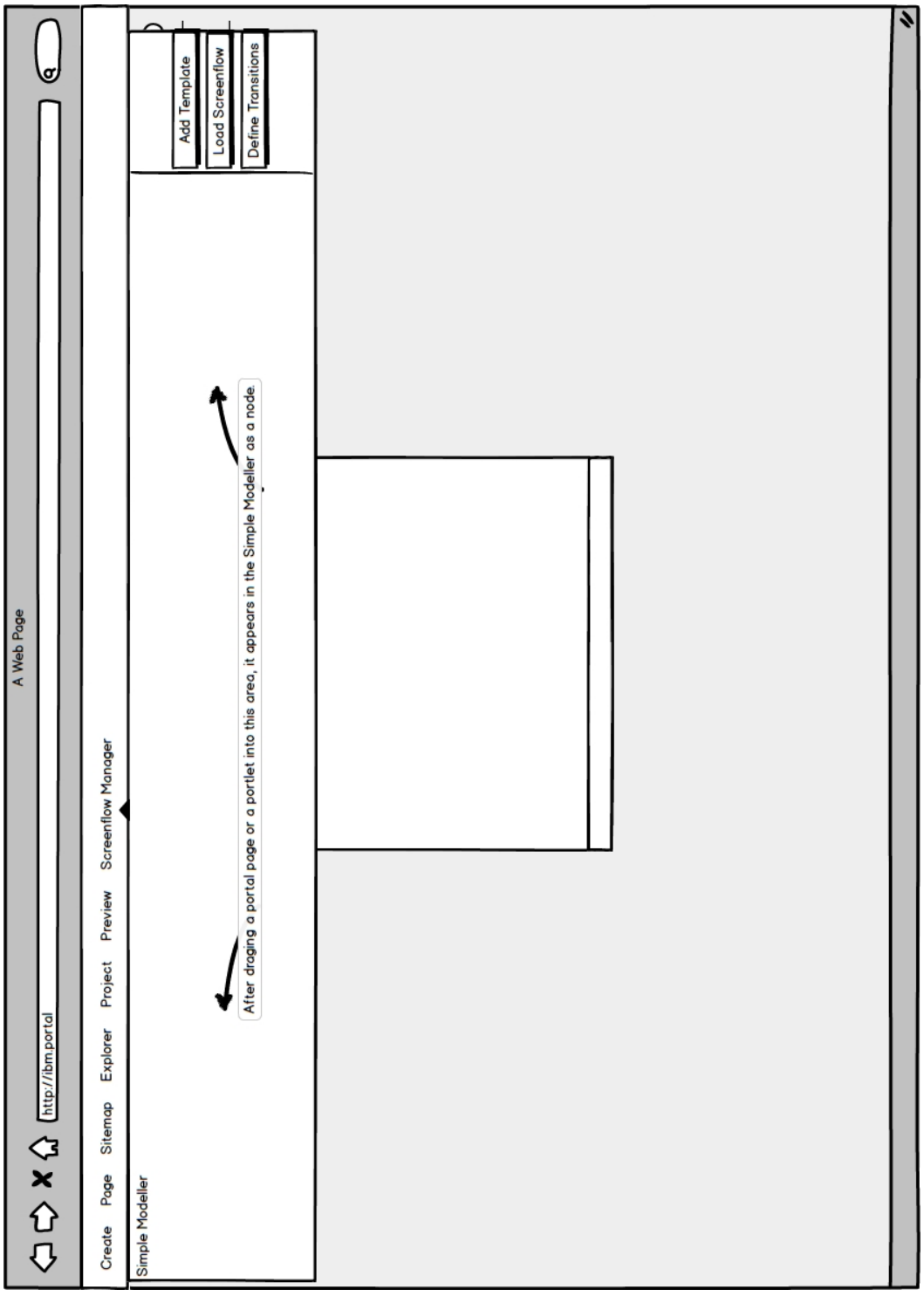

Abbildung A.7.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus und beschreibt, an welcher Stelle die Portlets und die Portalseiten dem Modellierungswerkzeug hinzugefügt werden.

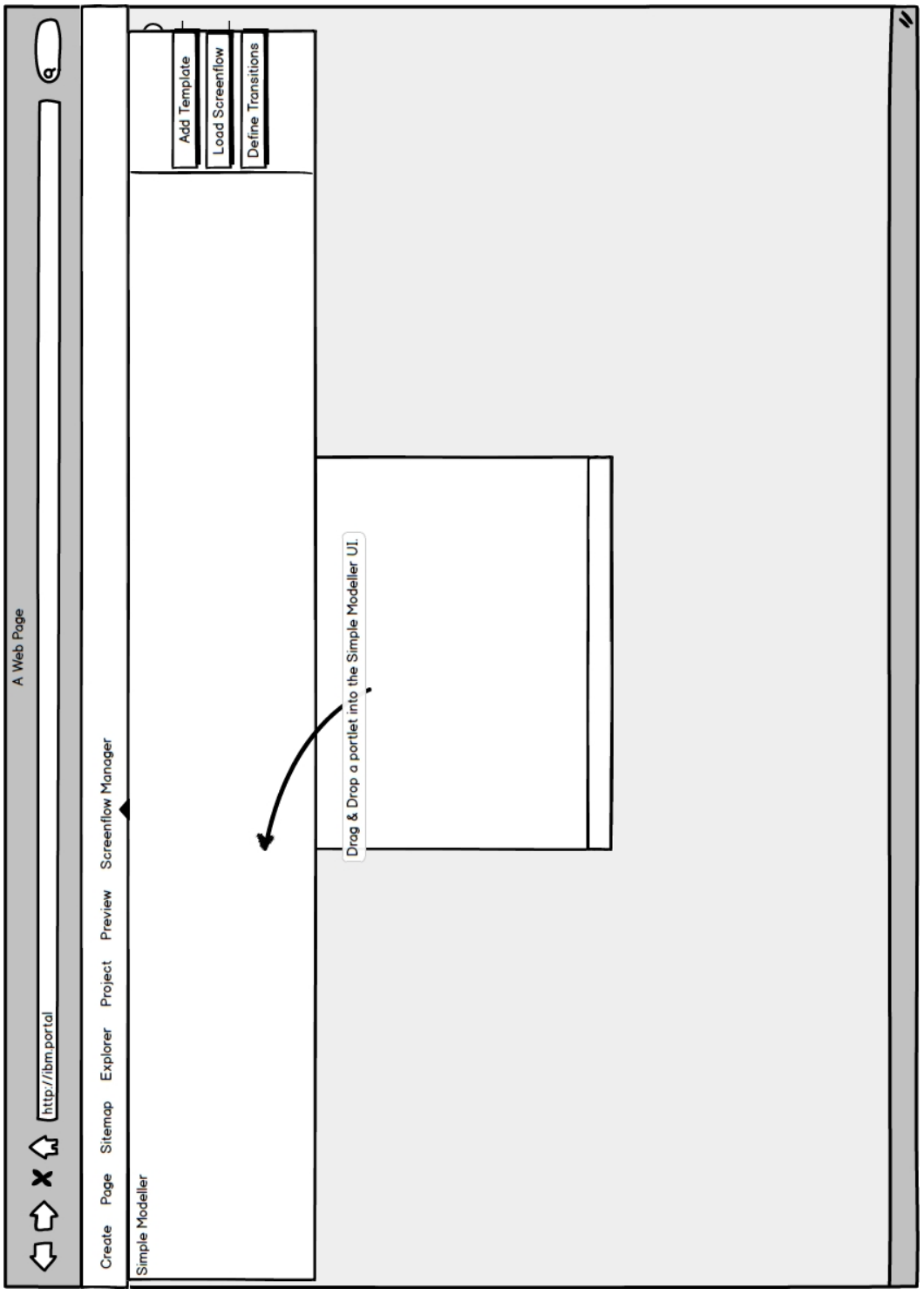

Abbildung A.8.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus und das Hinzufügen eines Portlets (Portlet A).

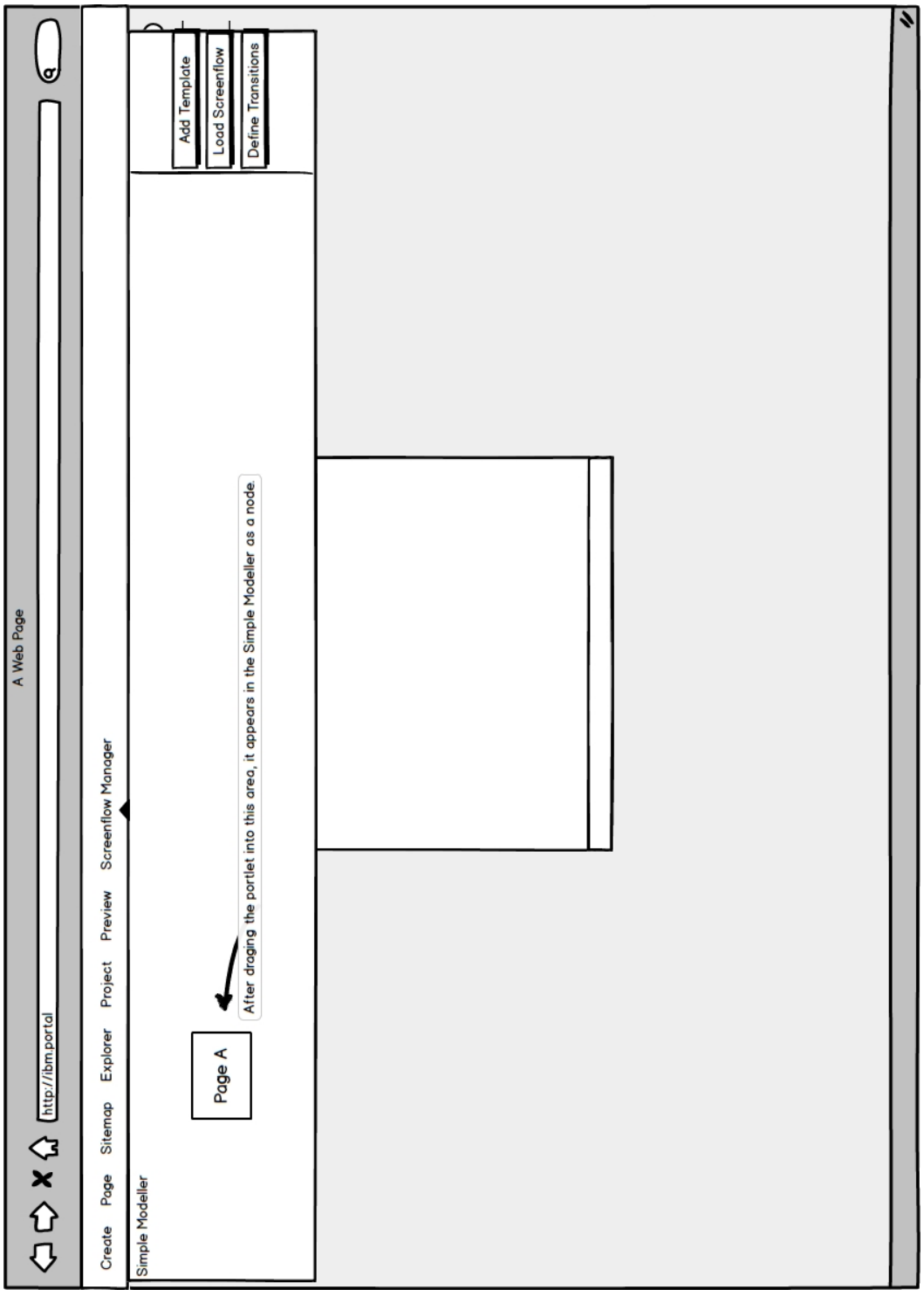

Abbildung A.9.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus und die hinzugefügte Portalseite des Portlets A. Ein Portlet muss immer in einer Portalseite enthalten sein. Da das Modellierungswerkzeug im einfachen Modus ist, wird die übergeordnete Portalseite des hinzugefügten Portlets angezeigt.

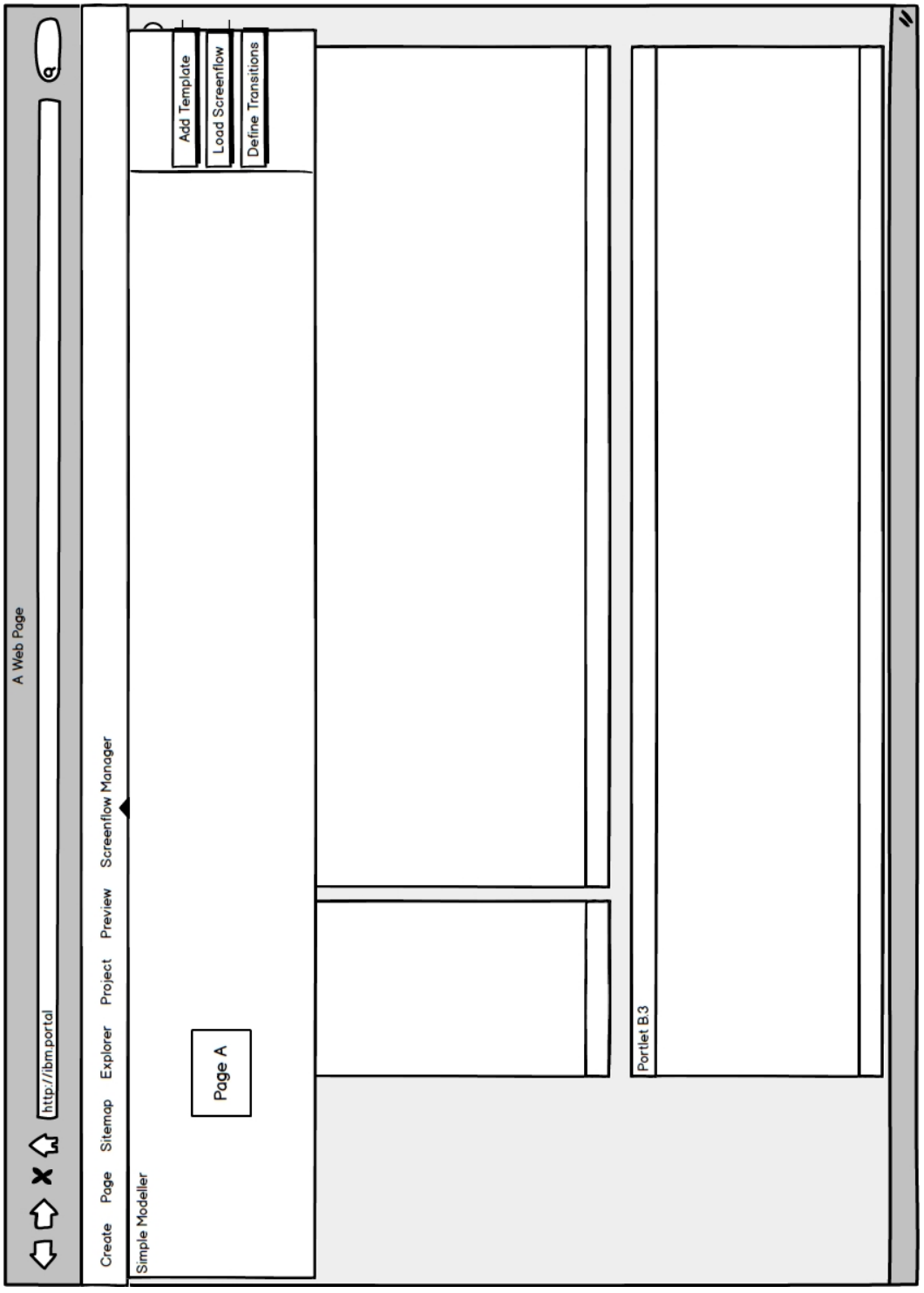

Abbildung A.10.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus auf der Portalseite B.

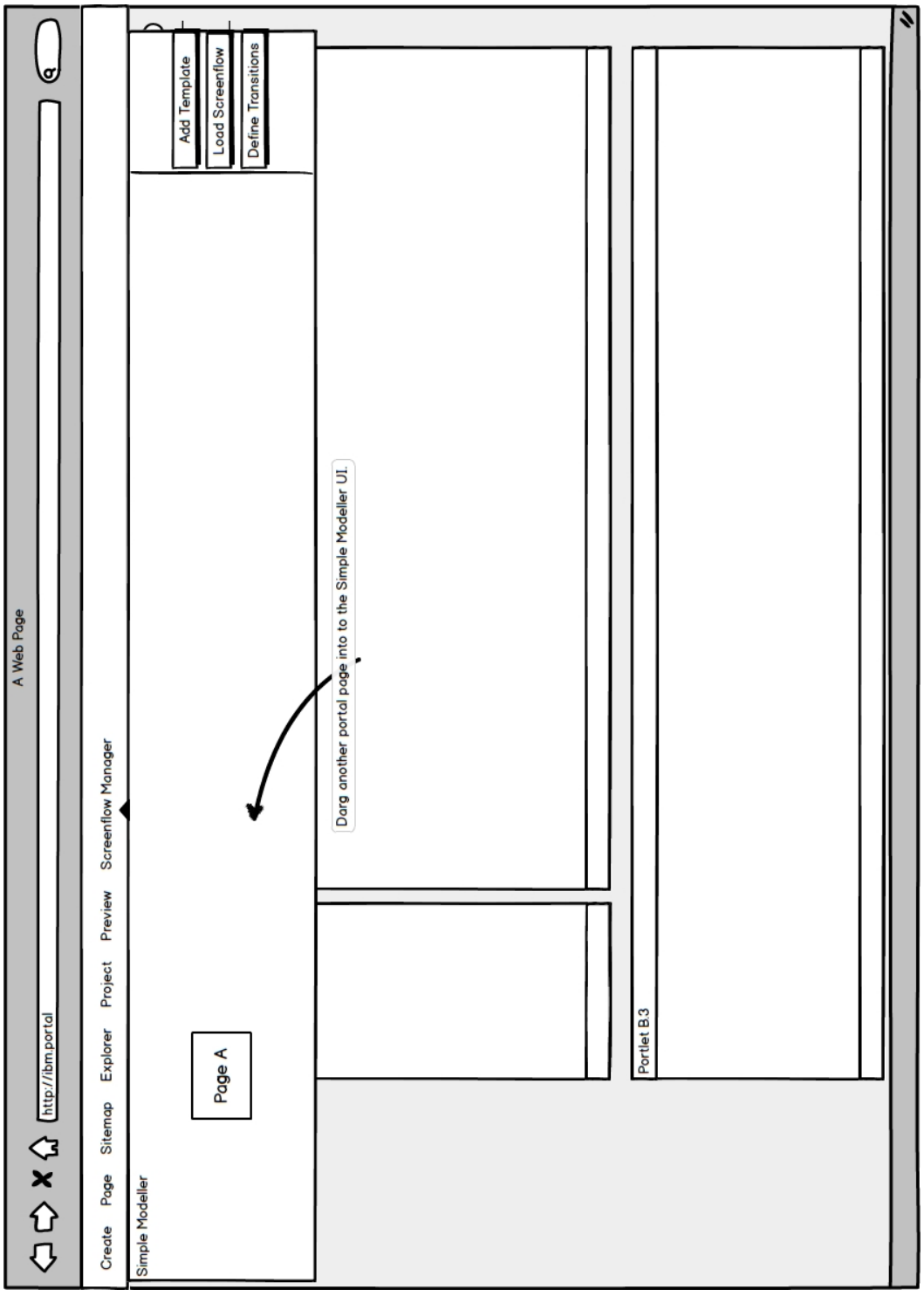

Abbildung A.11.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus und das Hinzufügen einer Portalseite (Portalseite B).

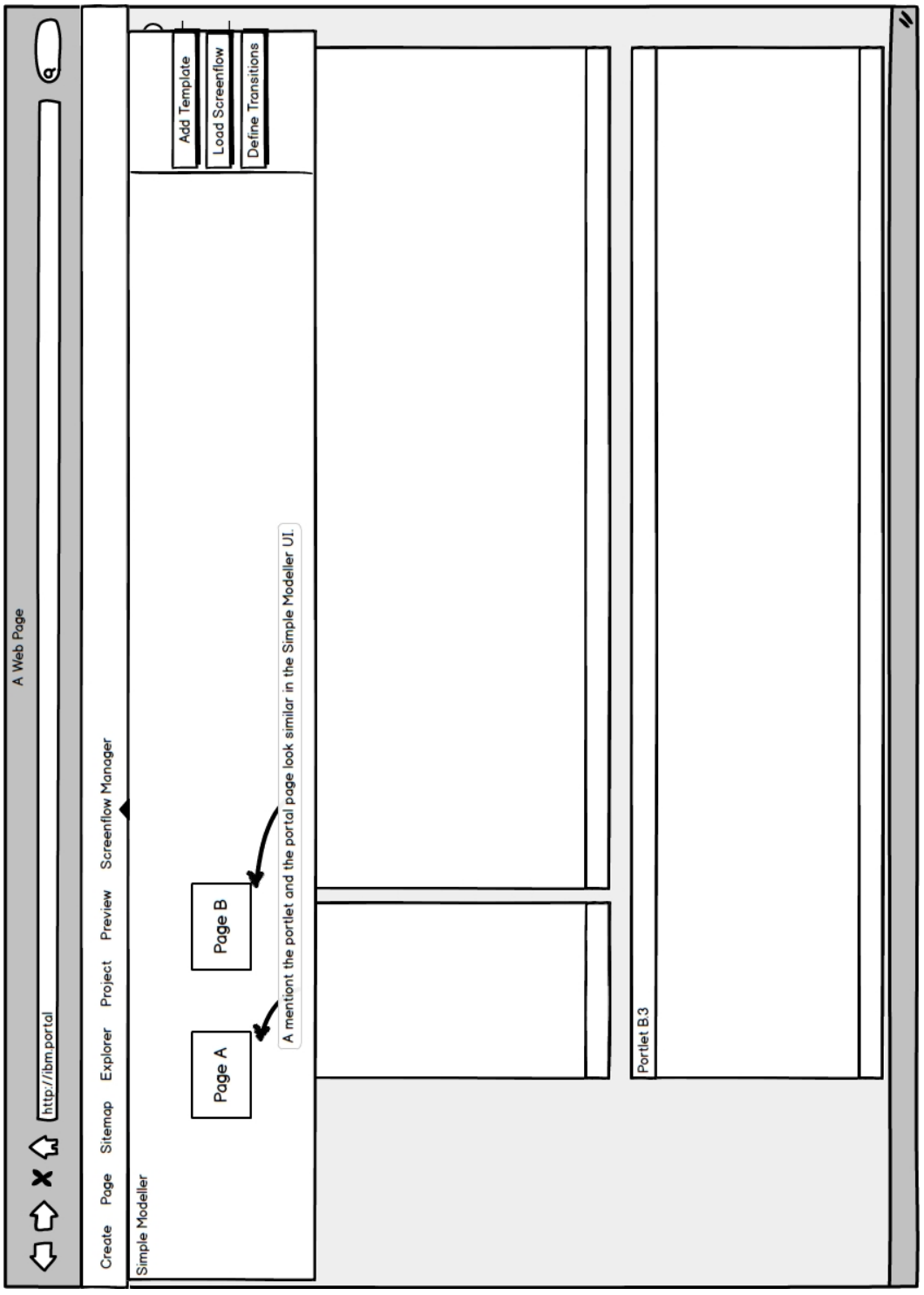

Abbildung A.12.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus, die Portalseite des Portlets A und die neu hinzugefügte Portalseite B.

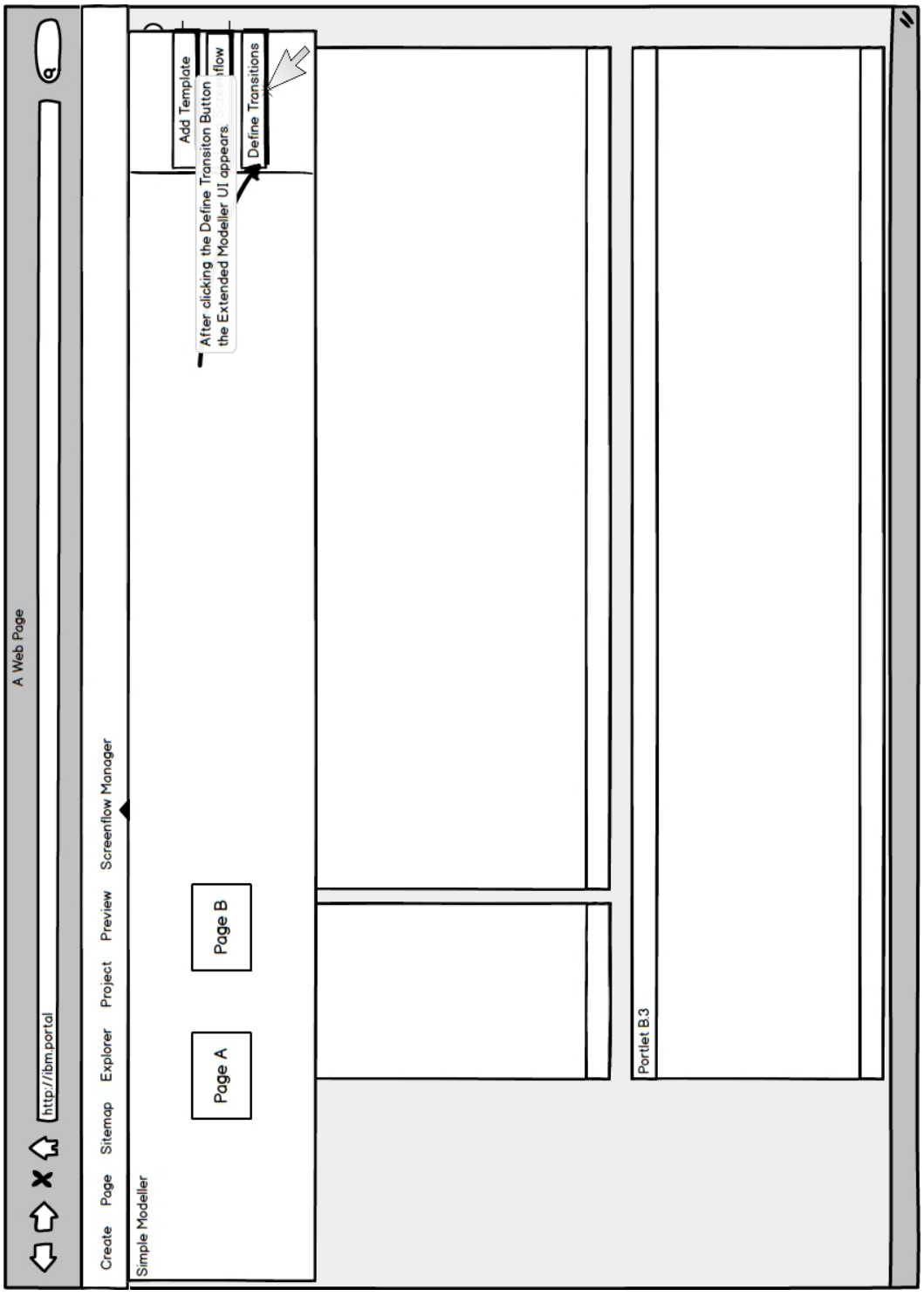

Abbildung A.13.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im einfachen Modus, vor dem Wechsel in den nächsten Schritt des Modellierungsprozess (*Define Transitions*).

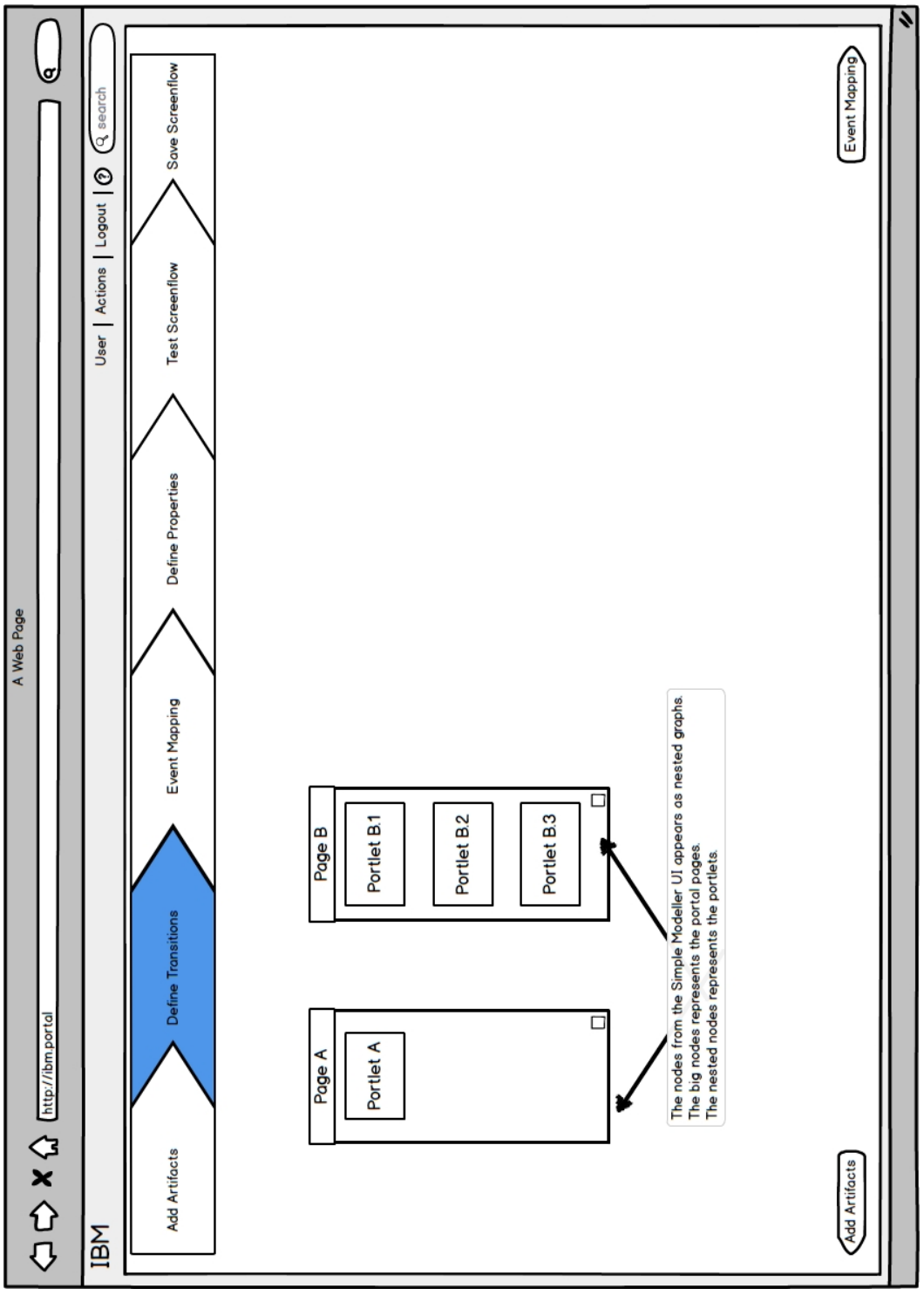

Abbildung A.14.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im erweiterten Modus, nach dem Wechsel in den Prozessschritt *Define Transitions*. Neben dem expandierten Graphen wird auch die Navigationsleiste angezeigt.

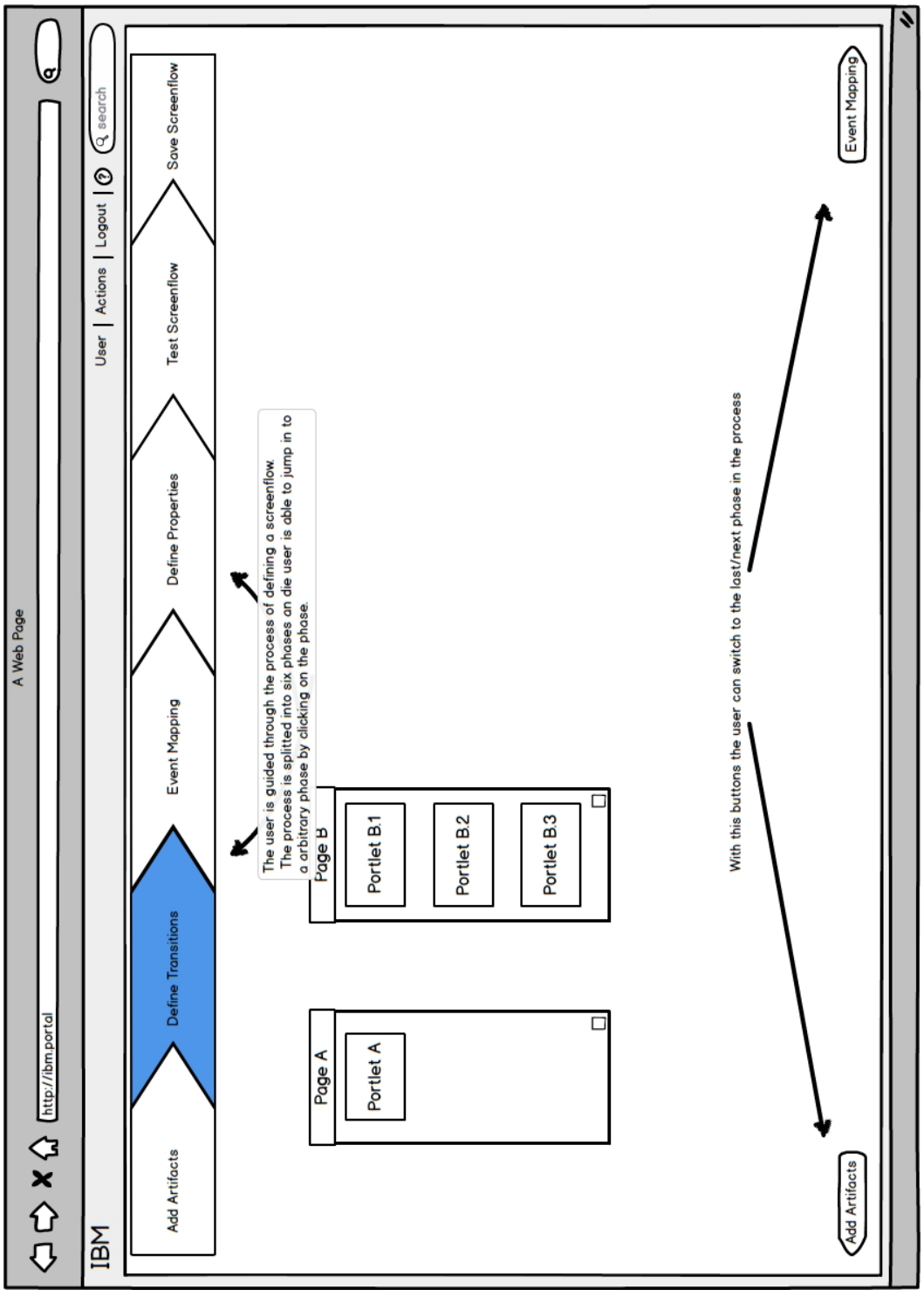

Abbildung A.15.: Dieses Mockup veranschaulicht das Modellierungswerkzeug in der Werkzeugleiste im erweiterten Modus, in dem Prozessschritt *Define Transitions*. Zusätzlich werden die Buttons für den Wechsel in den vorhergehenden oder folgenden Schritt des Modellierungsprozesses beschrieben.

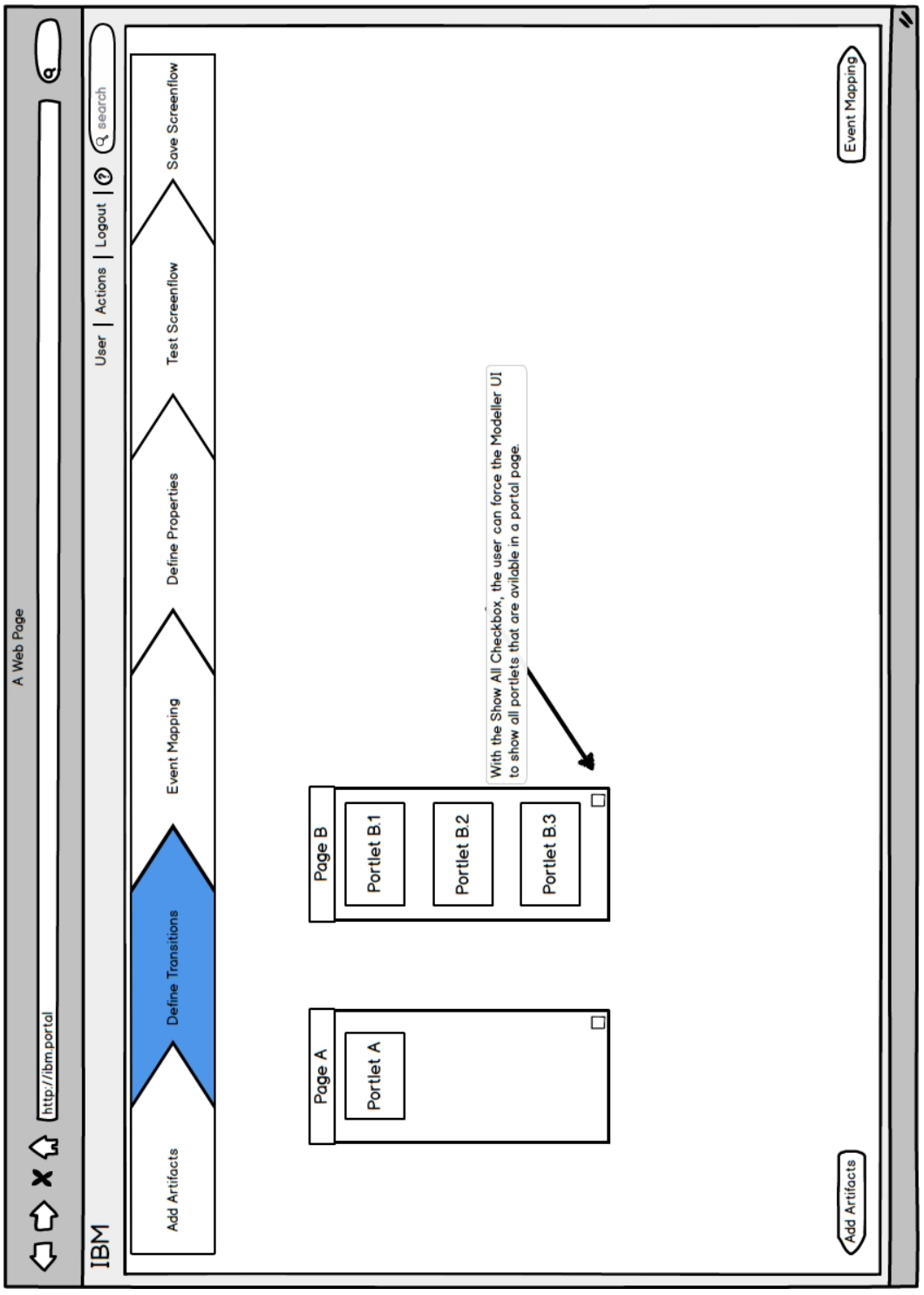

Abbildung A.16.: Dieses Mockup veranschaulicht und beschreibt den *Show All* Button. Mit diesem Button kann der Modellierer alle Portlets einer Portalseite einblenden (auch die Portlets, die zuvor nicht explizit hinzugefügt wurden).

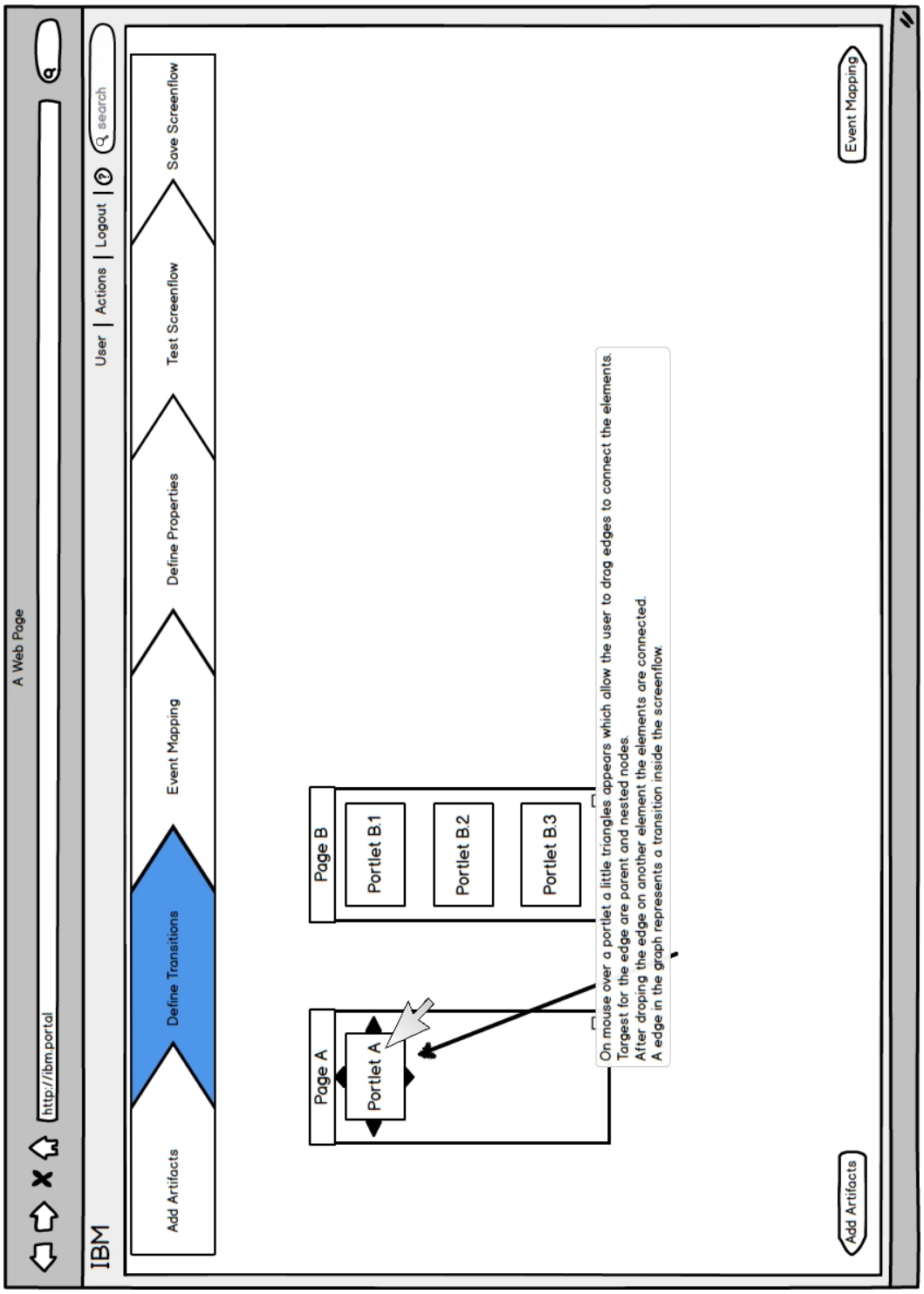

Abbildung A.17.: Dieses Mockup veranschaulicht das Werkzeug für das Verbinden von Knoten (erstellen einer Transition). Das Werkzeug wird durch berühren eines Knotens angezeigt. Dabei handelt es sich um vier Dreiecke, die um den entsprechenden Knoten dargestellt werden.

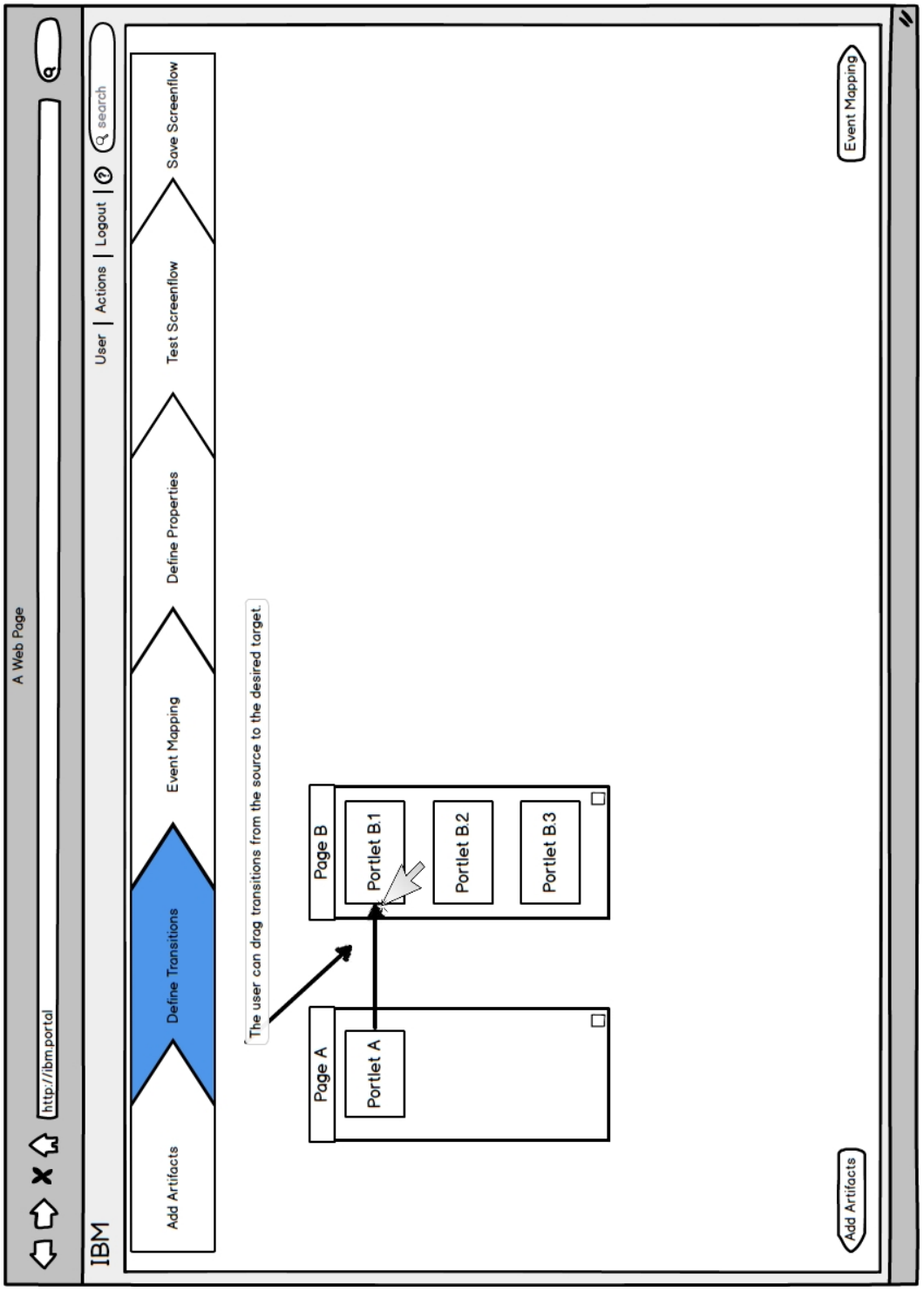

Abbildung A.18.: Dieses Mockup veranschaulicht den Graphen nach dem Verbinden der beiden Knoten.

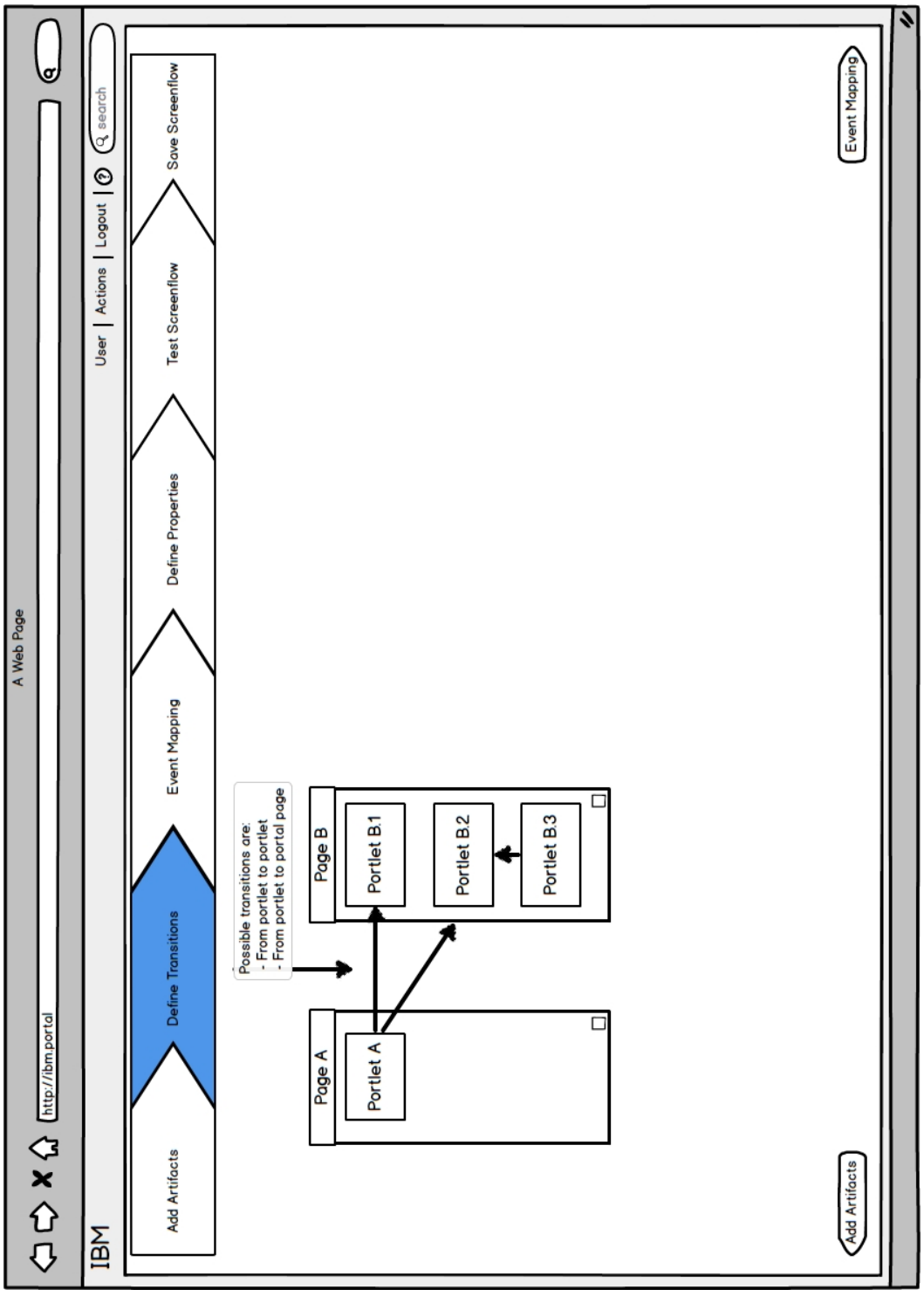

Abbildung A.19.: Dieses Mockup veranschaulicht den Graphen nach der Erstellung weiterer Transitionen. Dabei werden alle möglichen Arten von Transitionen dargestellt. Eine Transition zwischen Portlets auf derselben Portalseite, eine Transition zwischen Portlets auf unterschiedlichen Portalseiten und eine Transition zwischen einem Portlet und einer Portalseite (Broadcast).

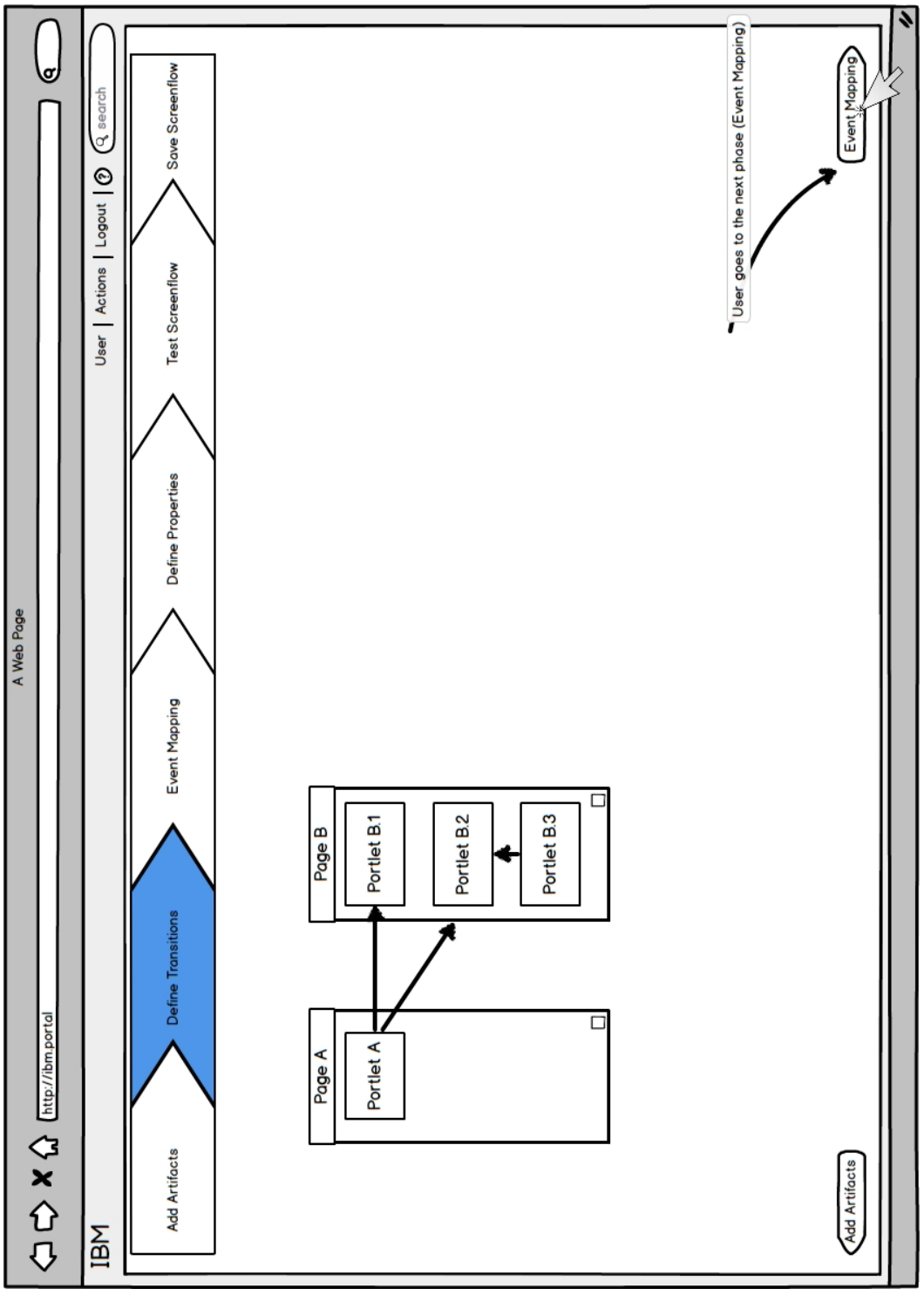

Abbildung A.20.: Dieses Mockup veranschaulicht das Modellierungswerkzeug vor dem Wechsel in den nächsten Schritt des Modellierungsprozess (*Event Mapping*).

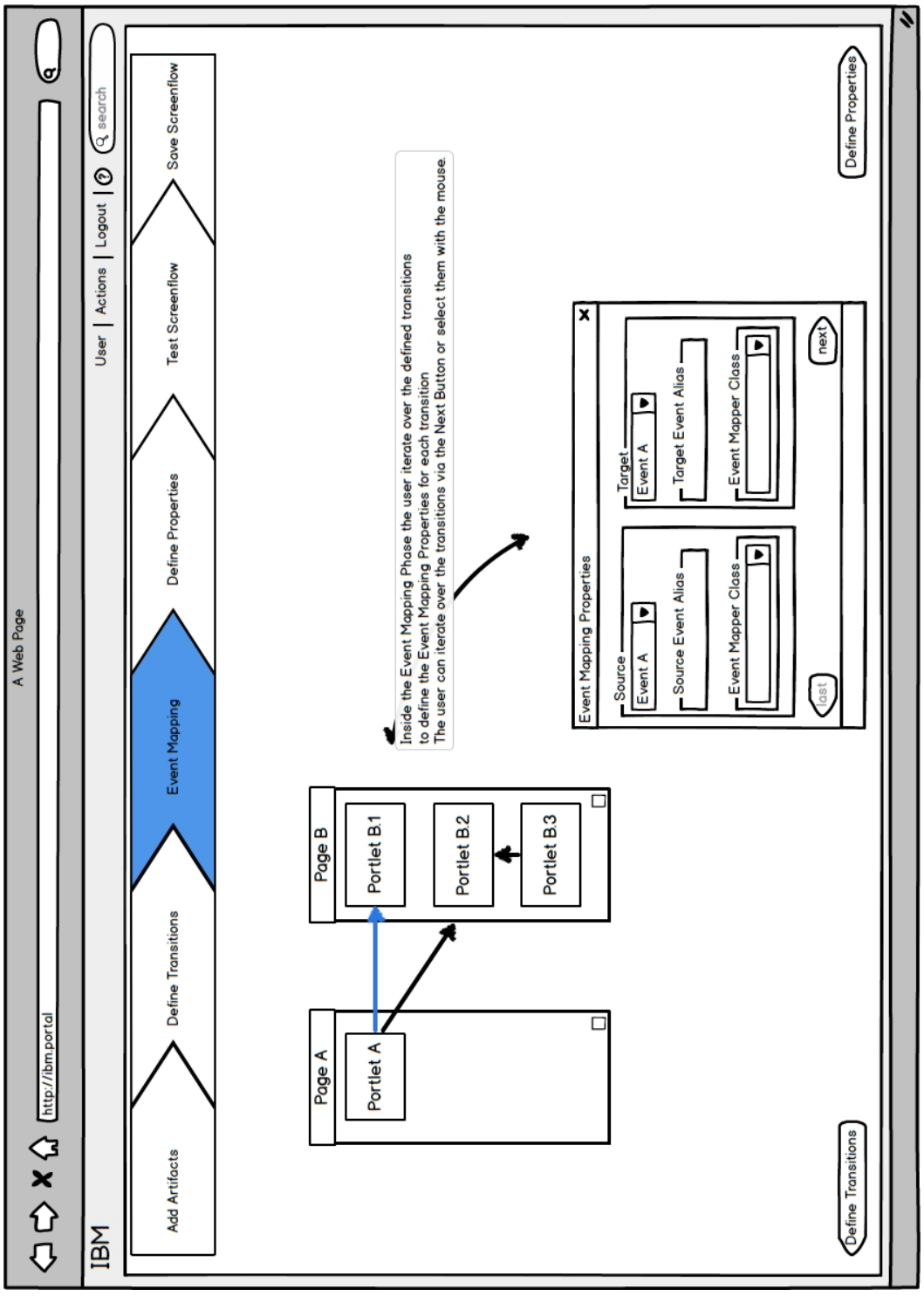

Abbildung A.21.: Dieses Mockup veranschaulicht das Modellierungswerkzeug nach dem Wechsel in den Prozessschritt *Event Mapping*. Zusätzlich zeigt es den Dialog für das Festlegen der Eigenschaften einer Transition. In dem Mockup ist die entsprechende Transition (blau) hervorgehoben.

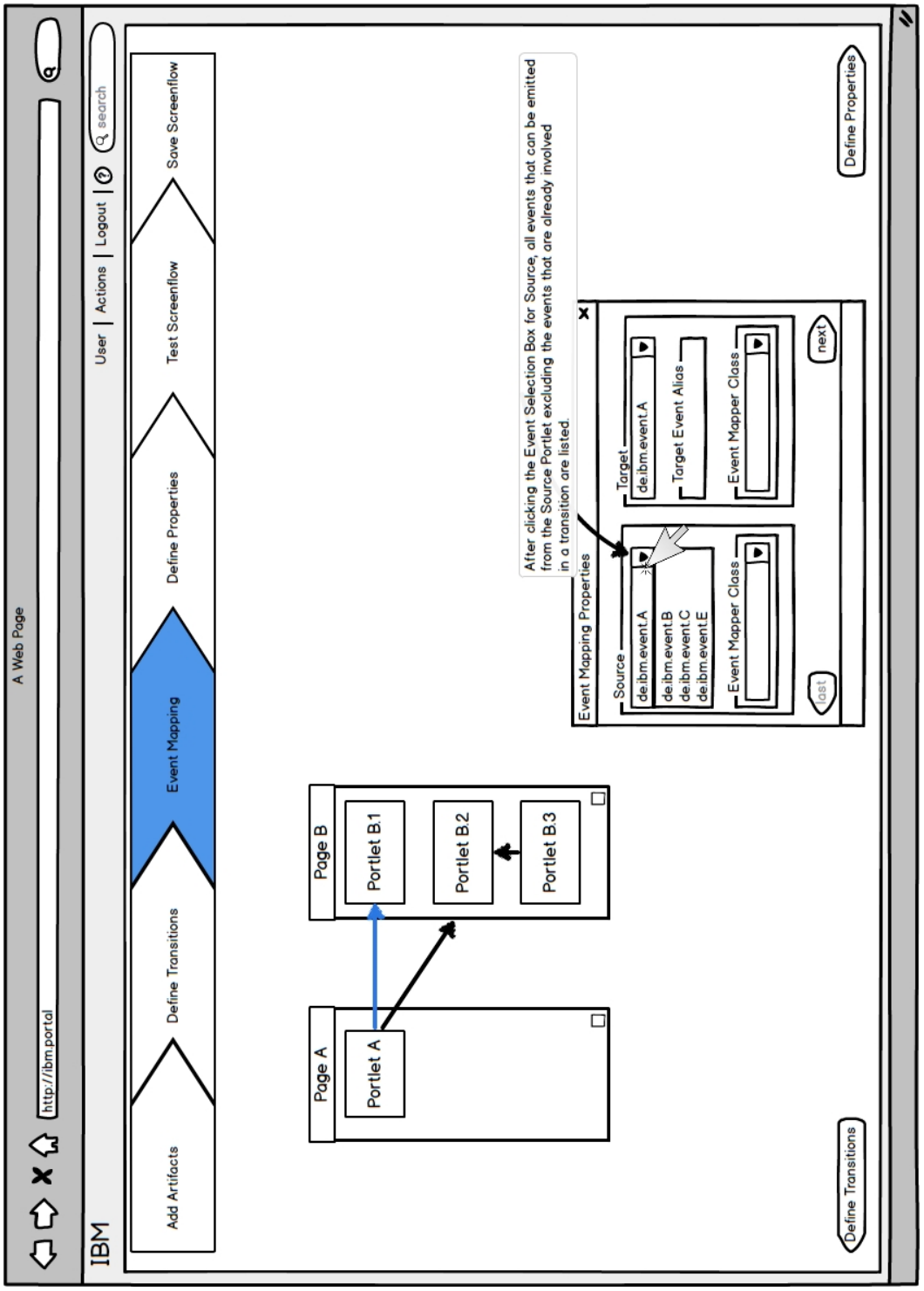

Abbildung A.22.: Dieses Mockup veranschaulicht das Zuweisen eines Ereignisses für den Quell Transition Endpoint der Transition.

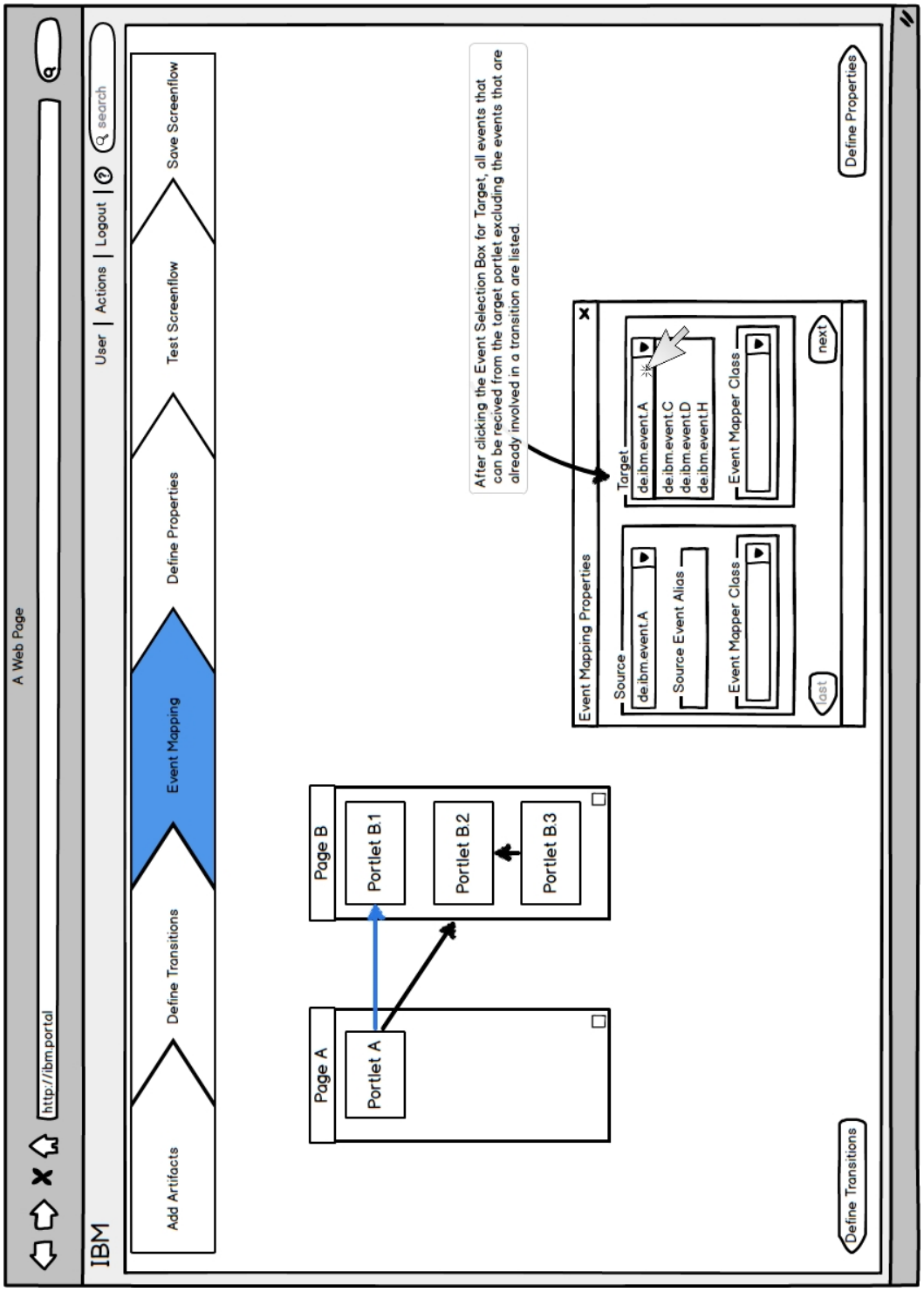

Abbildung A.23.: Dieses Mockup veranschaulicht das Zuweisen eines Ereignisses für den Ziel Transition Endpoint der Transition.

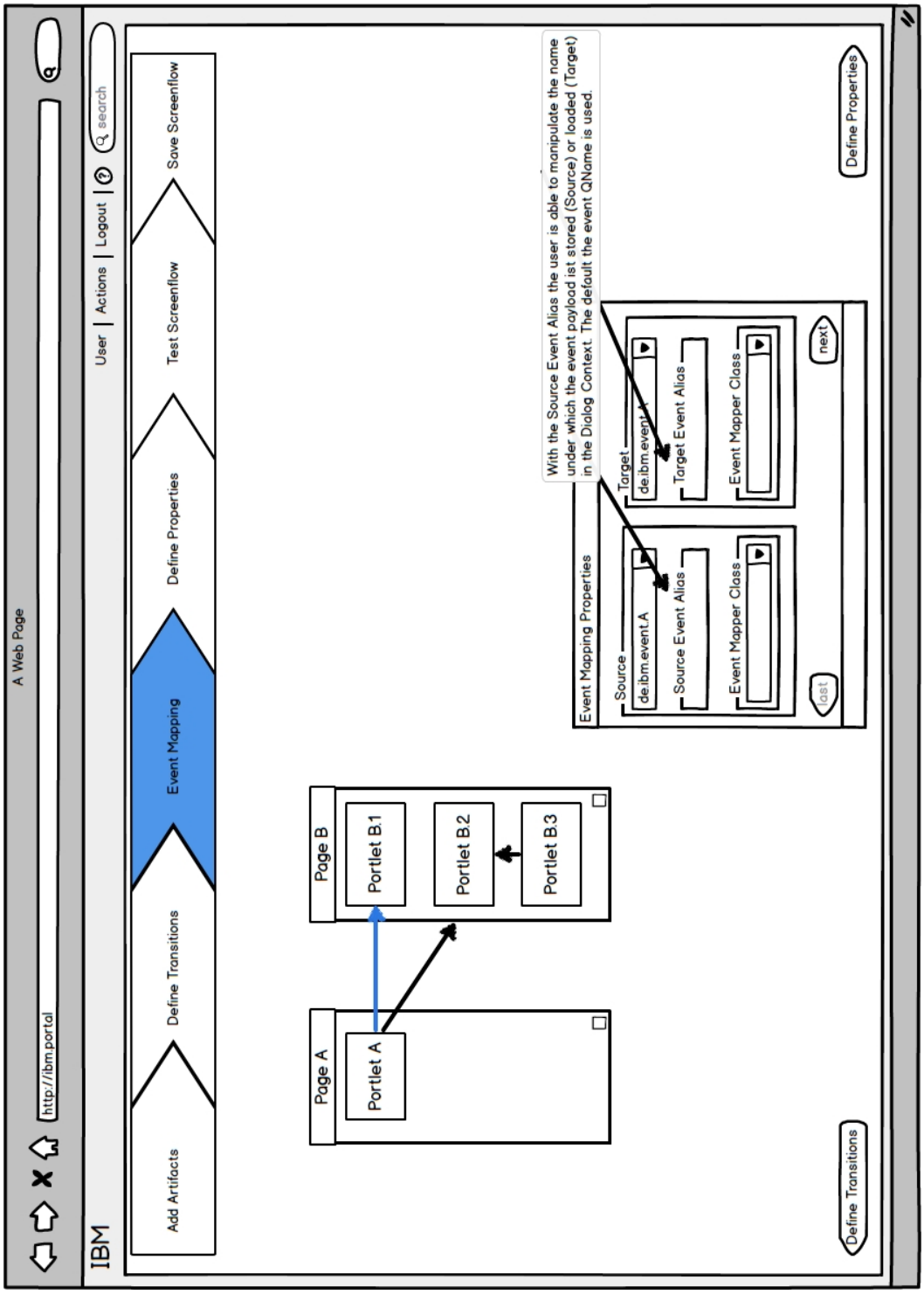

Abbildung A.24.: Dieses Mockup veranschaulicht die Möglichkeit einen Aliasnamen (DCX-Key) für das Quell- und Zielereignis zu wählen.

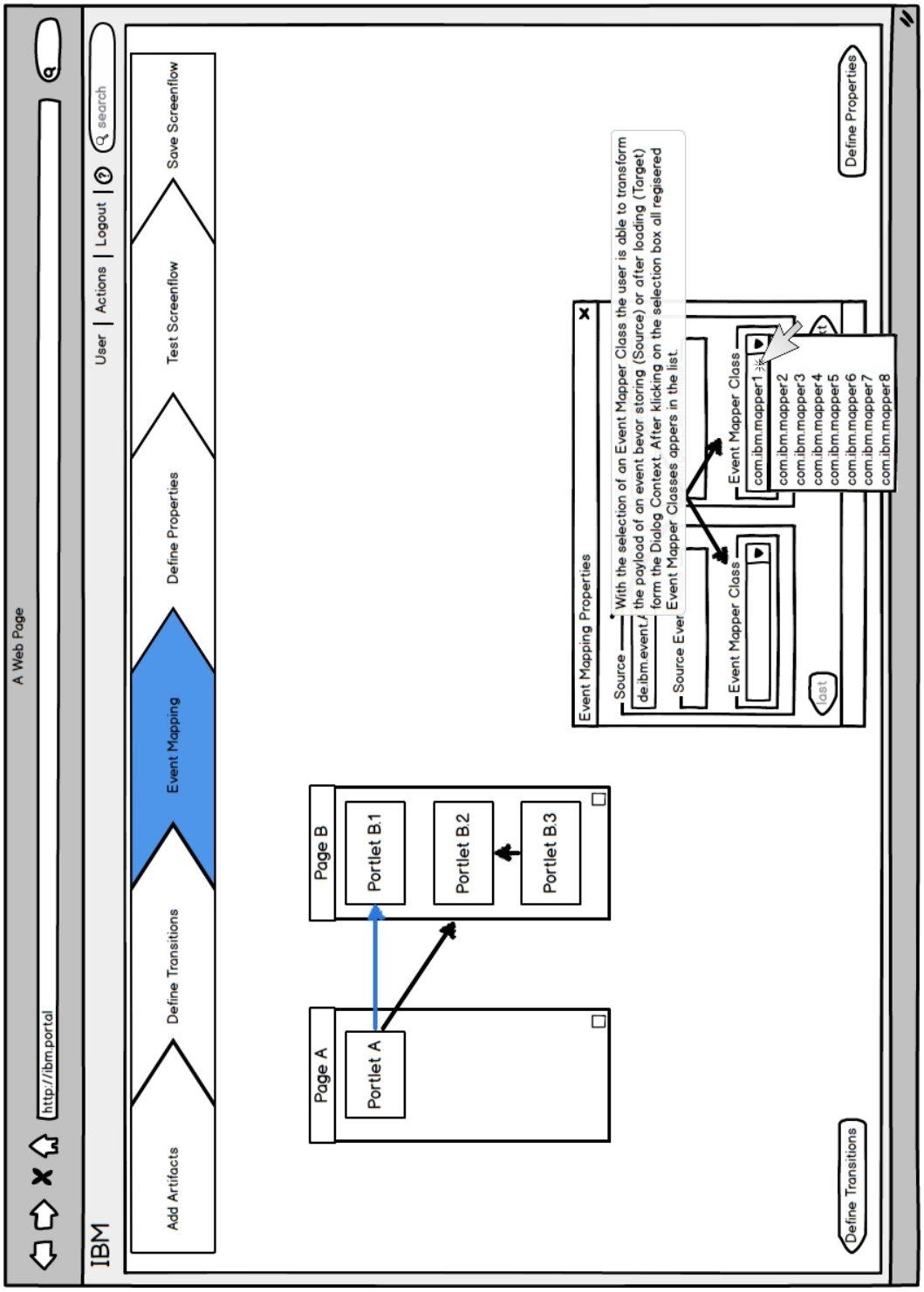

Abbildung A.25.: Dieses Mockup veranschaulicht die Auswahl einer Event Mapper Klasse (*ContentToPayloadMapper*) für das Zielereignis.

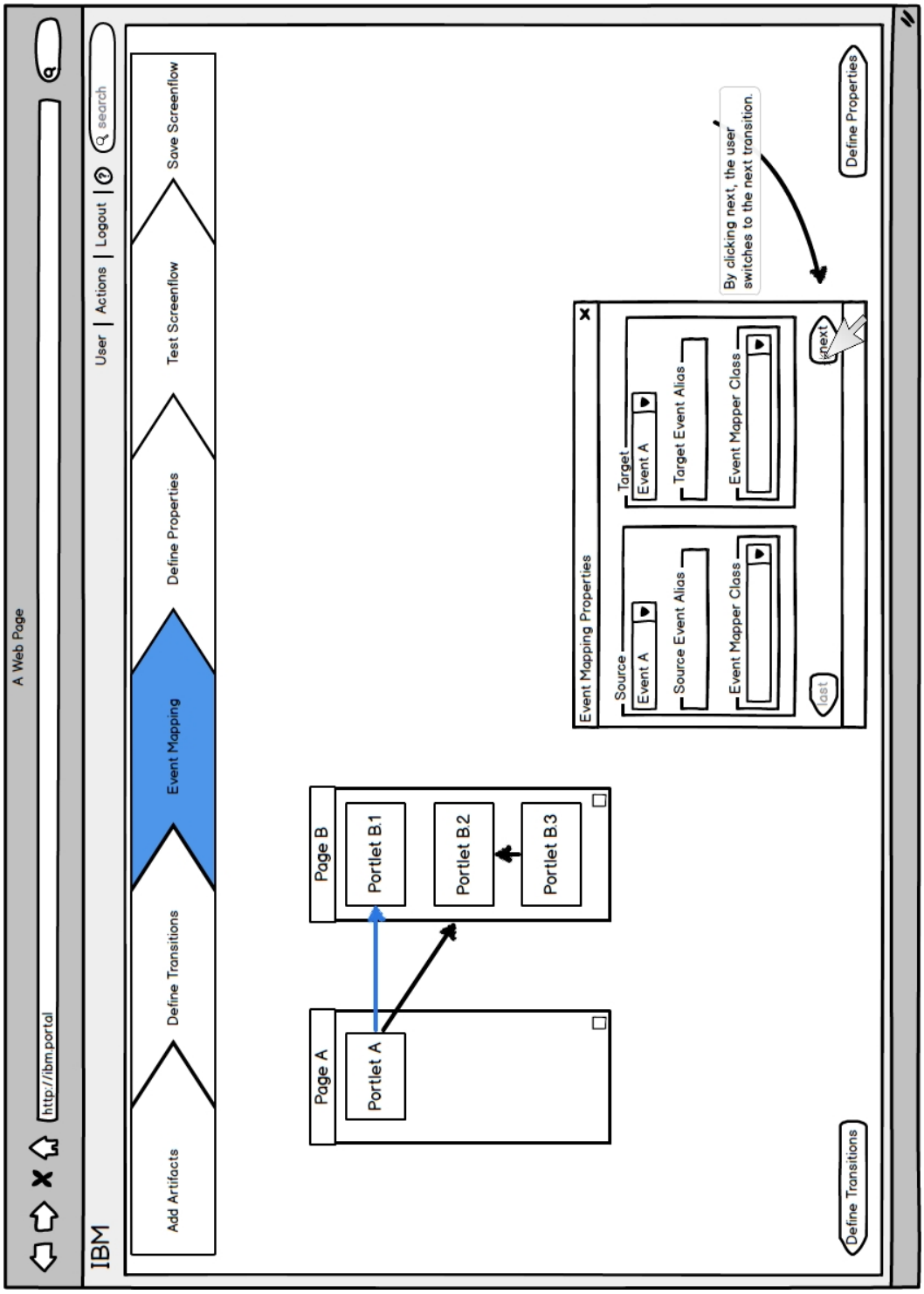

Abbildung A.26.: Dieses Mockup veranschaulicht das Wechseln zur nächsten Transition über den *next* Button.

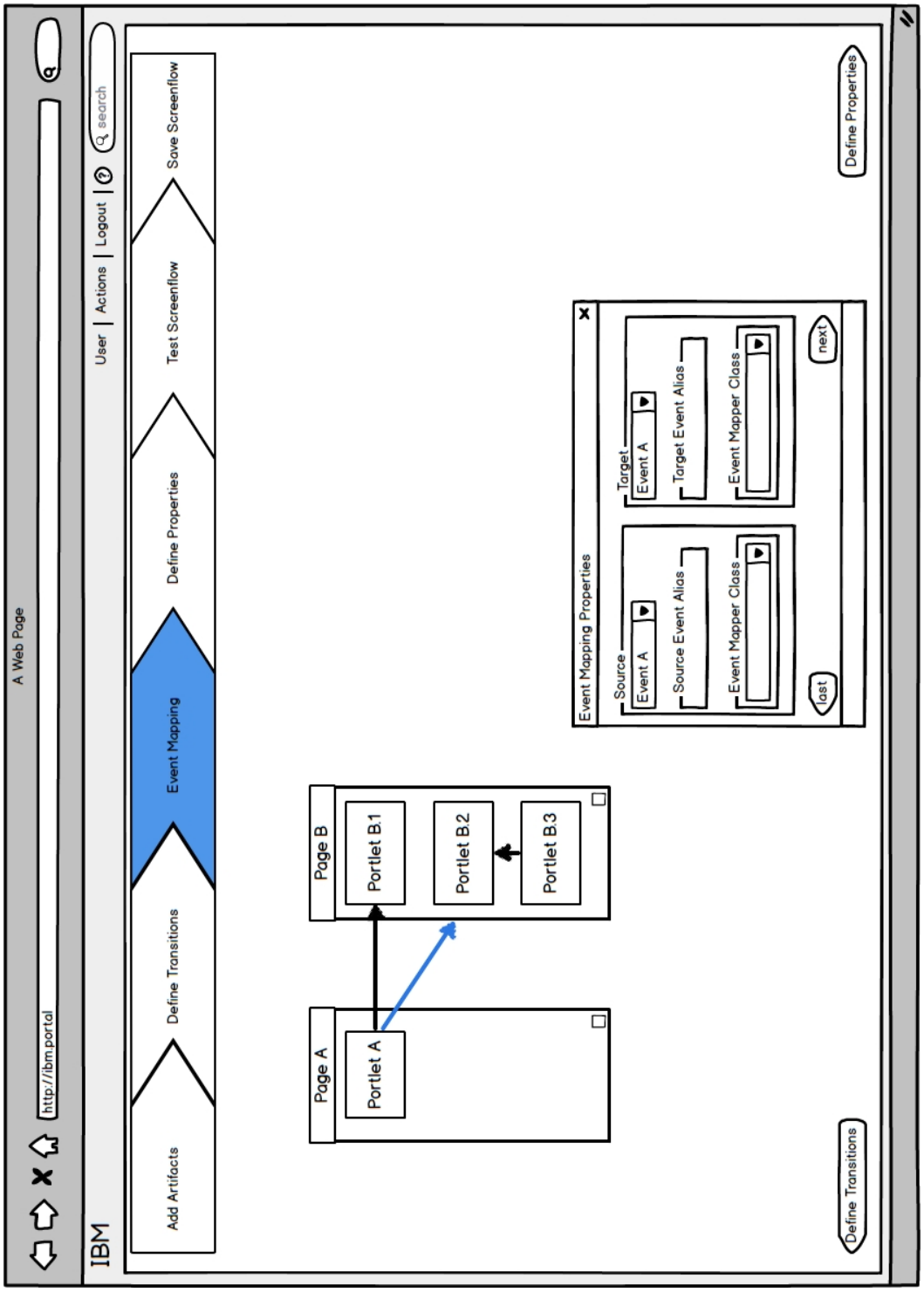

Abbildung A.27.: Dieses Mockup veranschaulicht den Dialog für das Festlegen der Eigenschaften für die nächste Transition.

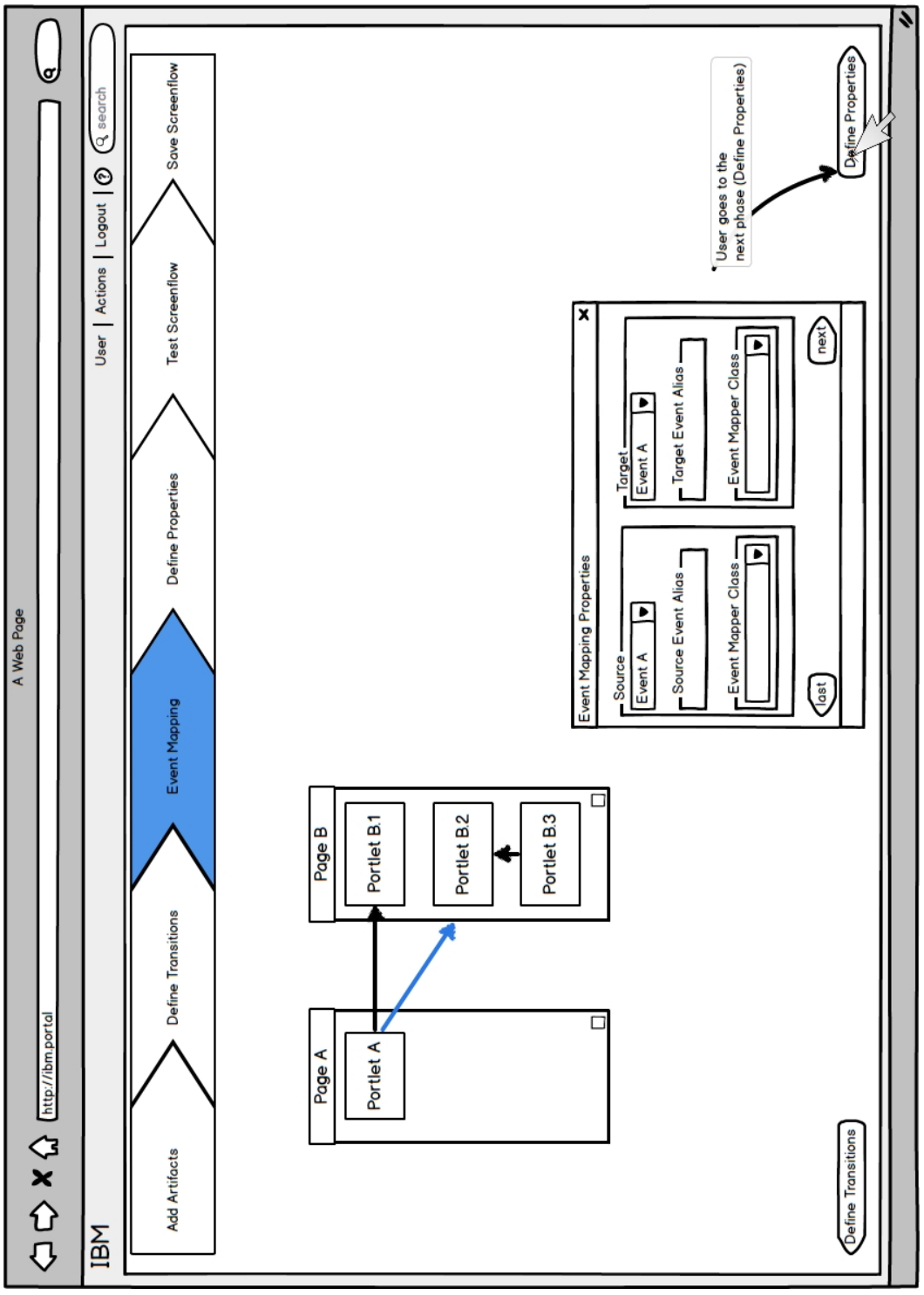

Abbildung A.28.: Dieses Mockup veranschaulicht das Modellierungswerkzeug vor dem Wechsel in den nächsten Schritt des Modellierungsprozesses (*Define Properties*).

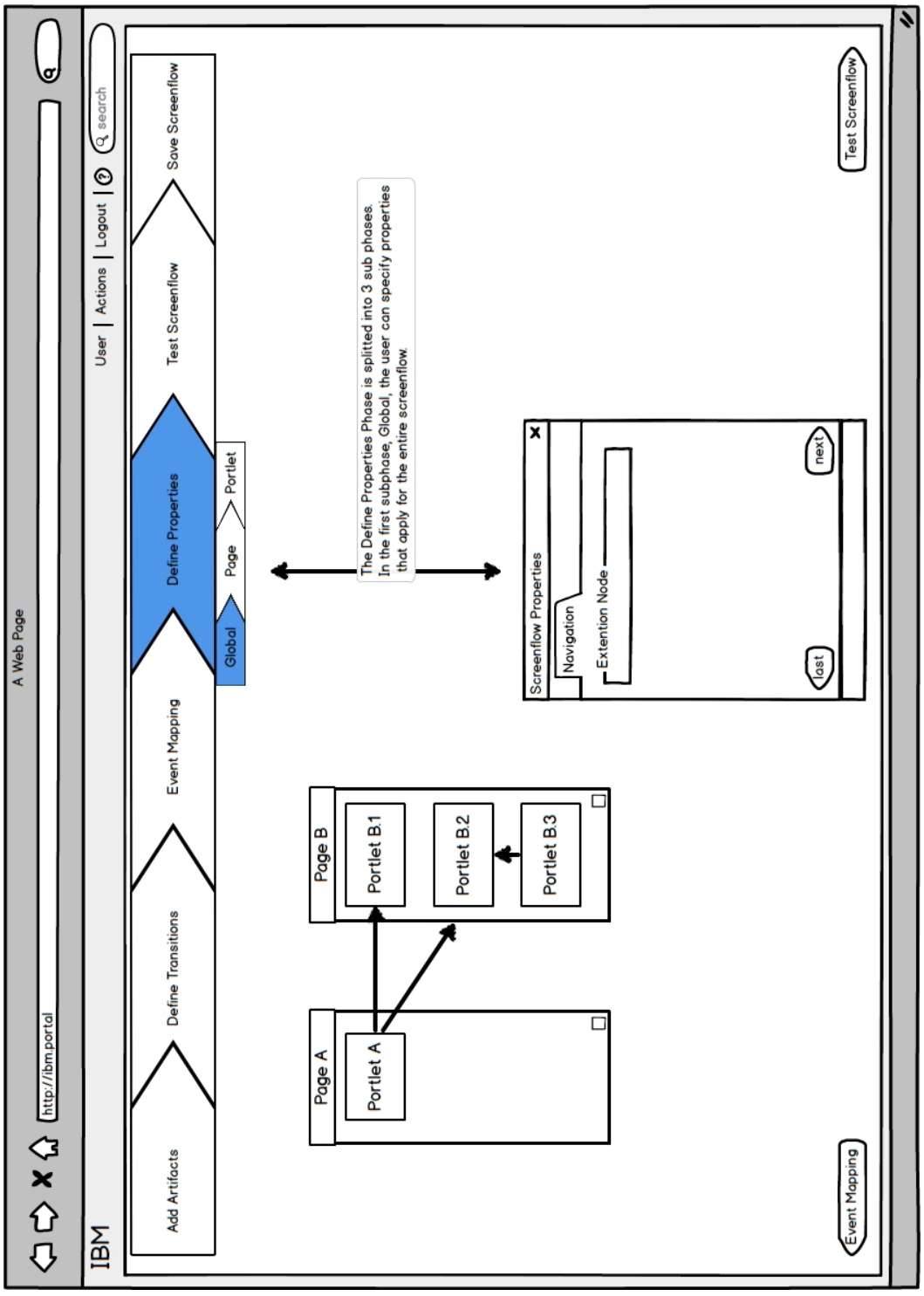

Abbildung A.29.: Dieses Mockup veranschaulicht das Modellierungswerkzeug nach dem Wechsel in den Prozessschritt *Define Proterties*. Zusätzlich zeigt das Mockup den Dialog für das Festlegen der Eigenschaften eines Screenflows. Hier kann der Modellierer einen Standard Extension Node vergeben.

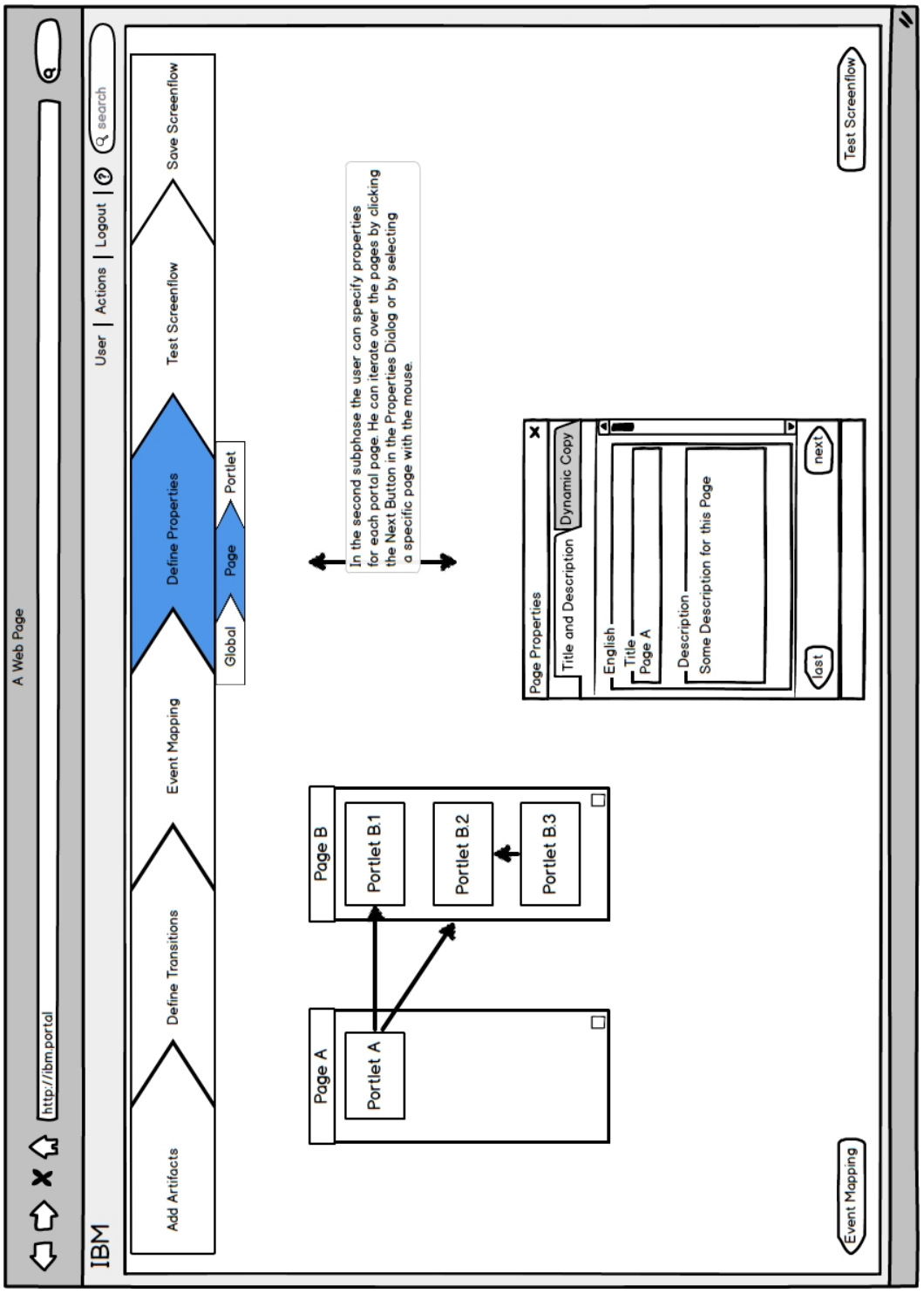

Abbildung A.30.: Dieses Mockup veranschaulicht den Dialog für das Festlegen der Eigenschaften einer Portalseite. Der Dialog besteht aus zwei Tabs. Im *Title and Description* Tab werden dem Modellierer Eingabefelder für einen Titel und eine Beschreibung für jede unterstütze Sprache untereinander aufgelistet. In diesem Mockup sind die Eingabefelder für die Sprache Englisch zu sehen.

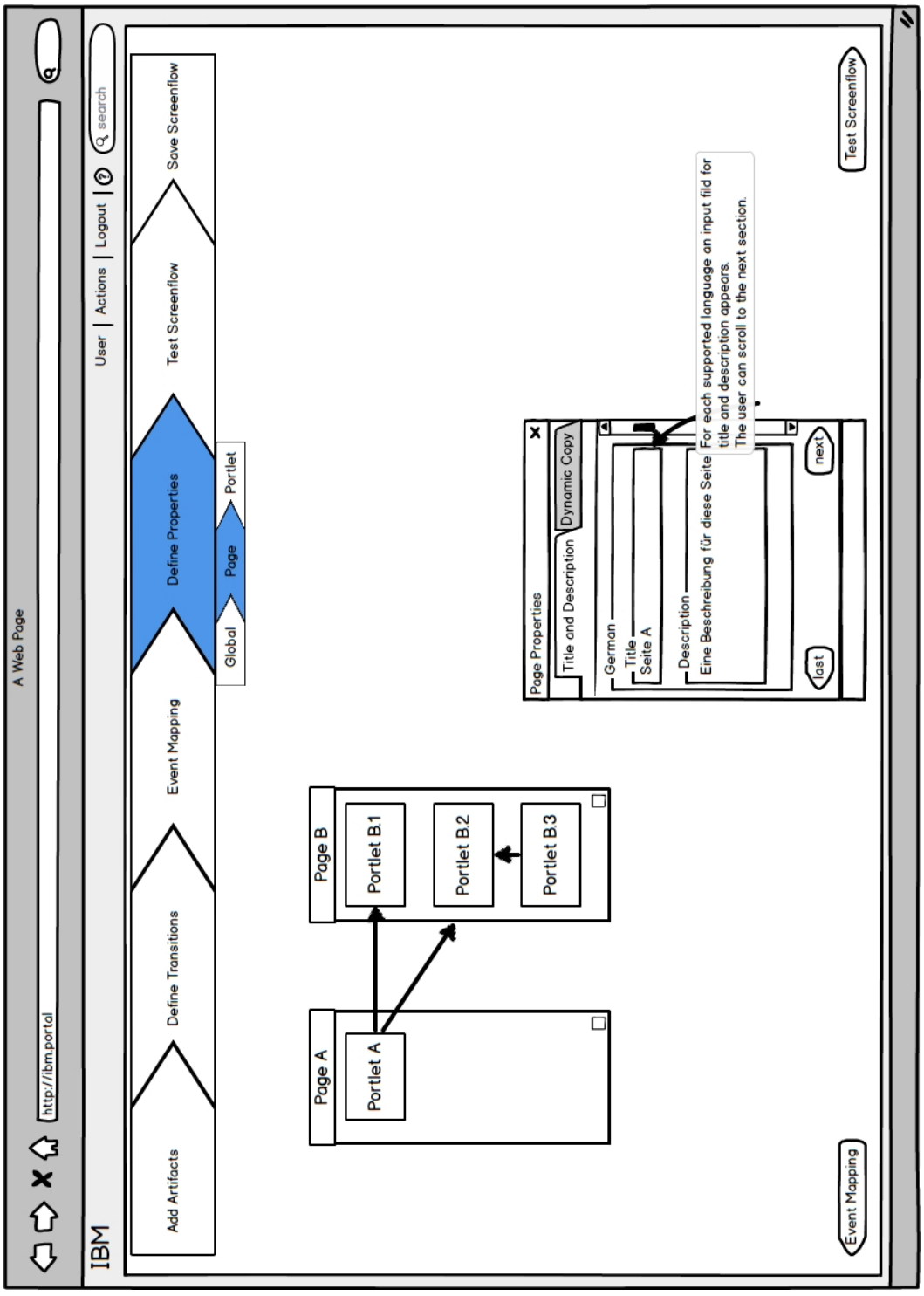

Abbildung A.31.: Dieses Mockup veranschaulicht den Dialog für das Festlegen der Eigenschaften einer Portalseite. In diesem Mockup sind die Eingabefelder für die Sprache Deutsch zu sehen.

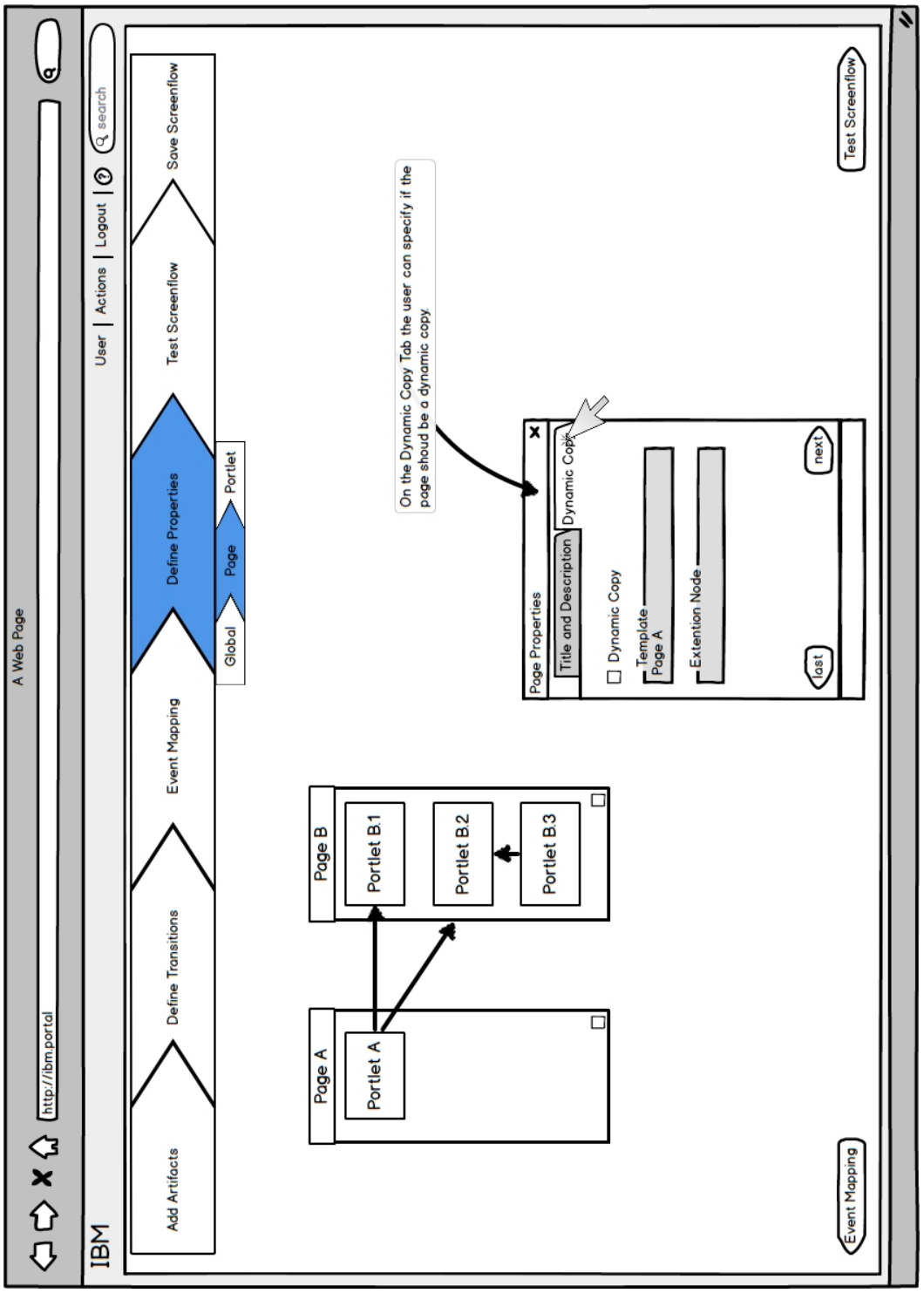

Abbildung A.32.: Dieses Mockup veranschaulicht den Dialog für das Festlegen der Eigenschaften einer Portalseite. Der Dialog besteht aus zwei Tabs. Im *Dynamic Copy* Tab kann der Modellierer wählen das die Portalseite dynamisch geladen werden soll.

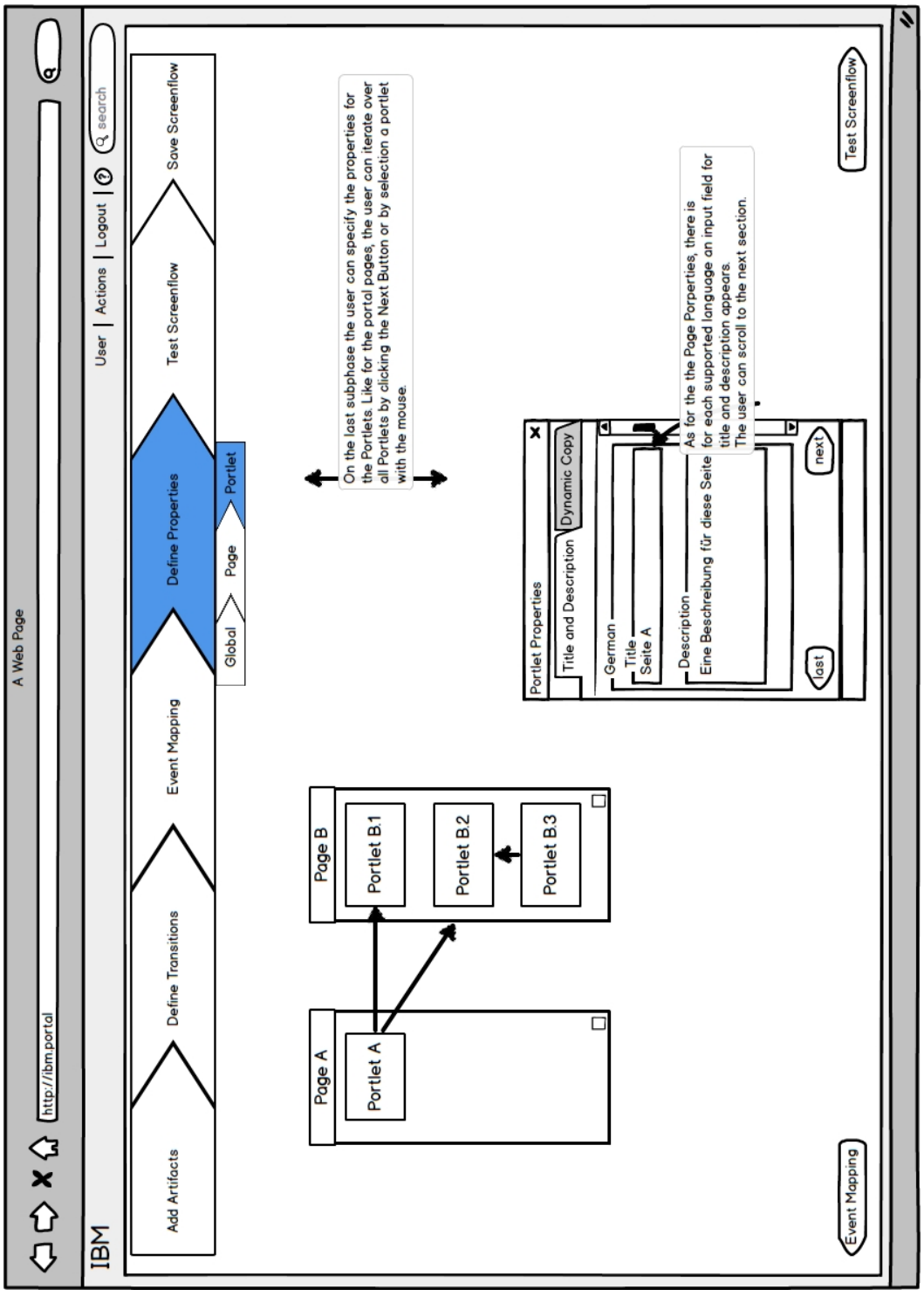

Abbildung A.33.: Dieses Mockup veranschaulicht den Dialog für das Festlegen der Eigenschaften eines Portlets. Der Dialog ist vom Aufbau identisch zum Dialog für das Festlegen der Eigenschaften einer Portalseite. Auf diesem Mockup wird das *Title and Description* Tab dargestellt.

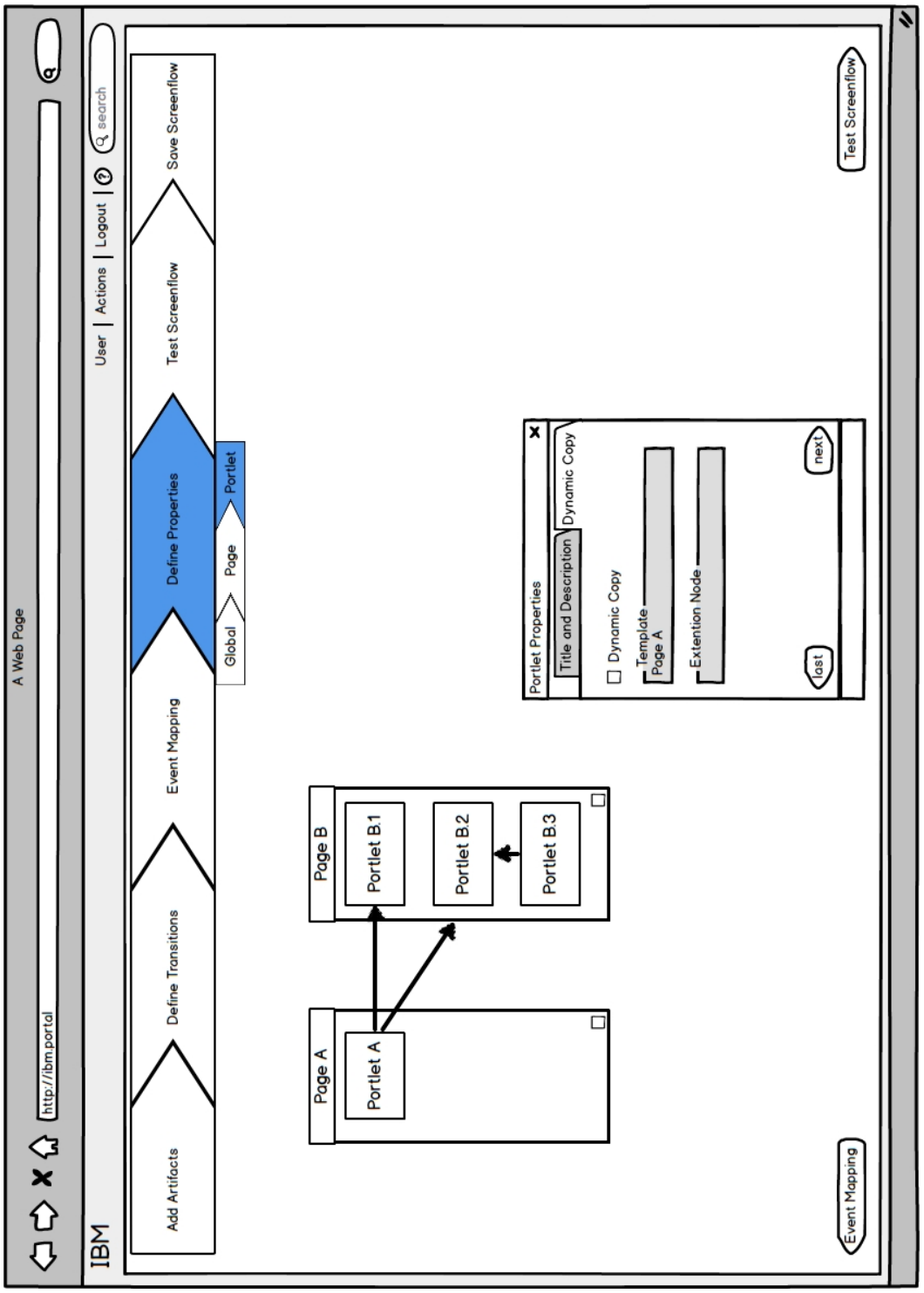

Abbildung A.34.: Dieses Mockup veranschaulicht den Dialog für das Festlegen der Eigenschaften eines Portlets. Auf diesem Mockup wird das *Dynamic Copy* Tab dargestellt.

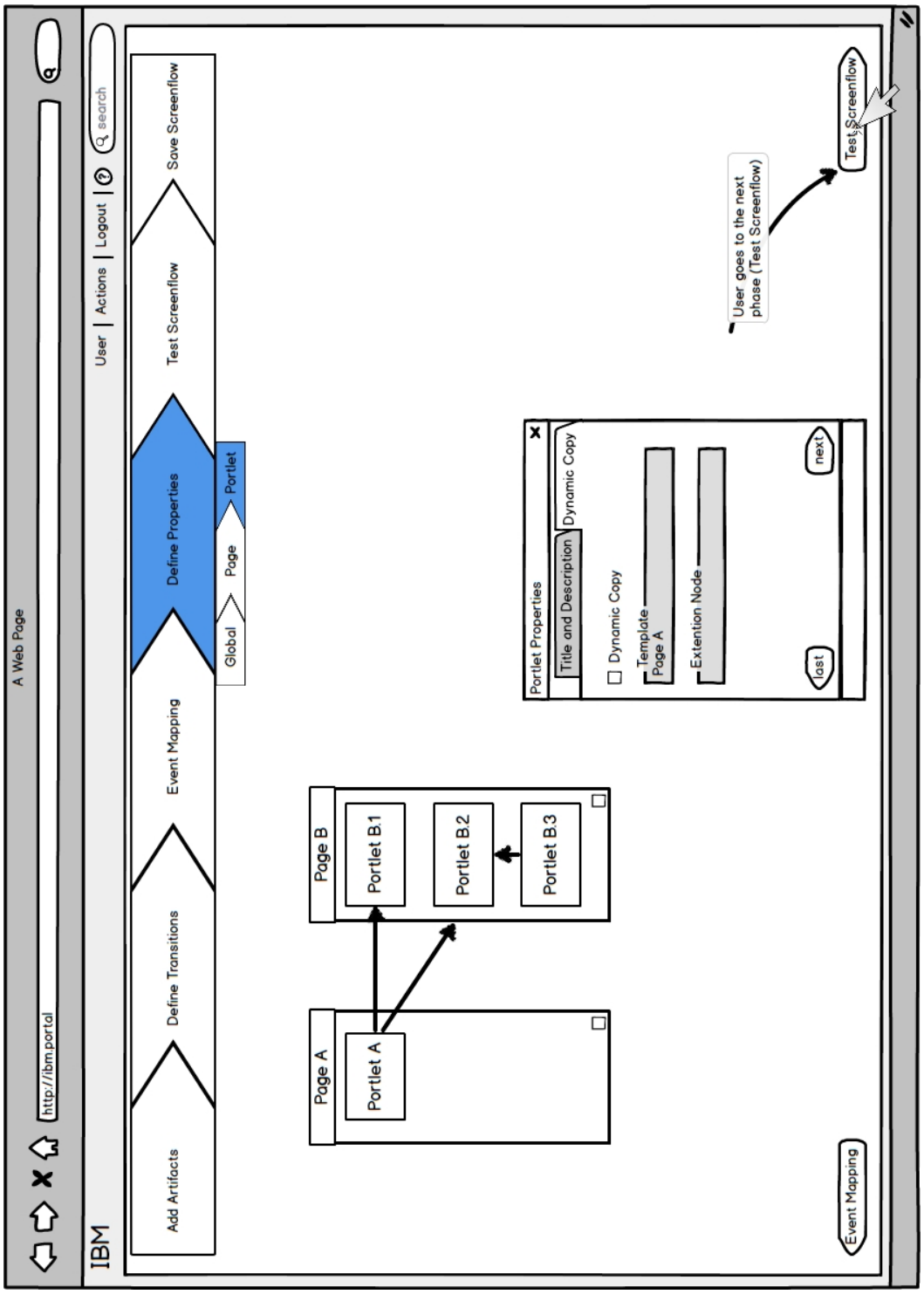

Abbildung A.35.: Dieses Mockup veranschaulicht das Modellierungswerkzeug vor dem Wechsel in den nächsten Schritt des Modellierungsprozesses (*Define Properties*).

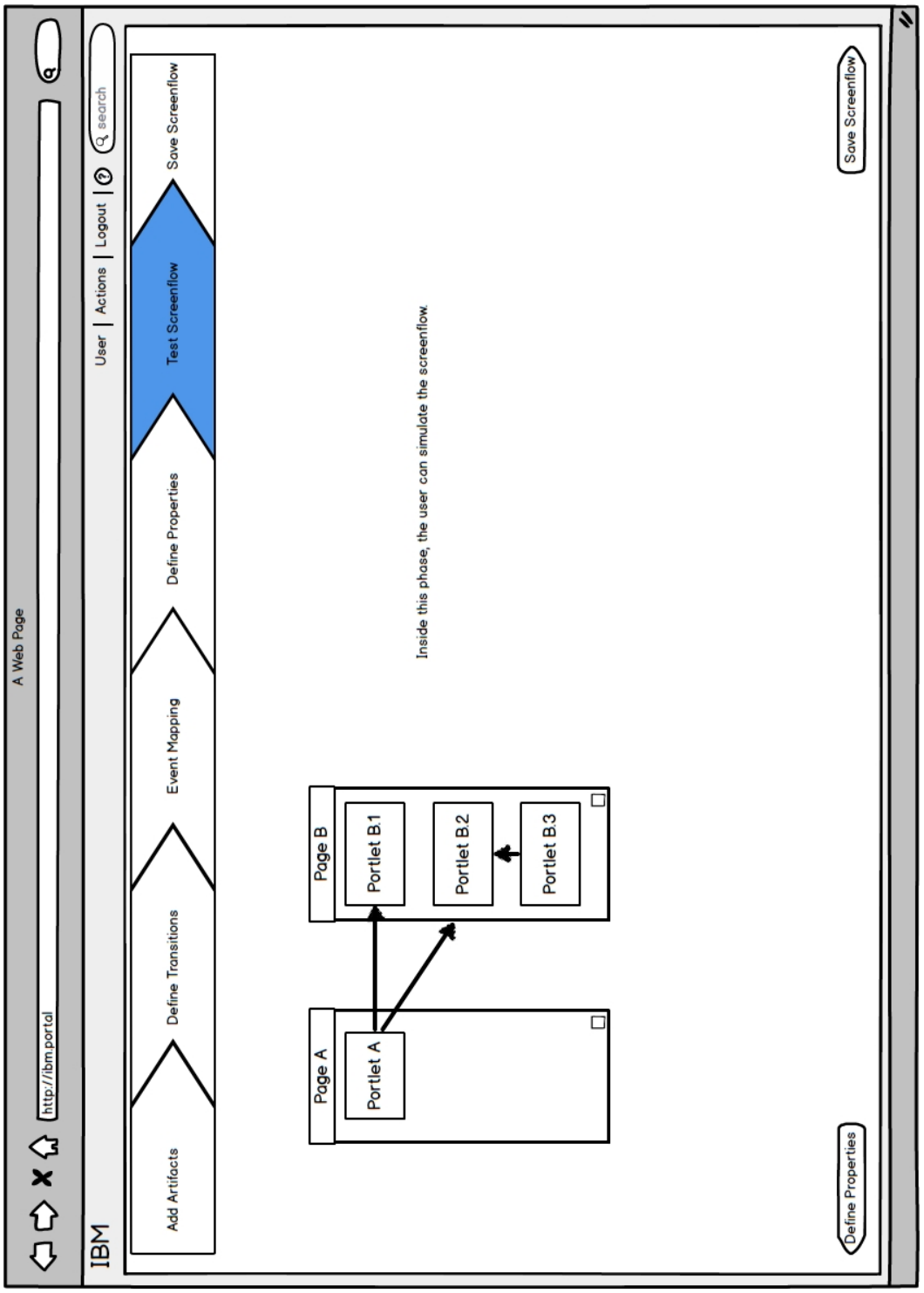

Abbildung A.36.: Dieses Mockup veranschaulicht das Modellierungswerkzeug nach dem Wechsel in den Prozessschritt *Test Screenflow*. Das Testen von Screenflows ist nicht Teil dieser Arbeit.

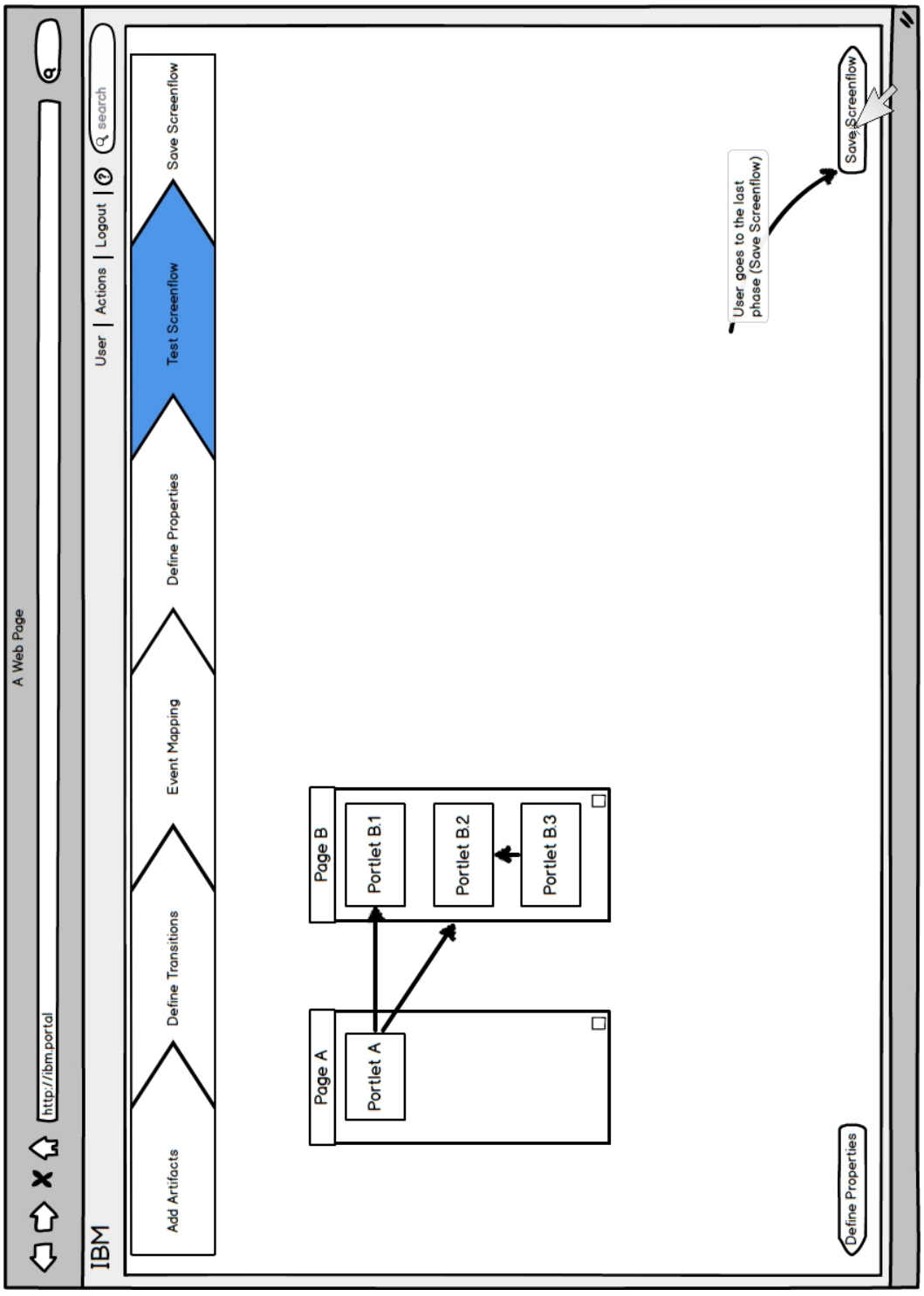

Abbildung A.37.: Dieses Mockup veranschaulicht das Modellierungswerkzeug vor dem Wechsel in den nächsten Schritt des Modellierungsprozesses (*Save Screenflow*).

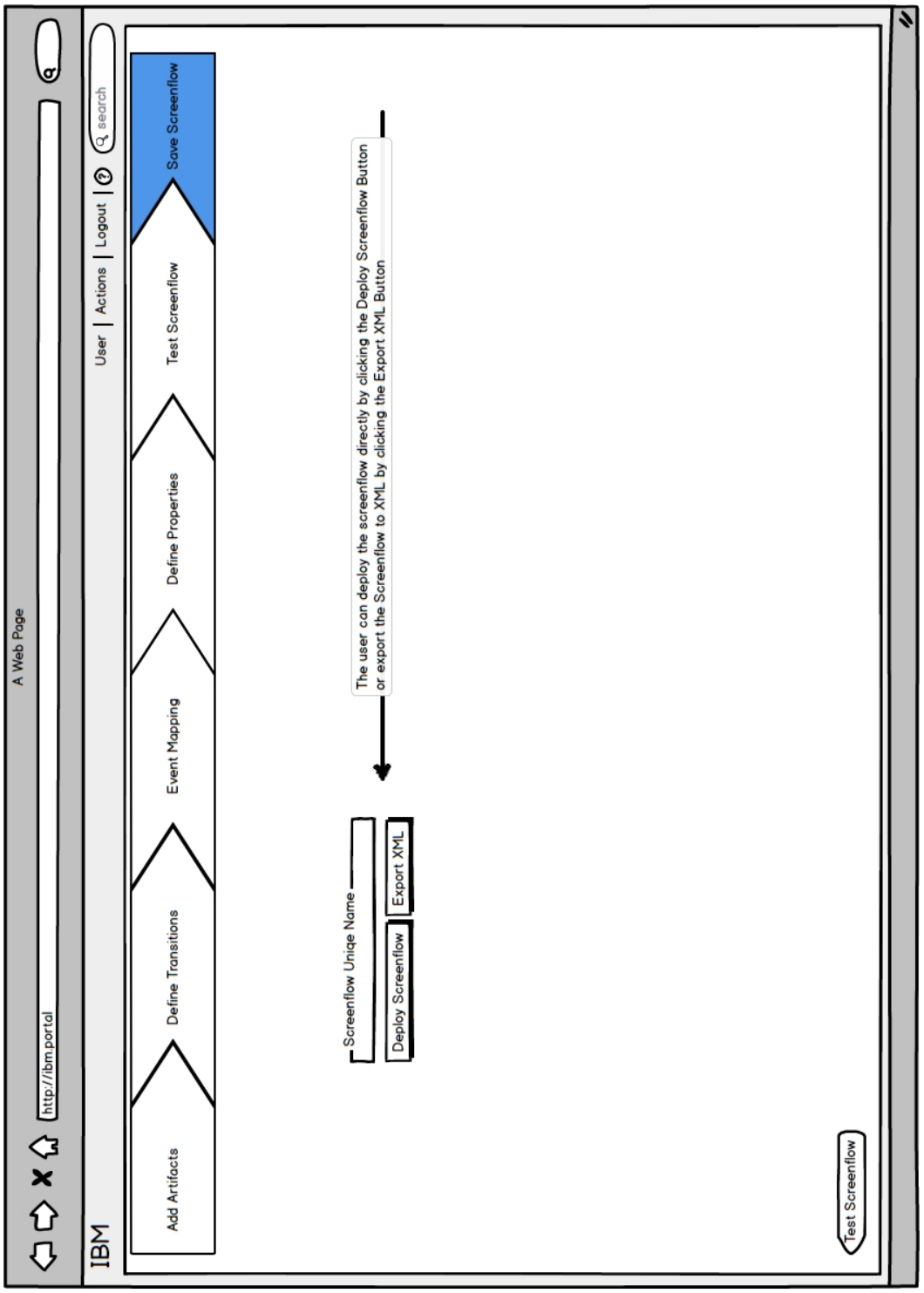

Abbildung A.38.: Dieses Mockup veranschaulicht das Modellierungswerkzeug nach dem Wechsel in den Prozessschritt *Save Screenflow*. Zusätzlich werden ein Eingabefeld angezeigt wo der Modellierer einen Namen für den Screenflow vergeben kann, sowie ein *Deploy* und ein *Save* Button.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Literaturverzeichnis

[AAA+07] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, A. E. Mark Ford, Y. Goland, A. Guízar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu. Web Services Business Process Execution Language, 2007.

[AAD+07a] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller. Web Services Human Task (WS-HumanTask), 2007.

[AAD+07b] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller. WS-BPEL Extension for People (BPEL4People), 2007.

[ABD+07] N. Ayachitula, M. Buco, Y. Diao, S. Maheswaran, R. Pavuluri, L. Shwartz, C. Ward. IT service management automation – A hybrid methodology to integrate and orchestrate collaborative human centric and automation centric workflows. 2007.

[Anto8] Anthony T. Holdener. *Ajax: The Definitive Guide*. O'Reilly, 2008. ISBN: 978-0-59-652838-6.

[Apa12] Apache Software Foundation. *Apache Cocoon - Control Flow*, 2012. URL http://cocoon.apache.org/2.1/userdocs/flow/.

[ASH12] J. Aubourg, J. Song, Hallvord R. M. Steen. XMLHttpRequest, 2012. URL http://www.w3.org/TR/XMLHttpRequest.

[Ban05] T. Banks. *Web Services Resource Framework (WSRF)*. Organization for the Advancement of Structured Information Standards, 1 Auflage, 2005. URL http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf.

[BG04] M. Book, V. Gruhn. Modeling Web-Based Dialog Flows for Automatic Dialog Control. In *19th IEEE International Conference on Automated Software Engineering (ASE 2004)* [Kto0], S. 100–109.

[Com06] R. W. Community. *RIFE Users Guide - Creating a more advanced RIFE application*, 2006. URL http://rifers.org/wiki/display/RIFE/GuideNumberguess.html.

[DL13] Dr. Andreas Nauerz, S. Liesche. *IBM UX Screen Flow Manager Documentation*. IBM WebSphere Portal, 2013. [Internes Dokument].

[Dr.12] Dr. Andreas Nauerz. *Adapting and Recommending Content and Expertise in Highly Collaborative Web Portals*. Dr. Hut, 2012. ISBN:978-3843905756.

[DS] L. DeMichiel, B. Shannon. Java Platform, Enterprise Edition (Java EE) Specification.

[DVG+] K. Donald, E. Vervaet, J. Grelle, S. Andrews, R. Stoyanchev. *Spring Web Flow Reference Guide*. Spring, 2.0.9 Auflage. URL http://static.springsource.org/spring-webflow/docs/2.0.x/reference/html/index.html.

[GHJV96] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Entwurfsmuster*, Band 5. Addison-Wesley, 1996. ISBN 3-8273-1862-9.

[GK] M. Großmann, H. Koschek. *Unternehmensportale*. Springer. ISBN: 978-1-84-628519-6.

[GSK+11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. *Guide to e-Science*, Kapitel Conventional Workflow Technology for Scientific Simulation. Springer, 2011. ISBN: 978-0-85729-438-8.

[Hee07] S. Heesen. *Cocoon - XML-basierte Webentwicklung Schritt für Schritt*. Open Source Press, München, 2007. ISBN: 978-3-93-751455-0.

[Hei02] M. Hein. *TCP/IP*, Band 6. mitp-Verlag, 2002. ISBN: 3-8266-4094-2.

[Hep08] S. Hepper. Java Portlet Specification, 2008.

[IBM] IBM. Dynamic UI Management. URL http://publib.boulder.ibm.com/infocenter/wpdoc/v6r1/topic/com.ibm.wp.ent.doc_v6101/dev/wpsdynui_cpts.html.

[IEDM07] Ian J. Taylor, Ewa Deelmann, Dennis B. Gannon, Matthew Shields. *Workflows for e-Science*. Springer, 2007. ISBN:978-1-84-628519-6.

[IUH+05] I. Foster Argonne, U. Chicago, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich. *The Open Grid Services Architecture*, 1 Auflage, 2005. URL http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.

[JRJ02] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexität*, Band 2. Addison-Wesley Longman Verlag, 2002. ISBN: 978-3827370204.

[KDS+12] D. Karastoyanova, D. Dentsas, D. Schumm, M. Sonntag, L. Sun, K. Vukojevic. Service-based Integration of Human Users in Workflow-driven Scientific Experiments. 2012.

[Kto00] N. Kassem, the Enterprise Team. Designing Enterprise Applications with the JavaTM 2 Platform, Enterprise Edition. 2000.

[Kus05] T. Kussmaul. DIe Java-Portlet-Spezifikation. *JavaSPEKTRUM*, 3, 2005.

[Loc12] A. Lochbihler. *Semantik von Programmiersprachen*. Lehrstuhl für Programmierparadigmen, Karlsruher Institut für Technologie, 2012. ISBN: 978-3-54-015163-0.

[LR00] F. Laymann, D. Roller. *Production Workflow Concepts and Techniques*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 2000. ISBN: 978-0-13-021753-0.

[Mat08] Mathew A. Russell. *Dojo - The Definitive Guide*. O'Reilly, 1 Auflage, 2008. ISBN: 978-0-596-51648-2.

[Mor09] R. Mordani. Java Servlet Specification, 2009.

[MS] K. Mehlhorn, P. Sanders. *Algorithms and Data Structures*. Springer. ISBN 978-3-54-077977-3.

[OAS] OASIS. Organization for the Advancement of Structured Information Standards. URL https://www.oasis-open.org/.

[OMG05] OMG. Unified Modeling Language (UML), 2005. URL http://www.omg.org/spec/UML/.

[OMG11] OMG. Business Process Model and Notation (BPMN), 2011. URL http://www.omg.org/spec/BPMN/2.0/.

[Ora] Oracle. Java BluePrints - Webseite. URL http://www.oracle.com/technetwork/java/javaee/blueprints/index.html.

[PP07] Prof. Dr. rer. nat. Jochen Ludewig, Prof. Dr. rer nat. Horst Lichter. *Software Engineering*. dpubkt.verlag, 2007. ISBN: 978-3898642682.

[Pro11] Prof. Dr. Frank Leymann. *Vorlesung - Web Services*. Institut für Architektur von Anwendungssystemen (IAAS) Universität Stuttgart, 2011.

[SDK10] M. Sonntag, F. L. Dimka Karastoyanova. The missing features of workflow systems for scientific computations. 2010.

[SK] M. Sonntag, D. Karastoyanova. Next generation interactive scientific experimenting based on the workflow technology.

[SLWM07] L. Shankar, D. Lee, M. Wynn-Mackenzie. *Building a human task-centric business process with WebSphere Process Server*. IBM, 2007. URL http://www.ibm.com/developerworks/websphere/library/techarticles/0702_shankar/0702_shankar.html.

[USA05] Urs B. Meyer, Simone E. Creux, Andrea K. Weber Marin. *Grafische Methoden der Prozessanalyse*. Carl Hanser Verlag GmbH & Co. KG, 1 Auflage, 2005. ISBN: 978-3446400412.

[W3C] W3C. Document Object Model. URL http://www.w3.org/DOM/.

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, Donald F. Ferguson. *Web Services Platform Architecture*. Prentice Hall, 2005. ISBN: 978-0131488748.

[Weg02] C. Wege. Portal Server Pechnology. 2002.

[Wes07] M. Weske. *Business Process Management - Concepts, Languages, Architectures*. Springer, 2007. ISBN: 978-3-540-73521-2.

Alle URLs wurden zuletzt am 10. 07. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschift