Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Patterns im Umfeld prozessgetriebener Anwendungen zur Dokumentenverarbeitung mit BPMN

Tim Mühlsteff [1] und Andreas Schäfer [2]

Abstract: An prozessgetriebene Anwendungen werden hohe Anforderungen bezüglich der Wartbarkeit und Bedienbarkeit gestellt. Standardpatterns, angepasst an die Bedürfnisse einer prozessgetriebenen Anwendung, erleichtern es Entwicklern diese Anforderungen zu erfüllen. Konkret wird in diesem Paper gezeigt, dass sich das MVP-Pattern mithilfe eines EventBus-Systems unkompliziert mit Vaadin einsetzen lässt. Über eine Factory in Verbindung mit einer sicher gestalteten Verwendung von Reflections können zusätzliche User Tasks flexibel in der UI verwendet werden, ohne großen Bearbeitungsaufwand im Quellcode zu verursachen. Es wird ein dreiteiliges Design-Pattern vorgestellt, das dem Benutzer alle Informationen übersichtlich darstellt. Genehmigungsprozesse, die in der Modellierung ein Statuskonzept integrieren, ermöglichen eine gute Nutzung von Prozessen über Abteilungsgrenzen hinweg. Weiter ermöglicht BPMN die Prozess-Einzelverarbeitung eines Dokumentes mithilfe eines weiteren Prozesses, um eine Mengenverarbeitung mehrerer Dokumente zu ergänzen.

Keywords: Prozess, User Interface, Pattern, Factory, MVC, MVP, BPMN, Camunda, Vaadin

1 Einleitung

In prozessgetriebenen Anwendungen kann es vorkommen, dass die Modellierung des Geschäftsprozesses sehr komplex wird. Dies kann insbesondere dann der Fall sein, wenn Prozessautomatisierung zum Einsatz kommt. Daher sollte die Systemarchitektur möglichst genau spezifiziert werden. Das führt dazu, dass der Quellcode dieser Anwendung besser strukturiert werden kann und damit wartbarer ist. Zudem sollte auch die Konzeption der User Interface (UI) Komponente bedacht werden. Sie sollte für den Nutzer möglichst leicht und intuitiv bedienbar sein sowie alle relevanten Informationen übersichtlich darstellen. Mit BPMN (Business Process Model and Notation) wird eine Grundlage für die Modellierung von Geschäftsprozessen geboten. Aktuell steht sie in der Version 2.0 zur Verfügung [OM11]. Die Notation ist so strukturiert, dass sich mit Hilfe nur weniger Komponenten Prozesse sowohl aus fachlicher als auch aus technischer Sicht gut abbilden lassen. Zentrale Bestandteile sind dabei vor allem Aktivitäten, Ereignisse sowie Gateways. [FR12, S.21]

Das Open Source Projekt Camunda stellt eine eigene BPM Plattform zur Verfügung, die eine Ausführung automatisierter Prozesse ermöglicht. Dies erfolgt mittels der Camunda Process Engine, die das technische Prozessmodell steuert. [FR12, S.6f]

[1] Fachhochschule Münster, Corrensstraße 25, 48149 Münster, muehlsteff@fh-muenster.de
[2] Fachhochschule Münster, Corrensstraße 25, 48149 Münster, andreas.schaefer@fh-muenster.de

Anhand von BPMN und der darauf aufbauenden Process Engine von Camunda werden im Folgenden Patterns beschrieben, die vor allem im Zusammenhang mit prozessgesteuerten sowie aufgabenorientierten Dokumentenverarbeitungsprozessen eingesetzt werden können.

Im Rahmen dieses Papers werden unterschiedliche Patterns zur Unterstützung automatisierter Prozesse vorgestellt und bewertet. So wird mit den User-Interface Patterns zunächst auf die Trennung von UI- und Geschäftslogik unter Verwendung des Model-View-Presenter (MVP) Patterns näher eingegangen. Dabei wird die Problematik betrachtet, dass der Presenter stets die Steuerung der Benutzeroberfläche übernehmen sollte. Es werden Wege aufgezeigt, wie Steuerungslogik aus der View in den Presenter verschoben werden kann. Ein weiterer Aspekt im Bereich UI ist die Gestaltung und der Aufbau der Verarbeitungsmaske und was in Verbindung mit aufgabenorientierten Prozessen besonders zu beachten ist. Zusätzlich wird auf das technische Mapping zwischen User Task und UI-Maske eingegangen und ein Weg aufgezeigt, der die möglichen Probleme bei der Verwendung von Reflections vermeidet. Im zweiten Teil des Papers stehen Process Patterns im Fokus. Zunächst werden unterschiedliche Möglichkeiten zur Erfassung von Zuständen bzw. Status eines Dokumentes in einem Geschäftsprozess erläutert. Im Weiteren wird darauf eingegangen, wie in einer BPMN-Anwendung Daten zu mehreren Einzeldokumente mit einem bestimmten Zustand in einer Maske erfasst und gespeichert werden können.

2 User-Interface Patterns

In diesem Kapitel geht es um die Vorstellung UI-orientierter Patterns, insbesondere der Anpassung des MVP-Patterns zur Trennung von Darstellungs- und Anwendungslogik, sowie um die wiederverwendbare Strukturierung von UI-Masken.

2.1 Anwendung des Model View Presenter Patterns mit Vaadin

Eine Anwendung soll auf verschiedenen Plattformen und Gerätearten angeboten werden. Dazu sind unterschiedliche Darstellungsformen notwendig. Zudem soll die Anwendungslogik, die im Kontext einer prozessgetriebenen Anwendung vor allem aus der Kommunikation mit der Process Engine und anderen externen Systemen besteht, nicht vermischt werden. Ein bekanntes Entwurfsmuster, das dieses Problem lösen soll, ist das Model-View-Controller (MVC) Pattern und dessen Weiterentwicklung, das Model-View-Presenter (MVP) Pattern. Im MVC Pattern kennt die View das Modell und den Controller, sie nutzt die Objekte aus dem Modell, um die Daten darzustellen. Zusätzlich bearbeitet sie auch Benutzerinteraktionen. Im MVP Pattern kennt die View das Modell nicht. Ebenfalls hat sie keinen direkten Zugriff auf den Presenter und enthält keine Logik zur Steuerung der UI (siehe Abb. 1). Es ist alleinige Aufgabe des Presenters auf das Modell zuzugreifen und die View zu manipulieren. [Fo06]

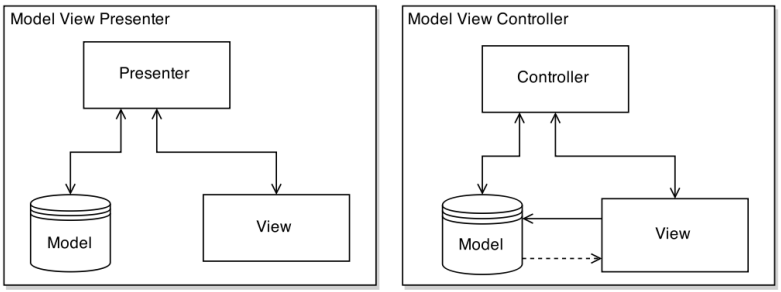

Abb. 1: Model View Presenter im Vergleich zum Model View Controller Pattern

Es besteht die Anforderung, dass die Anwendung über ein Web-Interface erreichbar sein soll. Für die Entwicklung einer Web-Oberfläche unter Java werden verschiedene Frameworks angeboten, die unterschiedliche Schwerpunkte setzen. Das Framework Vaadin bietet den Vorteil, eine Web-UI ausschließlich mit Java-Quellcode zu entwickeln. Es ist optional möglich eigene HTML-Templates zu verwenden. [VA15]

Das Framework Vaadin stellt Entwicklern Listenern auf Basis des Obersever-Patterns bereit, um auf Benutzerinteraktionen reagieren zu können. Es handelt sich dabei um eine Push-Variante in der eine definierte Funktion aufgerufen wird: [GM14, Kap. 5.14]

```
button.addClickListener(new Button.ClickListener() {
   public void buttonClick(ClickEvent event) { ... }
});
```

Im obigen Beispiel wird ein neuer Listener instanziiert und der Button-Variable zugewiesen. Dieser Codeausschnitt ist in der View platziert, die alle UI-Komponenten als Java Objekte beinhaltet. Nach dem MVP-Pattern darf an dieser Stelle keine Steuerungslogik implementiert werden, sondern der Presenter muss das Ereignis verarbeiten. Über EventBus-Systeme kann der Presenter informiert werden. Mit ihnen können aus allen Objekten heraus Ereignisse in einen Bus gesendet werden. Methoden, die als Subscriber beim EventBus registriert sind, werden bei Vorlage eines passenden Ereignisses aufgerufen. Damit kann folgendes Szenario realisiert werden: Aus der Klick-Methode der Schaltfläche kann eine Publish-Nachricht an den EventBus gesendet werden. Im Presenter wird eine Subscriber-Methode implementiert, die beim Anklicken des Buttons ausgeführt werden soll. Ein Open-Source Eventbussystem ist in den weit verbreiteten Google Guava Bibliotheken für Java enthalten. Die Identifikation des konkreten Eventstyps, z.B. *Benutzter klickt auf Schaltfläche A*, erfolgt über einen Java-Typ. Die instanziierten Objekte speichern ggf. Informationen zum Event in Variablen und stellen diese über Getter-Methoden dem Empfänger zur Verfügung. [WL12]

Auf der Webseite von Vaadin wird eine Alternative vorgestellt, die ausschließlich auf Vaadin-Komponenten basiert. Als Beispiel wird eine Taschenrechner-Anwendung verwendet (siehe Abb. 2) [GM13].

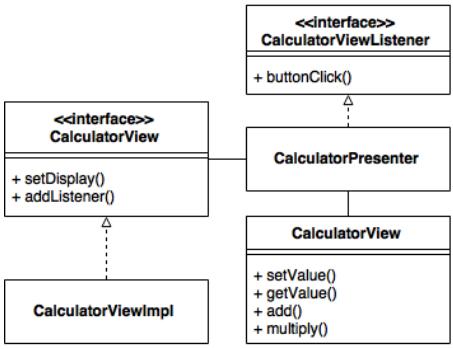

Abb. 2: UML-Klassendiagramm aus dem MVP-Beispiel von Vaadin [GM13]

In diesem Fall ist die View eine CustomComponent, die der Entwickler mit weiteren Komponenten, wie z.B. Schaltflächen, füllen kann. Der Entwickler muss zwei Interfaces anlegen. Ein Interface für die View sowie ein weiteres Interface (*CalculatorViewListener*), in der die Event-Methoden definiert werden. Dieses Interface wird im View-Interface platziert. Die konkrete View-Klasse erbt ein Interface (*CalculatorView*) und die benötigten Standard Vaadin-Listener, in diesem Fall den ClickListener. Das Interface ClickListener erfordert die Implementierung einer Methode, um auf einen Schaltflächenklick zu reagieren. Über den Konstruktor der Button-Klasse wird allen Schaltflächen im Beispiel die View als Listener zugeordnet. Sie ruft die Klick-Funktion aus dem ViewListener auf, der vom Presenter implementiert wird. Über die Schaltflächenbeschriftung, die übergeben wird, kann der Presenter die gewünschte Operation unterscheiden. [GM13]

Die vorgestellten Varianten ermöglichen die Nutzung von MVP mit Vaadin. An beiden lässt sich kritisieren, dass zunächst die View auf das originale Klick-Ereignis von Vaadin reagieren muss. Die Vor- und Nachteile der beiden Versionen sind zusammengefasst: Die Variante mit dem EventBus-System ist flexibel, verständlich und somit leicht umsetzbar. Zudem kann sie auch außerhalb des Vaadin Frameworks eingesetzt werden. Als Vorteil der Vaadin-Variante kann gesehen werden, dass eine zusätzliche Abhängigkeit verhindert wird. Allerdings ist die Umsetzung komplexer.

2.2 Darstellung von Aufgaben zu Dokumenten

Nach der technischen Realisierung der Trennung von Prozess- und UI-Steuerungslogik, behandelt dieses Unterkapitel die Darstellung einer Aufgabe mit sinnvollen Informationen aus einem Dokument im User Interface. Bereits die Erfassung nur weniger Daten zu einem Dokument kann sich in einem Anwendungssystem bereits als komplexe Herausforderung darstellen. Dies ist besonders dann der Fall, wenn sich die einzutragenden Angaben auf Daten beziehen, die bereits wesentlich eher im Verarbeitungsprozess erfasst wurden und somit nicht direkt ersichtlich sind. Zudem kann

das Problem auch dann bestehen, wenn ein Sachbearbeiter in einem mehrseitigen Formular nur an einer bestimmten Stelle die Daten erfassen muss. Um diesem Problem entgegen zu wirken, lassen sich die Daten des Formulars in zwei Kategorien einteilen. Zum einen gibt es Daten, die in dem Dokument bereits erfasst wurden. Sie dienen dem Sachbearbeiter als Wissensbasis und ggf. als Grundlage zur Erfassung weiterer Daten. Zum anderen müssen neue Informationen zu einem Dokument erfasst werden. Dabei kann es sich um Informationen, aber auch um Freigaben oder Ergebnisse einer Dokumentenprüfung handeln. Diese Erfassung von Informationen lässt sich als *Aufgabe* bezeichnen, die z.B. einem Sachbearbeiter zugewiesen ist. In einem Prozess nach BPMN 2.0 wird eine Aufgabe durch eine Benutzer-Aufgabe (User Task) definiert. Im Gegensatz zu einer manuellen Aufgabe kann sie durch eine Process Engine gesteuert und in dem Zuge einem Benutzer oder einer Benutzergruppe zugewiesen werden, indem sie in einer benutzerspezifischen Aufgabenliste (Task List) abgelegt wird. Zur Erledigung der Aufgabe muss eine entsprechende Interaktion, z.B. Eingabe oder Button-Klick, des Nutzers erfolgen. Erst dann wird der User Task beendet und der Prozess kann weiter verarbeitet werden. [FR12, S. 72]

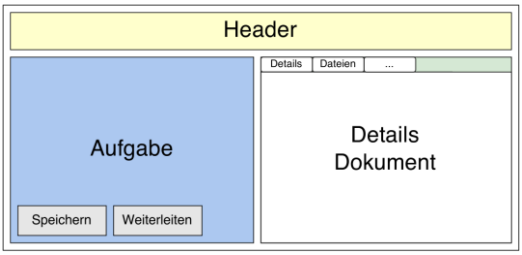

Abb. 3: Darstellung einer Aufgabe zu einem Dokument, angelehnt an [SN11, Abb.8-8]

Die User Tasks können aus der Process Engine abgefragt und auf einer Benutzeroberfläche (User Interface) dargestellt werden. Dies bietet dem Nutzer die Möglichkeit die Daten in einer Formularansicht zu erfassen und abschließend durch eine Schaltfläche zu bestätigen. Bei der Darstellung der Aufgabe sollte beachtet werden, dass benötigte Informationen zur Bearbeitung der Aufgabe entsprechend übersichtlich angezeigt werden. Es bietet sich an, die zu erfassenden und die benötigten Informationen nebeneinander zu platzieren (siehe Abb. 3:). Der linke Aufgabenbereich gestaltet sich entsprechend der jeweiligen Aufgabe, wohingegen der rechte Bereich das Dokument (Business Object) mit allen bereits erfassten Daten registerweise abbildet. Die Register sollten dabei in sich zusammenhängende Kategorien bilden und aufgabenspezifisch geöffnet werden. Besonders hervorzuheben ist, dass die Darstellung im rechten Bereich für alle Aufgaben identisch ist. Dies führt zu einer besseren Orientierung und Wiedererkennung. Um die gesamte Breite eines Bildschirms zur Visualisierung nutzen zu können, sollte zudem auf weitere Elemente in der horizontalen Ebene verzichtet werden. Dazu zählt auch ein vertikales Menü, das durch ein horizontales im oberen Bereich der Ansicht ersetzt werden sollte. Außerdem ist es hilfreich, die vertikale Trennung zwischen Aufgaben und Informationen dynamisch und variabel zu gestalten.

Dadurch lässt sie sich je nach Bedarf verschieben und die einzelnen Bereiche können gezielt vergrößert dargestellt werden.

Ein derartiges Vorgehen wird auch mit dem *Process/BO-Pattern* für BPM-Projekte beschrieben. Es sieht vor, dass zu jedem Prozess ein zentrales Business-Objekt (BO) gehört. Die aufgabenübergreifend wichtigsten Daten, wie z.B. ID, Bezeichnung und Ersteller, werden im Header abgebildet. [SN11, S. 192f] Eine Erweiterung des Patterns stellt das sogenannte *Process/BO-Portlet mit Task Data* dar. Sie definiert einen gesonderten Bereich für die Erfassung von Benutzereingaben in Form einer Art Formular. Daneben werden in einem weiteren Bereich die für die Erfassung relevanten vorhandenen Informationen angezeigt. [SN11, S. 280]

2.3 Mapping von User Task zu User Interface mit Vaadin und Camunda BPM

Nach dem grafischen Entwurf der Darstellung, muss dieser auch technisch umgesetzt werden. Es muss ein Mapping zwischen UI-Maske und User Task im Prozess realisiert werden. Zudem sollen die Masken, wie in Kapitel 2.2 beschrieben, als weitestgehend unabhängige und somit austauschbare Komponenten geladen werden. Der BPMN 2.0 Standard enthält keine Definitionen für das User Interface, daher unterscheidet sich das Vorgehen je nach Wahl der Process Engine. Ein Pattern zum Erstellen bzw. Laden von Objekten ist das Factory Design Pattern. Es bietet Methoden zum Erzeugen von Objekten von bestimmten Typen. Über Parameter entscheidet der Algorithmus der Factory, welches Objekt erstellt und welches zurückgeliefert wird.

In diesem Abschnitt werden der Einfachheit halber nur die Begriffe und Techniken von Camunda genannt und verwendet. Für das Mapping zwischen UI-Maske und User Task im Prozess existiert in Camunda BPM eine Variable namens FormKey. Hier kann bspw. der Pfad zu einer View-Komponente oder JSP-Seite hinterlegt werden. [CM13]

Die Darstellung der Aufgaben und des Dokuments erfolgt auf einer Webseite. Es gibt aufgrund der Verwendung des MVP-Patterns eine View und einen Presenter für die Bearbeitungsseite. Die User Tasks können über eine Vaadin CostumComponent dargestellt werden. In dieser Klasse können Standardkomponenten beliebig platziert und angesteuert werden. [GM14, Kap 5.29] Sie sind die Grundlage für die Aufgabenkomponenten. Der Presenter prüft beim Aufruf der Bearbeitungsseite, ob über die URL eine Identifikationsnummer eines Dokumentes übergeben wurde. Wenn dies der Fall ist, prüft er über das Model, ob es das Dokument gibt und ob dazu ein User Task vorliegt. Dann muss der Presenter dafür sorgen, dass die passende Aufgabenkomponente angezeigt wird. Die Instanziierung der passenden Aufgabenkomponente erfolgt nach dem Factory-Design-Pattern. Das trägt zu einem übersichtlichen Quellcode im Presenter bei und eröffnet die Möglichkeit, die Art der Instanziierung flexibel zu ändern, ohne den Presenter bearbeiten zu müssen. [CJ10]

Die Factory ist *static* deklariert und bietet eine Funktion zur Erzeugung der Aufgabenkomponenten. Als Übergabeparameter erwartet die Funktion den FormKey des

User Tasks aus der Prozessinstanz. Alle Aufgabenkomponenten erben von einer abstrakten *AbstractTaskComponent*-Klasse. Alle Klassen, die von der abstrakten Klasse erben, haben Zugriff auf den EventBus, der bei der Instanziierung über den Konstruktor gesetzt wird. Weiter sind zwei abstrakte Methoden definiert. Eine Methode zum Einlesen eines Datenobjektes, dessen Informationen in der UI-Komponente angezeigt werden soll, und eine Methode, die die Eingaben des Benutzers in das Datenobjekt zurückschreibt. Der Presenter braucht keine Kenntnis über den konkreten Typ der Aufgabekomponente, weil er die Komponenten ausschließlich mit den abstrakten Methoden der Oberklasse steuern kann. Die Schaltfläche zum Abschließen einer Aufgabe ist in der Aufgabenkomponente platziert. Somit empfängt sie das Klick-Ereignis und leitet es an den EventBus weiter. Die Subscriber-Methoden sind im Presenter platziert (siehe 2.1). Die Factory erzeugt die passende Instanz via Reflections. Der Einsatz von Reflections ist risikobehaftet. Es können Exceptions auftreten, die bei einem „normalen" Instanziieren einer Klasse nicht auftreten. Es sollten Maßnahmen getroffen werden, um die Risikoquellen gering zu halten: [OR14] Der FormKey wird nicht dazu genutzt, um den Pfad der konkreten Klasse zu hinterlegen, sondern fungiert als Key für die Map in der *TaskComponentFactory* (siehe Abb. 4). Nur wenn zu dem übergebenen Key eine Klasse hinterlegt ist, startet die Instanziierung. Ansonsten wird dem Benutzer eine Fehlermeldung angezeigt und die Entwickler über Logfile-Einträge informiert. So kann sichergestellt werden, dass bei einer Umbenennung der Komponente, beispielsweise im Rahmen eines Refactorings, das Prozessmodell und laufende Prozessinstanzen weiterhin funktionieren, ohne Exceptions zu erzeugen.

Eine weitere mögliche Fehlerquelle ist die Hinterlegung von unpassenden Klassen in der Map. Damit nur Klassen vom Typ *AbstractTaskComponent* hinterlegt werden können, ist der Typ-Parameter der Map mit dem Schlüsselwort *extends* definiert.

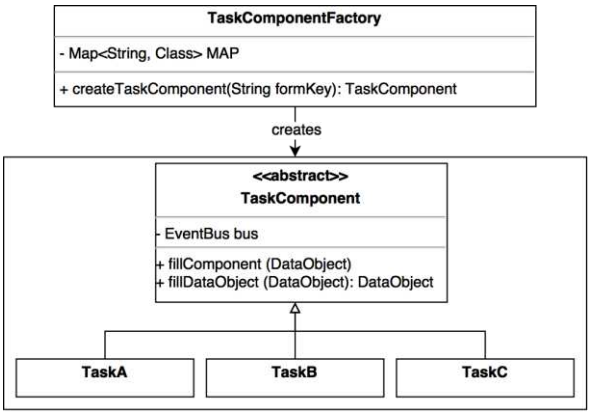

Abb. 4: UML-Klassendiagramm mit Factory und Vererbungsstruktur der Aufgaben

Ein neuer User Task erfordert nach diesem Ansatz lediglich die Entwicklung einer Aufgabenkomponente sowie das Hinzufügen eines Key-Value Paars in die Map. Diese

Lösung erlaubt die Umbenennung von Klassen, ohne die Prozesse anpassen zu müssen. Zudem kann der Compiler erkennen, ob die in der Map eingetragenen Klassen existieren und Typsicherheit gewährleisten. Die typischen Reflections-Probleme, die erst zur Laufzeit auftreten, werden so weitestgehend vermieden.

Eine weitere Möglichkeit wäre es, die Map aus einer Konfigurationsdatei zu erzeugen. Das hätte den Vorteil, dass die Komponenten im Betrieb auch für laufende Prozessinstanzen ausgetauscht werden können, z.B. wenn eine überarbeitete Version vorliegt, die einfacher für den Benutzer zu bedienen ist. Diese Variante bietet keine Typsicherheit und bei einem Refactoring muss die Konfigurationsdatei ggf. überarbeitet werden, um die Lauffähigkeit der Prozesse sicherzustellen. Ein Wenn-Dann-Konstrukt kommt nicht in Frage. Es ist starr und wird bei einer größeren Anzahl von User Tasks unübersichtlich. Eine Änderung der Konstruktor-Parameter beträfe mehrere Programm-Zeilen, während bei der Reflections-Variante nur eine Zeile verändert werden muss.

Auf der Webseite von Vaadin wird gezeigt, wie das Framework mit der Process Engine Activiti, von der Camunda BPM ursprünglich abstammt [NA13], eingesetzt werden kann. Das Vaadin-Beispiel setzt auch auf Reflections in Kombination mit einer typsicher gestalteten Map. Die Klasse *UserTaskFormContainer* aus dem Beispiel agiert auch hier als Fabrik und instanziiert die Objekte. Der Unterschied zur oben dargestellten Variante ist, dass die Klassen zur Laufzeit über eine Methode zur Map hinzugefügt werden können. Außerdem werden im Vaadin-Beispiel nicht Komponenten in eine Seite geladen, sondern es erfolgt eine Weiterleitung zur entsprechenden Seite. [MP11]

3 Process Patterns

Im Folgenden werden prozessorientierte Patterns im Zusammenhang mit der Verarbeitung von Dokumenten im BPMN Kontext dargestellt. Im Einzelnen wird näher auf die Erfassung von Dokumentenstatus sowie die gleichzeitige Verarbeitung mehrerer Dokumente in einem BPMN-Geschäftsprozess eingegangen.

3.1 Implementierung eines Statuskonzeptes in BPMN

Ein Dokument kann innerhalb eines Genehmigungsprozesses unter Umständen viele unterschiedliche Abteilungen oder gar Unternehmen durchlaufen. Die Definition eines Statuskonzeptes ermöglicht einen besseren Überblick und den gesamten Workflow. Dadurch lässt sich zu jeder Zeit der Zustand einer Prozessinstanz und damit der eines Dokumentes feststellen. Außerdem ist ersichtlich, welche Person der vorangegangene bzw. nachfolgende Sachbearbeiter ist.

Bei dem Einsatz von BPMN wird der Schwerpunkt auf den Prozessablauf (Sequenzfluss) gelegt. Andere Aspekte, wie z.B. Daten und Datenfluss, werden zweitrangig behandelt. [FR12, S. 103] Dennoch sind für die Abbildung des Status eines

Dokumentes in einer BPMN-Anwendung unterschiedliche Umsetzungsmöglichkeiten denkbar. So bietet die Modellierung von Datenobjekten (Data Object) innerhalb des Diagramms eine Art der Darstellung von Informationen. Ein Datenobjekt kann an der entsprechenden Stelle im Sequenzfluss platziert und neben der Bezeichnung auch ein Status in eckigen Klammern definiert werden (siehe Abb. 5:). Es legt dar, welche Daten zu einem bestimmten Zeitpunkt in Prozess vorhanden sein müssen und welchen Status sie besitzen sollten. Dementsprechend verhält es sich mit Dokumenten, die durch eine Aufgabe (Task) erzeugt oder manipuliert werden. Dies unterstützt den Leser eines BPMN-Diagramms durch die Anreicherung von Details und führt zu einem besseren Verständnis des Prozesses.

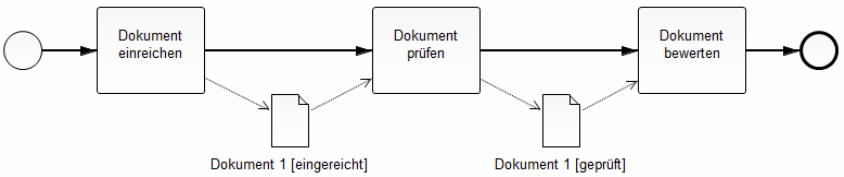

Abb. 5: Statuserfassung mit Datenobjekten in BPMN 2.0

Trotz der unterstützenden Funktionen bietet dieser Ansatz einen entscheidenden Nachteil: Es ist keine automatisierte Bearbeitung des Datenobjektes sowie dessen Status im Rahmen von ausführbaren BPMN-Modellen möglich. Wenn sich beispielsweise durch einen Task der Status eines Dokumentes von *eingereicht* in *geprüft* ändert, ist dieses im Diagramm zwar abbildbar, der Status eines Dokumentes kann jedoch nicht explizit abgefragt werden. Dafür wäre es notwendig den Status separat, z.B. in einer Datenbank, zu erfassen.

Eine alternative Möglichkeit bietet der Einsatz von Zwischenereignissen (Intermediate Events). Sie können durch den Prozess ausgelöst werden und einen Status beschreiben, der durch das Durchlaufen des Sequenzflusses erreicht werden kann. [FR12, S. 26]

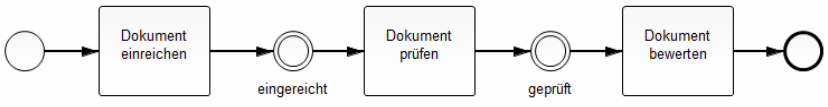

Abb. 6: Statuserfassung mit Intermediate Events in BPMN 2.0

In dem beispielhaften Prozess, der in Abb. 6: dargestellt ist, wird zunächst ein Dokument eingereicht. Unmittelbar im Anschluss daran folgt ein Intermediate Event, das den Status des Dokumentes als *eingereicht* kennzeichnet. Ebenso verhält es sich mit der Aktivität *Dokument prüfen* und dem darauf folgenden Status *geprüft*.

Durch die Zwischenereignisse lässt sich demnach, vergleichbar mit den Datenobjekten im Diagramm, der Status des Prozesses bzw. des darin verarbeiteten Dokumentes erkennen. Jedoch bieten die Events zusätzliche Funktionalitäten für die Verarbeitung in

ausführbaren BPMN-Anwendungen mit Camunda. So ist es möglich zu einem Ereignis einen Listener zu definieren und einen Ausdruck (Expression) zu hinterlegen, der bspw. das Aufrufen einer Methode bewirkt. Der Methode wird der entsprechende Status mit übergeben, sodass dieser zum Dokument gespeichert werden kann.

Innerhalb der Methode *veraendereStatus* wird zunächst das Dokument über eine ID ermittelt, die sich über die *execution*-Variable aus dem Prozess auslesen lässt. Zu dem betreffenden Dokument wird anschließend der Status in einer Datenbank erfasst. Der entscheidende Unterschied zur Umsetzung mit Datenobjekten ist, dass der Methodenaufruf mit dem entsprechenden Status direkt aus dem Modell erfolgt. Die Methode ist einmalig zu entwickeln, danach kann sie in jedem Intermediate Event verwendet werden und es sind keine weiteren Programmierungen außerhalb des Diagramms mehr notwendig. Dies hat den Vorteil, dass Änderungen und Erweiterungen im Prozess vereinfacht werden. Zudem erfolgt für Beteiligte des operativen Prozesses eine erhöhte Transparenz. So lässt sich der Zustand der Prozessinstanz und damit der Status eines Dokumentes zu jeder Zeit, z.B. über ein Verwaltungssystem, prüfen.

3.2 Mengenverarbeitungsprozess

Bei dem Einsatz einer BPMN-Anwendung werden Prozessinstanzen erzeugt. Dies geschieht, wenn in der Realität ein neuer Prozess gestartet wird [FR12, S. 23]. Somit kann man unter einer Instanz einen Prozess verstehen, den z.B. ein Dokument zur Bearbeitung durchläuft. Dieses wird durch unterschiedliche Sachbearbeiter solange editiert, bis alle erforderlichen Daten vorhanden sind. Dabei kann es vorkommen, dass zu einem bestimmten Ereignis mehrere Dokumente mit dem gleichen Status am selben Wert manipuliert werden. Dieser Fall tritt bspw. ein, wenn im Rahmen einer Sitzung Entscheidungen über die Prioritäten einer Vielzahl von Anträgen getroffen werden. Das Ziel ist die Datenerfassung über mehrere Anträge auf nur einer Ansichtsmaske, z.B. in einer Tabelle. Dieser Abschnitt beschäftigt sich mit der Frage, wie sich mehrere unabhängige Prozessinstanzen gebündelt innerhalb eines Tasks weiterbearbeiten lassen.

Die Verarbeitung des Prozesses wird in dem o.g. Beispiel durch das Einreichen eines Dokumentes initialisiert. Im BPMN-Prozess durchläuft die entsprechende Prozessinstanz verschiedene Tasks und Gateways, bis das Ereignis eintritt, in dem über eine Menge von Dokumenten gebündelt entschieden wird, z.B. im Anschluss an eine Bewertung des Dokumentes. An dieser Stelle lässt sich im BPMN-Prozess ein Intermediate Event platzieren, wodurch die eintreffenden Instanzen so lange warten, bis die Erfassung der Daten in den Dokumenten erfolgt ist. Als Ereignistyp wird ein Nachrichtenereignis (Message Event) eingesetzt, da sich so die Kommunikation zwischen der Einzel- und Mengenverarbeitung abbilden lässt (siehe Abb. 7).

Die Mengenverarbeitung erfolgt in einem gesonderten BPMN-Pool, der ausschließlich für diese Tätigkeit verwendet wird. Die Initialisierung einer Instanz in diesem Pool kann durch eine Nutzerinteraktion über das User Interface erfolgen, z.B. durch das Anklicken einer Schaltfläche. Auf der Benutzeroberfläche wird dem User eine Übersicht aller

Dokumente dargestellt, in der er eine bestimmte Eigenschaft zu allen Dokumenten erfassen kann. Durch das Anklicken der besagten Schaltfläche, z.B. Speichern, erfolgen mehrere Teilschritte, die sich zu zwei Schritten zusammenfassen lassen.

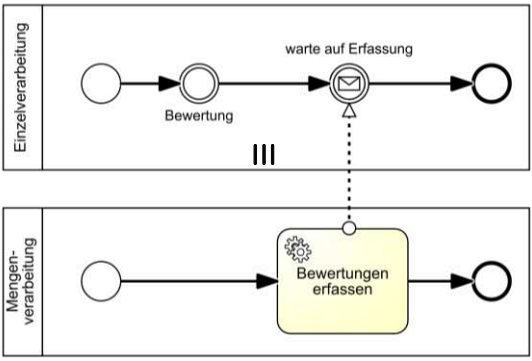

Abb. 7: Mengenverarbeitung mit Hilfe von zwei Pools in BPMN 2.0

Schritt 1: Datenerfassung in der Datenbank: Für alle dargestellten Dokumente werden die angegebenen Daten in einer Datenbank erfasst. Im o.g. Beispiel wären dies die Prioritäten zu den jeweiligen Dokumenten.
Schritt 2: Aufgabe zur Datenerfassung abschließen: Zunächst wird ein neuer Prozess in der Mengenverarbeitung instanziiert. Daraufhin wird die Aktivität *Bewertungen erfassen* in Form eines Service Tasks bearbeitet. Die Ausführung des Service Tasks erfolgt automatisiert. Sie versendet eine Nachricht, auf die das Message Event der jeweiligen Prozessinstanzen im Einzelverarbeitungsprozess (Gesamtprozess) wartet (s.o.).

Ein Problem dabei stellt die Transaktionssicherheit dar. Ein konsistenter und widerspruchsfreier Zustand ist für die Geschäfts- und die Prozessdaten nicht zu jeder Zeit gewährleistet. Sollte nach Schritt 1 z.B. ein Fehler auftreten und die Verarbeitung abbrechen, sind die Daten in der Datenbank erfasst. Der Prozess bildet dies jedoch nicht ab, da die Aufgabe nicht abgeschlossen wurde und die Prozessinstanzen weiterhin beim Message Event auf eine Nachricht warten.

4 Fazit

Probleme, die bei der Entwicklung einer prozessgetriebenen Anwendung auftauchen, können häufig durch Standardpatterns gelöst werden. Etablierte Patterns haben in der Praxis bewiesen, dass sie ein Problem gut und sinnvoll lösen können. Die Verwendung führt zu einer übersichtlicheren Architektur. In der Architekturbeschreibung kann auf Patterns verwiesen werden und neuen Entwicklern fällt die Einarbeitung leichter.

Die in diesem Paper vorgestellten Patterns in Kapitel 2.1 (Anwendung des Model View Presenter Patterns mit Vaadin) und 2.3 (Mapping von User Task zu User Interface mit

Vaadin und Camunda BPM) sind auf Vaadin und Camunda zugeschnitten.

Die Patterns zur Gestaltung der Benutzeroberfläche, die Implementierung eines Statuskonzeptes in BPMN und der Mengenverarbeitungsprozess können, wie bei Patterns üblich, mit geringen Anpassungen in jede Prozess-Anwendung übernommen werden. M. Fowler schrieb, dass Patterns nicht "blind" übernommen werden sollen, sondern "im Ofen ihres Projektes nachgaren" müssen. Die Verwendung von Patterns ohne Anpassungen an den Kontext führe zu schlechten Ergebnissen. [Fo09, S. 24]

Literaturverzeichnis

[CM13] Camunda Services GmbH: Tutorials & How Tos - User Interface, http://docs.camunda.org/latest/real-life/how-to/; Stand: 26.02.2015.

[CJ10] Cooper, J.: Java Design Patterns: A Tutorial. Addison Wesley, Reading Massachusetts, 2010.

[Fo06] Fowler, M.: http://www.martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp; Stand: 26.02.2015.

[Fo09] Fowler M: Patterns für Enterprise-Application-Architekturen. MITP-Verlag, Heidelberg, 2003.

[FR12] Freund, J.; Rücker, B.: Praxishandbuch BPMN 2.0. Hanser, München, 2012.

[GM13] Grönroos, M.: Model-View-Presenter Pattern with Vaadin. https://Vaadin.com/web/magi/home/-/blogs/2342730; Stand: 26.02.2015.

[GM14] Grönroos, M.: Book of Vaadin - Vaadin 7 Edition. 4. aktualisierte Auflage. Vaadin, 2014.

[MP11] Holmström, P.: Building Vaadin Applications on top of Activiti, https://vaadin.com/wiki?p_p_id=36&p_r_p_185834411_title=Building+Vaadin+Applications+on+top+of+Activiti; Stand: 25.02.2015.

[NA13] Neumann, A.: Camunda stellt BPM-Plattform unter Open-Source-Lizenz, http://www.heise.de/developer/meldung/camunda-stellt-BPM-Plattform-unter-Open-Source-Lizenz-1824467.html; Stand: 26.02.2015.

[OR14] Oracle: Trail: The Java Tutorials - The Reflection API, http://docs.oracle.com/javase/tutorial/reflect/; Stand 25.02.2015.

[SN11] Slama, D.; Nelius, R.: Enterprise BPM. dpunkt.verlag, Heidelberg, 2011.

[WL12] Wasseman, L.: Event Bus Explained, https://code.google.com/p/guava-libraries/wiki/EventBusExplained; Stand: 25.02.2015.

[VA15] Vaadin Ltd.: Introduction, https://Vaadin.com/introduction; Stand: 26.02.2015.