ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik

Institut für Datenbanken und Informationssysteme

Diplomarbeit

# Konzeption, Implementierung und Integration einer Komponente für die Erstellung intelligenter Formulare

Florian Micheler

9. Juni 2009

1. Gutachter: Prof. Dr. Manfred Reichert

2. Gutachter: PD Dr. Stefanie Rinderle-Ma

Betreuer: Dipl.-Inf. (U) Ulrich Kreher

**Kurzfassung**

Geschäftsprozesse in Unternehmen werden immer umfangreicher. Aus diesem Grund werden schon lange computergestützte Systeme zu deren Umsetzung eingesetzt. Dadurch ist es möglich, auch sehr komplexe Prozesse kontrollieren zu können. Die zunehmende Komplexität ist aber nicht das einzige Problem. Es können neue Prozesse hinzukommen, welche eine komplett andere Semantik haben; der Ablauf von vorhandenen Prozessen kann sich ändern. Daher setzen bereits viele Unternehmen bei der Umsetzung ihrer Geschäftsprozesse auf moderne Business-Process-Management-Systeme (BPM-Systeme). Diese Systeme sind auf die Gestaltung und Ausführung von Prozessen spezialisiert.

Ein wichtiger Aspekt bei der Ausführung von Prozessen stellt die Interaktion von Prozessen mit dem Benutzer dar. Diese Benutzer kommen dabei aus sehr unterschiedlichen Branchen. Die Interaktion mit dem BPM-System muss daher so gestaltet sein, dass sich alle Benutzer damit zurecht finden. Ein adäquates Mittel für eine unkomplizierte Interaktion stellen Formulare dar. Durch Formulare können dem Benutzer Informationen zur Verfügung gestellt oder von diesem entgegengenommen werden.

Neben einer einfache Handhabung gibt es jedoch noch weitere Anforderungen an Formulare. Formulare müssen ausreichend flexibel sein, um auch komplexeres Verhalten zu ermöglichen, beispielsweise um den Benutzer auf falsche Eingaben aufmerksam zu machen. Ein weiterer wichtiger Aspekt ist eine einheitliche Darstellung der Formulare innerhalb eines Unternehmens. Diese müssen immer gleich aufgebaut sein, unabhängig davon, wo sich ein Mitarbeiter befindet und auf welcher Plattform die Formulare dargestellt werden.

Die vorliegende Diplomarbeit befasst sich mit der Konzeption einer Softwarekomponente, welche in der Lage ist, solche Formulare zu erstellen und darzustellen. Diese Komponente besteht aus einer Entwicklungs- und einer Ausführungsumgebung. Die Entwicklungsumgebung dient der einfachen und schnellen Erstellung von Formularen. Die Ausführungsumgebung kümmert sich um die korrekte Darstellung der Formulare in unterschiedlichen grafischen Umgebungen. Die Komponente wird prototypisch implementiert und in ein vorhandenes BPM-System integriert. Hierfür wird der Prototyp von ADEPT2 verwendet. Bei ADEPT2 handelt es sich um ein BPM-System der nächsten Generation, welches an der Universität von Ulm [Ulm09] unter der Leitung von Prof. Dadam und Prof. Reichert entwickelt wurde.

Im Rahmen diese Arbeit werden auch vorhandene Produkte untersucht, die eine ähnliche Funktionalität zu der erstellenden Softwarekomponente bereitstellen. Dabei wird geprüft, welche Produkte für das Erstellen von Formularen geeignet sind und ob sie sich gegebenenfalls im Rahmen dieser Arbeit verwenden lassen.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Seit vielen Jahren vertrauen Unternehmen auf computergestützte Systeme, die bei der Umsetzung von Geschäftsprozessen helfen. In großen Unternehmen werden diese Prozesse jedoch immer komplexer und daher immer schwerer zu beherrschen. Hinzu kommt, dass sich Prozesse ändern können, was auch Auswirkung auf die Software dieser Systeme hat.

Eine Möglichkeit, die Komplexität zu bewältigen, ist für jeden Prozess eines Unternehmens eine individuelle Software zu entwickeln. Der Prozess dient dabei als Anforderung für die neue Software. Dadurch wird eine optimale Anpassung der Software an das jeweilige Umfeld gewährleistet. Allerdings hat dieses Verfahren einige Nachteile:

1. Die Entwicklung individueller Software erfordert ausgebildetete Informatiker und Softwareingenieure. Dieser Bedarf steigt mit der Komplexität des Prozesses, was zu hohen Personalkosten führt.

2. Unternehmen, die nicht in der Lage sind die Entwicklung selbst durchzuführen, lagern diese in Softwarehäuser aus, welche sich jedoch erst mit der Semantik der Geschäftsprozesse und den internen Abläufen des Unternehmens vertraut machen müssen. Dies führt neben den hohen Kosten zusätzlich zu einem hohen zeitlichen Aufwand für das Unternehmen.

3. Es gibt keine Trennung von Prozesslogik und Anwendungslogik. Dies erschwert Anpassungen der Software, wenn sich der Ablauf des Geschäftsprozesses verändert. Daraus wiederum ergeben sich hohe Folgekosten durch eine aufwendige Wartung der Software.

Aufgrund dieser Nachteile setzen immer mehr Firmen auf den Einsatz von modernen Business-Process-Management-Systemen (BPM-Systemen). Diese Systeme verfolgen andere Paradigmen als traditionelle Softwareentwicklung:

Anstatt eine eigene Software speziell für jeden einzelnen Geschäftsprozess neu zu programmieren, werden diese Prozesse bei BPM-Systemen modelliert. Dies führt zu einer klaren Trennung zwischen Prozess- und Anwendungslogik (Abbildung 1). Die Prozesslogik beschreibt darin den Kontrollfluss des Geschäftsprozesses. Damit wird festgelegt, welche Prozessschritte in welcher Reihenfolge ausgeführt werden. Die Anwendungslogik hingegen beschreibt den internen Ablauf eines Prozessschrittes. Sie ist in Form einzelner Anwen-

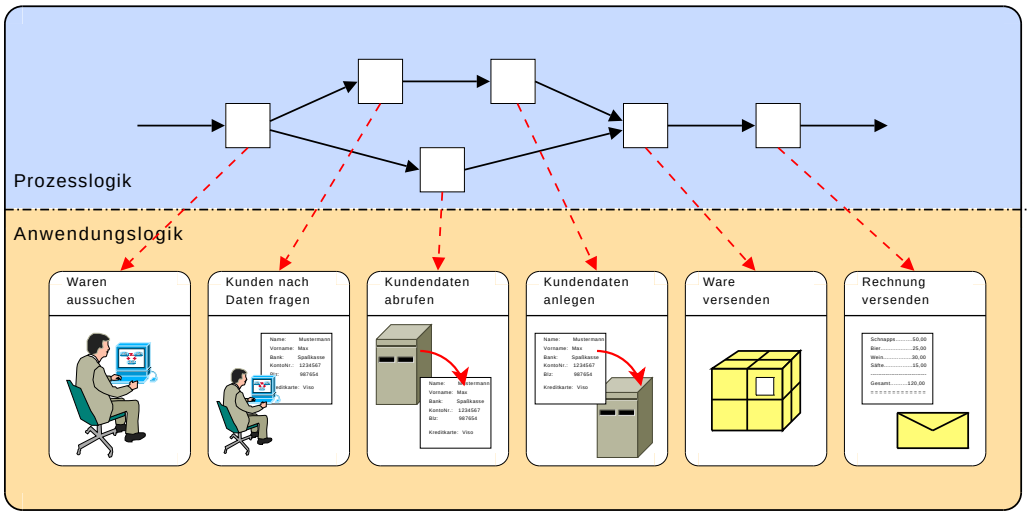


Abbildung 1: Trennung von Prozess- und Anwendungslogik

dungsbausteine in den Prozess integriert und kann mit Hilfe der Prozesslogik beliebig kombiniert werden, um die unterschiedlichsten Geschäftsprozesse umzusetzen. Diese Trennung hat gegenüber der traditionellen Softwareentwicklung einige entscheidende Vorteile:

Zum Einen sind hier Wartbarkeit, Erweiterbarkeit und Anpassbarkeit zu nennen. Bei Änderungen eines Geschäftsprozesses ist es beispielsweise ausreichend, den Kontrollfluss entsprechend anzupassen. Eine Änderung von Quellcode ist nicht erforderlich, solange sich der interne Ablauf eines Prozessschrittes nicht ändert.

Zum Anderen ist für die Modellierung kein ausgebildeter Informatiker notwendig. Um ein Modell des Prozesses zu erstellen oder zu ändern, reicht ein Mitarbeiter, der mit der Semantik des Prozesses vertraut ist. Dies führt zu einer erheblichen Senkung der Kosten und des zeitlichen Aufwands.

Bei den Prozessschritten werden unter anderem zwei Typen unterschieden:

- Prozessschritte für die Benutzerinteraktion
- Prozessschritte für die Datenverarbeitung

Bei den Prozessschritten für die Benutzerinteraktion geht es um den Austausch von Informationen zwischen Endanwender und System. Ein einfaches Mittel hierfür stellen Formulare dar. Diese beinhalten Felder, die einerseits dem Endanwender Informationen präsentieren und diesem andererseits die Möglichkeit bieten, Daten einzugeben. Formulare sind einfach zu erstellen

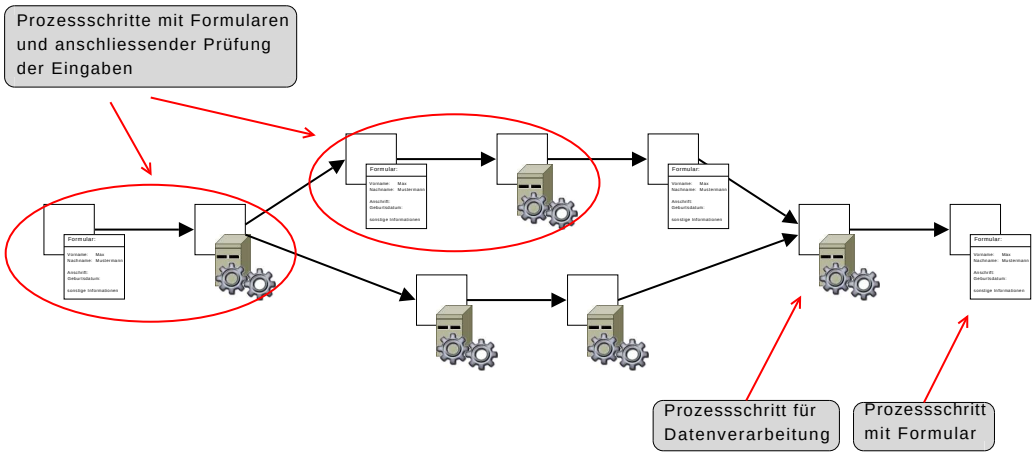


Abbildung 2: Prozess mit Dateneingaben und Datenverarbeitung

und zeigen den Endanwendern eine einheitliche Oberfläche. Formulare sind daher ein festes Konzept vieler BPM-Systeme. Allerdings enthalten Formulare in vielen Systemen kaum Anwendungslogik oder die Integration der Anwendungslogik geht zu Lasten der Bedienbarkeit.

Es gibt jedoch auch Prozessschritte, bei denen eine einfache Interaktion nicht ausreichend ist. Dies ist beispielsweise der Fall, wenn fehlerhafte Eingaben des Endanwenders erkannt werden sollen oder die eingegebenen Daten aufbereitet werden müssen, bevor sie an das System übergeben werden. Solche Anforderungen sind durch Formulare nicht optimal abgedeckt und benötigen zusätzliche Anwendungslogik.

Um diese Anforderungen dennoch zu erfüllen, werden zusätzlich Prozessschritte zur Datenverarbeitung in den Kontrollfluss des Geschäftsprozesses integriert. Diese Prozessschritte erlauben die Integration zusätzlicher Anwendungsbausteine, welche die benötigte Logik bereitstellen. Um die beispielhaft genannten Anforderungen umzusetzen, entwirft der Prozessmodellierer einen Prozessschritt mit einem Formular, welches die Eingaben des Endanwender entgegennimmt. Dem nachfolgenden Prozessschritt ordnet dieser Prozessmodellierer einen Anwendungsbausteine zu, welche die eingegebenen Daten prüft und gegebenenfalls auch weiterverarbeitet. Abbildung 2 zeigt einen Prozess mit mehreren Formularen und datenverarbeitenden Prozessschritten. Dieses Vorgehen führt jedoch zu Nachteilen:

1. Die Anwendungsbausteine, welche die zusätzlich benötigte Logik enthalten, müssen zunächst entwickelt werden. Im beschriebenen Beispiel muss ein Anwendungsbaustein entwickelt werden, welche die Einga-

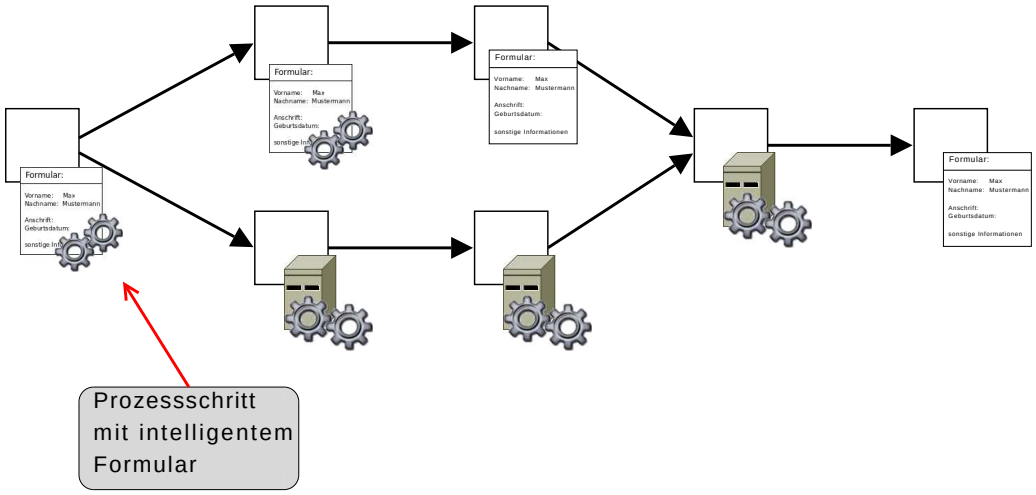


Abbildung 3: Vereinfachter Prozess mit intelligenten Formularen

ben des Endanwenders im vorangegangenen Prozessschritt auf deren Korrektheit hin überprüft. Dies bringt wiederum die Nachteile der traditionellen Softwareentwicklung mit sich, wie beispielsweise der Bedarf an qualifiziertem Personal und hohe Kosten.

2. Prozessschritte mit Formularen, die zusätzliche Logik erfordern, müssen in mehrere Prozessschritte aufgeteilt werden, was die Komplexität des Prozesses unnötig erhöht. Das Abfragen der Benutzereingaben und deren Prüfung ist semantisch gesehen ein einziger Schritt. Um diesen umsetzen zu können, sind jedoch zwei Prozessschritte notwendig.

Ein neuer Ansatz zur Bewältigung der genannten Probleme ist die Verwendung von „intelligenten“ Formularen. Sie bieten wie herkömmliche Formulare eine einfach erstellbare Oberfläche. Zusätzlich ist eine Integration von Anwendungslogik in den Formularen möglich, beispielsweise durch ausführbare Skripte. Dies erhöht die Funktionalität und verringert damit den Bedarf an zusätzlichen Prozessschritten. Für eine Prüfung der Eingaben eines Endanwender muss damit kein separater Anwendungsbaustein entwickelt und in den Prozess eingebunden werden. Das Formular kann diese Prüfung selbst durchführen.

Prozessschritte welche sich hauptsächlich mit der Datenverarbeitung beschäftigen, können durch intelligente Formulare nicht ersetzt werden. Diese Formulare helfen jedoch datenverarbeitende Prozessschritte einzusparen und vereinfachen damit die Prozesslogik eines Geschäftsprozesses. Abbildung 3 zeigt den selben Prozess, der auch in Abbildung 2 zu sehen ist. Es wurden

jedoch einige Prozessschritte durch intelligente Formulare ersetzt um den Prozess zu vereinfachen.

## 1.2 Aufgabenstellung

Im Rahmen dieser Arbeit wird eine Softwarekomponente entwickelt, mit deren Hilfe intelligente Formulare für Prozessschritte erstellt und dargestellt werden können.

Dafür wird untersucht, welche Anforderungen an solche Formulare gestellt werden. Hierzu zählen die Unterstützung von Formularfeldern für einfache und komplexe Datentypen sowie Listen, das Darstellen der Formulare mit unterschiedlichen Oberflächenbibliotheken sowie eine mögliche Automatisierung des Erstellvorgangs. Dabei ist auch zu evaluieren, inwieweit die externe Anbindung von Datenquellen sinnvoll ist.

Basierend auf diesen Anforderungen werden existierende Konzepte und Lösungen bezüglich der Anforderungen detailliert diskutiert. Falls diese Konzepte und Lösungen nicht adäquat sind, werden basierend auf der Diskussion Konzepte für neue, intelligente Formulare entwickelt. Diese stellen auf abstrakter Ebene Informationen zu verwendeten Oberflächenelementen, Layoutinformationen, Datenbindung sowie weitere Informationen zur Unterstützung der festgestellten Anforderungen zur Verfügung.

Anschliessend wird die Integration einer Softwarekomponente zur Unterstützung solcher Formulare für das ADEPT2-System entworfen und prototypisch implementiert. Dies beinhaltet zum Einen einen Editor zur Erstellung der Formulare inklusive der Anbindung an das entsprechende Prozessmodell. Zum Anderen wird eine Laufzeitumgebung benötigt, welche die vom Editor erzeugten Formulardaten in eine konkrete Benutzeroberfläche inklusive aller benötigten weitergehenden Funktionen transformiert. Diese kann dann dem Endanwender angezeigt werden. Wichtig hierbei ist die einfache Austauschbarkeit der verwendeten Oberflächenbibliothek.

## 1.3 Aufbau

Zunächst werden in Kapitel 2 die Grundlagen behandelt, die im weiteren Verlauf der Arbeit für das Verständnis nötig sind. Dazu zählen allgemeine Grundlagen über die Erstellung von Prozessen mit BPM-Systemen und eine Einführung von Formularen, die bereits in der ADEPT2-Implementierung

vorhanden sind. Ausserdem werden technische Grundlagen besprochen, wie die Verwendung von JavaScript [ISO02] und SWT (Standard Widget Toolkit [Ecl09a]).

In Kapitel 3 geht es um die Anforderungen, die an intelligente Formulare gestellt werden. Dafür werden aufbauend auf der Motivation zunächst die grundlegenden Probleme von Formularen erläutert. Anschliessend werden die beteiligten Endanwender des Systems identifiziert und deren Anforderungen an das System ausführlich diskutiert. Diese werden zusammen mit den allgemeinen Anforderungen in funktionale und nichtfunktionale Anforderungen eingeteilt.

Nach der Vorstellung der Anforderungen werden in Kapitel 4 darauf basierend Kriterien zusammengestellt, welche für eine Evaluierung bereits existierender Produkte notwendig sind. Dabei wird geprüft, ob es Produkte gibt, welche die gestellten Anforderungen bereits erfüllen. Diese Evaluierung befasst sich mit vollständigen BPM-Systemen ebenso, wie mit eigenständigen Programmen zur Gestalltung von Oberflächen und Formularen. Ausserdem wird eine neue Technologie zur Darstellung von Formularen in Web-Browsern untersucht.

In Kapitel 5 werden Lösungsansätze für eine eigene Entwicklung präsentiert. Zunächst wird die Architektur der neuen Softwarekomponente diskutiert. Diese beinhaltet unter anderem den Editor, mit dessen Hilfe Formulare für einen Prozessschritt erstellt werden und die Umgebung für die Darstellung dieser Formulare. Neben der Architektur werden noch andere Aspekte besprochen, wie zum Beispiel das Datenmodell der Formulare und ein geeignetes Format zum Transport von Formularen. Anschliessend werden verschiedene Konzepte für die einzelnen Aspekte der Architektur vorgestellt, welche die urspünglichen Anforderungen aus Kapitel 3 adäquat umsetzen.

Um die Arbeit abzurunden, wird die neue Softwarekomponente unter Verwendung der vorgestellten Konzepte prototypisch implementiert und in das ADEPT2-System integriert. Das Ergbnis wird in Kapitel 6 beschrieben. Dabei werden auch Probleme angesprochen, die bei der Implementierung aufgetreten sind.

Kapitel 7 fasst das Ergebnis der Arbeit abschliessend zusammen. Es wird beschrieben, was im Rahmen dieser Arbeit erreicht wurde und welche Probleme nicht gelöst werden konnten. Zuletzt wird ein Ausblick darauf gegeben, in welche Richtung sich die Entwicklung in Zukunft richten wird.

# 2 Grundlagen

Dieses Kapitel führt einige Grundlagen aus dem Softwareentwicklungs- und BPM-Bereich ein, die für das Verständnis dieser Arbeit benötigt werden. Zunächst wird das allgemeine Vorgehen beim Modellieren von Geschäftsprozessen erläutert. Darauf aufbauend werden ausgewählte Aspekte von ADEPT2 vorgestellt. Im Anschluss werden die Grundlagen von JavaScript und der Oberflächenentwicklung mit SWT [Ecl09a] diskutiert. Dieses Wissen wird für die Entwicklung der Konzepte der neuen Komponente benötigt.

## 2.1 Grundlagen von BPM-Systemen

### 2.1.1 Der Informationsbegriff

Neben der Koordinierung von Arbeitsabläufen ist vor allem auch der Transport von Informationen ein wichtiger Aspekt von BPM-Systemen. Diese Informationen werden beispielsweise durch Benutzereingaben erzeugt, durch das BPM-System verwaltet und den Endanwendern präsentiert, um diese bei ihrer Arbeit zu unterstützen.

Durch die Benutzereingaben werden zunächst jedoch nur Daten zur Verfügung gestellt. Aus diesen Daten müssen Informationen gewonnen werden. Im Verlauf dieser Arbeit spielt die Unterscheidung zwischen Daten und Informationen eine wichtige Rolle. Daher sollen diese beiden Begriff im Folgenden voneinander abgegrenzt werden.

Bei Daten handelt es sich um Zeichen oder Zeichenketten. Sie sind durch die DIN ISO/IEC 2382 [Int93] der International Organization for Standardization [ISO09] definiert. Um daraus Informationen zu gewinnen, müssen die Daten interpretiert werden. Die Daten selber geben allerdings keinen Aufschluss darüber wie diese zu interpretieren sind. Aus ihnen lassen sich daher keine Informationen gewinnen.

Um Daten in einem BPM-System interpretierbar zu machen, wird jedem Datum ein Typ zugewiesen, welcher Aufschluss darüber gibt, wie die Daten zu interpretieren sind. Daten, welche beispielsweise vom Typ String sind, lassen sich als Text interpretieren. Zu diesem Zweck gibt es in den meisten BPM-System bereits vordefinierte Datentypen für einfache Informationen, wie sie auch in vielen gebräuchlichen Programmiersprachen benutzt werden. Einige solcher Typen sind:

- String
- Integer
- Float
- Boolean

Zeichen oder Zeichenfolgen ohne einen Datentyp werden fortan als Daten bezeichnet. Zeichen oder Zeichenfolgen mit einem Datentypen werden als Informationen klassifiziert.

### 2.1.2 Benutzer eines BPM-Systems

BPM-Systeme sind häufig sehr komplex und umfangreich. Wird in einem Unternehmen ein solches System eingesetzt, sind daher in der Regel viele unterschiedliche Benutzer mit unterschiedlichen Aufgaben und speziellem Fachwissen daran beteiligt. Diese Benutzer lassen sich in drei Gruppen einteilen (Abbildung 4).

- Prozessmodellierer: Dies sind Personen, welche mit dem BPM-System Geschäftsprozesse entwickeln. Sie sind in der Regel nicht selber in die Ausführung eines Prozesses involviert.
- Anwendungsentwickler: Sie sind für die Entwicklung von Anwendungsbausteinen verantwortlich, die beispielsweise in einen Geschäftsprozess integriert werden können. Auch diese Personen sind nicht an der Ausführung eines Prozesses beteiligt.
- Endanwender: Dies sind die Personen, welche während der Ausführung eines Geschäftsprozesses in einem Unternehmen einbezogen sind, um beispielsweise Daten einzugeben, oder um Arbeitsschritte durchzuführen, welche nicht unbdeingt durch ein computergestütztes System erledigt werden können (z. B. Verpacken der Ware).

### 2.1.3 Verwendung eines BPM-Systems

Um einen Geschäftsprozess mit einem BPM-System umzusetzen, gibt es mehrere Möglichkeiten. Eine sehr verbreitete Methode ist die Erstellung eines Modells des Geschäftsprozesses (Prozessmodell) mit Hilfe eines Graphen. Es gibt verschiedene Systeme, welche diese Methode umsetzen, beispielsweise Inubit [Inu09a] oder ADEPT2 ([Rei00], [Dad09]). Da es zwischen solchen Systemen Unterschiede in den Details der Umsetzung und im Vokabular gibt, wird im

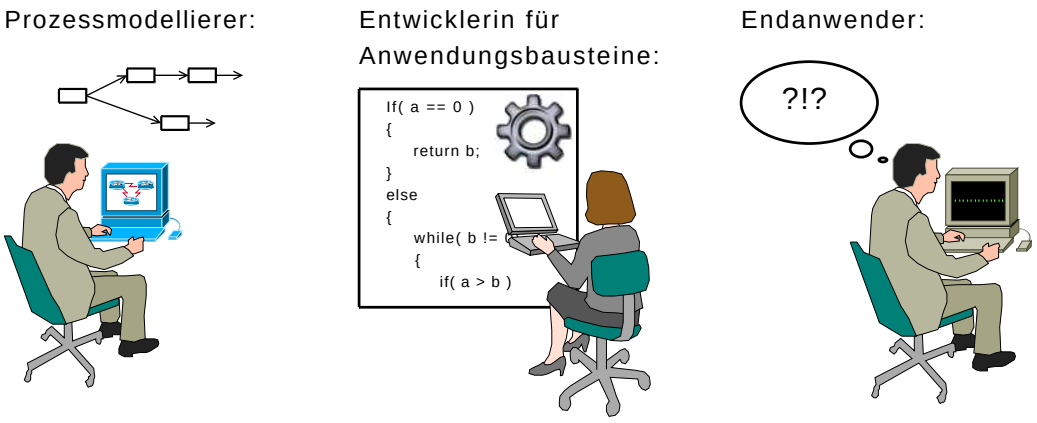


Abbildung 4: Benutzer eines BPM-Systems

Verlauf dieser Arbeit das Modell von ADEPT2 als Grundlage verwendet. Die darin enthalten Prinzipien lassen sich jedoch auch auf andere Systeme übertragen. ADEPT2 ist ein BPM-System der nächsten Generation, das auf den Forschungsarbeiten [Dad06] von Prof. Reichert [Rei00] und Prof. Dadam seit 1997 beruht und an der Universität von Ulm entwickelt wird. Bei der Umsetzung von Geschäftsprozessen mit ADEPT2 sind mehrere Aspekte zu beachten:

- Kontrollfluss
- Datenfluss
- Aktivitäten

Es gibt noch weitere Aspekte, wie beispielsweise die Behandlung von Ausnahmen, welche jedoch für diese Arbeit nicht relevant sind.

Mit dem Kontrollfluss wird die Reihenfolge der einzelnen Schritte des Geschäftsprozesses (im weiteren Verlauf Prozessschritte genannt) festgelegt. Hierzu wird zunächst der Geschäftsprozess mit Hilfe eines Graphen modelliert. Dieser Graph besitzt Knoten, welche die einzelnen Prozessschritte darstellen. Diese Prozessschritte sind wiederum durch gerichtete Kanten miteinander verbunden, welche den Kontrollfluss repräsentieren. Jeder Graph hat einen Start- und einen Endknoten. Innerhalb des Kontrollflusses sind Konstrukte wie Schleifen oder Verzweigungen möglich. Wie ein Kontrollfluss aussehen darf, ist in [Rei00] beschrieben. Abbildung 5 zeigt beispielhaft das Modell eines Prozesses.

Der Datenfluss beschreibt den Transport von Informationen innerhalb des Kontrolflusses. Hierzu werden dem Graphen zunächst sogenannte Datenele-

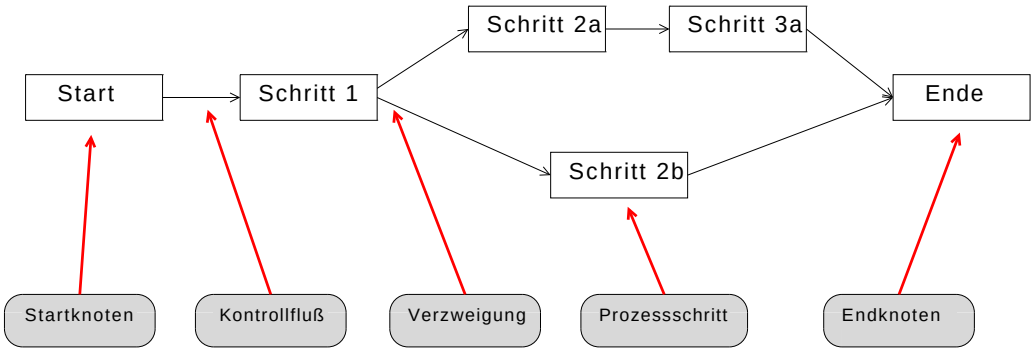


Abbildung 5: Prozessmodell mit Prozessschritten und Kontrollfluss

mente in Form weiterer Knoten hinzugefügt. Jedes Datenelement repäsentiert dabei eine bestimmte Information. Um die Informationen zwischen den Prozessschritten zu transportieren, werden die Datenelemente jedem Prozessschritt, in dem sie benötigt werden, durch eine weitere gerichtete Kante zugeordnet. Dabei spielt die Richtung der Kante eine wichtige Rolle. Eine Kante vom Prozessschritt zum Datenelement kennzeichnet einen sogenannten schreibenden Zugriff. Dies bedeutet, dass bei einem entsprechenden Prozessschritt das Datenelement mit Informationen versorgt wird. Eine Kante vom Datenelement zum Prozessschritt bedeutet, dass die Informationen aus dem Datenelement in dem Prozessschritt verwendet werden (lesender Zugriff). Weitere Informationen zu Datenelementen sind in [Rei06] zu finden. In Abbildung 6 wird das Modell aus Abbildung 5 um den Datenfluss erweitert.

Ein weiterer wichtiger Aspekt sind die Aktivitäten. Diese beschreiben die Semantiken eines Prozessschrittes und den zugeordneten Datenelementen.

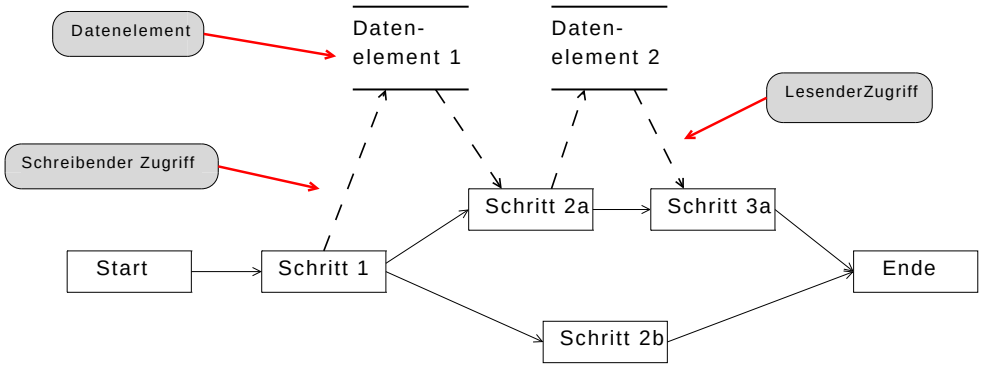


Abbildung 6: Prozessmodell mit Datenfluss

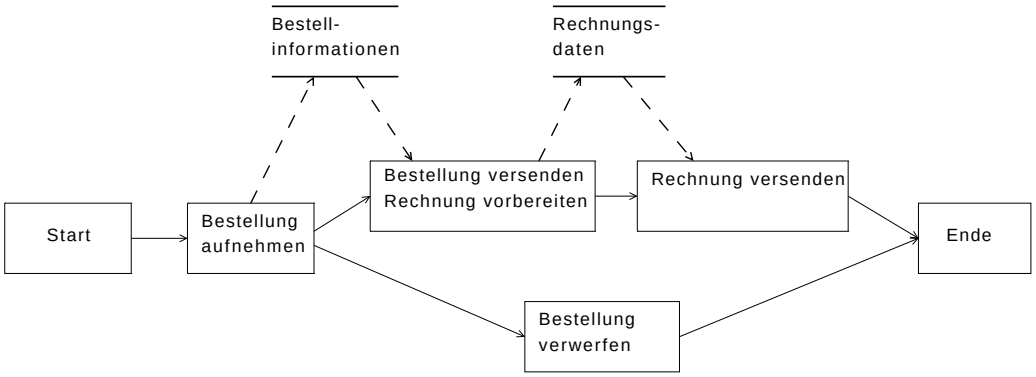


Abbildung 7: Prozessmodell mit Aktivitäten

Eine Aktivität definiert, was während der Ausführung eines Prozessschrittes geschehen soll, beispielsweise mit einer Anwendung, welche dem Prozessschritt hinterlegt wird. Anwendungen, die als Aktivität in einen Prozess integriert sind, werden in dieser Arbeit als Anwendungsbausteine bezeichnet, um diese von alleinstehenden Anwendungen abzugrenzen. Die Datenelemente der Prozessschritte dienen den Aktivitäten dabei als Parameter. Im weiteren Verlauf der Arbeit werden Datenelemente daher als Prozessschrittparameter bezeichnet. Lesender Zugriff auf Datenelemente sind Eingabeparameter der Aktivität. Das bedeutet, dass die Informationen aus den Datenelementen zu Beginn der Ausführung einer Aktivität dieser zur Verfügung gestellt wird, um damit arbeiten zu können. Ein schreibender Zugriff auf ein Datenelement bedeutet, dass die Aktivität nach der Ausführung Informationen für das Datenelement bereitstellt. Abbildung 7 zeigt ein Prozessmodell mit Aktivitäten.

Ein auf diese Weise erstelltes Modell eines Geschäftsprozesses kann nun in einer Ausführungsumgebung instanziiert und ausgeführt werden. Diese Umgebung startet zunächst die Aktivität des ersten Prozessschrittes. Nachdem die Aktivität beendet wurde, wird gemäß dem Kontrollfluss der nächste Prozessschritt ermittelt und dessen Aktivität gestartet. Die Ausführung des Modells ist beendet, wenn der Endknoten erreicht wird. Bei der Ausführung des Modells müssen nicht alle Prozessschritte durchlaufen werden. Durch bedingte Verzweigungen können Bereiche des Graphen ausgelassen werden. Abbildung 8 zeigt die Ausführung einer Prozessinstanz.

In ADEPT2 ist eine Entwicklungsumgebung (*Process Template Editor*) integriert. Mit diesem Editor ist der Prozessmodellierer in der Lage, alle beschriebenen Aspkete der Erstellung eines Prozessmodells durchzuführen. Abbildung 9 zeigt den *Process Template Editor*. Das Modellieren erfolgt

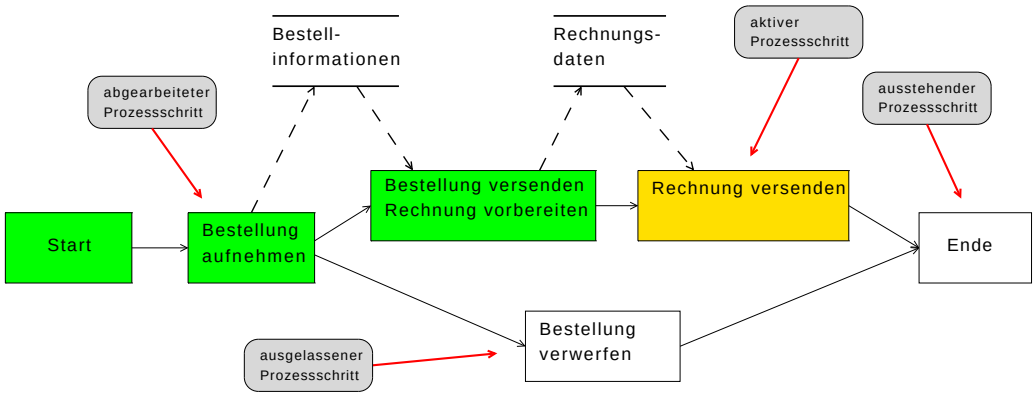


Abbildung 8: Instanz eines Prozessmodells

dabei mit Hilfe von Änderungsoperationen, welche den Graphen immer in einem gültigen Zustand halten. Die Funktionsweise dieser Operationen ist in [Jur06] beschrieben. Weitere Informationen zu dem Editor werden in den entsprechenden Abschnitten dieser Arbeit erläutert.

## 2.2 Unterstützung von Formularen bei ADEPT2

Sehr häufig werden für Prozessschritte zwei Arten von Aktivitäten verwendet:

- Aktivitäten zur Interaktion mit einem Endanwender (Benutzerinteraktion), um beispielsweise Informationen von diesem zu erfragen.
- Automatische Aktivitäten zur Durchführung von datenverarbeitenden Schritten.

Für Aktivitäten mit Benutzerinteraktionen gibt es in ADEPT2 bereits eine Softwarekomponente, welche hierfür einfache Formulare bereitstellt. Diese werden automatisch auf Basis der Prozessschrittparameter eines Formulares erstellt. Das bedeutet, dass für das Formular zum Einen Felder generiert werden, welche die Eingabeparameter anzeigen. Zum Anderen werden Eingabefelder generiert, welche zur Laufzeit durch den Endanwender ausgefüllt werden. Dessen Eingaben stellen die Ausgabeparameter der Aktivität dar. Allerdings ergeben sich durch die Automatisierung auch einige Einschränkungen:

1. Auf das Layout des Formulars kann kein Einfluss genommen werden. Für jeden Parameter wird ein entsprechendes Formularfeld erstellt. Diese Felder werden alphabetisch aufgelistet.

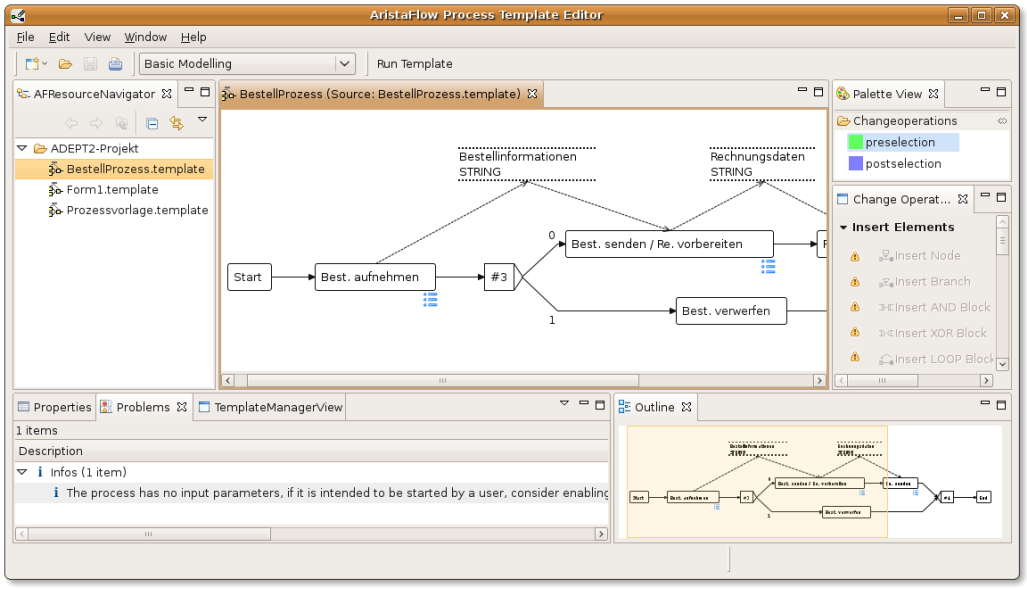


Abbildung 9: *Process Template Editor* von ADEPT2

2. Es ist nicht möglich, das Verhalten der Formulare durch Anwendungslogik flexibel zu gestalten. Auf semantisch falsche Eingaben durch den Endanwender kann nicht reagiert werden, beispielsweise bei der Eingabe einer negativen Zahl in einem Formularfeld für eine Altersangabe.

In Abbildung 10 ist ein Modell eines Prozesses zu sehen, mit dem ein Urlaubsantrag erstellt und genehmigt werden kann. Der Prozess besteht aus zwei Schritten. Die Aktivität des zweiten Schrittes hat dabei drei Eingabeparameter, welche die Daten des Antragsstellers beinhalten und zwei Ausgabeparameter, welche durch einen Antragsprüfer angegeben werden müssen. Diese Ausgabeparameter beinhalten die Entscheidung über die Genehmigung des Urlaubs und gegebenenfalls eine Begründung. Abbildung 11 zeigt das automatisch generierte Formular für die zweite Aktivität.

## 2.3 Anwendungsintegration

Für Prozesse, bei denen diese Formulare keine ausreichende Funktionalität bieten, müssen neue Anwendungsbausteine entwickelt und als Aktivitäten in das BPM-System integriert werden. Im Folgenden Abschnitt wird der Aufbau der Ausführungsumgebung von ADEPT2 besprochen, welche integrierte Anwendungsbausteine zur Laufzeit eines Prozesses ausführt.

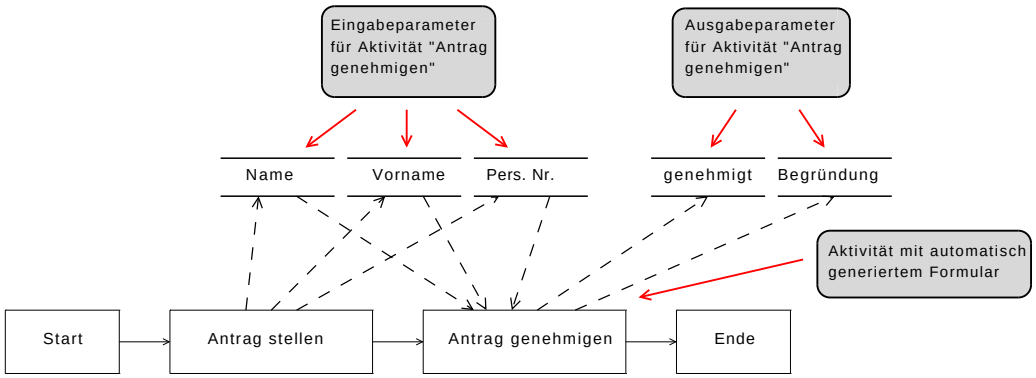


Abbildung 10: Prozessmodell für einen Urlaubsantrag mit Formularaktivitäten

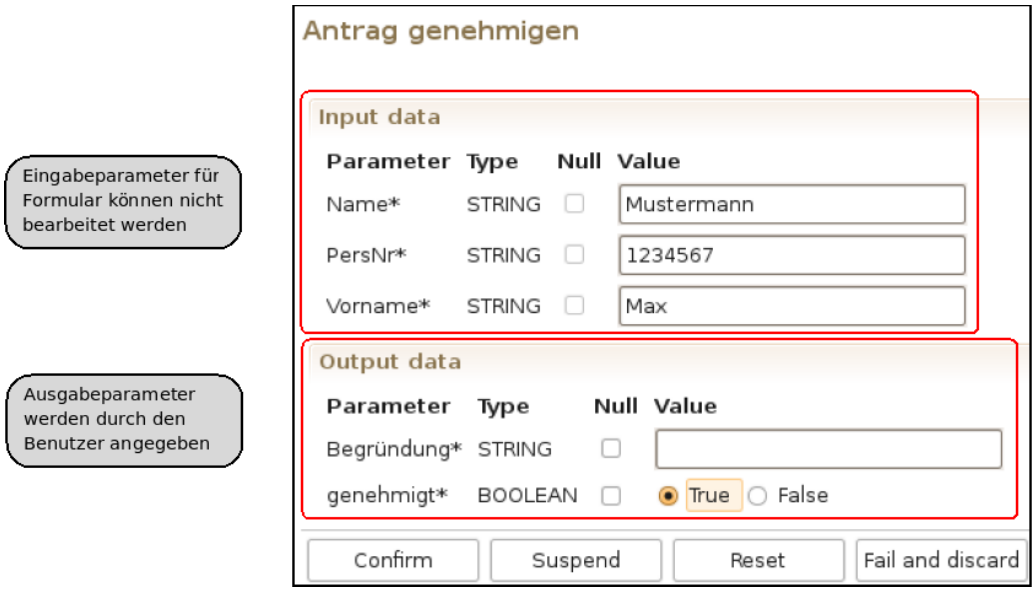


Abbildung 11: Generiertes Formular für eine Aktivität

Ausgangspunkt der Entwicklung eines neuen Anwendungsbausteines für das System ist die Schnittstelle *ExecutableComponent*. Diese Schnittstelle beschreibt die Methoden, welche von dem Baustein bereitgestellt werden müssen, damit diese in der Ausführungsumgebung verwendet werden können (Quellcode 1).

Um den erstellten Anwendungsbaustein innerhalb des *Process Template Editor* als Aktivität nutzen zu können, muss für die Implementierung der *ExecutableComponent* noch eine *ExecutableComponentDescription* angelegt wer-

den. Diese Beschreibung enthält Metainformationen zu der neuen Komponente. Beispielsweise den Klassenpfad und den Klassennamen. Angaben über benötigte Prozessschrittparameter sind ebenfalls möglich.

```
1  public interface ExecutableComponent {
2    public void init(SessionContext sessionContext);
3    public void initResume(int savePointID, SessionContext
       sessionContext);
4    public void run();
5    public boolean close();
6    public boolean reset();
7    public boolean signal(int signal);
8    public boolean suspend();
9    public boolean kill();
10   public Vote prepareCommit() throws
       IsNotSupportedException;
11   public boolean rollback() throws
       IsNotSupportedException;
12   public void commit() throws IsNotSupportedException;
13 }
```

Quellcode 1: Schnittstelle der *ExecutableComponent*

Zur Laufzeit führt das BPM-System den Prozess regulär aus. Dabei werden die Aktivitäten der Prozessschritte gestartet. Enthält eine Aktivität dabei eine integrierte Anwendung, wird diese gestartet und die Werte der Eingabeparameter übergeben. Die Ausführung des Prozesses wird unterbrochen, bis die Anwendung beendet ist und die Werte für die Ausgabeparameter bereitstellt. Diese werden in die entsprechenden Datenelemente geschrieben, bevor mit der Ausführung des Prozesses fortgefahren wird. Dieser Vorgang ist in Abbildung 12 nochmals dargestellt.

## 2.4 JavaScript

Im späteren Verlauf dieser Arbeit wird verstärkt auf JavaScript für den Einsatz in Formularen zurückgegriffen. Daher werden nun die wichtigsten Grundlagen dieser Sprache erläutert.

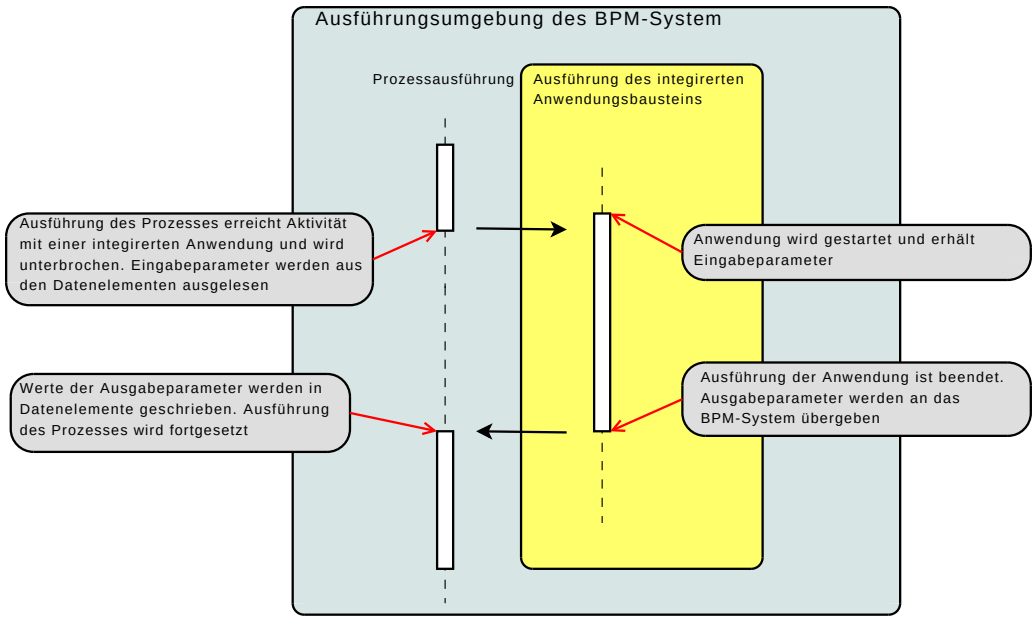


Abbildung 12: Integration eines Anwendungsbausteins zur Laufzeit

### 2.4.1 Allgemeine Einführung

JavaScript ist eine prozedurale Skriptsprache, welche 1995 für den Netscape Navigator entwickelt und später unter dem Namen ECMAScript [ISO02] standardisiert wurde. Mittlerweile beherrschen aber die meisten gängigen Web-Browser diese Sprache. Die Web-Browser stellen eine spezielle Laufzeitumgebung zur Verfügung, innerhalb der ein Skript interpretiert wird. Das Skript selber wird dabei in ein HTML-Dokument eingebettet oder in einer separaten JavaScript-Datei zur Verfügung gestellt. Innerhalb eines solchen Skriptes hat der Entwickler Zugriff auf die Inhalte des HTML-Dokumentes und des Web-Browser, wie beispielsweise Texte, Bilder, Formularelemente, Links aber auch auf Cookies, Fenster des Web-Browser oder die Liste der besuchten Webinhalte. Ein Zugriff auf Ressourcen ausserhalb des Web-Browser ist nicht möglich.

```
1 <html><head><title>Test</title>
2 <script type="text/javascript">
3 <!--
4 function Quadrat()
5 {
6    var Ergebnis = document.Formular.Eingabe.value *
        document.Formular.Eingabe.value;
```

```
7     alert("Das␣Quadrat␣von␣" + document.Formular.Eingabe.
          value + "␣=␣" + Ergebnis);
8  }
9  //-->
10 </script></head>
11 <body>
12    <form name="Formular" action="">
13       <input type="text" name="Eingabe" size="3">
14       <input type="button" value="Quadrat␣errechnen"
             onclick="Quadrat()">
15    </form>
16 </body></html>
```

Quellcode 2: JavaScript eingebettet in HTML [Mü08a]

Die Ausführung der Skripte basiert häufig auf Ereignissen, die beispielsweise durch Benutzeraktionen im HTML-Dokument ausgelöst werden. Quellcode 2 zeigt ein einfaches JavaScript-Programm, welches in ein HTML-Dokument eingebettet ist und durch das Aktivieren einer Schaltfläche ausgeführt wird. In Abbildung 13 ist ein Web-Browser zu sehen, welcher das HTML-Dokument darstellt.

Ein wichtiges Konzept von JavaScript ist die Objektorientierung [Joh95]. Dies bedeutet, dass alle Zugriffe, beispielsweise auf das HTML-Dokument, über entsprechende Objekte erfolgen. Wie in vielen anderen objektorientierten Programmiersprachen auch, besitzen Objekte in JavaScript Attribute und Methoden.


Abbildung 13: Ausführung von JavaScript in einem Web-Browser

Das Konzept der Klassen gibt es in JavaScript dagegen nicht. Statt dessen werden Prototypen verwendet. Um eigene Objekte mit individuellen Eigenschaften und Methoden zu erhalten, werden Objekte zunächst kopiert. Diesen Kopien wiederum werden die benötigten Eigenschaften und Methoden hinzugefügt. Auf diese Weise ist auch Vererbung möglich, da die Methoden und Eigenschaften des kopierten Objektes auch in dem neuen Objekt verfügbar sind, wenn dies gewünscht ist. Dazu wird das kopierte Objekt als Prototyp des neuen Objektes deklariert.

Eine weitere Eigenschaft von JavaScript ist die dynamische Typisierung. Dadurch werden Variablen erst zur Laufzeit an einen bestimmten Typ gebunden. Zur Entwurfszeit ist nicht bekannt, welchen Typ eine Variable hat. Es ist jedoch möglich zur Laufzeit den Typ einer Variable zu ermitteln und diese Typinformation im Programmfluss zu verwenden.

Aufgrund der steigenden Beliebtheit von Web 2.0-Anwendungen [Alb07], wird JavaScript immer häufiger dazu eingesetzt, den Aufbau des HTML-Dokumentes dynamisch zu verändern. Damit ist es möglich, während der Benutzer mit dem Dokument arbeitet, neue Textfelder einzufügen, zu entfernen oder zu ändern. Durch diese Möglichkeit lassen sich HTML-Dokumente nicht mehr nur für das Präsentieren von Informationen einsetzen. Es ist möglich damit komplette, auf HTML basierte Anwendungen mit einer aufwendigen Benutzerinteraktion zu entwickeln.

### 2.4.2 JavaScript ausserhalb eines Web-Browsers

Die Verwendung von JavaScript ist nicht auf den Kontext eines Web-Browsers beschränkt. JavaScript kann in jeder Anwendung genutzt werden, die in der Lage ist, diese Sprache zu interpretieren. Eine entsprechende Laufzeitumgebung zu erstellen ist dabei recht aufwändig. Es gibt jedoch vorgefertigte Umgebungen, welche in Anwendungen integriert werden können, um JavaScript-Programme innerhalb dieser Anwendung zu interpretieren. Eine Möglichkeit hierfür bietet die *ScriptEngine* [Ull09, Kapitel 10.6.3], welche in Java 1.6 zur Verfügung steht. Diese ScripteEngine erlaubt die Ausführung von JavaScript im Kontext eines Java-Programms.

Dabei ist zu beachten, dass sich die Kontexte grundlegend unterscheiden: Im Kontext eines Web-Browsers kann beispielsweise mit Hilfe des *document*-Objekt, auf den Inhalt des HTML-Dokument zugegriffen werden. Dies ist im Kontext eines Java-Programms nicht möglich.

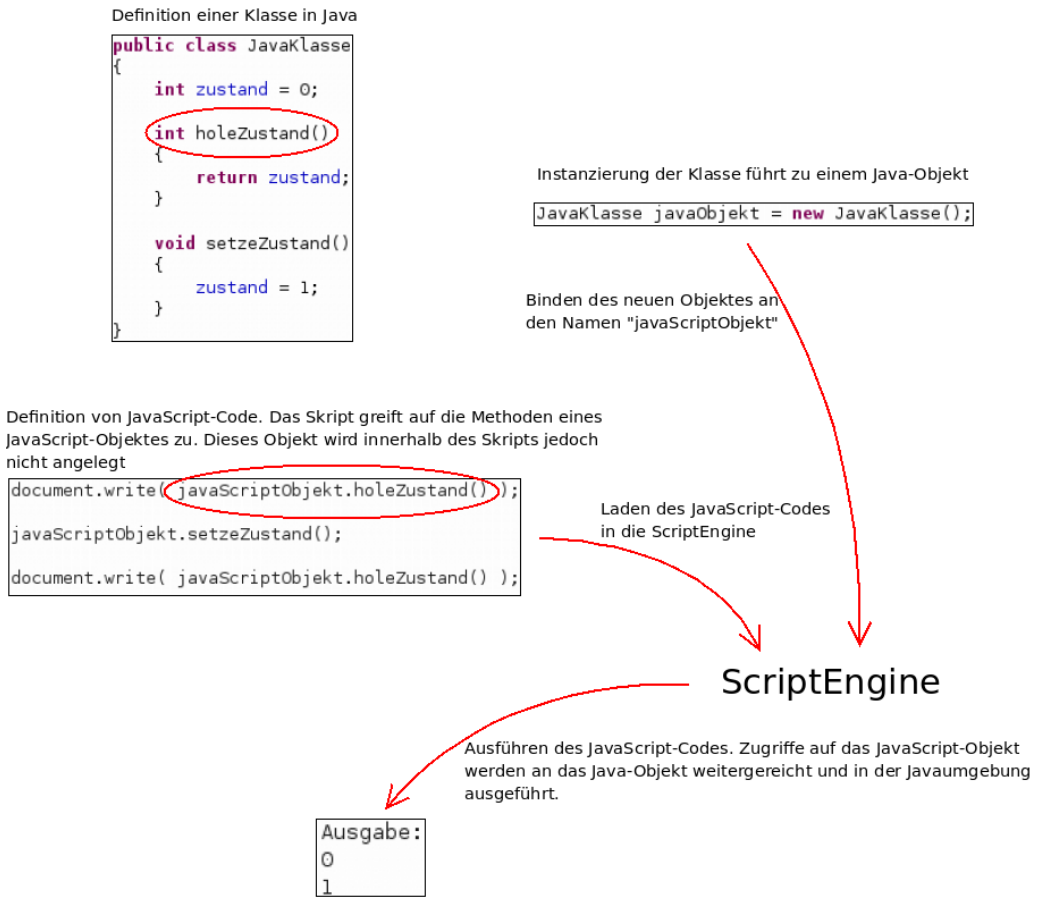


Abbildung 14: Funktionsweise der ScriptEngine

Es ist jedoch möglich, innerhalb der JavaScript-Programme auf ausgewählte Objekte aus dem Kontext des laufenden Java-Programms zuzugreifen. Hierfür wird der *ScriptEngine* der Quellcode des JavaScript-Programms übergeben. Zusätzlich können Objekte aus dem Java-Kontext übergeben werden. Diese Objekte werden dabei an Namen gebunden, mit denen innerhalb des JavaScript-Programms auf die Objekte zugegriffen werden kann. Dabei können alle Attribute und Methoden des Java-Objektes verwendet werden. Abbildung 14 zeigt die Verwendung der *ScriptEngine*.

Wenn während der Ausführung des JavaScript-Programms eine Methode eines Objektes aufgerufen wird, welches an ein Java-Objekt gebunden ist, wird dieser Aufruf durch die *ScriptEngine* an die Methode des Java-Objektes ausserhalb des JavaScript-Programms weitergereicht und dort ausgeführt. Für das JavaScript-Programm ist dieser Vorgang völlig transparent.

## 2.5 Grafische Kontexte

Wenn im Rahmen eines BPM-Systems grafische Oberflächen angezeigt werden sollen spielt der grafische Kontext eine wichtige Rolle, da nicht jede Oberfläche auf jedem PC dargestellt werden kann. Warum dies so ist, wird im Folgenden Abschnitt erläutert. Weiterführend wird die Entwicklung von Oberflächen mit Hilfe von SWT beschrieben.

### 2.5.1 Allgemeines

In großen Unternehmen gibt es eine Vielzahl von Mitarbeitern an unterschiedlichen Arbeitsplätzen. Diese Arbeitsplätze bieten nicht alle das selbe Umfeld für den Einsatz eines BPM-Systems. Beispielsweise kann sich bereits das verwendete Betriebssystem an jedem Arbeitsplatz unterscheiden. Einige Mitarbeiter können Mac OS X [App07] an ihrem Desktop PC einsetzen, während andere Palm OS [Pal09] auf ihren mobilen PDA einsetzen. Damit dennoch alle Mitarbeiter in einen Geschäftsprozess mit eingebunden werden können, muss sich ein BPM-System an unterschiedliche Systeme anpassen können. Die wichtigste Aufgabe des BPM-Systems für den Mitarbeiter ist die Interaktion mit dem Endanwender. Diese erfolgt meist durch die Verwendung einer grafischen Oberfläche. Fast alle Betriebssystem stellen hierfür eine oder mehrere Möglichkeiten zu deren Darstellung zur Verfügung. Diese Möglichkeiten werden im Verlauf dieser Arbeit als *grafische Kontexte* bezeichnet. Jeder grafischer Kontext hat dabei bestimmte Voraussetzungen an das Betriebssystem, um diesen nutzen zu können.

Daher bieten nicht alle Betriebssysteme die selben grafischen Kontexte an. Das BPM-System muss deshalb in der Lage sein, mit unterschiedlichen grafischen Kontexten umgehen zu können. Ein Beispiel für einen grafischen Kontext ist eine HTML-Umgebung. Hiermit werden die Oberflächen zur Benutzerinteraktion mit Hilfe eines HTML-Dokumentes dargestellt. Dies setzt jedoch voraus, dass das Betriebssystem eine Anwendung zur Verfügung stellt, um HTML-Dokument interpretieren zu können. Eine weitere Möglichkeit für einen grafischen Kontext ist die Verwendung von PDF-Dokumenten. Auch hier wird eine eigene Anwendung zur Darstellung vorausgesetzt.

### 2.5.2 Oberflächenentwicklung mit SWT

Eine andere Möglichkeit für einen grafischen Kontext sind Bibliotheken, welche grafische Komponenten für die Entwicklung einer Oberfläche be-

reitstellen. Diese Bibliotheken werden meist in sogenannten Rahmenwerken angeboten. Beispiele für Rahmenwerke sind [Wik09a]:

- AWT: Abstract Window Toolkit für Java [Sun09a]
- GTK: Gimp Toolkit für verschiedene Plattformen [GTK09]
- MFC: Microsoft Foundation Classes für ältere Windowssystme [Mic09a]
- Motif: Älteres Rahmenwerk für Posix-Systeme [Ope09]
- QT: Toolkit von Trolltech für verschiedene Plattformen [Tro09]
- Swing: Erweiterung von AWT [Sun09b]
- SWT: Standard Widget Toolkit aus dem Eclipse-Projekt [Ecl09a]
- Windows Forms: Rahmenwerk für .NET-Umgebungen [Mic09f]
- wxWidgets: Objektorientiertes Rahmenwerk für verschiedene Plattformen [WX09]

Welche dieser Rahmenwerke verwendet werden können, ist dabei nicht nur vom verwendeten Betriebssystems abhängig, sondern auch von der eingesetzten Programmiersprache. AWT und Swing sind für den Einsatz in Java-Programmen entwickelt und betriebssystemunabhängig. SWT ist ebenfalls für Java entwickelt, dessen Umsetzung unterscheidet sich allerdings je nach Betriebssystem. Da die Implementierungen der grafischen Oberflächen von ADEPT2 SWT verwenden, wird auch im Rahmen dieser Arbeit auf SWT für die Entwicklung der Oberflächen zurückgegriffen. SWT ist ein quelloffenes Rahmenwerk, dass entwickelt wurde, um einen performanten und portablen Zugriff auf die Bibliotheken für Benutzerschnittstellen des Betriebssystems zu gewähren. SWT greift dabei auf andere existierende Rahmenwerke zurück, welche auf dem jeweiligen Betriebssystem verfügbar sind. Auf einem GNU/Linux System wird beispielsweis GTK verwendet. Die Einbindung in die Java-Umgebung erfolgt dabei durch das *Java Native Interface* [Ull09, Kapitel 27], wodurch diese Kapselung vor dem Anwendungsentwickler verborgen wird.

Die grafischen Benutzeroberflächen sind in SWT aus *Composite* aufgebaut. Ein *Composite* ist eine Repäsentation für ein Steuerelement, welches zur Darstellung von Informationen oder zur Interaktion mit dem Benutzer verwendet wird. Die Bibliotheken von SWT enthalten *Composite* für die gebräuchlichsten Steuerelemente, wie beispielsweise Textfelder, Beschriftungsfelder, Schaltflächen, Listen, Tabellen oder Baumansichten. Für komplexere

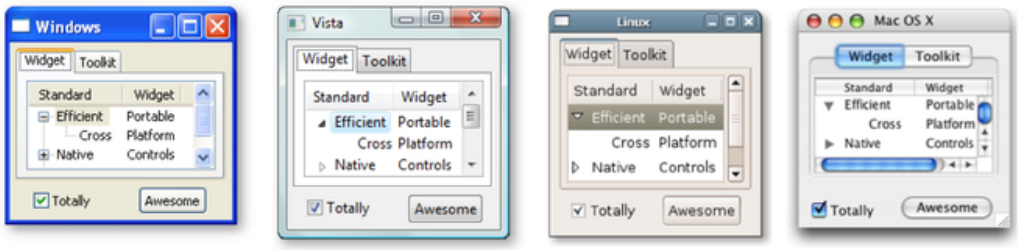


Abbildung 15: Einige *Composite* von SWT in unterschiedlichen Betriebssystemen [Ecl09a]

Steuerelemente ist es möglich, neue *Composite* anzulegen. Diese werden dabei hierarchisch aus bereits vorhandenen *Composite* aufgebaut. Abbildung 15 zeigt eine Oberfläche bestehend aus einem Register, einer Baumansicht, einem Kontrollkästchen und einer Schaltfläche. Diese Oberfläche wird in unterschiedlichen Betriebssystemen dargestellt.

Um die *Composite* anzuordnen, wird dem übergeordneten *Composite* ein *Layout* [Ecl09b] zugeordnet. Dieses *Layout* ist für die Positionierung der *Composite* verantwortlich. Hierfür gibt es mehrere Implementierungen mit einem unterschiedlichen Verhalten. Alternativ können die Positionsangaben auch manuell gesetzt werden.

Um auf Aktionen von Benutzern reagieren zu können, verwendet SWT eine Variante des *Observer-Patterns* [Joh95]. Alle *Composite* können verschiedene Ereignisse auslösen. Um über ein bestimmtes Ereignis informiert zu werden, können sich interessierte Objekte an dem entsprechendem *Composite* für das Ereignis registrieren. Diese Objekte müssen dafür jedoch eine bestimmte Schnittstelle implementieren, welche Methoden enthält, die beim Eintreten des Ereignisses aufgerufen werden.


Abbildung 16: Darstellung des SWT-*Composite*

Quellcode 3 zeigt eine Klasse, welche ein *Composite* mit einem Beschriftungsfeld, einem Textfeld und einer Schaltfläche definiert. Durch das Auslösen der Schaltfläche wird das Quadrat der Zahl berechnet, die in dem Textfeld angegeben wurde. Quellcode 4 demonstriert die Anwendung des neuen *Composite*. Abbildung 16 zeigt dessen Darstellung.

```
1  public class NeuesComposite extends Composite implements
       SelectionListener {
2    Text textFeld;
3    Label beschriftungsFeld;;
4    Button schaltflaeche;
5
6    public NeuesComposite(Composite parent, int style) {
7      super(parent, style);
8      setLayout(new GridLayout(3, false));
9      beschriftungsFeld = new Label(this, SWT.None);
10     beschriftungsFeld.setLayoutData(new GridData(GridData
         .BEGINNING, GridData.BEGINNING, false, false));
11     beschriftungsFeld.setText("Eine␣Zahl␣eingeben");
12     textFeld = new Text(this, SWT.BORDER);
13     textFeld.setLayoutData(new GridData(GridData.
         BEGINNING, GridData.BEGINNING, false, false,
14     2, 1));
15     schaltflaeche = new Button(this, SWT.PUSH);
16     schaltflaeche.setLayoutData(new GridData(GridData.
         BEGINNING, GridData.BEGINNING, false, false));
17     schaltflaeche.setText("Quadrat␣berechnen");
18     schaltflaeche.addSelectionListener(this);
19   }
20
21   public void widgetSelected(SelectionEvent e) {
22     int num = Integer.parseInt(textFeld.getText());
23     MessageBox mb = new MessageBox(getShell(), SWT.OK);
24     mb.setText("Berechnung");
25     mb.setMessage("Das␣Quadrat␣ist␣" + Integer.toString(
         num * num));
26     mb.open();
27   }
28 }
```

Quellcode 3: Benutzerdefiniertes Steuerelement als SWT-Composite

```
1  public static void main(String[] args) {
2    Display display = new Display ();
3    Shell shell = new Shell(display);
4    shell.setLayout(new FillLayout());
5    Composite composite = new NeuesComposite(shell, SWT.
       None);
6    shell.pack();
7    shell.open();
8
9    while(!shell.isDisposed()) {
10     if(!display.readAndDispatch()) {
11         display.sleep();
12     }
13   }
14   display.dispose ();
15 }
```

Quellcode 4: Verwendung eines *Composite*

# 3 Anforderungen an eine Komponente zur Formularerstellung

Aufbauend auf den Grundlagen über BPM-Systeme, werden die Anforderungen diskutiert, die sich für den Entwurf einer neuen Softwarekomponente (im Folgenden als Komponente bezeichnet) zur Erstellung von Formularen ergeben. Dabei spielt der bereits besprochene Konflikt zwischen der Fähigkeit, Formulare automatisch zu generieren und einer ausreichenden Funktionalität eine Rolle. Es gibt aber noch eine Vielzahl weiterer Aspekte, welche berücksichtigt werden müssen. Im folgenden Abschnitt wird zunächst auf die grundlegenden Problemstellungen der Generierung und Darstellung von Formularen eingegangen. Im darauf folgenden Abschnitt werden die genauen Anforderungen im Detail diskutiert.

## 3.1 Problemstellung

Der Endanwender soll durch Formulare bei seiner Arbeit mit einem BPM-System unterstützt werden. Die Formulare sollen dabei sinnvoll auf die Eingaben des Endanwenders reagieren können, um ihn beispielsweise auf Fehler aufmerksam zu machen oder die eingegebenen Daten aufzubereiten. Die bisher eingesetzten Formulare, beispielsweise von ADEPT2, sind hierfür aufgrund der fehlenden Anwendungslogik nicht geeignet. Um diese Ansprüche erfüllen zu können werden intelligente Formulare benötigt. Dabei handelt es sich um Formulare, deren Verhalten durch Anwendungslogik präzise gesteuert werden kann.

Um das Problem besser zu verstehen, wird ausgehend von dem Modell eines Geschäftsprozesses erläutert, wie intelligente Formulare in das Prozessmodell integriert und dem Endanwender bereitgestellt werden. Dazu wird das Vorgehen zum Erstellen von Prozessmodellen aus Kapitel 2 erweitert.

Zunächst wird wie bisher ein Modell des Geschäftsprozesses durch den Prozessmodellierer erstellt. Dieser legt Prozessschritte und Parametern an. Beim Zuordnen von Aktivitäten kommt eine Unterscheidung hinzu. Hier gibt es drei mögliche Arten von Aktivitäten:

1. Aktivitäten für datenverarbeitende Prozessschritte
2. Aktivitäten zur Interaktion mit dem Endanwender ohne zusätzliche Anwendungslogik

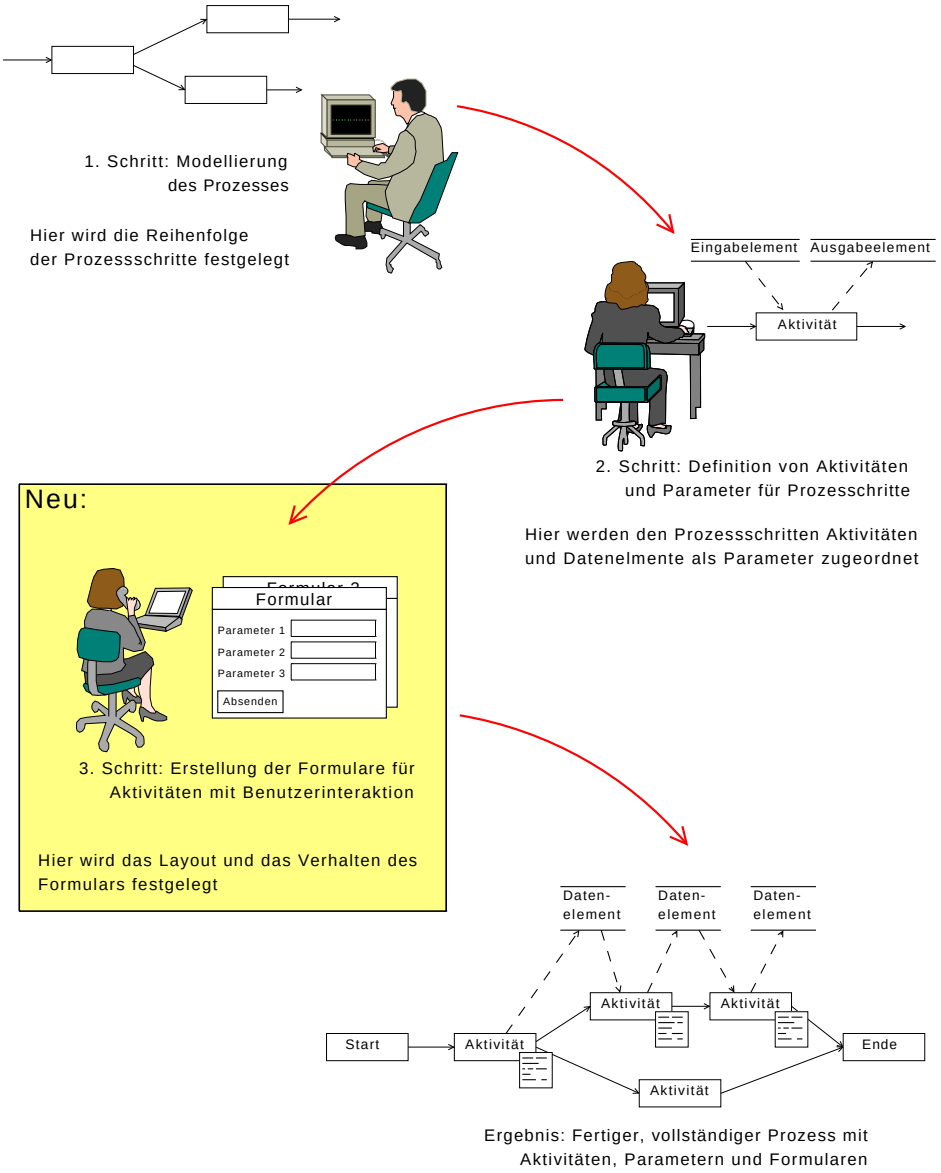


Abbildung 17: Entwurf des Prozesses mit Prozessschritten, Aktivitäten Parametern und Formularen

3. Aktivitäten zur Interaktion mit dem Endanwender mit zusätzlicher Anwendungslogik

Die ersten beiden Arten sind für den weiteren Verlauf dieser Arbeit nicht weiter interessant. Sie wurden bereits in Kapitel 2 erläutert. Bei Aktivitäten für Benutzerinteraktion mit zusätzlicher Anwendungslogik weicht das weitere

Verfahren jedoch ab. Bei einer solchen Aktivität kann das Formular nicht automatisch durch das BPM-System erstellt werden. Es sind zusätzliche Informationen nötig, um beispielsweise die Semantik der Anwendungslogik oder das Layout festzulegen.

Hierfür muss das bereits bestehende BPM-System um eine Komponente erweitert werden, welche die Erstellung eines intelligenten Formulars ermöglicht. Um die Verantwortlichkeiten während des Erstellvorgangs eines Prozessmodells suber zu trennen wird im folgenden davon ausgegangen, dass für die Erstellung des Formulars nicht der Prozessmodellierer, sondern ein eigener Formularentwickler zuständig ist. Die Erweiterung des Vorgangs zur Erstellung eines Prozessmodells wird in Abbildung 17 dargestellt. In dieser Abbildung wird ein fester Ablauf bei der Erstellung des Prozessmodells vorausgesetzt. Auch wenn die Reihenfolge beliebig ist, soll für den weiteren Verlauf dieser Arbeit die in der Abbildung verwendeten Reihenfolge beibehalten werden.

Nachdem der Prozess mit seinen Schritten, Aktivitäten und Formularen fertiggestellt ist, kann dieser durch die Ausführungsumgebung gestartet werden. Die einzelnen Prozessschritte werden dabei wie gehabt der Reihe nach abgearbeitet. Sobald ein Schritt erreicht wird, für den ein Formular entworfen wurde, wird das zugeordnete Formular dem Endanwender übergeben. An dieser Stelle muss das bestehende System ebenfalls erweitert werden, da dieses noch nicht in der Lage ist intelligente Formulare darzustellen. Nach der Bearbeitung des Formulars durch den Endanwender, kann das System aus dessen Eingaben die Ausgabeparameter bestimmen und mit der Ausführung des Prozesses fortfahren. Abbildung 18 verdeutlicht das Vorgehen.

Das Ziel ist im Folgenden herauszufinden, wie diese Komponente aufgebaut sein muss und wie deren Arbeitsablauf im Detail aussieht. Aus dem eben beschriebenen Ablauf lassen sich bereits einige Rahmenbedingungen ableiten:

- Die zu entwickelnde Komponente wird kein eigenständiges Produkt. Sie ist abhängig von einem bereits vorhandenen BPM-System und wird immer in dessen Kontext ausgeführt werden.

- Der Ablauf vom Erstellen des Prozesses bis zum fertigen, durch den Endanwender abgearbeiteten Formular ist in zwei Phasen unterteilt. Die erste Phase ist die Entwurfszeit, zu der der Prozess, die Prozessschritte, die dazugehörigen Parameter und Aktivitäten, sowie die Formulare entworfen werden (Abbildung 17). Die zweite Phase ist die Laufzeit, in welcher der Prozess gestartet und die einzelnen Prozess-

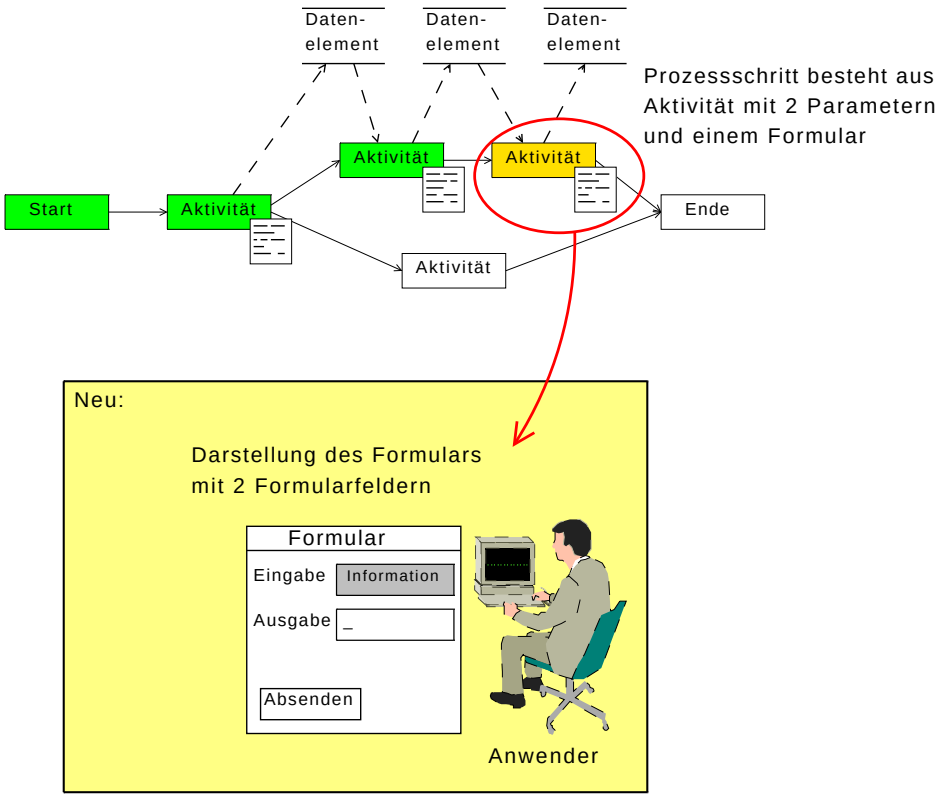


Abbildung 18: Ausführung eines Prozesses mit Formularen

schritte vom System abgearbeitet werden. Die Formulare werden zur Bearbeitung an die Endanwender verteilt (Abbildung 18).

Um mehr über die Anforderungen der Komponente zu erfahren, ist eine tiefergehende Analyse der beiden angeprochenen Phasen notwendig.

Während der Entwurfszeit wird ein Formular ausgehend von den Parametern des Prozessschrittes erstellt. Hiefür werden Formularfelder angelegt und mit den Parametern verknüpft. Die Parameter sind dabei an den Prozess gebunden und können daher nur innerhalb des Prozesses geschrieben oder gelesen werden. Mit welchen Werten die Parameter versorgt werden, wird daher nur von den Aktivitäten bestimmt, denen diese Parameter zugeordnet sind. Da das Verhalten von Aktivitäten jedoch dynamisch ist, lassen sich zur Entwurfstzeit keine Aussagen über die Werte der Parameter machen.

Dies bedeutet, dass zur Entwurfszeit nur ein abstraktes Formular erstellt werden kann. Um daraus ein konkretes Formulars zu machen, muss das abstrakte Formular mit Werten versorgt werden. Dies kann allerdings erst zur

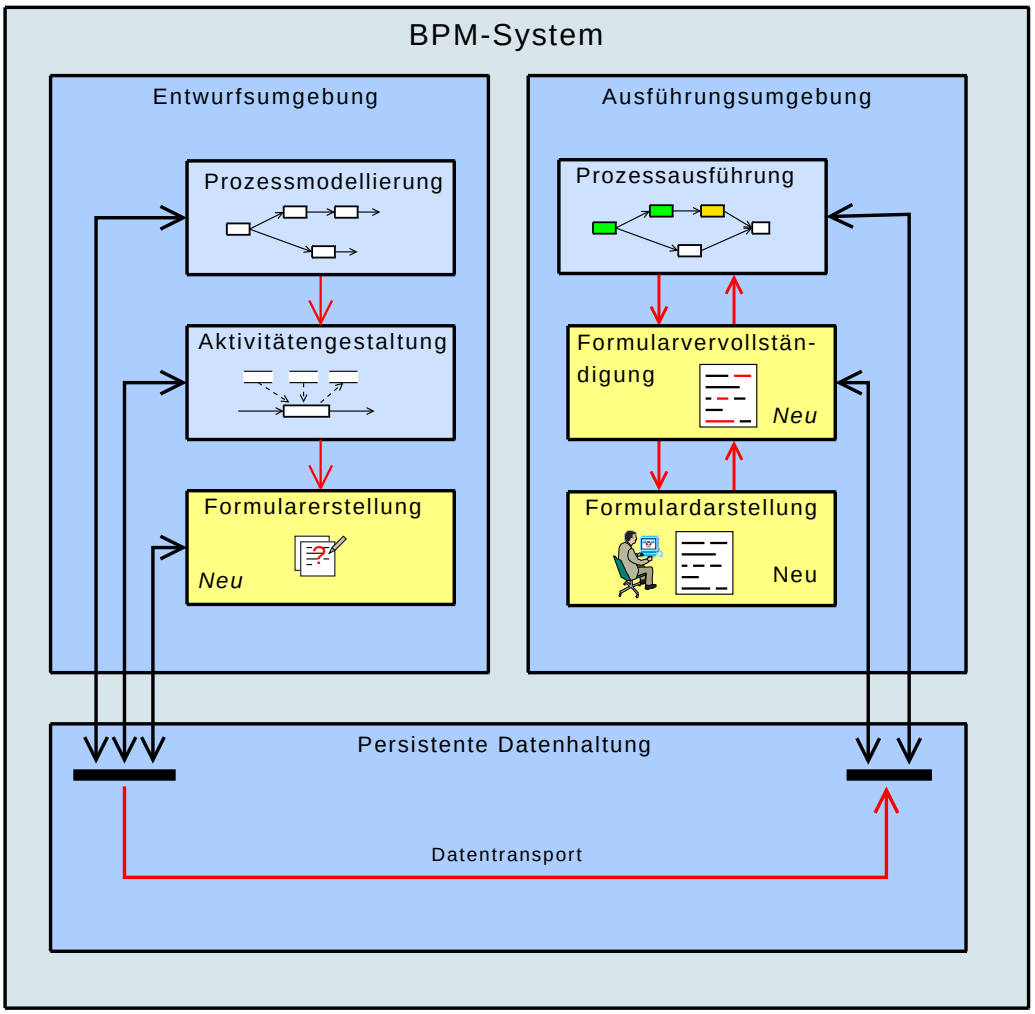


Abbildung 19: Überblick über das Gesamtsystem

Laufzeit geschehen, wenn die Werte für die Parameter vorliegen. Nachdem ein konkretes Formular erstellt wurde kann es dem Endanwender angezeigt werden.

Daraus ergebn sich zwei Subkomponenten, die jeweils die Aufgaben aus den beiden Phasen erledigen. Diese Subkomponenten müssen in das BPM-System integriert werden. Wie diese dabei einzuordnen sind, zeigt Abbildung 19.

Die Subkomponente für die Entwicklungsumgebung dient der Erstellung der Formulare. Ihre Aufgabe ist es, den Formularentwickler dabei zu unterstützen, aus den Parametern der Prozessschritte Formularfelder zu erstellen. Dies ist Schritt 3 aus Abbildung 17. Aufgrund der Tatsache, dass die Werte

der Parameter noch nicht zur Verfügung stehen, wird nur ein abstraktes Formular mit Informationen über das Aussehen und das Verhalten des Formulares erzeugt.

Die Subkomponente für die Ausführungsumgebung ist dafür verantwortlich, die Formulare den Endanwendern bereitzustellen. Dies erfolgt in zwei Schritten. Zunächst wird das zuvor erstellte abstrakte Formular mit den Werten der Parameter versorgt, um ein vollständiges Dokumente zu erhalten. Anschliessend wird das Formular grafisch dem Endanwender angezeigt.

Diese beiden Subkomponenten werden an unterschiedlichen Stellen des BPM-Systems eingebettet. Daher ist es außerdem noch nötig, die Informationen über das erstellte Formular zwischen der Erstellungs- und Anzeigesubkomponente zu transportieren. Die meisten BPM-Systeme bieten hierfür einen Dienst für die persistente Verwaltung von Daten an. Es geht bei dieser Anforderung daher nicht darum, wie das abstrakte Formular gespeichert werden kann, sondern in welchem Format dies geschieht. Dieses Format muss alle notwendigen Informationen kapseln und von beiden Subkomponenten verstanden werden.

## 3.2 Anforderungen

Im Folgenden werden die Details der einzelnen Anforderungen diskutiert. Dabei fließen auch die gewonnen Erkentnisse über die Aufteilung in die Subkomponenten ein. Die Anforderungen sind in zwei Kategorien eingeteilt: Nichtfunktionale Anforderungen und funktionale Anforderungen. Zuerst werden die nicht funktionalen Anforderungen besprochen, da diese allgemeine Bedingungen für das System definieren. Anschließend werden die funktionalen Anforderungen näher erläutert, welche das Verhalten der Subkomponenten betreffen.

### 3.2.1 Nichtfunktionale Anforderungen

**3.2.1.1 Integrationsfähigkeit** Eine Komponente zur Erstellung von Formularen wird nicht als eigenständiges Programm genutzt werden. Das bedeutet, dass der Einsatz nur im Kontext eines BPM-Systems sinnvoll ist. Wenn die Komponente zur Formulargenerierung nicht zusammen mit dem BPM-System entwickelt wird, sondern das BPM-System bereits existiert und gegebenenfalls auch bereits im Einsatz ist, ist eine Integration der neuen

Komponente in das Host-System notwendig. Dabei gibt es zwei Arten der Integration, die beachtet werden müssen.

- logische Integration
- technische Integration

**Logische Integration** Die logische Integration findet bereits in der konzeptionellen Phase der Softwareentwicklung statt. Hier geht es darum, die zu integrierende Komponente inhaltlich an das Host-System anzupassen. Dies bedeutet, dass die Konzepte, welche in dem bereits existierendem System umgesetzt worden sind, auch in der Komponente genutzt werden. Ist dies nicht der Fall, können einige Apskete des BPM-Systems unter Umständen durch den Einsatz der neuen Komponente nicht mehr genutzt werden.

Im Falle der Komponente für die Erstellung von Formularen, sind die BPM-Datentypen ein Beispiel für ein solches Konzept. Im Idealfall werden die Datentypen aus dem BPM-System auch in der neuen Komponente genutzt und dort nicht erneut implementiert. Unter Umständen kann es jedoch sinnvoller sein, eigene Datentypen für die Komponente zu implementieren, beispielsweise um die Informationen aus den Prozessschrittparameter unterteilen zu können. In diesem Fall ist es notwendig, dass mindestens auch alle Datentypen aus dem BPM-System erneut implementiert werden.

**Technische Integration** Gegenstand der technischen Integration sind die Schnittstellen des BPM-Systems und der Komponente. Um die Komponente in der BPM-Suite nutzen zu können, müssen die beiden Programme aufeinander abgestimmt sein und eine gemeinsame Schnittstellen verwenden. Es gibt zwei Möglichkeiten eine gemeinsame Schnittstelle zu erreichen:

1. BPM-System an die Komponente anpassen
2. Komponente an das BPM-System anpassen

Die erste Möglichkeit hat den Vorteil, dass die neue Komponente den vollen Funktionsumfang, den das existierende System anbietet, nutzen kann. Dies setzt allerdings auch eine logische Integration der Komponente in das BPM-System voraus. Die zu integrierende Komponente ist dann optimal auf das Host-System zugeschnitten. Dem gegenüber steht allerdings ein wesentlich höherer Aufwand, wenn die selbe Komponente in ein anderes System eingebunden werden muss. Dies erfordert ein erneutes Anpassen und Implementieren der Komponentenschnittstellen.

Bei der zweite Möglichkeit geht es um die Entwicklung einer generischen Komponente, die zu einer Vielzahl möglicher BPM-Systeme kompatibel ist. Die Komponente muss, nachdem sie einmal entwickelt wurde, nicht mehr geändert werden. Diese Möglichkeit hat allerdings auch den Nachteil, dass man bei der Entwicklung des integrierenden Systems immer an die Schnittstellen der Komponente halten muss. Ein weiterer Nachteil ist, dass eine generische Komponente nicht logisch in ein BPM-System integriert werden kann, da diese nicht für ein einzelnes BPM-System entwickelt wird.

**Anforderung für diese Arbeit** Die Komponente, welche im Rahmen dieser Arbeit entwickelt wird, soll logisch in eine bereits existierende Implementierung von ADEPT2 integriert werden, um die Vorteile bereits implementierter Konzepte nutzen zu können. Da es sich bei der Implementierung dieser Komponente im Rahmen dieser Arbeit um einen Prototypen handelt, spielt die Generizität zumindest bei der Implementierung keine übergeordnete Rolle. Wichtig jedoch ist eine modulare Struktur des neuen Programms, um das Ergebnis dieser Arbeit mit geringem Aufwand weiterentwickeln und an andere BPM-Systeme anpassen zu können.

#### 3.2.1.2 Einfache Bedienung

Die Forderung nach einer einfachen Bedienbarkeit ist zunächst sehr abstrakt. Deshalb wird diese in den nächsten Abschnitten konkreter formuliert. Es gibt im Grunde zwei Personengruppen, welche mit der Komponente interagieren müssen:

- Die Formularentwickler
- Die Endanwender

Für beide Gruppen gelten dabei jeweils andere Anforderungen.

**Einfachheit für Formularentwickler** Für die Formularentwickler bezieht sich die Anforderung nach einer einfachen Bedienbarkeit auf die Entwicklungsumgebung für Formulare. Es wird nicht davon ausgegangen, dass es sich bei einem Formularentwickler immer um einen Informatiker handelt. Daher wird keine tiefergehende technische Programmierkenntnis vorausgesetzt. Trotzdem muss das System leicht erlern- und bedienbar sein. Im Idealfall ist die Entwicklungsumgebnug so intuitiv zu bedienen, dass nicht einmal eine Schulung notwendig ist, um aus den Parametern des Prozessschrittes ein Formular zu erstellen.

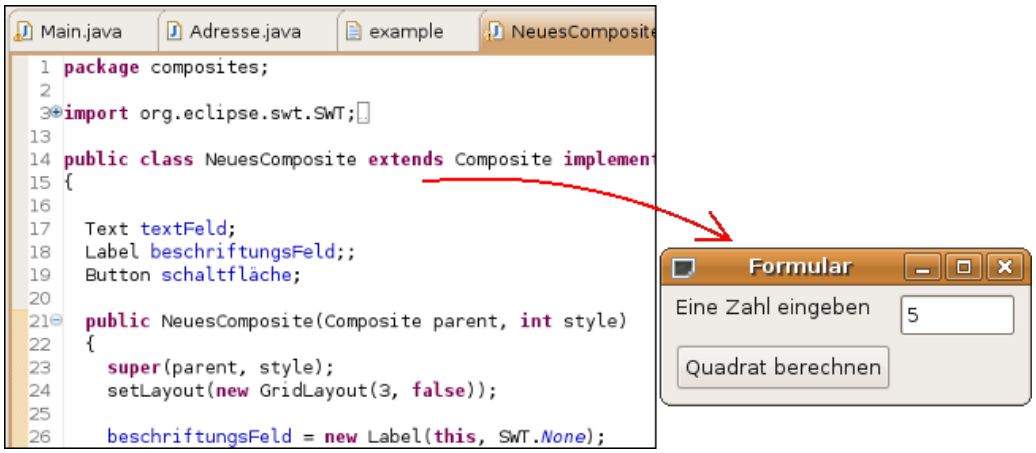


Abbildung 20: Traditionelle Entwicklung einer Oberfläche

Dieses Ziel kann durch einen Assistenten erreicht werden, der den Formularentwickler durch gezielte Fragen den Hauptteil der Arbeit abnimmt und triviale Aufgaben selbständig erledigt. Bei der Entwicklung von traditionellen Oberflächen, wie besispielswese SWT, muss der Entwickler Quellcode angeben, der die Oberfläche zur Laufzeit anlegt. Dies ist in Abbildung 20 dargestellt. Mit speziellen Programmen können die Oberflächen auch grafisch erstellt werden. Der Quellcode wird hierbei automatisch generiert. Der Entwickler muss das Layout jedoch immer noch selbst gestalten.

Ein Assistent ist in einfachen Fällen in der Lage auch das Layout zu generieren. Ein einfacher Fall liegt dann vor, wenn es sich bei den Parametern für den Prozessschritt um einfache Daten handelt, und das Layout keine besondere Anpassungen erfordert. Dies ist in Abbildung 21 dargestellt. In kom-

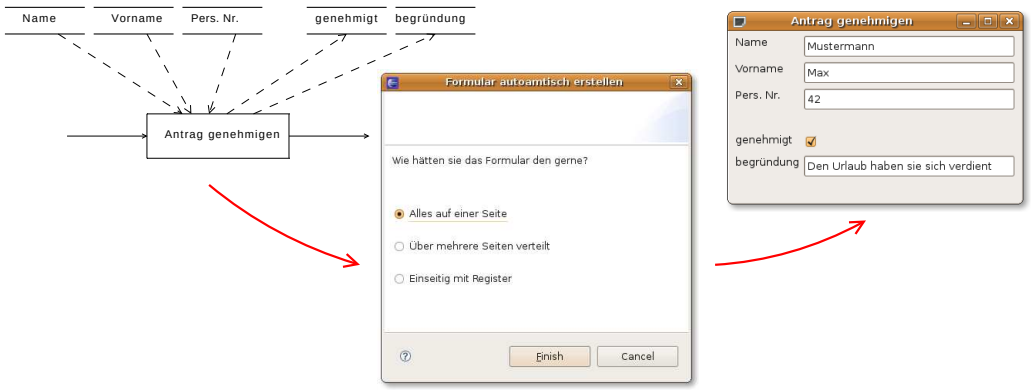


Abbildung 21: Entwicklung eines Formulars mit einem Assistenten

plizierteren Fällen kann der Assistent den Formularentwickler involvieren, um durch gezielte Fragen fehlende Informationen zu vervollständigen und spezielle Wünsche und Anforderungen in das Formular zu integrieren.

Die Automatisierung der Formularerstellung ist dabei immer nur eine Option für den Formularentwickler. Um die Gestaltung der Formulare flexibel zu halten und den Formularen bei Bedarf mehr Funktionalität, beispielsweise durch zusätzliche Anwendungslogik hinzuzufügen, muss eine manuelle Änderung des Formulars durch den Formularentwickler immer möglich sein.

**Einfachheit für Endanwender** Für die Endanwender bezieht sich die Einfachheit der Bedienung auf die Formulare selbst. Ein Formular ist dann einfach zu bedienen, wenn ein Endanwender intuitiv in der Lage ist, ein neues, bisher unbekanntes Formular zu lesen und zu bearbeiten. Das heißt, dass keinerlei Schulungen im Umgang mit Formularen notwendig sind. Diese Forderung ist wichtig, da die Endanwender aus den unterschiedlichsten Branchen kommen können. Viele dieser Endanwender verfügen daher über kein oder nur geringes technisches Verständnis. Für diese muss ein Formular ebenso selbstverständlich benutzbar sein, wie für einen Informatiker.

Um dies zu erreichen, ist es wichtig, bei der Erstellung eines Formulares immer auf die Komplexität zu achten. Die Komplexität eines Formulars hängt zum Einen von Prozessschrittparameter ab und zum Anderen von deren Darstellung im Formualr. Für diese Darstellung gibt es zwei Faktoren, die zu beachten sind: Der Formularentwickler und der eben angesprochene Assistent. Die Erstellung durch einen Formularentwickler kann nur durch hinreichende Schulung bezüglich anwendungsfreundlicher Formularerstellung beeinflusst werden. Für den Erstellvorgang durch den Assistenten hingegen ist die Komponente verantwortlich. Es ist beim Entwurf eines Assistenten für die Formularerstellung darauf zu achten, dass ein generiertes Formular auch komplexe Parameter übersichtlich darstellt.

**Zusammenfassung** Für die Entwicklung der neuen Komponente ist es sehr wichtig, dass sie den Formularentwickler bei seiner Arbeit unterstützt und automatisierbare Aufgaben selbständig durchführt. Der Formularentwickler muss mit einem Minimum an Aufwand zu einem gutem Ergebnis kommen, aber trotzdem die Möglichkeit haben, das Formular teilweise oder vollständig, individuellen Bedürfnissen anzupassen. Es ist ebenfalls von großer Bedeutung, dass das vollständig generierte Formular auch bei komplexen

Ein- und Ausgabeparametern immer so einfach gehalten wird, dass auch fachfremde Endanwender problemlos damit zurecht kommen können.

**3.2.1.3 Lizenz** Ein weiteres wichtiges Thema bei der Entwicklung von Software ist die Lizenz, welche für die Software verwendet wird. Besonders bei Projekten, bei denen Software von Drittanbietern integriert wird, ist es wichtig, sich mit dieser Frage auseinander zu setzen. Welche Lizenz für eine Software verwendet werden darf, hängt hierbei von mehreren Faktoren ab.

Auf der einen Seite wird die Lizenz für eine Komponente von der verwendeten Lizenz des Host-Systems beeinflusst. Ein vorhandene System kann unter einer Lizenz stehen, welche dazu verpflichtet, alle Quellen des Systems (einschließlich der aller Komponenten) offen zu legen, wie zum Beispiel die GPL [GNU91]. In diesem Fall darf die Komponente nicht mit einer Lizenz versehen werden, welche genau dies verhindert. Wenn die Komponente für ein spezielles System entwickelt wird und nicht unbedingt generisch sein muss, richtet sich die Lizenz für die Komponente nach der Lizenz des Systems, in welches die Komponente integriert werden soll.

Auf der anderen Seite ist es natürlich ebenso möglich, dass die Komponente selbst wieder aus bereits existierenden Komponenten und Programmen besteht. Diese stehen ebenfalls wieder unter einer Lizenz, welche beachtet werden muss. Eine Komponente eines Drittanbieters, welche nur für den privaten Gebrauch der Software lizenziert ist, darf nicht in die eigene Komponente integriert und so vertrieben werden. In diesem Fall muss die Lizenz der eigenen Software, also auf die Lizenzen der verwendeten Software, abgestimmt sein.

Es gibt auch Fälle, in denen beide Aspekte zutreffen. Wenn eine Komponente für ein System entwickelt wird und selber wieder bereits existierende Software von anderen Herstellern integriert, müssen sowohl die Lizenzen der integrierten Software, als auch die des integrierenden Host-Systems beachtet werden.

Um das Ergebnis dieser Arbeit auch in Zukunft weiterentwickeln und diese Entwicklungen veröffentlichen zu können, kommt es darauf an, dass die Komponente zur Erstellung von Formularen keine Lizenz verwendet, die diesbezüglich Restriktionen aufweist. Insbesondere bedeutet dies, dass auch die Quellen offen gelegt werden müssen. Bei der Verwendung von Software welche zur Einbettung in die Komponente vorgesehen ist, ist auf die Kompatibilität der Lizenz zu achten. Da die Quellen der ADEPT2-Implementierung momentan nicht frei verfügbar sind, ist beispielsweise die GPL [GNU91] keine Option im Rahmen dieser Arbeit.

### 3.2.2 Funktionale Anforderungen

Nachdem im vorherigen Abschnitt die grundlegenden nichtfunktionalen Anforderungen der neuen Komponente besprochen worden sind, werden in den folgenden Abschnitten die Details für das Verhalten dieser Komponente diskutiert. Begonnen wird mit den allgemeinen Askpekten. Dazu zählen die Anforderungen zu den grafischen Kontexten, den Datentypen und auf diesen aufbauend die grafischen Komponenten, die für die Darstellung der Daten verantwortlich sind. Anschliessend werden die Anfoderungen zur Anwendungslogik und der damit eng verbundenen Restriktionen für Formulare vorgestellt. Die darauf folgenden Abschnitte beschäftigen sich mit den spezielleren Anforderungen, wie Formularhistorie, Lokalisierung und externen Datenquellen.

**3.2.2.1 Grafische Kontexte** Formulare dienen der Interaktion mit dem Endanwender und sind daher Bestandteil der grafisschen Oberfläche eines BPM-Systems. Für die Erstellung von Formularen gelten deshalb die selben Rahmenbedingen, wie für alle anderen grafischen Oberflächen des Systems. Welche Rahmenbedingen dies sind, ist wiederum vom verwendeten grafischen Kontext abhängig, da unterschiedliche Kontexte auch unterschiedliche Möglichkeiten bieten. Die Beschreibung des Layouts eines Formulars muss jedoch unabhängig von den Möglichkeiten eines grafischen Kontextes erfolgen.

**Motivation** Ein BPM-System muss nicht auf einen grafischen Kontext beschränkt sein. Prinzipiell können durch das System beliebig viele Kontexte unterstützt werden. Das Problem dabei ist jedoch, dass nicht unbedingt bei jedem Endanwender alle Kontexte umgesetzt werden können, beispielsweise, weil die erforderliche Software zum Betrachten eines PDF-Dokumentes nicht installiert ist. Es ist daher zwar zur Entwurfszeit bekannt, welche grafischen Kontexte das BPM-System generell unterstützt, jedoch nicht, welche dieser Kontexte bei welchem Endanwender verfügbar sein werden. Daher ist es nicht möglich sich zur Entwurfszeit auf einen grafischen Kontext festzulegen, mit dem das Formular beschrieben wird. Um Formulare denoch bei jedem Endanwender darstellen zu können ist es notwendig, den grafischen Kontext, der für die Darstellung eines Formulars verwendet wird erst zur Laufzeit festzulegen. Dazu muss das Formular zur Entwurfszeit jedoch kontextunabhänig beschrieben werden. Diese Beschreibung muss dann zur Laufzeit für einen der vorhandenen Kontexte übersetzt werden.

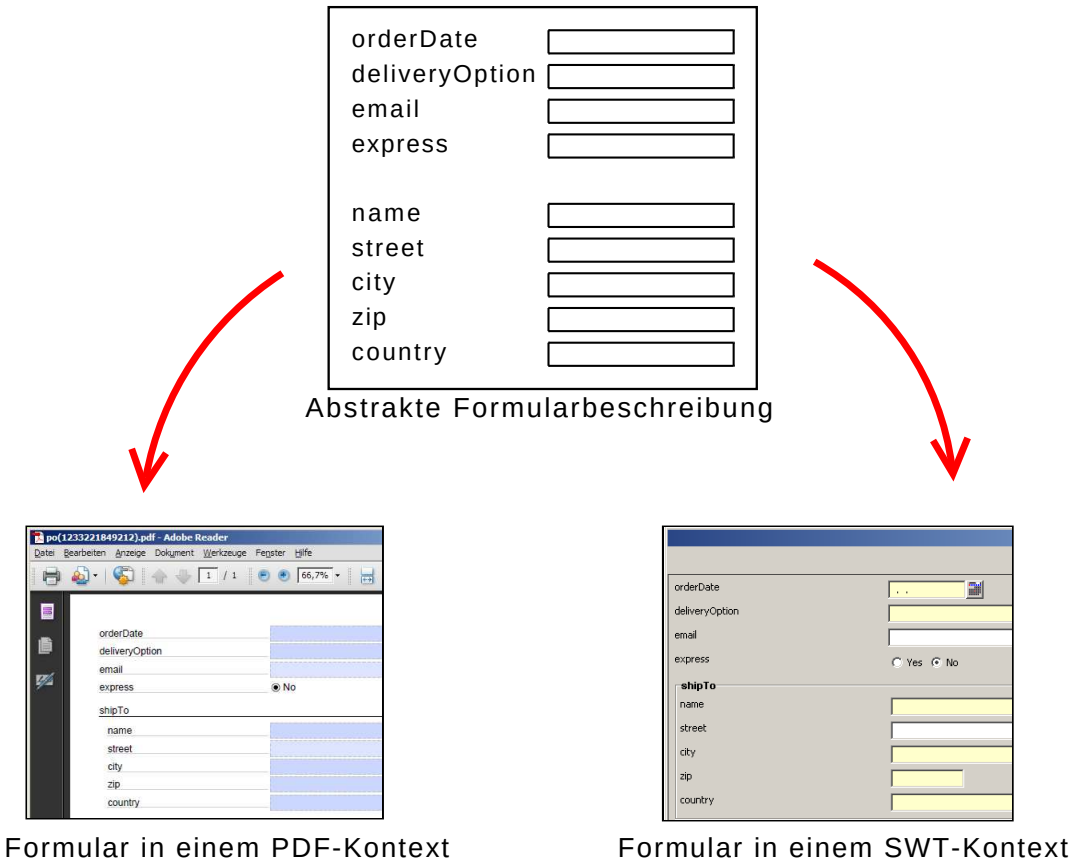


Abbildung 22: Formulare in unterschiedlichen Kontexten mit ähnlichem Aussehen

**Probleme** Diese Anforderung erfordert beim Entwurf der Komponente die Festlegung auf ein eigenes kontextunabhängiges Format zur Formularbeschreibung. Dieses Format muss alle Informationen beinhalten, die nötig sind, um das Formular zur Ausführungszeit für jeden Kontext zu übersetzen, den das BPM-System zur Verfügung stellt. Ein Problem sind die unterschiedlichen Möglichkeiten der einzelnen Kontexte. In einem Formular innerhalb eines SWT-Kontextes ist es beispielsweise möglich, baumartige Strukturen darzustellen. Bei einer Beschreibung des Formulars durch HTML ist dies nur mit einigem Aufwand zu erreichen. Es muss jedoch sichergestellt werden, dass ein Formular in allen vom BPM-System bereitgestellten Kontexten ein ähnliches Aussehen und Verhalten hat.

Bei der Festlegung eines Formates für eine abstrakte Formularbescheibung ist deshalb darauf zu achten, dass die Beschreibung in allen Kontexten auch

umgesetzt werden kann. Ein Beispiel für die Umsetzung eines abstrakten Formulars für zwei Kontexte zeigt Abbildung 22. Um ein Format für die Beschreibung festzulegen, dürfen nur Elemente verwendet werden, welche von allen Kontexten unterstützt werden.

**Zusammenfassung** Zur Beschreibung eines Formulars stehen meherere grafischen Kontexte zur Verfügung, welche jedoch nicht alle beim Endanwender umgesetzt sein müssen. Um trotzdem die korrekte Darstellung bei jedem Endanwender garantieren zu können, muss die Beschreibung des Formulars zur Entwurfszeit kontextunabhängig erfolgen. Zur Laufzeit muss diese Beschreibung in jeden Kontext übersetzbar sein. Für die Beschreibung sind deshalb nur Möglichkeiten zugelassen, die sich in allen Kontexten umsetzen lassen.

#### 3.2.2.2 Datentypen

Die Hauptaufgabe von Formularen in einem BPM-System ist die Präsentation und das Abfragen von Informationen. Wie bereits aus Kapitel 2 bekannt ist, werden Datentypen benötigt, um die Daten aus dem BPM-System als Informationen interpretieren zu können. In diesem Abschnitt wird erläutert, welche Art von Daten für das BPM-System wichtig sind, und welche Datentypen benötigt werden, um diese Daten zu interpretieren.

**Einfache Informationen** Ein großer Teil der Informationen die in einem Geschäftsprozess vorkommen, lassen sich in mehrere Gruppen einteilen. Dazu zählen Wahrheitswerte, Text und Zahlen. Datentypen für solche Informationen sind in vielen Programmiersprachen und BPM-Systemen bereits vorhanden. Einige solcher Typen sind:

- String
- Integer
- Float
- Boolean

**Komplexe Informationen** Es gibt jedoch auch komplexere Informationen, welche sich nicht durch einen dieser einfachen Datentypen beschreiben lassen, wie beispielsweise eine Adresse.

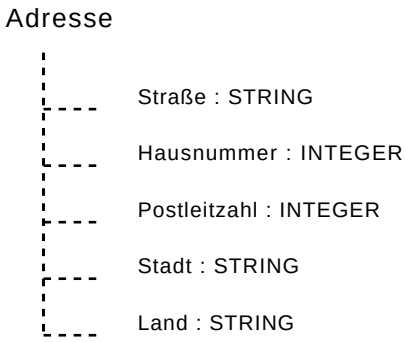


Abbildung 23: Strukturierter Datentyp für eine Adresse

Eine solche Information besteht in der Regel aus mehreren Elementen. Diese Elemente, wiederum stellen eigene, getrennte Informationen dar. Die Adresse besteht zum Beispiel aus den Elementen *Straße, Hausnummer, Postleitzahl, Stadt* und *Land* (Abbildung 23).

Eine Adresse lässt sich daher nicht mittels eines einfachen Datentyps wie beispielsweise einem String beschreiben. Die Informationen der einzelnen Elemente der Adresse würden verloren gehen, wenn diese in einem gemeinsamen Text gespeichert werden. Es lässt sich nicht zweifelsfrei festlegen welcher Teil des Textes die Stadt und welcher Teil die Straße beinhaltet. Dafür ist eine einheitliche Kodierung der Elemente innerhalb des Textes notwendig. Die Verwendung einer entsprechenden Kodierung kann bei sehr komplexen Informationen jedoch zu unübersichtlichen Texten führen.

Wie eine komplexe Information zu interpretieren ist, ist immer von dessen Struktur abhängig. Daher ist es möglich komplexe Informationen mit Hilfe von Datentypen zu beschreiben, welche ebenfalls eine komplexe Struktur besitzen. Eine mögliche Darstellung der interpretierten Daten einer Adresse wird in Abbildung 24 gezeigt.

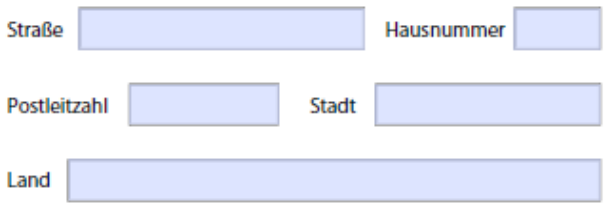


Abbildung 24: Daten interpretiert als Adressinformation

**Dateien als Informationen** Es gibt auch komplexe Informationen, die sich selbst mit strukturierten Datentypen nur schwer beschreiben lassen. Beispielsweise die Informationen, die in einer Excel-Tabelle oder einem Adobe-Dokument zu finden sind. Besonders schwierig ist die Strukturierung von Daten aus binären Dateien. Deren Inhalt ist zwar ebenfalls eine Information, jedoch lässt sich für beliebige binäre Daten keine einheitliche Struktur angeben.

Deshalb müssen Dateien als Informationen in einem Formular separat behandelt werden. Auch hierfür soll die neue Komponente entsprechend vorbereitet werden. Es muss möglich sein, Dateien sowohl als Eingabe- wie auch als Ausgabeparameter von Prozessschritten zu verwenden. Wird eine Datei als Eingabeparameter verwendet, muss der Endanwender im Formular die Möglichkeit erhalten diese zu speichern oder lokal in einer dafür vorgesehenen externen Anwendung zu betrachten. Wird die Datei als Ausgabeparameter angegeben, muss der Endanwender eine Datei auf seinem lokalen System dem Formular hinzufügen können, um diese dem BPM-System zu übergeben.

**Listenwertige Informationen** Ebenso wichtig wie strukturierte Informationen sind listenwertige Informationen, wie beispielsweise eine Einkaufsliste in einem Onlineshop. Eine solche Liste könnte zwar durchaus als komplexer Datentyp gesehen werden, der aus mehreren Komponenten mit jeweils dem selben Datentyp besteht (Abbildung 25). Allerdings hat dies den Nachteil, dass die Länge der Liste durch die Anzahl der Komponenten nicht veränderbar ist. Für eine längere Liste muss ein neuer Datentyp mit mehr Komponenten erstellt werden.

Dies ist eine unzureichende Lösung, da die Anzahl der Elemente in der Liste zur Ausführungszeit des Formulars variabel bleiben muss. Bei listenwertigen Datentypen handelt es sich daher um einen weiteren Sonderfall, welcher von der neuen Komponente beachtet werden muss.

**Zusammenfassung** Grundlage für Informationen sind die Daten aus den Parametern der Prozessschritte. Diese Daten müssen interpretiert werden, um Informationen zu erhalten. Es gibt verschiedene Arten von Informationen, die in einem BPM-System von Bedeutung sind. Welche Informationen konkret verwendet werden, ist zur Entwurfszeit nicht bekannt. Um das System also vor laufenden Änderungen zu bewahren, und auch in Zukunft mit neuen Arten von Informationen umgehen zu können, muss die neue Komponente in der Lage sein, die Art der Informationen zu beschreiben. Hierfür


Abbildung 25: Liste als strukturierter Datentyp mit fester Länge

sollen die bereits vorhandenen Datentypen des BPM-Systems, strukturierte Datentypen, listenwertigen Datentypen oder Datentypen für Dateien verwendet werden.

**3.2.2.3 Grafische Elemente** Im letzten Abschnitt sind die Datentypen besprochen worden, die dabei helfen, die Daten aus dem BPM-System als Informationen zu interpretieren. In diesem Abschnitt geht es um die grafische Repräsentation dieser Informationen im Formular. Es ist die Aufgabe des Formularentwicklers die Parameter des Prozessschrittes im Formular auf grafische Elemente abzubilden, welche die Informationen dieser Parameter korrekt und sinnvoll anzeigen. Dabei gibt es zwei wichtige Aspekte zu beachten:

**Endanwendergewohnheiten** BPM-Systeme werden von Unternehmen aus allen Branchen benötigt. Die Endanwender, die mit diesem System arbeiten, sind daher meistens nicht der IT-Branche zuzuordnen. Der Umgang mit einem PC hat sich in der Geschäftswelt dennoch stark etabliert. Daraus ergeben sich Gewohnheiten der Endanwender, was auch die Benutzung von grafischen Oberflächen betrifft. Endanwender kennen aus der täglichen Arbeit eine Reihe von grafischen Standardelementen, welche daher bereits vertraut sind. Einige davon sind in Abbildung 26 aufgezeigt. Da das System den Endanwender unterstützen soll, muss sich das System den Erwartungen und Gewohnheiten des Endanwenders anpassen.

Eine weitere Gewohnheit der Mitarbeitern von Unternehmen betrifft den Umgang mit nichtdigitalen Formularen. Digitale Formulare finden erst seit einigen Jahren Verbreitung in Unternehmen. Um den Endanwender den Um-

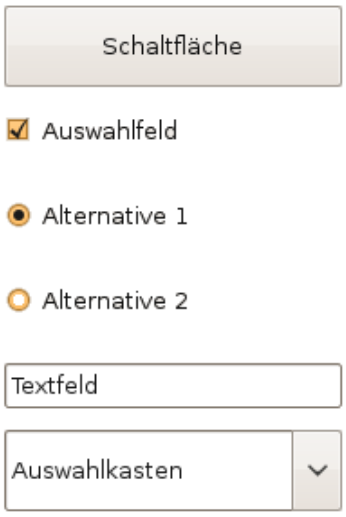


Abbildung 26: Standardelemente grafischer Oberflächen

stieg auf die neue Form der Formulare zu erleichtern ist es notwendig, dass Formulare, welche von einem BPM-System erstellt werden, ein grafisches Pendant der traditionellen Formulare darstellen. Abbildung 27 zeigt ein Formular für eine elektronische Überweisung. Die verwendeten Felder sind denen eines traditionellen Überweisungsformulars (Abbildung 28) sehr ähnlich.

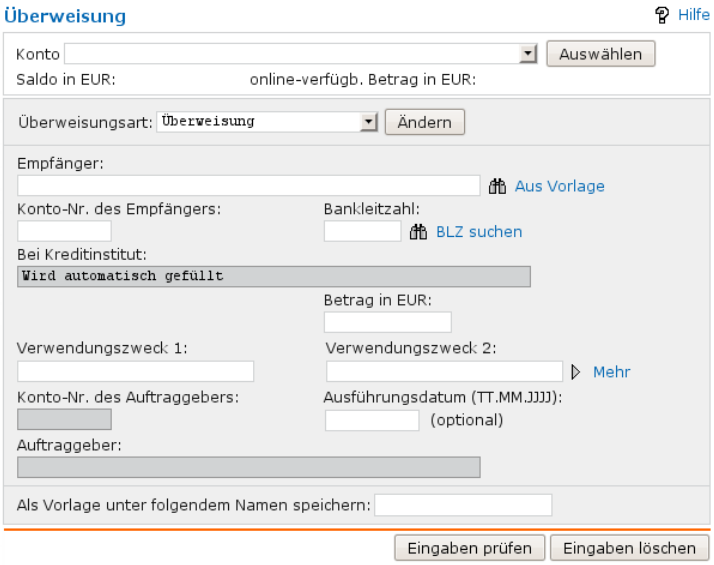


Abbildung 27: Beispielformular für elektronische Überweisungen [FID09]

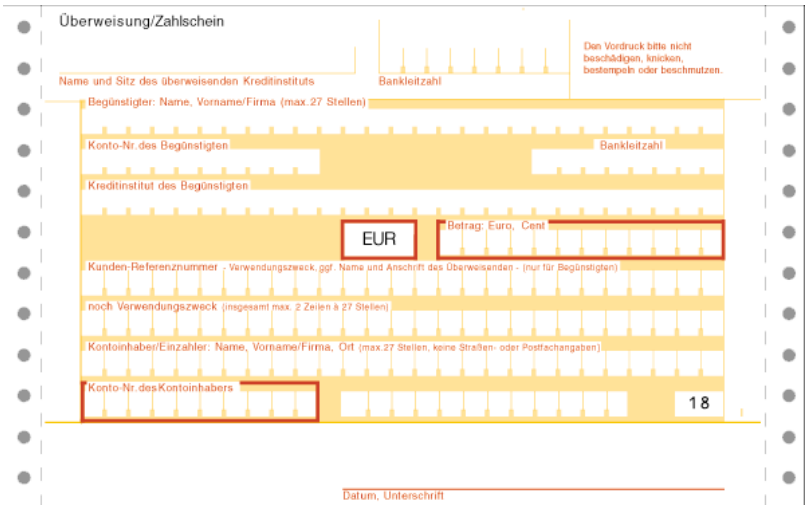


Abbildung 28: Traditionelles Überweisungsdokument

**Grafische Elemente in unterschiedlichen Kontexten** Neben den Gewohnheiten der Endanwender gibt es noch einen weiteren Punkt, der die bereits angedeutete Auswahl von grafischen Komponenten einschränkt: In unterschiedlichen Kontexten gibt es teilweise unterschiedliche grafische Elemente. In einer HTML-Umgebung stehen dem Formularentwickler beispielsweise grafische Elemente zur Verfügung um Dateien zu übergeben (Abbildung 29, links). In einem *SWT-Composite* muss ein solches Element erst erstellt werden (Es gibt zwar einen sogenannten File-Dialog, es gibt jedoch kein grafisches Element, welches diesen automatisch öffnet und die angegeben Dateien verwaltet).

Auf der anderen Seite gibt es in einem SWT-Kontext grafische Elemente um Baumstrukturen darstellen zu können (Abbildung 29, rechts). Dafür gibt es in einem HTML-Kontext noch keine fertigen grafischen Elemente.

Selbst bei grafischen Elementen, welche in mehreren Kontexten vertreten sind, kann es sein, dass sich diese Komponenten bezüglich ihrer Anwendung oder ihres Verhaltens unterscheiden. Es gibt in einem HTML-Kontext beispielsweise Schaltflächen, welche den Inhalt der grafischen Elemente des HTML-Dokuments an einen entfernten Server senden. Im SWT-Kontext gibt es ebenfalls Schaltflächen, jedoch haben diese ein anderes Verhalten. Dies muss bei der Umsetzung der Formulare zur Ausführungszeit berücksichtigt werden.

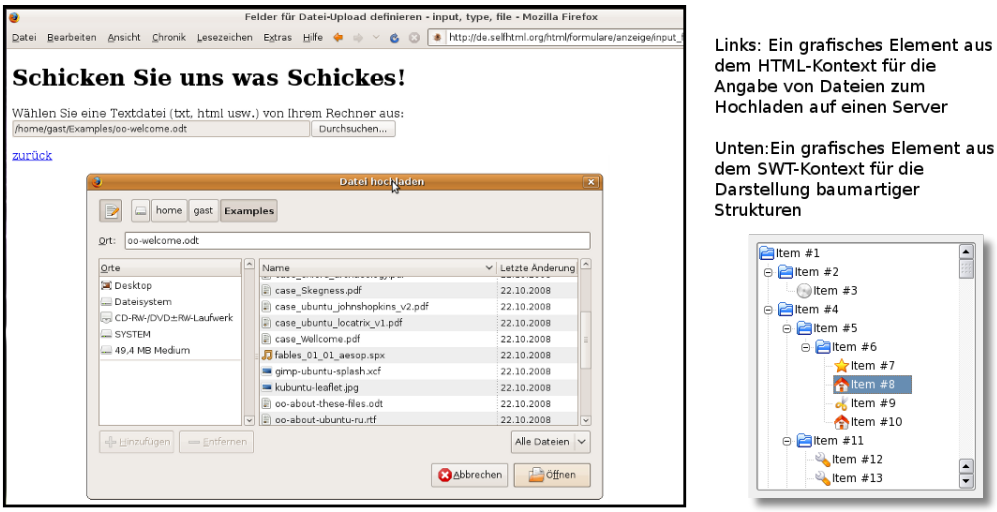


Abbildung 29: Kontextbezogene grafische Elemente [Mü08b].

**Zusammenfassung** Der Formularentwickler hat die Aufgabe, die Informationen der Parameter grafisch auf das Formular abzubilden. Dabei stehen verschiedene grafische Elemente zur Verfügung. Welche grafischen Elemente eingesetzt werden können, orientiert sich zum Einen an den Gewohnheiten des Endanwenders. Hier ist der Umgang mit traditionellen Formularen und der Umgang mit grafischen Oberflächen von Softwaresystemen zu beachten. Zum Anderen wird die Auswahl der Elemente durch die unterstützten grafischen Kontexte beeinflusst.

#### 3.2.2.4 Integration von Anwendungslogik

In den letzten Abschnitten wurden grundlegende Anforderungen bezüglich grafischen Kontexten und Elementen und Datentypen dargelegt. Im folgenden Abschnitt soll darauf aufbauend über die Integration von Anwendungslogik in Formulare diskutiert werden. Es wurde bereits erwähnt, dass ein Formular die Aufgabe hat, Daten zu präsentieren und Daten von Endanwendern abzufragen. Für viele Formulare sind diese Möglichkeiten ausreichend. Bei komplexeren Formularen ist es jedoch auch sinnvoll, bereits im Formular mit diesen Daten arbeiten zu können.

**Gründe für Anwendungslogik in Formularen** Es gibt Fälle in denen eine präzisere Steuerung des Verhaltens von Formularen gewünscht ist. Ein Beispiel mit einem Formular eines Onlineshops soll dies zeigen: Bei dem Formular soll es möglich sein, verschiedene Artikel auszuwählen, welche alle


Abbildung 30: Formular für Einkaufsliste mit automatischer Preisberechnung [Ein09]

einen definierten Preis haben. Das Formular muss den Gesamtpreis aller ausgewählter Artikel berechnen und dem Endanwender anzeigen. Bei einer Änderung der Posten der Einkaufsliste muss dieser Preis neu berechnet werden. Abbildung 30 zeigt ein entsprechendes Formular.

Eine solche Anforderung kann auch mit normalen Formularen ohne zusätzliche Anwendungslogik gelöst werden, in dem man dem BPM-System die Verantwortung der Berechnung übergibt. Dafür muss man nach jeder Änderung des Formulars dieses abschicken und anschließend in einem separaten Prozessschritt den neuen Preis berechnen. Dannach wird ein neues Formular mit der geänderten Einkaufsliste und dem geänderten Preis erstellt.

Diese Lösung ist jedoch nicht sehr elegant, da das BPM-System dabei einen Teil der Anwendungslogik übernehmen muss, der nicht zur Semantik des Prozesses gehört. Außerdem verursacht jede Änderung durch das Erstellen eines neuen Formulares eine Wartezeit für den Endanwender. Besser ist es, die Berechnung innerhalb des Formulars durchführen zu können und damit Zeit zu sparen und das BPM-System nicht unnötig zu belasten.

**Probleme mit Anwendungslogik in Formularen** Wenn man Berechnungen durch das BPM-System in separaten Prozessschritten durchführt, muss sich der Formularentwickler keine Gedanken um die Ausführung von Anwendungslogik machen. Wird die Berechnung jedoch im Formular durchgeführt, muss auch sichergestellt werden, dass diese Geschäftslogik innerhalb der Ausführzeit des Formulars ausgeführt wird. Dies kann zu einem Problem werden, wenn man sich nochmal mit den unterschiedlichen Kontex-

ten befasst, in denen ein Formular ausgeführt werden kann. Die Kontexte müssen die Verarbeitung von Geschäftslogik unterstützen. Wenn ein Formular als SWT-Anwendung realisiert wird, kann die Anwendungslogik als zusätzlicher Quellcode implementiert werden. Bei einer Realisierung des Formulars in Form eines PDF-Dokumentes ist zusätzliche Anwendungslogik jedoch ein Problem. Bei diesem Dokumenttyp ist es nicht möglich, das Verhalten des Dokumentes zu beeinflussen.

Selbst wenn bei allen eingesetzten Kontexten die Unterstützung zusätzlicher Anwendungslogik möglich ist, gibt es noch ein weiteres Problem: Das Verhalten der Logik muss in allen Kontexten identisch sein. Die Logik wird vom Formularentwickler zur Erstellzeit des Formulars beschrieben. Zu diesem Zeitpunkt ist es jedoch nicht möglich zu sagen, in welchem Kontext das Formular tatsächlich ausgeführt wird. Daher ist es notwendig, dass die Anwendungslogik auf abstrakte Weise beschrieben wird, ohne Details über den Kontext beachten müssen.

Das löst das Problem jedoch nicht vollständig. Ähnlich wie bei den grafischen Elementen ist die Ausführungsumgebung dafür verantwortlich die Anwendungslogik semantiktreu von der abstrakten Beschreibung in eine zum Kontext kompatible Logik zu übersetzen.

**Anforderung an die neue Komponente** Die neu zu entwickelnde Komponente hat die Aufgabe, dem Formulardesigner die Entwicklung von Formularen mit einer hohen Funktionalität und einem intelligentem Verhalten zu ermöglichen. Der Formularentwickler muss dabei die Möglichkeit haben, das Formular auf Benutzereingaben reagieren zu lassen, beliebige Berechnungen mit den Daten des Formulars durchzuführen, und das Ergebnis dieser Berechnungen dynamisch zur Ausführzeit wieder im Formular zu präsentieren. Dieses durch den Formularentwickler definierte Verhalten darf sich in verschiedenen Kontexten nicht unterscheiden.

**3.2.2.5 Lokalisierung** In diesem Abschnitt wird die Anpassungsfähigkeit der Formulare an Endanwender erläutert. Zuerst werden mögliche Gründe für die Fähigkeit von Formularen sich an Endanwender anpassen zu können aufgezählt. Anschließend folgt eine Diskussion über zwei Ansatzpunkte für eine Anpassung des Formulars und ihre Probleme.

**Motivation** Häufig agieren große Unternehmen nicht nur auf nationaler Ebene, sondern weltweit. Dies führt dazu, dass in dem Unternehmen Men-

Abbildung 31: Barrierefreie Tastatur [BWF09]

schen aus unterschiedlichen Ländern zusammenarbeiten. Diese Mitarbeiter arbeiten alle mit den selben Prozessen und daher auch mit den selben Formularen. Da man nicht annehmen darf, dass alle Mitarbeiter die selben Sprachen beherrschen und die selbe Schrift benutzen, ist es für ein BPM-System wichtig, Formulare anzubieten mit der alle Beteiligten zurechtkommen.

Ein weiterer wichtiger Grund für die Anpassungsfähigkeit von Formularen ist die Berücksichtigung von Menschen, die auf Barrierefreiheit angewiesen sind. Mitarbeiter, welche keine Sehfähigkeiten besitzen, müssen den Inhalt von Formularen auf eine andere Weise wahrnehmen können. Die traditionelle grafische Lösung ist hier nicht möglich. Der Inhalt muss verbal mitgeteilt werden oder auf einer speziellen Hardware mit Unterstützung für Blindenschrift präsentiert werden, wie sie in Abbildung 31 dargestellt ist.

**Ansatzpunkte** Eine Möglichkeit die Anpassung vorzunehmen, ist zur Entwicklungszeit des Prozesses für jede Endanwendergruppe ein eigenes Formular zu erstellen, welches genau auf die Zielgruppe zugeschnitten ist. Zur Ausführungszeit des Formulars hat der Endanwender dann die Möglichkeit das für ihn am besten geeignete Formular auszuwählen und mit diesem dann zu arbeiten.

Dieses Vorgehen ist aus mehreren Gründen sehr aufwändig. Zum Einen muss für jede Endanwendergruppe ein eigenes Formular erstellt werden. Zum Anderen können sich die Anforderungen an die Lokalisierung ändern, wenn beispielsweise eine weitere Sprache angeboten werden muss. Dies erfordert ein Anpassen aller bisher entworfenen Formulare.

Bessere ist es, wenn jedes Formular nur einmal entworfen wird und die Ausführungsumgebung das Formular für den jeweiligen Endanwender automatisch zur Ausführungszeit anpasst. Dazu benötigt diese jedoch Informationen über den entsprechenden Endanwender. Ein großes Problem bei diesem Ansatzpunkt ist, dass die Sprache automatisch in eine vom Endanwender beherrschte Sprache übersetzt werden muss. Die Technologie für eine sowohl grammatikalisch, also auch semantisch korrekte Übersetzung ist jedoch bei weitem noch nicht ausgereift, was zu Verwirrung bei unklar übersetzten Textbausteinen führen kann.

**Zusammenfassung** Es gibt verschiedene Gründe für eine Lokalisierung der Formulare an verschiedene Endanwender. Zum Einen sind das die sprachlichen Unterschiede der Endanwender. Zum anderen stellt die Barrierefreiheit für manche Endanwender ein wichtiges Kriterium dar.

Hier sind zwei grundsätzliche Herangehensweisen denkbar: Die Anpassung kann durch den Formularentwickler, durch das Erstellen mehrerer Formulare für den selben Prozessschritt geschehen oder durch die Laufzeitumgebung durch dynamisches Anpassen eines einzelnen Formulars.

Die neue Komponente für die Formularerstellung soll in der Lage sein, dem Endanwender ein für ihn angepasstes Formular zu bieten, welches keine inhaltlichen Unklarheiten durch die Lokalisierung enthält. Der Formularentwickler soll dennoch nicht gezwungen sein, für jede neue Sprache alle Formulare neu zu erstellen.

**3.2.2.6 Anbindung externer Datenquellen** Bis jetzt wurde davon ausgegangen, dass alle Informationen, welche in dem Formular präsentiert werden, auf den Parametern des Prozessschrittes basieren. Im Idealfall ist dies auch so, da dadurch das BPM-System die volle Kontrolle über die Informationen hat, welche dem Endanwender dargestellt werden. Es ist allerdings möglich, dass dem Endanwender die Informationen aus den Prozessschrittparametern nicht ausreichen, um das Formular Bearbeiten zu können. Daher ist es wünschenswert, Daten und Informationen außerhalb des Zuständigkeits-

bereiches des BPM-Systems in das Formular einbinden zu können, beispielsweise von einer externen Datenbank.

Allerdings ist dabei zu beachten, dass die Informationen bei jedem Aufruf des Formulars variieren können, da sie nicht durch die Prozesslogik kontrolliert werden. Ein Endanwender, der das Formular heute bearbeitet, kann mit anderen Werten arbeiten, wie ein Endanwender, welcher das selbe Formular im selben Prozessschritt zu einem anderen Zeitpunkt bearbeitet.

**3.2.2.7 Restriktionen für Formularfelder** Dieser Abschnitt beschäftigt sich mit der Beschränkung von Endanwendereingaben in Formularen, wie zum Beispiel die Länge der Eingabe in einem Textfeld.

**Motivation** Nicht immer sind alle Eingaben sinnvoll, die in einem Formularfeld möglich sind. Welche Eingaben möglich sind, kann bereits durch Angabe eines Datentyps beeinflusst werden. Dies verhindert, dass in ein Feld ein Name eingegeben wird, obwohl beispielsweise eine Jahreszahl erwartet wird. Etwas komplizierter wird es jedoch, wenn eine Eingabe eine bestimmte Codierung (Syntax) erwartet. Bei einem Textfeld, welches für die Eingabe eines deutschen Autokennzeichens gedacht ist, ist die Eingabe von *Kräutertee* im Sinne des Datentyps durchaus korrekt. Die Syntax stimmt jedoch nicht. Noch schwieriger wird die semantische Analyse der Eingabe. In einem Feld, welches an einen ganzzahligen Datentypen gebunden ist und die Eingabe des Alters einer menschlichen Person erwartet, ergibt die Eingabe *4711* oder *-3* wenig Sinn. Es ist zwar möglich, jedoch ist die Eingabe semantisch nicht korrekt.

Um die Benutzung von Formularen zu erleichtern, muss der Endanwender auf falsche Eingaben hingewiesen werden oder im Idealfall sogar daran gehindert werden. Die Formularfelder müssen dazu mit Restriktionen belegt werden, welche den gültigen Wertebereich für die Eingaben beschränken.

**Möglichkeiten der Restriktion** Es gibt verschiedene Varianten solcher Beschränken für Felder in Formularen.

- Untere und obere Schranken für Zahlenfelder
- Mindest- und Maximallänge für Textfelder
- Übereinstimmung mit regulärem Ausdruck
- Obligatorische Zeichen oder Zeichengruppen

Eine Angabe von Schranken bei Zahlenfeldern lässt nur sinnvolle Eingaben von Altersangaben zu. Mit regulären Ausdrücken kann die Korrektheit der Syntax bei Textfeldern geprüft werden. Und durch die obligatorische Verwendung bestimmter Zeichen bei Textfeldern kann die Sicherheit bei Eingabe von Passwörtern verbessert werden.

Für die korrekte Angabe der Restriktionen ist der Formularentwickler während der Erstellzeit verantwortlich. Es stellt sich jedoch die Frage, wie die Restriktionen umgesetzt werden. Durch die Verwendung der bereits diskutierten Anwendungslogik ist der Formularentwickler in der Lage bei jedem Feld während der Eingabe zu prüfen, ob der Endanwender korrekte Angaben macht. Dies führt allerdings zu einem erhöhten Aufwand bei der Erstellung, da für jedes beschränkte Feld eine aufwendige Logik notwendig ist. Besser ist es, die oben aufgezählten Restriktionsmöglichkeiten direkt an Formularfelder binden zu können. Der Formularentwickler kann dann beim Anlegen eines Formularfeldes direkt eine oder mehrere Beschränkungen mit angeben und muss keine zusätzliche Anwendungslogik zu jedem Feld erstellen.

**Probleme von Restriktionen** Wie bereits im Abschnitt über Anwendungslogik angesprochen, stellt sich auch hier die Frage nach der Kompatibilität mit unterschiedlichen Kontexten, in denen das Formular bearbeitet werden kann. Der Kontext muss die Möglichkeit Formularfelder zu beschränken unterstützen. Ist dies nicht der Fall, ist eine Überprüfung der Angaben während der Ausführungszeit nicht möglich. Der Inhalt des Formulars muss dann, nachdem der Endanwender die Bearbeitung abgeschlossen hat, bezüglich der vorhandenen Restriktionen geprüft werden. Wenn in den Eingaben des Endanwenders Fehler gefunden werden, muss dieser das Formular erneut bearbeiten, um die Fehler zu beseitigen. Dies macht zum Einen zusätzliche Prozessschritte nötig und zum Anderen erschwert dies die Bearbeitung des Formulars durch den Endanwender. Für diesen ist es besser, bereits bei der Eingabe auf Fehler hingewiesen zu werden.

**Zusammenfassung** Nicht alle Eingaben der Endanwender bei Formularen ergeben Sinn. Syntaktisch oder inhaltlich falsche Eingaben müssen durch Restriktionen verhindert werden. Der Formularentwickler kann dies durch Geschäftslogik oder durch direkte Angabe der Beschränkungen erreichen. Die Prüfung der Restriktionen kann während der Eingabe oder nach dem Abschluss der Arbeiten an dem Formular durchgeführt werden. Wie bei der Anwendungslogik gibt es die Problematik mit inkompatiblen Kontexten. Einer

Prüfung der Eingaben nach der Bearbeitung macht zusätzliche Prozessschritte nötig und führt zu einem größeren Aufwand für den Endanwender.

Die Komponente für Erstellung von Formularen ist so zu gestalten, dass der Formularentwickler die Möglichkeit hat, Beschränkungen ohne großen Aufwand umzusetzen. Für den Endanwender soll dies möglichst transparent und komfortabel sein.

#### 3.2.2.8 Wiederverwendung historischer Formulardaten

**Motivation** Ein Prozess besteht in der Praxis nicht nur aus einem einzelnen Prozessschritt mit einem Formular, sondern aus zahlreichen Prozessschritten mit vielen unterschiedlichen Formularen. Dabei stellen die einzelnen Formulare keine abgeschlossenen semantischen Einheiten dar. Vielmehr bauen die Formulare aufeinander auf. Die Inhalte der Formulare befassen sich dabei alle mit einer gemeinsamen Semantik, welche durch den gemeinsamen Prozess definiert ist. Viele Formulare greifen dabei auf dieselben Prozessschrittparameter zurück, oder nehmen aufeinander Bezug, indem ein Formular auf ein anderes Formular aus einem bereits abgearbeiteten Prozessschritt verweist.

**Probleme möglicher Lösungsansätze** Um den Endanwender bei seiner Arbeit mit vielen unterschiedlichen und komplexen Formularen zu unterstützen, ist es von Vorteil, wenn in Formularen das Ergebnis bereits abgearbeiteter Formulare des selben Prozesses mit dargestellt wird, wie es in Abbildung 32 dargestellt ist. Der Endanwender hat damit einen besseren Überblick über bisher im Prozess verwendeten Daten.

Hierfür gibt es die Möglichkeit diese Verantwortung entweder der Laufzeitumgebung oder dem Formularentwickler zu übertragen. Die Laufzeitumgebung kann für ein Formular die Ergebnisse aller bisherigen Formulare in das aktuelle Formular einfügen. Der Endanwender hat dann alle Informationen, welche er möglicherweise für die Bearbeitung des aktuellen Formulars benötigt. Dies hat allerdings auch den Nachteil, dass das Formular schnell unübersichtlich werden kann, wenn ein Prozess aus vielen Schritten besteht.

Eine Alternative ist die selektive Auswahl von Formularen und Formularfeldern durch den Formularentwickler bereits zur Erstellungszeit. Hierbei werden nur diejenigen Informationen dargestellt, welche zusätzlich zu den Prozessschrittparametern des Formulars zur Bearbeitung des Formulars relevant

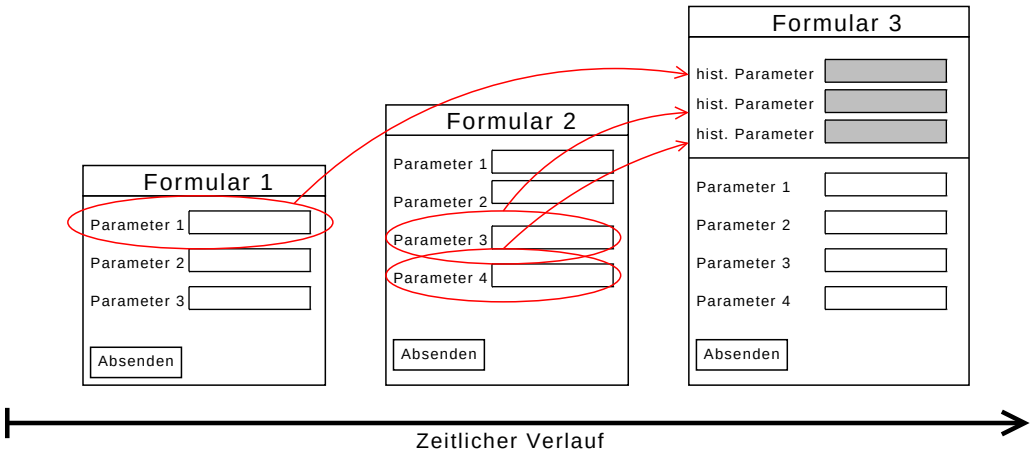


Abbildung 32: Verwendung von Formularfeldern vorangegangener Formulare

sind. Für den Formularentwickler führt dies jedoch zu einem zusätzlichen Aufwand durch das Aussuchen der Informationen.

**Anforderung** Es ist wichtig, dass die neue Komponente zur Erstellung von Formularen dazu in der Lage ist, den Endanwender mit allen Informationen zu versorgen, welche zusätzlich zu den gegebenen Prozessschrittparametern für eine erfolgreiche Bearbeitung des Formulars nötig sind. Hierbei soll verhindert werden, dass das Formular aufgrund der Informationsmenge unübersichtlich wird, was die Bearbeitung erschwert. Die Erstellung der Formulare hingegen darf für den Formularentwickler durch die Auswahl historischer Formulardaten nicht mit großem zusätzlichem Aufwand verbunden sein.

**3.2.2.9 Kontrolle der Komplexität** Der letzte Abschnitt hat unter anderem die Probleme der zunehmenden Komplexität von Formularen durch die Einbeziehung historischer Formularinformationen aufgezeigt. Dieser Abschnitt geht auf die allgemeine Komplexität von Formularen ein. Zunächt wird die Entstehung von Komplexität untersucht. Im Anschluss werd der mögliche Umgang mit Komplexität diskutiert.

**Entstehung von Komplexität** Selbst wenn ein Formular keine Daten vorangegangener Schritte beinhaltet, kann es vorkommen, dass ein Formular unübersichtlich wird. Dies ist dann der Fall, wenn ein Formular auf einen Prozessschritt mit zahlreichen Parametern basiert. Falls die Felder für diese

Parameter nicht zusammenhängend im Formular angeordnet werden oder nicht angeordnet werden können, führt das zu einer nicht unwesentlichen Erhöhung der Komplexität für den Endanwender, der das Formular bearbeiten muss.

Eine andere mögliche Ursache für zu umfangreiche Formulare ist die Komplexität der Parameter selbst. Wenn die Formulardaten auf Parametern mit strukturierten Datentypen basieren, wie sie im entsprechenden Abschnitt vorgestellt wurden, können diese unter Umständen nicht durch einfache grafische Elemente dargestellt werden. Es müssen komplexere grafische Elemente verwendet werden, um die Informationen aus den Parametern vollständig darzustellen. Erweist sich die Struktur eines solchen Datentyps als äußert umfangreich, mit einer hohen Schachtelungstiefe, wirkt sich das auch auf das Layout des Formulars selbst aus. Es werden mehr grafisch anzeigende Elemente benötigt und dem Endanwender fällt es schwerer den Überblick zu bewahren.

**Möglicher Umgang mit Komplexität** Komplexität entsteht bereits zur Entwurfszeit. Dies fängt jedoch nicht beim Erstellen des Formulars an, sondern bereits bei der Erstellung des Prozesses. Der einfachste Weg ein Formular überschaubar zu halten ist, Prozessschritte mit zu vielen Parametern oder mit Parametern, welche auf sehr komplexen Datentypen basieren, auf mehrere Prozessschritte aufzuteilen und somit auch die Komplexität auf mehrere Formulare aufzuteilen.

In manchen Situationen ist dies jedoch nicht möglich, beispielsweise wenn die Parameter eines Prozessschrittes logisch eine Einheit bilden. Wenn zusammenhängende Daten aufgeteilt werden, geht für die einzelnen Komponenten der logische Zusammenhang verloren. Ein alternativer Ansatz ist es, dem Formularentwickler die Möglichkeit zu geben, Parameter für ein Formular zusammenzufassen, um diese zu strukturieren. Somit ist es möglich, die Informationen für ein Formular logisch in Blöcken zu gruppieren. Diese Blöcke können dann beispielsweise in Subformularen dargestellt werden. Diese Subformulare wiederum können als separate Dokumente behandelt oder in einem einzelnen Formular durch ausblendbare Komponenten des Formulars dargestellt werden. Eine Lösung, welche den Formularentwickler mit einbezieht, führt jedoch auch hier zu einer Erhöhung des Aufwandes beim Erstellen des Dokumentes durch den Formularentwickler.

## 3.3 Zusammenfassung

Dieses Kapitel hat die wichtigsten Probleme und Fragen erläutert, die für einen Entwurf einer Komponente zur Erstellung von Formularen eine Rolle spielen. Zunächst wurde der Vorgang besprochen, der nötig ist, um Formulare zu erstellen, in den Prozess einzubinden und darzustellen. Dabei hat sich herausgestellt, dass die neue Komponente aus einer Entwurfsumgebung und einer Laufzeitumgebung besteht. Anschliessend sind die funktionalen und nichtfunktionalen Anforderungen erörtert worden. Die nichtfunktionalen Anforderungen hingegen betreffen:

- Einfache Bedienbarkeit
- Lizenz für die neue Komponente
- Integrationsfähigkeit in ein vorhandenes System

Für eine einfache Bedienbarkeit ist es beispielsweise wichtig, dass die Komponente dazu in der Lage ist, Formulare größtenteils selbstständig zu generieren, um den Formularentwickler bei seiner Arbeit zu unterstützen. Die funktionale Anforderungen betreffen:

- Grafische Kontexte
- Datentypen
- Grafische Elemente
- Integration von Anwendungslogik
- Lokalisierung von Formularen
- Anbindung externer Datenquellen
- Restriktionen für Formularfelder
- Wiederverwendung historischer Formularfelder
- Kontrolle der Komplexität

Die grafischen Kontexte haben dabei eine besondere Bedeutung, da diese massive Auswirkungen auf die anderen Anforderungen haben. Jeder grafische Kontext stellt nicht nur unterschiedliche grafische Elemente zur Verfügung, sondern ist auch für die Anwendungslogik, sowie Restriktionen verantwortlich. Es ist daher äusserst wichtig die Formulare während der Entwicklungszeit abstrakt zu beschreiben, um die Unterschiede zwischen verschiede-

nen grafischen Kontexten zu verbergen. Die Umsetzung dieser abstrakten Beschreibung ist die Aufgabe der Laufzeitumgebung.

# 4 Evaluierung vorhandener Systeme

Die zusammengestellten Anforderungen des letzten Kapitels sind die Voraussetzung für die Evaluierung vorhandener Systeme. Die Evaluierung zeigt die Stärken und Schwächen der einzelnen bereits verfügbaren Produkte auf. Dabei wird auch untersucht, ob die Produkte als Modul für die neue Komponente geeignet sind.

Zunächst wird ein Überblick über Produkte gegeben, welche sich mit der Benutzerinteraktion basierend auf Formularen beschäftigen und daher für eine Evaluierung im Rahmen dieser Arbeit geeignet sind. Anschliessend werden, aufbauend auf den bereits diskutierten Anforderungen, Kriterien für einen Vergleich der Produkte erläutert. Im darauf folgenden Abschnitt wird das Ergebnis dieses Vergleichs besprochen und die Vor- und Nachteile der einzelnen Systeme dargestellt.

## 4.1 Existierende Technologien und Produkte

Es gibt mehrere Produkte und Technologien, welche für eine Evaluierung im Rahmen dieser Arbeit beachtet werden müssen. Eine wichtige Produktgruppe stellen BPM-Systeme dar, welche bereits Formulare zur Benutzerinteraktion integriert haben. Ein Beispiele hierfür ist Inubit [Inu09a]. Bei Inubit handelt es sich um eine Suite für Modellierung, Simulation und Durchführung branchenübergreifender Geschäftsprozesse.

Zwei weitere Produkte, welche im Rahmen dieser Arbeit evaluiert werden sind Microsoft Infopath [Mic09f] und Adobe Lifecycle Designer [Ado09b]. Dabei handelt es sich jedoch nicht um BPM-Systeme. Diese Systeme sind von ihrer Idee her nicht dazu ausgelegt, Prozesse zu gestalten oder auszuführen, sondern konzentrieren sich auf den Aspekt der Oberflächenerzeugung.

Adobe Lifecycle Designer ist eine Softwarekomponente aus Adobe Lifecycle Enterprise Suite (ES) [Ado09c] und ermöglicht die Erstellung interaktiver Formulare, welche mit den aktuellen Versionen des Adobe Reader [Ado09d] ausgefüllt werden. Infopath ist ein Produkt von Microsoft, welches ebenfalls der Erstellung von interaktiven Formularen dient.

Neben diesen Produkten gibt es auch Produkte, die nicht alleinstehend arbeiten, sondern für die Integration in andere Programme vorgesehen sind, beispielsweise JAXFront [JAX]. Hierbei handelt es sich ebenfalls um ein Produkt zur Erstellung und Darstellung von Formularen für die Benutzerin-

teraktion. Das Einbeziehen der Formulare in einen Geschäftsprozess ist mit JAXFront jedoch nicht möglich. Dafür wird allerdings eine Schnittstelle angeboten, welche anderen Programmen die Integration von JAXFront erlaubt.

Ein weiteres Produkt, welches sich ausschliesslich mit der Erstellung von Formularen befasst, ist WJP Form [WJP09]. Dabei handelt es sich um einen Editor, der eingesetzt wird, um Formulare für den Prototypen von ADEPT2 zu gestalten. Dieses Produkt ist explizit auf ADEPT2 zugeschnitten und ist nicht für den Einsatz in aderen BPM-Systemen ausgelegt. Das Produkt verfolgt einen sehr pragmatischen Ansatz.

Neben diesen Produkten gibt es noch einen Standard des W3C (World Wide Web Consortium [W3C09a]), der sich ebenfalls mit Formularen beschäftigt. Dabei handelt es sich um XForms [W3C07], eine auf XML [W3C08] basierte Erweiterung von HTML für die Darstellung von Formularen.

Die nachfolgende Aufzählung fasst die zu evaluierenden Produkte und Technologien zusammen:

- Inubit
- Adobe Lifecycle Designer ES
- Microsoft Infopath
- JaxFront
- WJP Form
- XForms

## 4.2 Vergleichskriterien

Um die Stärken und Schwächen dieser Produkte ermitteln zu können, müssen Kriterien für die Untersuchung der einzelnen Produkte festgelegt werden. Ziel dieses Abschnittes ist es, einen Katalog von Vergleichskriterien zusammenzustellen, mit dessen Hilfe jedes Produkt bewertet wird. Die Basis dieses Katalogs bilden die Anforderungen aus Kapitel 3.

Das erste Kriterium, welches bei der Evaluierung untersucht wird, ist eine einfache Bedienbarkeit des Systems. Die Akzeptanz eines Systems hängt stark davon ab, wie schnell Anwender in der Lage sind, produktiv mit dem System zu arbeiten.

Ein weiteres Merkmal, welches bereits zu Beginn der Untersuchung eines Produkts überprüft wird, ist die Lizenz, unter der das Produkt steht. Dabei spielt nicht die Lizenz eine Rolle, welche zum Testen verwendet wird, sondern die Lizenz, unter welcher das Produkt produktiv eingesetzt werden kann. Hierbei wird auch untersucht, ob die Lizenz eine Integration in ein vorhandenes System erlaubt. Falls dies möglich ist, wird untersucht, wie die Integration technisch durchgeführt werden kann.

Nachdem ermittelt worden ist, ob eine technische Integration möglich ist, muss untersucht werden, ob das Produkt auch logisch integriert werden kann. Die Schnittstelle zwischen dem BPM-System und der neuen Komponente stellen die typisierten Parameter dar, welche einen Datentyp besitzen. Für die Formularerstellung muss das Produkt daher auch mit diesen Datentypen umgehen können, um die Parameter entsprechend ihres Datentyps korrekt darzustellen. Dies ist ein weiteres Kriterium.

Um auch Daten in das Formular zu integrieren, welche nicht im Datenfluß eines Prozesses vorkommen, wird auch die Möglichkait von externen Datenquellen untersucht. Dabei wird festgestellt, ob bei der Erstellung des Formulars beispielsweise Datenbankverbindungen angegeben werden können.

Anschliessend folgt die Untersuchung der verfügbaren grafischen Elemente. Dabei wird geprüft, welche grafischen Elemente zur Verfügung stehen und gegebenenfalls wie damit die Parameter eines Prozesses dargestellt werden. Da Parameter auch strukturierte Daten oder Listen sein können, ist ebenfalls wichtig, wie das Produkt Strukturen und Listen grafisch umsetzt.

Aufbauend auf der Evaluierung der grafischen Elemente der Produkte wird anschliessend geprüft, wie sich das Verhalten dieser Elemente steuern lässt und wie Anwendungslogik in das Formular integriert werden kann.

Um die Formulare später auch bei allen Endanwendern einsetzen zu können, ist es wichtig, dass das Produkt die Formulare für alle grafischen Kontexte erstellen kann, die das BPM-System unterstützt. Für dieses Kriterium wird deshalb ermittelt, für welche grafischen Kontexte die Formulare erzeugt werden können.

Ein weiterer Aspekt, der bei der Verteilung der Formulare an unterschiedliche Mitarbeiter eine Rolle spielt, ist die Sprache der Benutzer. Dafür muss das Formular an die bevorzugten Sprachen der einzelnen Mitarbeiter anpassbar sein. Für diesen Vergleichspunkt wird daher gepüft, ob eine Lokalisierung des Formulars angeboten wird und wie diese umgesetzt wird.

Die folgende Aufzählung fasst die Kriterien für den Vergleich nochmals zusammen:

- Einfache Bedienbarkeit durch den Formularentwickler
- Lizenz und Integrationsfähigkeit in vorhandene BPM-Systeme
- Unterstützte Datentypen
- Anbindung externer Datenquellen
- Verfügbare grafische Elemente
- Integration von Anwendungslogik
- Unterstützte grafische Kontexte
- Möglichkeit der Lokalisierung

## 4.3 Vergleich

### 4.3.1 Inubit

Inubit ist ein eigenständiges BPM-System. Dies beinhaltet auch die Verwaltung sogennanter Tasks. Tasks sind Aktivitäten, welche unter anderem die Interaktion mit dem Endanwender ermöglichen. Ein sogenannte Taskgenerator dient dabei der Erstellung von Formularen für diese Interaktion. Die Ein- und Ausgabeparameter für das Formular sowie die Beschreibung des Formulars selber werden mit Hilfe von XML- und CSV[RFC05]-Kontainern verwaltet.

**Bedienbarkeit** Für die Bedienung von Inubit sind umfangreiche technische Kenntnisse notwendig. Zum Einen sind Kenntnisse über die Konzepte des gesamten Systems notwendig. Dies zeigt sich unter anderem beim Anlegen neuer Formulare. Hier müssen für die Grundkonfiguration des Tasks Angaben gemacht werden, welche sich auf andere Aspekte des Systems beziehen, wie beispielsweise Organigramme [Inu09b]. Bei Organigrammen handelt es sich um Diagramme, welche die Organisationsstrukturen darstellen, etwa welche Bearbeiter welche Rollen einnehmen.

Zum Anderen basiert die Kommunikation zwischen dem BPM-System und den Formularen auf XML-Kontainern. Dies wird vor dem Formularentwickler jedoch nicht verborgen. Für die Gestaltung des Formulars muss der Inhalt des

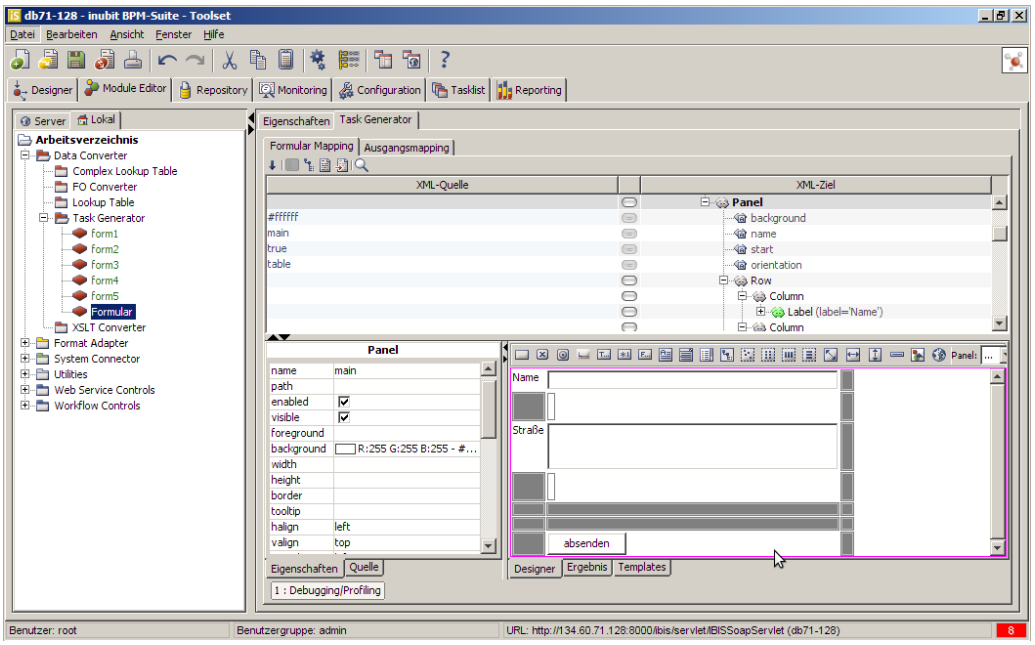


Abbildung 33: Entwicklungsumgebung für Formulare von Inubit

Kontainers teilweise direkt bearbeitet werden. Hierzu sind weitgehende XML-Kenntnisse notwendig. Darauf wird auch im Handbuch [Inu09b, Kap 4.1] hingewiesen.

Aufgrund dieser Anforderungen an den Entwickler erfüllt Inubit das Kriterium nach einer einfachen Bedienbarkeit nicht. Das Produkt ist von diesem Standpunkt aus nicht dazu geeignet, Formulare innerhalb der neuen Komponente zu erstellen. Abbildung 33 zeigt die Oberfläche der Formularentwicklungsumgebung von Inubit auf der unter anderem rechts oben eine Baumansicht für den XML-Kontainer zu sehen ist.

**Lizenz und Integrationsfähigkeit** Bei *Inubit* handelt es sich um ein kommerzielles Produkt, welches nicht weitergegeben oder kopiert werden darf. Dieses Produkt ist aus Modulen aufgebaut. Auch wenn es möglich ist, nur einzelne Module zu lizenzieren, sind diese logisch in das BPM-System integriert und nur in dessen Kontext lauffähig. Eine Integration in ein anderes BPM-System ist daher nicht möglich.

**Unterstützte Datentypen** Die Prozessschrittparameter werden den Formularen entwender in Form eines XML- oder eines CSV-Kontainers über-

geben. Inbuit biete hierfür auch eine Konvertierung zwischen diesen beiden Kontainern an. Innerhalb des XML-Kontainers beispielsweise werden die Struktur und die Werte der Parameter beschrieben. Somit stehen den Formularen alle Datentypen zur Verfügung, welche sich durch XML beschreiben lassen. Dies schliesst auch beliebig strukturierte und listenwertige Datentypen ein.

**Externe Datenquellen** In Inubit gibt es sogenannte *Connectors*, die es ermöglichen auf Datenbestände ausserhalb des BPM-Systems zuzugreifen und innerhalb des Systems zu nutzen. Diese Daten können anschliessend in einem XML- oder CSV-Kontainer in ein Formular integriert werden.

**Grafische Elemente** Für die Gestaltung des Formulares und die Abbildung der Parameter des Formulares stehen dem Entwickler folgende grafische Elemente zur Verfügung:

- Kontrollkästchen
- Auswahlfelder
- Beschriftungsfelder
- Felder für Passwörter
- Elemente für die Darstellung von Bildern
- Auswahllisten
- Schaltflächen

Für listenwertige Texte stehen ausserdem Listenfelder zur Verfügung. Für strukturierte Datentypen und strukturierte listenwertige Datentypen gibt es eine Baumansicht, welche die Struktur des Datentypes grafisch darstellt. Ein weiteres grafisches Element erlaubt das Einbinden von Dateien.

Mit diesen grafischen Elementen hat Inubit einen ausreichenden Vorrat für die Präsentation von Informationen. Dadurch, dass für strukturierte Datentypen nur eine Baumansicht verwendet werden kann, fehlt es dem Produkt jedoch an Flexibilität, was die Darstellung strukturierter Datentypen angeht.

**Anwendungslogik** Inubit besitzt eine Laufzeitumgebung für JavaScript. Diese Umgebung ermöglicht die dynamische Ausführung von JavaScript-Programmen während der Laufzeit eines Prozesses. Das Skript wird dabei direkt

in den XML-Kontainer des Formulars eingebettet. Bei Schaltflächen innerhalb des Formulars, besteht die Möglichkeit den hinterlegten JavaScript-Code bei einem Auslösen der Schaltfläche auszuführen. Auf diese Weise lässt sich die Anwendungslogik in Form von JavaScript-Programmen in das Formular integrieren.

**Grafische Kontexte** Die erzeugten Formulare lassen sich für zwei unterschiedliche grafische Kontexte erstellen:

- HTML-Kontext
- Swing-(Java-)Kontext

Damit lassen sich Formulare bereits in einem großen Umfeld einsetzen. Die Gestaltung des Formulars ist jedoch nicht vollständig kontextunabhängig. In beiden Kontexten gibt es unterschiedliche grafische Elemente. Ein weiterer Nachteil ist, dass der Kontext, in welchem das Formular ausgeführt werden soll, bereits zur Entwicklungszeit festgelegt werden muss.

**Lokalisierung** Es ist möglich, statische Texte des Formulares sprachlich an bestimmte Benutzergruppen anzupassen. Hierfür lassen sich Texte, welche im Formular verwendet werden, automatisch in eine Excel-Tabelle exportieren. Diese Tabelle muss anschliessend manuell übersetzt und unter Angabe der Sprache wieder in das Formular importiert werden.

### 4.3.2 Adobe Lifecycle

**Bedienbarkeit** Adobe Lifecycle bietet eine übersichtliche Benutzeroberfläche, welche intuitiv zu bedienen ist. Das Erstellen von Formularen ist durch einen sogenanten WYSIWYG (What you see is what you get)-Editor [Wik09b] möglich (Abbildung 34). Ein Bearbeiten von XML-Kontainern, wie bei Inubit, ist nicht erforderlich.

**Lizenz und Integrationsfähigkeit** Adobe Lifecycle Designer ist Bestandteil von Adobe Lifecycle SE. Allerdings ist Lifecycle Designer auch als alleinstehendes Produkt verfügbar und lauffähig. Das Produkt bietet jedoch keine Programmierschnittstellen, um es in bestehende Systeme zu integrieren.

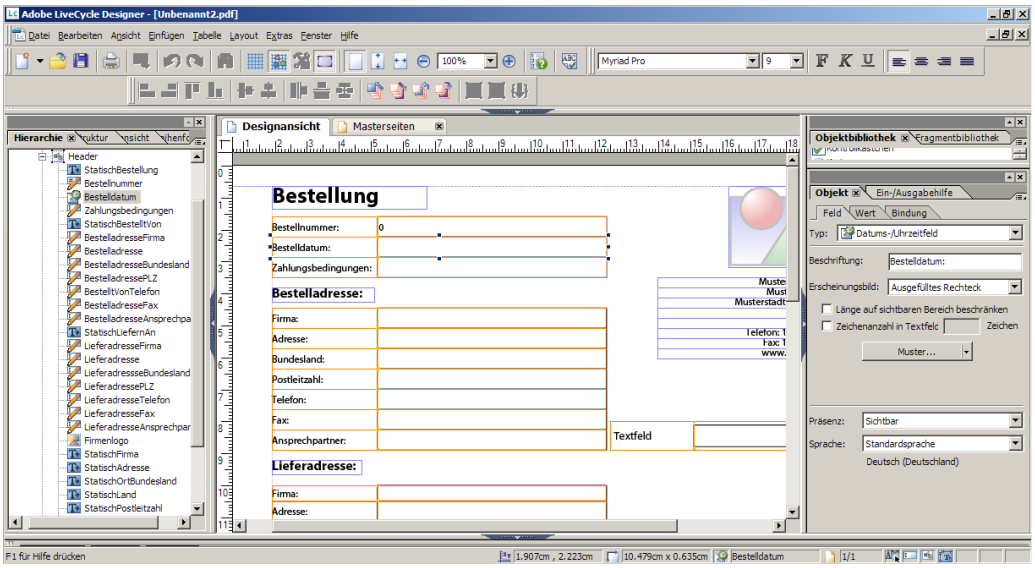


Abbildung 34: Entwicklungsumgebung für Formulare von Adobe Lifecycle Designer

**Unterstützte Datentypen** In Adobe Lifecycle Designer werden die Informationen nicht anhand der Datentypen unterschieden. Welchen Typ ein Datum hat, hängt allein von der grafischen Elementen, welche für die Anzeige verwendet wird, ab. Es gibt jedoch die Möglichkeit, die Benutzereingaben durch Angabe eines Musters einzuschränken. Damit lässt sich ein Textfeld beispielsweise dazu verwenden, um eine Adresse oder ein Autokennzeichen in einem bestimmten Format anzugeben.

Für die Verwendung in einer neuen Komponente ist Adobe Lifecycle Designer nicht geeignet, da es für eine Integration erforderlich ist, dass das Produkt mit den Datentypen der Prozessschrittparameter umgehen kann.

**Externe Datenquellen** Um Informationen dynamisch in das Formular zu integrieren, stehen sogenannte Datenverbindungen zur Verfügung. Diese Datenverbindungen erlauben das Importieren von Informationen aus folgenden Quellen:

- Datenbanken
- XML-Dateien
- Web-Services

Abhängig von der gewählten Datenverbindung lassen sich noch weitere Details angeben, um spezifische Datensätze aus der Datenverbindung zu erhalten. Bei einer Datenbank ist beispielsweise die Angabe von Abfrage-Kommandos möglich. Eine Datenverbindung kann dabei direkt auf ein grafisches Elemente abgebildet werden, um die Informationen aus der Datenverbindung mit diesem Element darzustellen.

**Grafische Elemente** Adobe Lifecycle Designer bietet eine sehr große Auswahl an grafischen Elementen für die Verwendung in Formularen. Dies beinhaltet unter anderem auch Elemente, welche häufig in Anwendungen zu finden sind. Typische Vertreter sind:

- Listenfelder
- Textfelder
- Optionsfelder
- Kombinationsfelder
- Schaltflächen
- Tabellen

Es sind auch grafische Elemente vorhanden, die weniger häufig verwendet werden. Diese sind für die Darstellung von Parametern mit speziellen Datentypen geeignet. Einige Beispiele dafür sind:

- Eingabefelder für E-Mail-Adressen
- Auswahlfelder für Länder
- Barcodes
- Felder für digitale Unterschriften
- geometrische Elemente wie Kreise oder Rechtecke

Ausserdem können sogenante Fragmente erstellt werden. Ein Fragment ist ein wiederverwendbarer Ausschnitt eines Formulars. Abbildung 35 zeigt einige der verfügbaren grafischen Elemente.

**Anwendungslogik** Jedes Steuerelement kann mehrere unterschiedliche Ereignisse auslösen, zum Beispiel ein Ereignis, welches das Ändern des Inhalts durch den Benutzer signalisiert. Für jedes Ereignis kann durch ein Skript

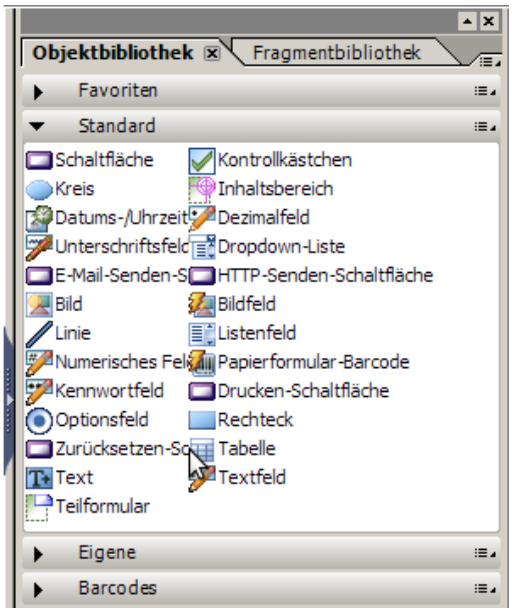


Abbildung 35: Grafische Elemente für Formulare in Adobe Lifecycle Designer

das Verhalten beim Eintreten des Ereignisses festgelegt werden. Als Skriptsprachen stehen dabei JavaScript, sowie *FormCalc* [Ado09a] zur Verfügung. *FormCalc* ist eine von Adobe entwickelte Skriptsprache für Formulare. Damit bietet Adobe Lifecycle Designer ausreichende Funktionalität für den Einsatz in einer neuen Komponente.

**Grafische Kontexte** Adobe Lifecycle Designer erstellt aus den Formularen PDF-Dateien, welche die Formulare darstellen. In Abbildung 36 wird ein PDF dargestellt, das aus einem Formular erstellt wurde. Ausgabeformate für andere grafische Kontexte sind nicht möglich. Die Erstellung von PDF-Dateien ist jedoch nicht ausreichend für eine Verwendung in der neuen Komponente.

**Lokalisierung** Die Angabe von Texten in unterschiedlichen Sprachen ist nicht möglich.

### 4.3.3 Microsoft Infopath

**Bedienbarkeit** Das Produkt ist intuitiv zu bedienen und bietet für Anwender anderer Microsoftprodukte eine vertraute Umgebung. Zum Erstellen

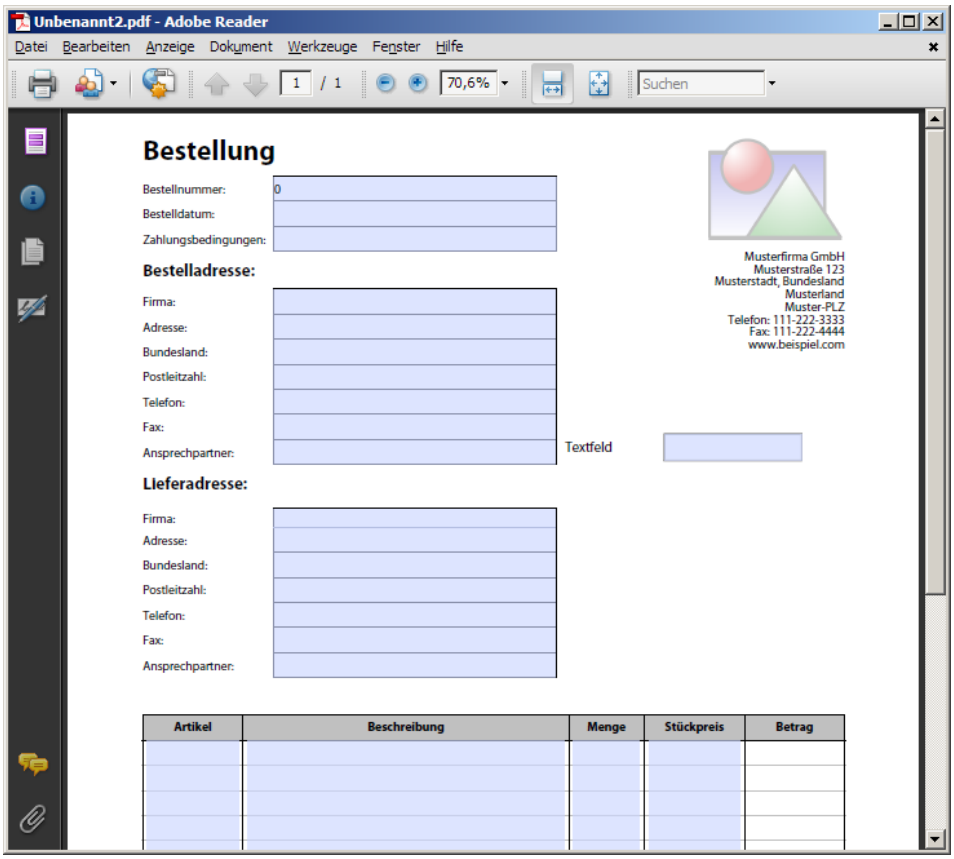


Abbildung 36: Mit Adobe Lifecycle Designer generiertes PDF-Dokument

von Formularen steht hier ebenfalls ein WYSISWYG-Editor zur Verfügung. Abbildung 37 zeigt die Entwicklungsumgebung für Formulare.

**Lizenz und Integrationsfähigkeit** Microsoft Infopath ist unabhängig von anderen Produkten lauffähig, lässt sich jedoch mit anderen Produkten von Microsoft kombinieren. Microsoft SharePoint [Mic09g] (Dokumentenmanagementsystem, mit Weboberfläche zum verteilten arbeiten an gemeinsam genutzen Dokumenten) kann beispielsweise genutzt werden, um die Formulare öffentlich zugänglich zu machen. Für das Anlegen von zusätzlichen Steuerelementen oder die Integration von Anwendungslogik wird eine .NET-Entwicklungsumgebung [Mic09b] benötigt. Eine Integration von Infopath in bereits existierende Systeme ist nicht möglich.

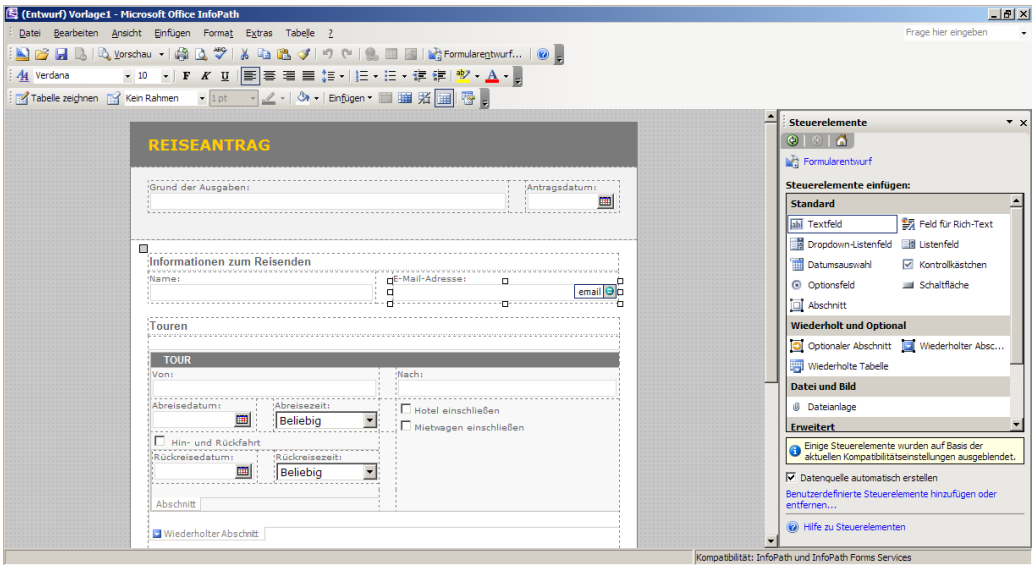


Abbildung 37: Entwicklungsumgebung für Formulare von Microsoft Infopath

**Unterstützte Datentypen** Microsoft Infopath unterstützt folgende Datentypen:

- String
- Integer
- Double
- Boolean
- URI
- Date
- Time
- DateTime

Strukturierte oder listenwertige Datentypen werden nicht untersützt. Für eine Verwendung in einer neuen Komponente ist das Produkt daher nur bedingt geeignet.

**Externe Datenquellen** Microsoft Infopath bietet die Möglichkeit, Informationen aus verschiedenen externen Datenquellen dynamisch in das Formular zu integrieren. Zur Auswahl stehen dabei folgende Quellen:

- Datenbankanfragen (Nur Microsoft SQL-Server [Mic09c])
- Webservices
- XML-Dateien
- Inhalte einer Microsoft SharePoint-Bibliothek

Ebenso wie bei Adobe Lifecycle Designer ist die Auswahl eines definierten Datensatzes möglich. Es ist auch möglich, während der Laufzeit des Formulars Informationen an externe Dienste zu senden.

**Grafische Elemente** Für die Gestaltung des Layouts stehen dem Entwickler folgende grafische Elemente zur Verfügung:

- Felder für einfachen Text
- Felder für formatierten Text
- Kombinationsfelder
- Listenfelder
- Felder für Datumsauswahl
- Kontrollkästchen
- Optionsfelder
- Schaltflächen
- Kontainer für Dateien

Ausserdem können selbst erstellte grafische Elemente als Formularfelder verwendet werden. Das Erstellen eigener grafischer Elemente ist in Infopath jedoch nicht möglich. Dafür wird eine separate Entwicklungsumgebung benötigt.

Des weiteren bietet Infopath die Möglichkeit optionale oder sich wiederholende grafische Elemente zu verwenden. Dadurch lassen sich auch ohne listenwertige Datentypen Listen von beliebigen Formularfeldern realisieren, wie zum Beispiel eine Einkaufsliste.

**Anwendungslogik** Um das Verhalten des Formulares zu beeinflußen, kann jedes Formularfeld durch sogenannte Regeln auf Benutzereingaben reagieren und dabei bestimmte Aktionen ausführen. Folgende Aktionen sind möglich:

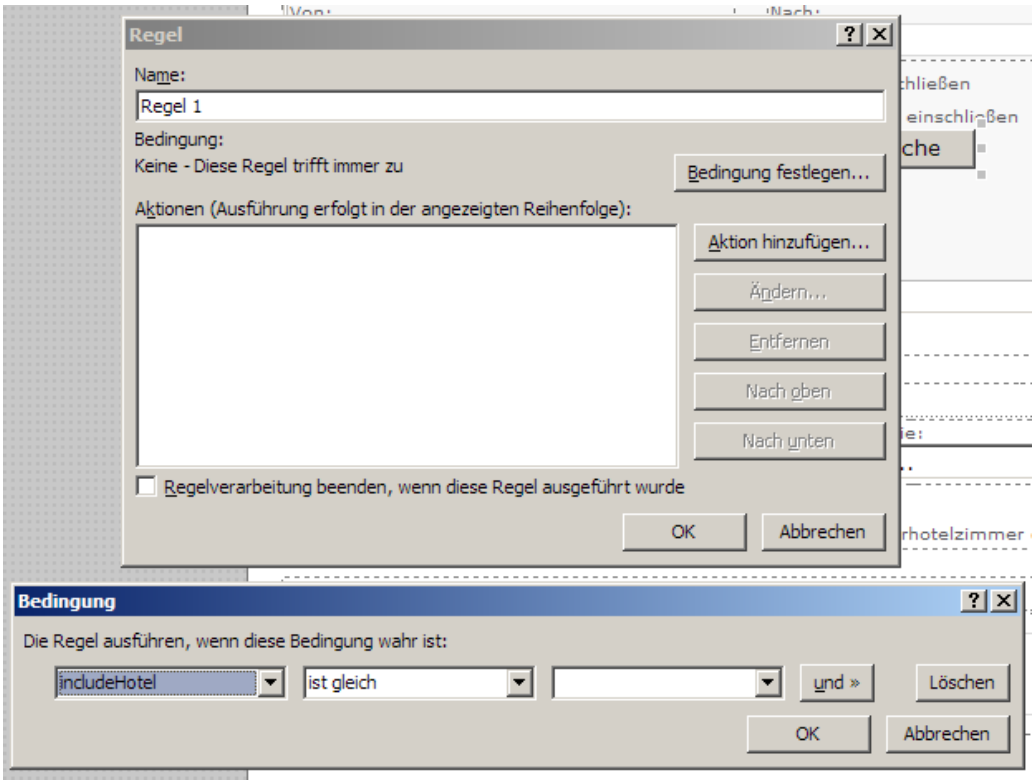


Abbildung 38: Dialoge zum Erstellen von Regeln für Formularfelder in Microsoft Infopath

- Dialogfeldmeldungen anzeigen
- Wert eines Formularfeldes setzen
- Daten einer externen Quelle abfragen
- Daten zu einer externen Quelle senden.
- Öffnen eines weiteren Formulars

Beim Aktivieren einer Schaltfläche ist ausserdem auch die Angabe von Anwendungslogik möglich. Hierfür wird die Skriptsprache VBA (Visual basic for Application [Mic09e]) verwendet. Abbildung 38 zeigt einen Dialog zum Erstellen von Regeln für ein grafisches Element.

**Grafische Kontexte** Das Formular wird wahlweise in einem proprietärem Format auf dem lokalem Arbeitsplatz gespeichert oder auf einem SharePoint-Server hinterlegt. In beiden Fällen wird für das Darstellen des Formulars ein

eigener Betrachter verwendet. Es ist jedoch auch möglich, das Formular in andere Formate zu exportieren. Dafür müssen jedoch entsprechende Exportfilter nachinstalliert werden. Diese können auch selbst entwickelt werden.

Eine Verwendung der erstellten Formulare für die neue Komponente ist daher umständlich, jedoch durchaus möglich.

**Lokalisierung** Das Einstellen der Sprache der Formularfelder ist zur Entwurfszeit möglich. Ein Anpassen der Sprache zur Laufzeit des Formulars wird jedoch nicht unterstützt.

### 4.3.4 JAXFront

JAXFront ist ein Produkt zum Generieren grafischer Oberflächen auf Basis eines XML-Kontainers. Der Endanwender hat dabei die Möglichkeit die Daten dieses Kontainers zu bearbeiten. Im Gegensatz zu den XML-Kontainern von Inubit wird die Struktur von XML jedoch vor dem Entwickler verborgen.

Die Architektur von JAXFront sieht eine Trennung der Oberflächenbeschreibung, dem Geschäftsmodell und der Instanzdaten vor. Das Geschäftsmodell beschreibt den syntaktischen Aufbau der Informationen in Form eines XML-Schemas. Die Oberflächenbeschreibung legt die grafische Repräsentation dieser Informationen fest und beinhaltet Regeln für das Verhalten der Oberfläche. Die Instanzdaten legen zum Einen die Werte fest, die beim Starten der Oberfläche dargestellt werden und zum Anderen die Werte, die der Benutzer eingegeben hat. Diese Daten werden in Form eine XML-Datei gespeichert. Abbildung 39 stellt die Architektur von JAXFront dar.

Das Produkt besteht aus zwei Architekturkomponenten:

1. Dem XUI-Editor, der es dem Entwickler erlaubt ausgehend von dem Geschäftsmodell eine grafische Oberfläche zu erstellen.
2. Einem Renderer, der aus dem Geschäftsmodell, der Oberflächenbeschreibung und den Instanzdaten die eigentliche Oberfläche erstellt.

Das Erstellen einer Oberflächenbeschreibung für das Geschäftsmodell ist dabei optional. Der Renderer ist in der Lage, basierend auf dem Geschäftsmodell, eine Oberfläche selbständig zu generieren. Wird in der erstellten Oberfläche jedoch ein bestimmtes Verhalten benötigt, muss die entsprechende Anwendungslogik durch den XUI-Editor manuell hinzugefügt werden.

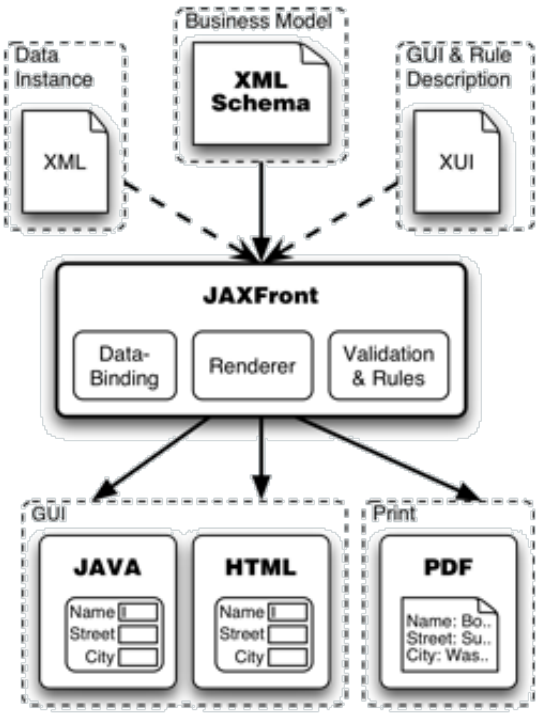


Abbildung 39: Architektur von JAXFront [JAX]

Um die Oberflächenbeschreibung mit Hilfe des XUI-Editors zu erstellen oder durch den Renderer automatisch zu generieren, muss die XML-Datei mit Geschäftsmodell bereits vorliegen. Ein Editor um dieses zu erstellen ist nicht vorhanden.

**Bedienbarkeit** Der XUI-Editor ermöglicht das Anlegen und Bearbeiten der grafischen Elemente, welche zur Darstellung der einzelnen Elemente aus dem XML-Schema des Geschäftsmodells dienen. Die Möglichkeiten dabei sind sehr vielfältig. Es ist dafür eine umfangreiche Dokumentation vorhanden, welche alle wichtigen Aspekte beschreibt.

Mit diesem Editor können jedoch nur Angaben über die Eigenschaften der grafischen Elemente gemacht werden. Ein einfache Gestaltung der Oberfläche in Form eines WYSIWYG-Editors ist nicht möglich. Abbildung 40 zeigt den XUI-Editor von JAXFront. Leider zeigte der Editor in der vorliegenden Version ausgeprägte Schwächen hinsichtlich seiner Stabilität. Um JaxFront produktiv einsetzen zu können, müssen diese Schwächen seitens der Hersteller behoben werden.

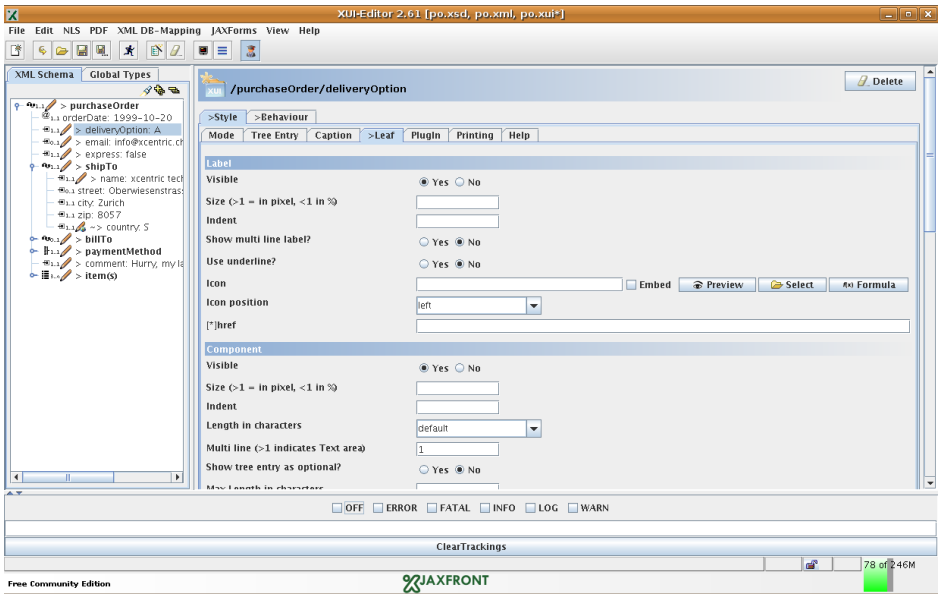


Abbildung 40: XUI-Editor von JAXFront

**Lizenz und Integrationsfähigkeit** Der Editor von JAXFront ist ein alleinstehendes Produkt und kostenlos verfügbar. Der Renderer ist in fünf verschiedenen Edition verfügbar. Tabelle 1 zeigt eine Übersicht dieser Edition und der entsprechenden Kosten für die Lizenz.

Bis auf die „Formserver Edition“ sind alle Editionen für die Integration in bestehende Systeme ausgelegt. Der Formserver ist ein Form-Solution-Service auf Basis von J2EE [Sun06]. Dabei handelt es sich um ein vollständiges Produkt zur Erstellung, Verwaltung und Verteilung von Formularen. Dies beinhaltet auch einen Datenhaltungsdienst und einen einfachen Workflow-Dienst.

Die „Community Editon“ ist frei erhältlich, um das Produkt testen und evaluieren zu können. Diese Edition darf auch in vorhandene Produkte integriert werden. Jedoch wird beim Rendern der Oberflächen ein Markenzeichen eingeblendet. Die restlichen Edition sind der „Community Edition“ ähnlich, bieten jedoch einen höheren Funktionsumfang und beinhalten kein Markenzeichen auf den Oberflächen.

Für die Integration in Java-Projekte steht eine spezielle Schnittstelle bereit, die es erlaubt den Renderer zu verwenden.

| Edition | Lizenzkosten |
|---|---|
| Free Community Edition | 0,- |
| Standard Edition | 600,- |
| Professional Edition | 1.200,- |
| Enterprise Edition | 5.400,- |
| Formserver Edition | 10.800,- |

Tabelle 1: Übersicht über die verfügbaren Editionen des JAXFront Renderers und der Lizenzkosten in Euro

**Unterstützte Datentypen** Die Datentypen für die Informationen werden im XML-Schema des Geschäftsmodells festgelegt. Dadruch können alle Datentypen verwendet werden, welche in XML möglich sind.

**Externe Datenquellen** In JAXFront ist es möglich Teile der Instanzdaten von externen Datenbanken zu beziehen. Dies setzt das Vorhandensein eines geeigneten JDBC (Java Database Connectivity [Sun09c]) voraus. Damit ist es möglich, bestimmte Felder aus den Tabellen der Datenbank auf die Instanzdaten für ein Formularabzubilden. Die Werte, die sich zur Laufzeit in den angegebenen Feldern der Tabelle befinden, werden in das Formular integriert.

**Grafische Elemente** JAXFront bietet für alle Datentypen aus dem Geschäftsmodell ein oder mehrere adäquate grafische Elemente zu deren Darstellung an. Stehen für den Datentyp eines Elementes mehrere grafische Repräsentationen zur Verfügung, hat der Entwickler die Möglichkeit ein passendes Elemente zu wählen. Elemente für das Exportieren oder Importieren von Dateien gibt es nicht.

**Anwendungslogik** Für jedes Element können unabhängig von dem verwendeten grafischen Element Regeln erstellt werden mit deren Hilfe das Verhalten der Oberfläche angepasst werden kann. Dabei ist es möglich, Bedin-

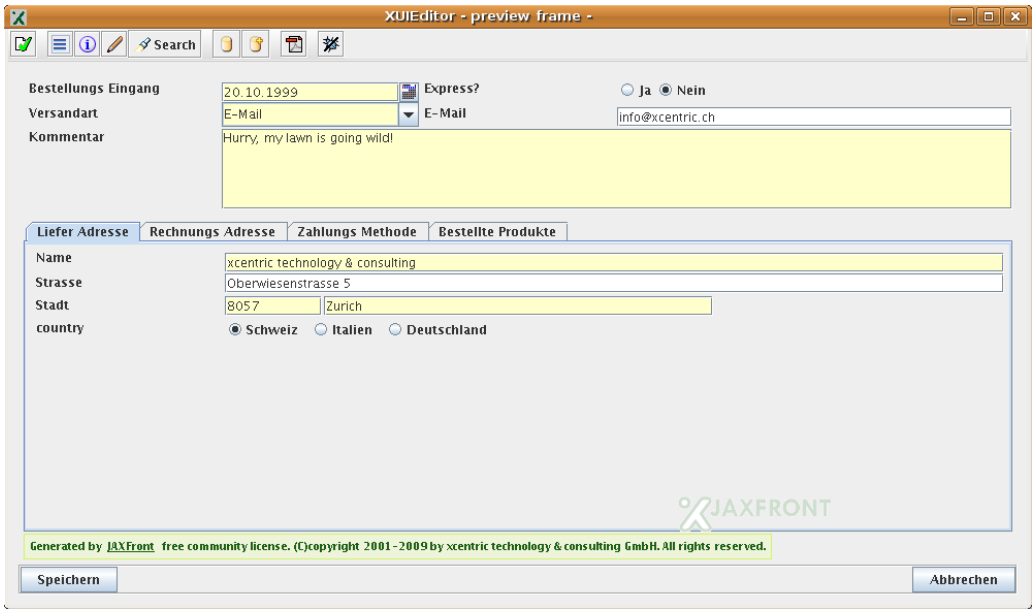


Abbildung 41: Oberfläche für Swing gerendert

gungen und Aktionen festzulegen. Bei den Aktionen stehen vordefinierte Methoden bereit, welche durch den Entwickler ausgewählt werden können, beispielsweise das Ändern von Eigenschaften eines Formularfeldes. Die Angabe einer Java-Klasse mit zusätzlicher Anwendungslogik ist ebenfalls möglich. Die Klasse muss dafür eine bestimmte Schnittstelle von JAXFront implementieren.

**Grafische Kontexte** Der Renderer unterstützt das Darstellen der Oberfläche für die folgenden Kontexte:

- PDF
- HTML
- Swing (Java)

Damit bietet JAXFront ein breites Spektrum an Einsatzmöglichkeiten. Welche Kontexte tatsächlich genutzt werden können, hängt von der verwendeten Edition ab. Abbildung 41 zeigt eine Oberfläche, die für den Swing-Kontext gerendert ist. Im Gegegnsatz zu Inubit muss zur Entwurfszeit noch kein grafischer Kontext festgelegt werden. Diese Wahl wird zur Laufzeit durch das integrierende System getroffen.

**Lokalisierung** Alle Texte der grafischen Oberfläche können durch einen *National Language Support* (NLS) in mehreren Sprachen angegeben werden. Zur Laufzeit kann die entsprechende Sprache durch das Host-System ausgewählt werden.

### 4.3.5 WJP Form

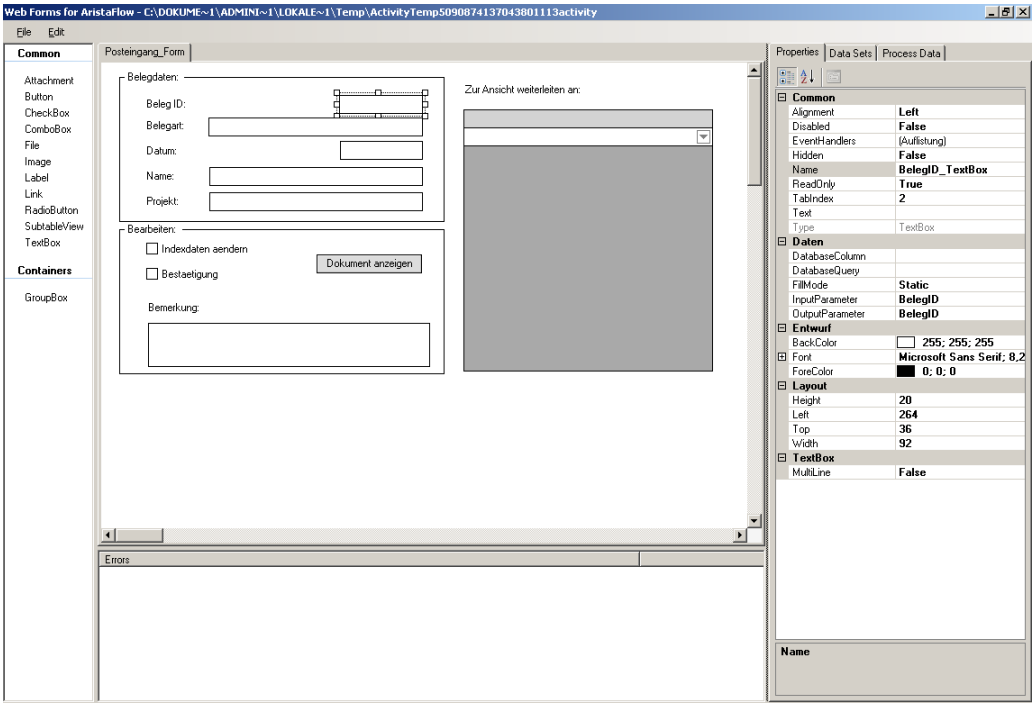


Abbildung 42: Editor für Formulare von WJP

**Bedienbarkeit** Die Bedienung von WJP Form ist intuitiv und schnell erlernbar. In WJP Form steht ein WYSIWYG-Editor zur Verfügung um die Oberfläche der Formulare zu erstellen. Der Editor wird in Abbildung 42 gezeigt.

**Lizenz und Integrationsfähigkeit** WJP Form wurde speziell für die Integration in die auf ADEPT2-beruhende kommerzielle „AristaFlow BPM Suite“ [Ari09] entwickelt. Eine Integration in ein anderes System ist daher nicht vorgesehen. Es ist zwar möglich, eine WJP Form-Lizenz zu erwerben. Sinnvoll ist dies jedoch nur mit dem oben genannten Produkt von AristaFlow

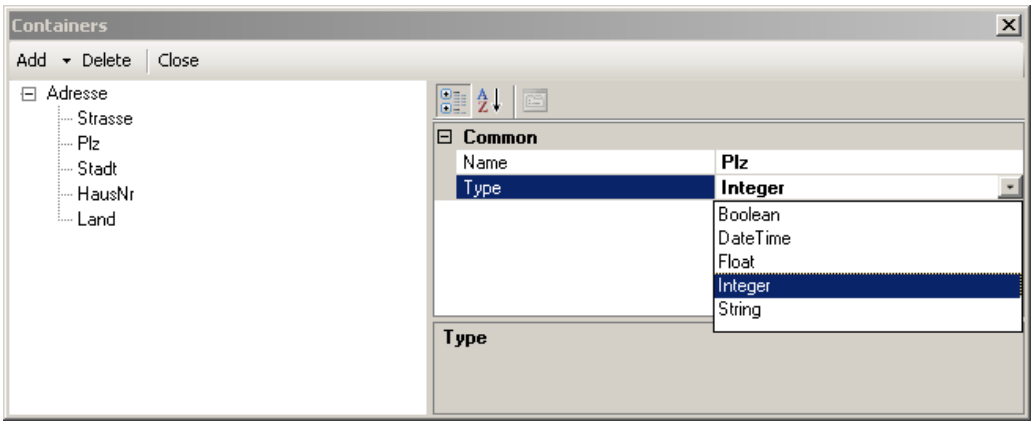


Abbildung 43: WJP Editor für *Container*

**Unterstützte Datentypen** Es werden die Datentypen aus dem Metamodell von ADEPT2 unterstützt.

- Boolean
- DateTime
- Float
- String
- Integer
- UDT

UDT-Datentypen (User-Defined-Datatype) sind Datentypen mit einer frei definierbaren Struktur. Diese können mit WJP Form genauer spezifiziert werden. Hierfür steht ein Editor zur Verfügung mit dem sogenante Kontainer angelegt werden können. Dabei handelt es sich um hierarchische Strukturen deren Elemente, aus den Basisdatentypen bestehen. Die Prozessschrittparameter können dabei mit solchen Kontainern assoziiert werden, um auf die Elemente der Parameter zugreifen zu können. Abbildung 43 zeigt den Editor für die Kontainer. Listenwertige Datentypen werden vom ADEPT2-Modell momentan nicht unterstützt und stehen daher bei WJP Form nicht zur Verfügung.

**Externe Datenquellen** Um zusätzliche Informationen in das Formular zu integrieren, welche nicht als Prozessschrittparameter zur Verfügung stehen, können Datenbankverbindungen zu Instanzen des Microsoft-SQL Server

angegeben werden. Für diese Verbindungen lassen sich zusätzlich Kommandos erstellen, welche Anfragen an die Datenbank darstellen und zur Laufzeit des Formulars Informationen aus der Datenbank abrufen. Abbildung 44 zeigt eine Datenbankverbindung und ein Kommando für eine Abfrage innerhalb dieser Datenbank.

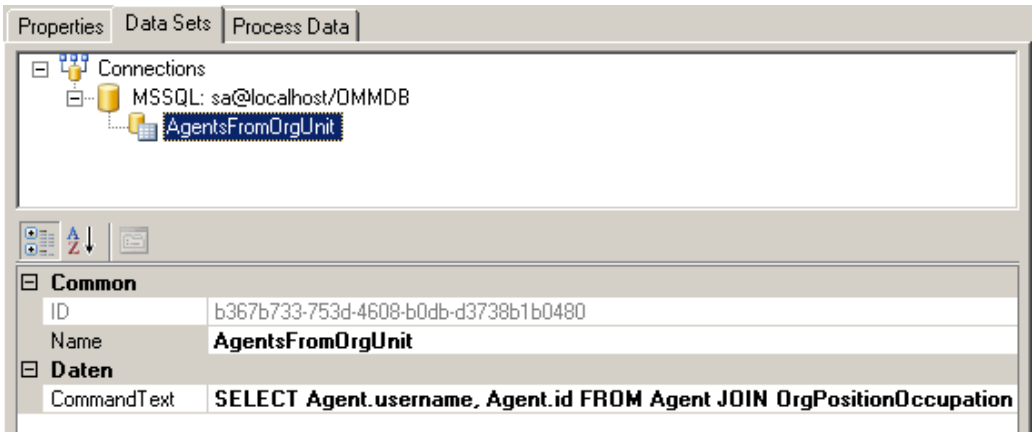


Abbildung 44: Übersicht über externe Datenbankverbindungen in WJP Form

**Grafische Elemente** Für die Gestaltung des Layouts stehen folgende grafischen Elemente zur Verfügung:

- Schaltflächen
- Listenfelder
- Textfelder
- Optionsfelder
- Beschriftungsfelder
- Hypertextfelder
- Elemente für den Im- und Export von Dateien
- Steuerelemente zur Darstellung von Bildern
- Kombinationsfelder

Die Prozessschrittparameter können dabei auf die im Formular verwendeten grafischen Elemente abgebildet werden und dienen als Eingabedaten für diese. Bei Prozessschrittparametern, die mit einem Kontainer assoziiert sind,

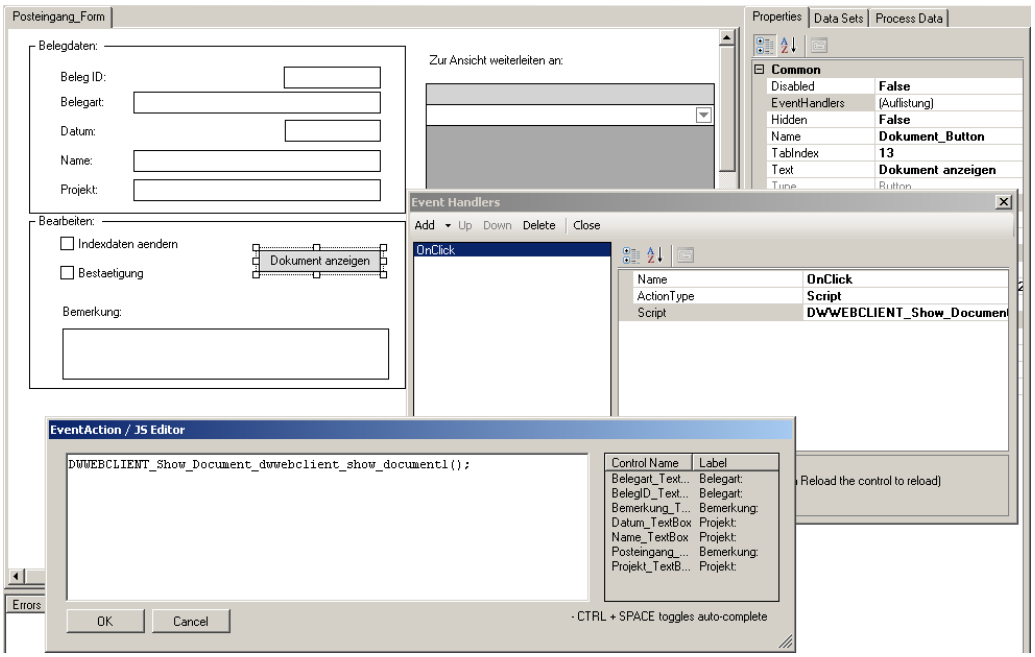


Abbildung 45: Dialog für die Behandlung von Ereignissen eines grafischen WJP-Elements

können die Elemente des Kontainers als Eingabedaten für ein grafisches Element verwendet werden. Alternativ können auch externe Datenquellen als Eingabedaten für die grafischen Elemente verwendet werden. Zur Laufzeit des Formulars stellen die grafischen Elemente die Werte der ihr zugeordneten Parameter oder Datenquellen dar.

Eine Erstellung von eigenen grafischen Elementen für die Wiederverwendung in anderen Formularen ist nicht möglich.

**Anwendungslogik** Für die Steuerung des Verhaltens von Formularen steht die Verwendung von JavaScript oder VB (Visual Basic [Mic09d]) zur Verfügung. Alle grafischen Elemente können unterschiedliche Ereignisse auslösen. Diese Ereignisse können entweder Behandlungsroutinen in Form von Skripten ausführen oder ein erneutes Laden der Eingabedaten der grafischen Elemente auslösen. Durch ein erneutes Laden der Eingabedaten werden Werte aktualisiert, welche aus einer externen Datenquelle bezogen werden. Abbildung 45 zeigt im Hintergrund rechts der Mitte einen Dialog der die Verwaltung der Ereignisse von grafischen Elementen ermöglicht. Im Vorder-

grund links unten ist ein Editor für die Behandlungsroutine in JavaScript zu sehen.

Für häufig verwendetes Verhalten in einem Formular können auch globale Methoden definiert werden, welche innerhalb der Behandlungsroutine aufgerufen werden können.

**Grafische Kontexte** Das Formular kann zur Laufzeit nur als HTML-Dokument dargestellt werden. Das ist nicht ausreichend für dein Einsatz in einer Komponente zur Formularerstellung.

**Lokalisierung** Eine Lokalisierung des Formulars ist nicht möglich.

### 4.3.6 XForms

XForms ist eine Spezifikation des W3C-Konsortiums. Es handelt sich dabei um einen Standard Formulardarstellung. Eine mögliche Anwendung ist die Integration von XForms in (X)HTML-Dokumente. Ein Web-Browser der dieses Dokument darstellen soll, muss in der Lage sein XForms zu interpretieren. Quellcode 5 zeigt wie ein XForms-Dokument in ein HTML-Dokument integriert wird. In Abbildung 46 ist die Darstellung dieses Dokuments in einem XForms-fähigen Web-Browser zu sehen. Als Web-Browser wird Firefox [Moz09] verwendet. Damit dieser jedoch das Formular darstellen kann, ist ein zusätzliches Plugin [Ree08] nötig.

```
1  <html xmlns="http://www.w3.org/1999/xhtml"
2        xmlns:xf="http://www.w3.org/2002/xforms">
3      <head>
4          <title>2.1.a Introductory Example No. 1</title>
5          <link rel="stylesheet" href="../driverPages/forms
               /TestSuite11.css" type="text/css" />
6          <xf:model>
7              <xf:instance>
8                  <ecommerce xmlns="">
9                      <method>cc</method>
10                     <number />
11                     <expiry />
12                 </ecommerce>
13             </xf:instance>
```

```
14              <xf:submission id="submit" method="post"
                    action="http://xformstest.org/cgi-bin/echo
                    .sh" />
15          </xf:model>
16      </head>
17      <body>
18          <xf:group>
19              <xf:label class="title">2.1.a Introductory
                    Example No. 1</xf:label>
20          </xf:group>
21          <xf:select1 ref="method" xmlns="">
22              <xf:label>Select Payment Method:</xf:label>
23              <xf:item>
24                  <xf:label>Cash</xf:label>
25                  <xf:value>cash</xf:value>
26              </xf:item>
27              <xf:item>
28                  <xf:label>Credit</xf:label>
29                  <xf:value>cc</xf:value>
30              </xf:item>
31          </xf:select1>
32          <xf:input ref="number" xmlns="">
33              <xf:label>Credit Card Number:</xf:label>
34          </xf:input>
35          <xf:input ref="expiry" xmlns="">
36              <xf:label>Expiration Date:</xf:label>
37          </xf:input>
38          <xf:submit submission="submit">
39              <xf:label>Submit Now</xf:label>
40          </xf:submit>
41      </body>
42  </html>
```

Quellcode 5: XForms-Dokuments, eingebettet in einem HTML-Dokument [W3C09b](vereinfacht)

**Bedienbarkeit** Da es sich bei XForms um eine Spezifikation handelt, kann über die Bedienung keine Aussage gemacht werden. Die Bedienung ist abhängig von der verwendeten Implementierung.

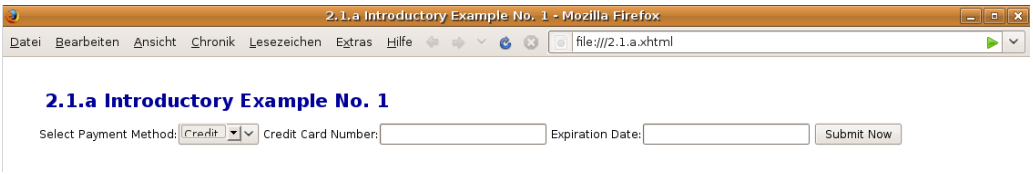


Abbildung 46: Darstellung eines XForms-Dokuments in einem XForms-fähigem Web-Browser [W3C09b](vereinfacht)

**Lizenz und Integrationsfähigkeit** Die W3C-Spezfikation von XForms ist frei zugänglich und kann daher in jedem beliebigen System implementiert werden.

**Unterstützte Datentypen** Die XForms-Spezifikation bietet ein große Auswahl an Datyentypen, wie zum Beispiel:

- String
- Integer
- Byte
- Date
- Time
- AnyURI
- Email
- Base64

Häufig werden Datentypen noch weiter unterteilt. Den Integer-Datentyp gibt es beispielsweise mit und ohne Vorzeichen sowie in verschiedenen Größen.

Ausserdem gibt es einen Datentyp der Listen beliebiger anderer Datentypen repräsentiert. Durch die Verwendung von XML lassen sich aus diesen Datentypen auch komplexe Strukturen zusammenstellen.

**Externe Datenquellen** Die Spezifikation von XForms sieht keine Anbindung von Daten aus externen Datenquellen vor.

**Grafische Elemente** Folgende grafischen Elemente stehen zur Verfügung

- Textfelder
- Auswahlfelder
- Optionsfelder
- Kontrollkästchen
- Menüs
- Dateiauswahl
- Passwortfelder
- Schaltflächen
- Bildfelder
- Gruppierungsfelder
- Schieberegler

Durch die hierarchische Struktur von XML können hier, ähnlich wie bei den Datentypen, beliebige Strukturen von grafischen Elementen erstellt werden.

**Anwendungslogik** Der XForms-Standard stellt für alle grafischen Elemente Ereignisse bereit. Diese Ereignisse können entweder eine JavaScript-Methode aufrufen, welche beispielsweise durch den Browser ausgeführt wird, oder eine URL aufrufen. Bei dieser URL kann es sich beispielsweise um ein Skript handeln, welches auf Seiten des Servers ausgeführt wird. Die Antwort des Servers kann dazu verwendet werden, um Teile des Dokumentes neu aufzubauen. Innerhalb von XForms ist es möglich festzulegen, welche Teile des XForms-Dokumentes durch die Antwort neu aufbgebaut werden soll. Durch diese Techniken ist es, ähnlich wie mit Ajax [Wen06], möglich die Anwendungslogik auf den Client und einen Server aufzuteilen. Damit ist beispielsweise die Gestaltung eines Formulars denkbar, welches Details und Abbildungen zu Produkten eines Onlineshops anzeigt. Bei einer Vielzahl an Produkten ist es nicht sinnvoll ein Formular zu generieren, welches bereits alle Details und Abbildungen zu allen Produkten enthält. Das Dokumente wäre unter Umständenen mehrere hundert Megabyte groß. Sinnvoller ist, es in einem Bereich des Formulars nur eine Liste der verfügbaren Produkte anzuzeigen und in einem anderen Bereich die Details und Abbildungen eines Produktes dynamisch nachzuladen.

**Grafische Kontexte** Die XForms-Spezifikation macht keine Angaben zu untersetützten grafischen Kontexten. In welchem grafischen Kontext ein XForms-Dokument dargestellt werden kann, hängt von der Implementierung des XForms-Interpreter ab. Ein Web-Browser, der das XForms-Format versteht, ermöglicht beispielsweise einen HTML-Kontext. Eine Implementierung, welche aus einem XForms-Dokument ein reines HTML-Dokument erzeugen kann, stellt ebenfalls eine Lösung für einen HTML-Kontext dar. Eine sind aber auch Implementierungen denkbar, welche aus einem XForms-Dokument ein SWT-Composite erstellt. Dadurch kann das XForms-Dokument in einem SWT-Kontext dargestellt werden. Es ist dabei die Aufgabe der Implementierung die grafischen Elemente und das Verhalten der Dokuments korrekt abzubilden.

**Lokalisierung** Der aktuelle XForms-Standard enthält momentan noch keine Möglichkeiten, Texte in einem XForms-Dokument an die Sprachen der Benutzer anzupassen. In einer eigenen Implementierung kann diese Möglichkeit jedoch integriert werden.

## 4.4 Fazit

Die Evaluierung hat gezeigt, dass es bereits verschiedene Ansätze für die Erstellung und Integration von Formularen in anderen Systemen gibt. Jedoch war keines der untersuchten Produkte in der Lage, alle Evaulierungskriterien zu erfüllen.

Inubit ist für die Verwendung in einer Komponente zur Generierung von Formularen ungeeignet, da die Entwicklungsumgebung ein fester Bestandteil des BPM-Systems von Inubit ist. Dieses System ist nicht auf Erstellung von allgemein verwendbaren Formularen ausgelegt.

Adobe Lifecycle Designer und Microsoft Infopath sind sehr ähnliche Produkte. Sie bieten eine komfortable Entwicklungsumgebung für die Gestaltung von Formularen. Ihr größter Nachteil ist jedoch, dass die erzeugten Formulare kontextbezogen sind.

JAXFront wurde entwickelt um Benutzeroberflächen für Anwendungen zu generieren und darzustellen. Die Integration in ein bestehendes System wird dabei durch entsprechende Programmierschnittstellen ermöglicht. Aufgrund der hohen Lizenzkosten und der fehleranfälligen Entwicklungsumgebung wird JAXFront im Rahmen dieser Arbeit jedoch nicht verwendet.

WJP Form wird derzeit zum Erstellen von Formularen für die „AristaFlow BPM Suite“ verwendet. Allerdings fehlen diesem Produkt wichtige Funktionsmerkmale wie zum Beispiel Lokalisierbarkeit der Formularoberflächen. Zudem können WJP-Formulare derzeit nur für den HTML-Kontext übersetzt werden.

XForms stellt einen guten Ansatz dar, da es ein frei verfügbarer Standard ist, der einen großzügigen Funktionsumfang für die Beschreibung von Formularen bereitstellt. Diese Technologie ist jedoch noch nicht ausreichend verbreitet. Dies liegt unter anderem auch daran, dass es bisher kaum Implementierungen gibt. Für die Verwendung in einer Komponente eines BPM-System ist XForms daher noch nicht geeignet.

Tabelle 2 zeigt die Ergebnisse der Evaluierung im Überblick.

| Kriterium | Inubit | Adobe Lifecycle | MS Infopath | JAXFront | WJP Form | XForms |
|---|---|---|---|---|---|---|
| Bedienbarkeit | - | + | + | 0 | + | 0 |
| Lizenz und Integrations-fähigkeit | - | - | - | 0 | 0 | + |
| Unterstützte Datentypen | + | - | 0 | + | 0 | + |
| Externe Datenquellen | + | + | + | + | + | - |
| Grafische Elemente | 0 | + | 0 | 0 | 0 | + |
| Anwendungs-logik | + | + | + | + | + | + |
| Grafische Kontexte | 0 | - | 0 | + | - | 0 |
| Lokalisierung | 0 | - | 0 | + | - | 0 |
| Gesamt | + | 0 | ++ | +++++ | + | +++ |

Tabelle 2: Zusammenfassung der Evaluierungsergebnisse

# 5 Entwurf einer Komponente zur Formularerstellung

Kapitel 4 hat gezeigt, dass es auf dem Markt Produkte gibt, welche einzelne Probleme bei der Formularerstellung lösen. Jedoch kann keines der getesteten Produkte die Anforderungen vollständig erfüllen.

In dieses Kapitel wird der Entwurf für eine Komponente zur Formularerstellung vorgestellt. Dafür wird zunächst die Architektur dieser Komponente und ihre Einordnung in das BPM-System grob erläutert. Anschließend wird der generelle Ablauf der Formularerstellung und -darstellung besprochen. Im Anschluss daran werden Konzepte diskutiert, welche die Anforderungen aus Kapitel 3 erfüllen. Diese Konzepte werden zu einem Entwurf zusammengefasst, der die Grundlage für eine prototypische Implementierung darstellt.

Die Konzepte sind weitestgehend unabhängig von einem BPM-System. Um eine Integration in ADEPT2 zu ermöglichen, müssen jedoch einige Details bereits beim Entwurf an ADEPT2 angepasst werden. In diesen Fällen wird explizit darauf hingewiesen.

## 5.1 Architektur

### 5.1.1 Aufbau

Kapitel 3 zeigte, dass für die neue Komponente eine Entwicklungsumgebung und eine Ausführungsumgebung benötigt wird. Dies bildet den Ausgangspunkt für den Entwurf. Für einen vollständigen Entwurf wird jedoch noch ein Datenmodell für die Beschreibung von Formularen und ein Modul für den Transport des Modells zwischen den beiden Umgebungen benötigt. Abbildung 47 beschreibt den Aufbau der neuen Komponente und deren Einordnung in das vorhandene BPM-System.

Um die Funktionsweise der Entwicklungs- und Ausführungsumgebung nachvollziehen zu können, wird zunächst das Datenmodell erläutert. Dieses Datenmodell ermöglicht die Beschreibung von Formularmodellen. Ein Formularmodell enthält die Informationen des abstrakten Formulars und die Werte für die Ein- und Ausgabeparameter. Das abstrakte Formular beinhaltet dabei drei Aspekte:

- Das Aussehen: Das Aussehen wird durch grafische Elemente wie Textfelder, Listen und Schaltflächen definiert.

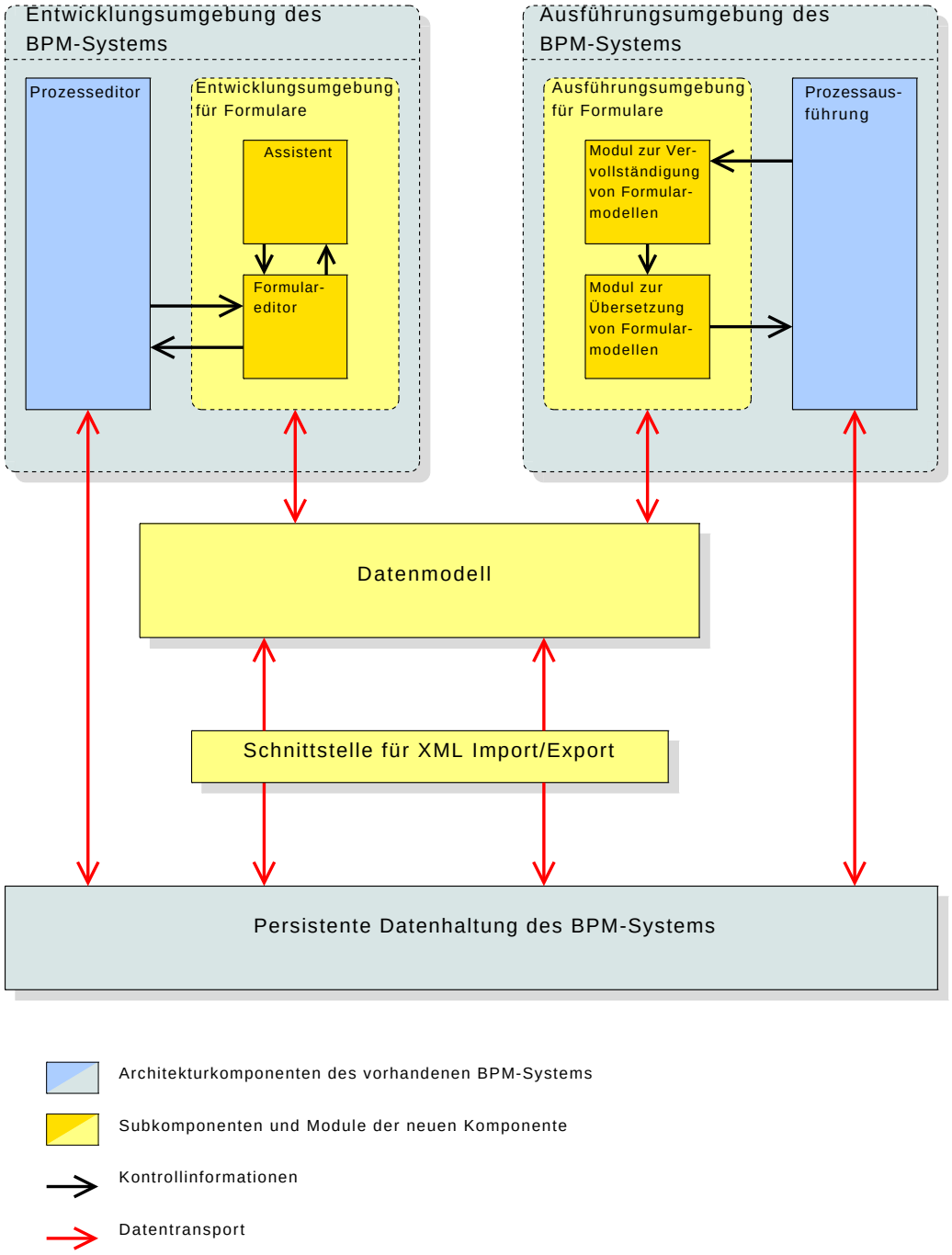


Abbildung 47: Architektur der neuen Komponente und Eingliederung in das BPM-System

- Das Verhalten: Das Verhalten wird durch die Anwendungslogik festgelegt, welche durch JavaScript-Programme beschrieben ist.
- Die Abbildung von Prozessschrittparametern auf die grafischen Elemente des Formulars.

Die Entwicklungsumgebung ist für die Erstellung dieses Formularmodells zuständig. Dafür benötigt die Entwicklungsumgebung zwei Module. Das erste Module ist ein Assistent, der Teile des Formularmodells unter Berücksichtigung der Prozessschrittparameter automatisch generiert. Das zweite Modul ist ein Editor, mit dem das Aussehen des Formularmodells verfeinert und angepasst wird. Die Entwicklungsumgebung wird in die bereits vorhandene Entwicklungsumgebung für Prozesse aus dem BPM-System integriert.

Die Ausführungsumgebung übersetzt das Formularmodell in ein konkretes Formular. Für diesen Vorgang werden zwei Module benötigt. Ein Modul versorgt das Formularmodell zunächst mit den Werten der Prozessschrittparameter. Das zweite Modul führt die eigentliche Übersetzung des Formularmodells durch. Dabei ensteht ein konkretes, kontextabhängiges Dokument. Dieses Dokument wird der Ausführungsumgebung des BPM-Systems übergeben, welche dem Endanwender das Dokument dastellt. Die Ausführungsumgebung wird in die bereits existierende Ausführungsumgebung des BPM-Systems integriert. Für die Integration der prototypischen Implementiertung in das ADEPT2-System wird die aus Kapitel 2 vorgestellte Anwendungsintegration angewendet.

Die Erstellung eines Formularmodells und dessen Übersetzung zu einem konkreten Formular sind zeitlich und räumlich voneinander getrennt. Für den Transport des Formularmodells muss dieses persistent gespeichert werden. Diese Aufgabe übernimmt das BPM-System. Um das Modell dem BPM-System zu übergeben, ist eine weitere Subkomponente notwendig, da das BPM-System mit einem Formularmodell nicht umgehen kann. Diese Subkomponente erstellt einen Kontainer, in welchem das Modell gekapselt wird. Dafür wird ein XML-basiertes Format verwendet. Dieses lässt sich einfach serialisieren und dem System so übergeben. Um das Datenmodell wieder zu deserialisieren, wird der Kontainer aus dem BPM-System geholt und die Informationen rekonstruiert.

### 5.1.2 Ablauf

**5.1.2.1 Erstellung des Formularmodells** Um mit der Erstellung des Formularmodells zu beginnen, wird die Entwicklungsumgebung für Formul-

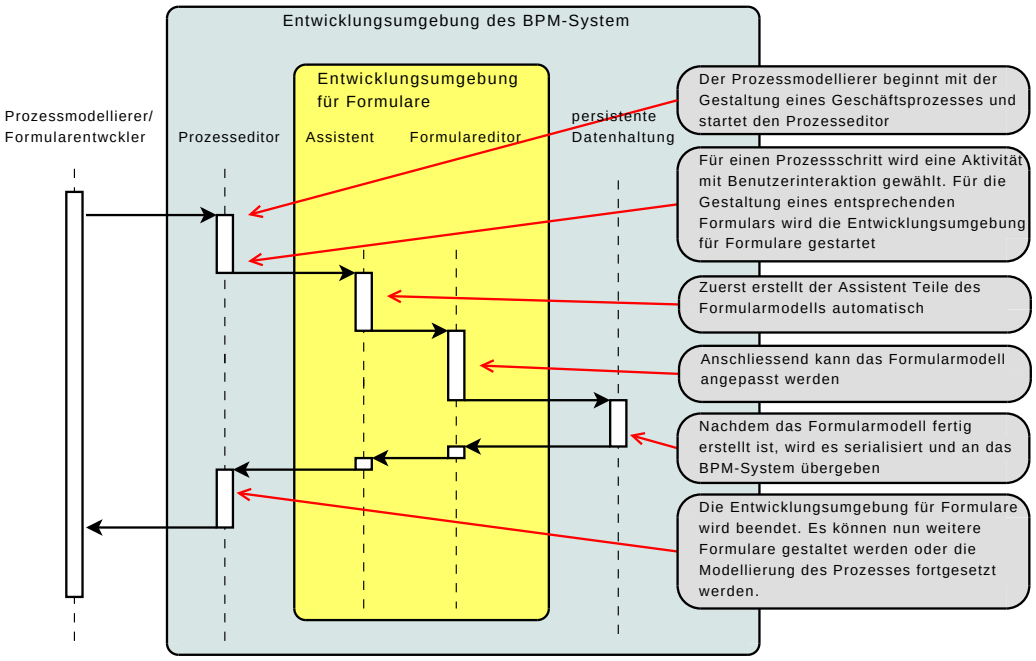


Abbildung 48: Erstellung eines Formularmodells

are von dem Prozesseditor gestartet. Dabei werden auch alle Ein- und Ausgabeparameter der Aktivität übergeben, für die ein Formular erstellt werden soll. Die Entwicklungsumgebung für Formulare startet den Assistenten, welcher mit Hilfe der verfügbaren grafischen Elemente, Formularfelder generiert. Diese Formularfelder dienen der Darstellung der Eingabeparameter.

Nachdem der Assistent diese Aufgabe erledigt hat, hat der Formularentwickler die Möglichkeit, das Layout des Formulars nachträglich anzupassen oder zusätzliche Anwendungslogik in das Formular einzubinden. Nach dem Abschluss der Arbeiten wird das erstellte Formularmodell in das XML-Format serialisiert und dem BPM-System übergeben. Die Werte für die Parameter sind zu diesem Zeitpunkt noch nicht im Formularmodell enthalten. Anschließend wird die Kontrolle wieder an den Prozesseditor zurückgegeben. Diese Vorgehen ist in Abbildung 48 dargestellt.

**5.1.2.2 Ausführung des Formulars** Um aus dem Formularmodell ein konkretes Formular zu erzeugen, welches durch einen Endanwender bearbeitet werden kann, sind mehrere Schritte notwendig. Diese Schritte sind in Abbildung 49 dargestellt.

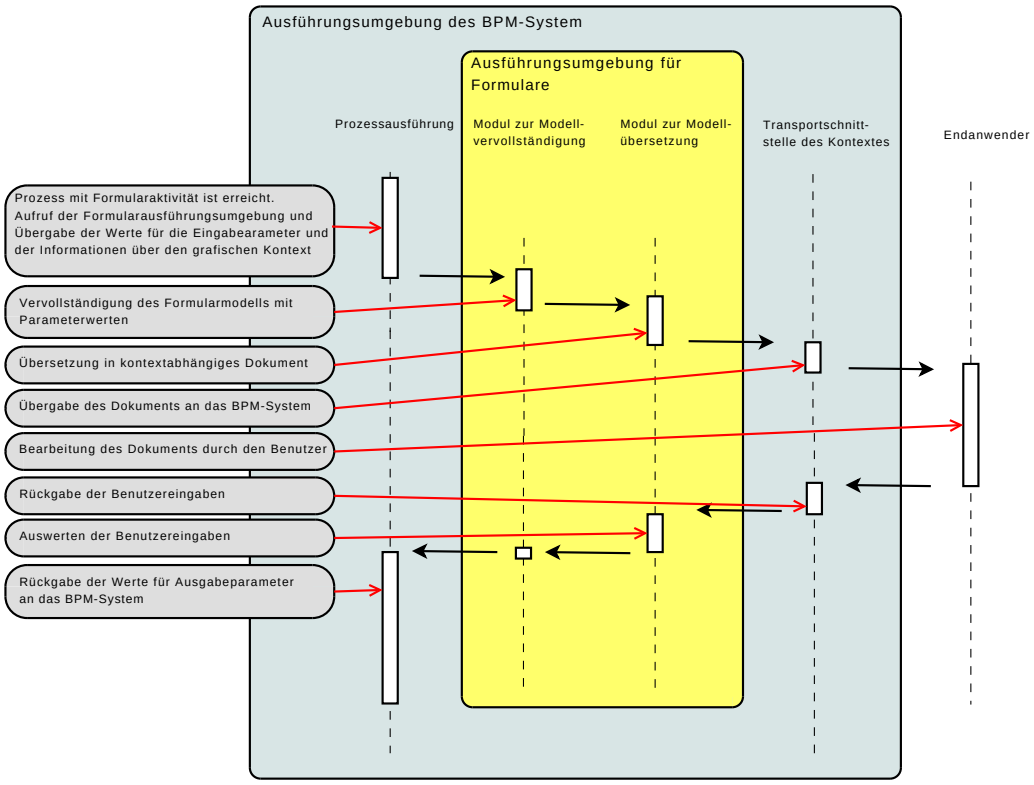


Abbildung 49: Ablauf einer Formularausführung

Der erste Schritt ist das Starten der Ausführungsumgebung für Formulare. Dies geschieht während der Ausführung eines Prozesses durch das BPM-System, sobald ein Prozessschritt mit einer Formularaktivität erreicht wird. Dabei werden auch die Werte für die Prozessschrittparameter und die Informationen über den grafischen Kontext übergeben.

Die Ausführungsumgebung kann nun den XML-Kontainer mit dem erstellten Formularmodell deserialisieren. Diesem Modell werden anschließend die Werte der Eingabeparameter hinzugefügt. Danach wird das Modell dem Übersetzer geben, welcher das nun vollständige Formularmodell mit Hilfe der Kontextinformationen in ein entsprechendes Dokument erstellt. Hierbei handelt es sich beispielsweise um ein HTML-Dokument für einen Web-Browser. Dieses Dokument wird dem BPM-System übergeben, welches dafür verantwortlich ist, das Formular dem Anwender bereitzustellen und diesem die Bearbeitung zu ermöglichen.

Während der Anwender das Formular bearbeitet, hat dieser die Möglichkeit, Eingabeparameter zu betrachten und Werte für Ausgabeparameter anzuge-

ben. Die Eingabeparameter sind dabei an Formularfelder gebunden, die nicht verändert werden können. Die Ausgabeparameter sind ebenfalls an Formularfelder gebunden, welche zu Beginn jedoch noch keine Werte enthalten. Diese müssen erst durch den Endanwender eingegeben werden.

Sobald die Bearbeitung durch den Endanwender abgeschlossen ist, stellt das BPM-System der Formularausführungsumgebung die Eingaben des Endanwenders zur Verfügung. Diese Eingaben sind dabei ebenfalls in einem kontextabhängigem Format. In einem HTML-Kontext werden diese durch eine GET- bzw POST-Anfrage des HTML-Dokuments zur Verfügung gestellt. In einem SWT-Kontext können die EIngaben des Benutzer durch sogenannte Getter-Methoden abgerufen werden.

Das Übersetzungsmodul wertet die Eingaben des Endanwenders aus, um die Werte für die Ausgabeparameter des Prozessschrittes zu erhalten. Diese Werte werden zum Schluss an das BPM-System zurückgegeben. Der Prozessschritt ist damit vollständig abgearbeitet und die Ausführung des Prozesses wird fortgesetzt.

## 5.2 Entwurf des Datenmodells

Das Datenmodell beinhaltet mehrere Konzepte, die für die Gestaltung und Übersetzung von Formularmodellen benötigt werden:

- Datentypen: Bilden die Grundlage für alle weiteren Konzepte.
- Werte: Repräsentieren Informationen und haben einen Datentyp.
- Formularfelder: Dienen der Darstellung von Werten in Formularen.
- Formularmodelle: Beschreiben das Verhalten und das Aussehen von Formularen.

Diese Konzepte werden in den folgenden Abschnitten eingehend erklärt.

### 5.2.1 Datentypen

Um die Daten, welche vom Endanwender abgefragt oder diesem präsentiert werden, als Informationen interpretieren zu können, werden Datentypen benötigt. Das ADEPT2-Modell bietet bereits eigene Datentypen an, jedoch fehlen bei diesen Datentypen wichtige Konzepte wie listenwertige oder strukturierte Datentypen. Daher muss für die neue Komponente ein eigenes Modell

für Datentypen erstellt werden. Neben den Grunddatentypen wie STRING, BOOLEAN, INTEGER, FLOAT und DATE sollen dabei noch Datentypen für Aufzählungen (ENUM), Datentypen für beliebige binäre Daten (BLOB) sowie Datentypen für die genannten listenwertigen und strukturierten Datentypen zur Verfügung gestellt werden.

Strukturierte Datentypen bestehen aus Elementen, welche einen Namen und einen Datentyp haben. Der Name eines Elements muss dabei innerhalb des strukturierten Datentyps eindeutig sein. Der Datentyp eines Elements kann beliebig gewählt weren. Dies bedeutet, dass auch andere strukturierte Datentypen als Datentypen für die Elemente verwendet werden dürfen. Eine wichtige Bedingung ist dabei jedoch, dass hierbei keine zyklischen Abhängigkeiten zwischen strukturierten Datentypen entstehen.

Listenwertige Datentypen sind Datentypen mit beliebig vielen Elementen, welche alle den selben Datentyp besitzen. Dieser gemeinsame Datentyp ist der Basistyp der Liste. Der Basistyp kann dabei jeder beliebige Datentyp sein. Auch bereits vorhandene Listen sind möglich. Anders als bei strukturierten Datentypen haben die Elemente einer Liste keine Namen, sondern werden über ihren Index referenziert. Die Anzahl der Elemente einer Liste ist dynamisch, das bedeutet, dass zur Laufzeit Elemente hinzugefügt oder entfernt werden können.

Um diese neuen Datentypen in die neue Komponente zu integrieren, muss eine Abbildung der neuen Datentypen auf die Datentypen des ADEPT2-Metamodell vorgenommen werden. Dies ist wichtig, da die Parameter für das Formular mit ADEPT2-Datentypen versehen sind. Die Abbildung muss daher bijektiv sein, da zum Einen Werte mit ADEPT2-Datentypen aus dem BPM-System kommen (Eingabeparameter) und zum Anderen Werte mit den neuen Datentypen an das BPM-System übergeben werden (Ausgabeparameter).

Die fünf Grunddatentypen (STRING, BOOLEAN, INTEGER, FLOAT und DATE) sind im ADEPT2-Metamodell bereits vorhanden. Daher können diese direkt abgebildet werden. Die restlichen Datentypen werden als UDT abgebildet. UDTs haben einen eindeutigen Namen, der dazu genutzt wird, den korrespondierenden Datentyp der neuen Komponente zu identifizieren. Tabelle 3 zeigt die Beziehung zwischen den Datentypen von ADEPT2 und denen der neuen Komponente.

| Formulardatentyp | ADEPT2-Datentyp | Beispiel |
|---|---|---|
| String | String | “Herr Müller“<br>“Textnachricht“ |
| Boolean | Boolean | true<br>false |
| Integer | Integer | 4711<br>12358132134 |
| Float | Float | 3,14159265<br>1.41421356 |
| Date | Date | 01/01/1970 00:00:00<br>01/04/2009 12:30:00 |
| Blob | Udt | *Inhalt einer Datei* |
| Aufzählung | UDT | Nord, Ost, Süd, West : Süd<br>Gelb, Grün, Rot, Blau : Rot |
| Liste | Udt | String[ ]: {“Hund“, “Katze“}<br>Float[ ]: {1.2, 9.6, 5.0} |
| Struktur | Udt | Adresse:<br>• Straße : String<br>• Postleitzahl : Integer<br>• Ort : String<br>• Land : Aufzählung |

Tabelle 3: Zuordnung der neuen Datentypen auf ADEPT2 Datentypen

### 5.2.2 Werte

Um innerhalb des Datenmodells Werte mit einem bestimmten Datentyp verwenden zu können, wird eine Repräsentation für Werte benötigt. Diese Werte können beispielsweise dazu verwendet werden, um die Benutzereingaben von Formularfeldern zu speichern. Eine Repräsentation von Werten wird ausserdem benötigt, um die Werte der Eingabeparameter von Prozessschritten im Formularmodell speichern zu können. Ein weitere Einsatzzweck für eine Repräsentation von Werten ist die Möglichkeit, Standardwerten für Formularfelder bereits zur Entwicklungszeit anzugeben. Diese Standardwerte müssen ebenfalls im Modell hinterlegt werden.

Werte haben immer einen festen Datentyp, der nicht geändert werden kann. Ein Wert für einen Prozessschrittparameter kann beispielsweise nur von einem Formularfeld dargestellt werden, welches für diesen Datentyp ausgelegt ist. Eine Besonderheit der Werte ist, dass jeder Wert auch NULL sein kann. Diese Forderung kommt aus dem ADEPT2-Metamodell, welches NULL für Werte generell zulässt. Ein Prozessschrittparameter der NULL ist muss daher auch innerhalb der Ausführungsumgebung für Formulare NULL sein. Wie NULL-Werte durch Formularfelder dargestellt werden, muss für jedes Formularfeld separat entschieden werden.

Um den Umgang mit Werten in der Entwicklungsumgebung zu erleichtern, ist es möglich, bei Werten mit einem strukturierten Datentyp direkt auf die Werte des Elemtes zuzugreifen. Hierfür wird die Punktnotation vereinbart, welche bereits in den meisten Programmiersprachen verwendet wird, um auf die Elemente strukturierter Objekte zuzugreifen. Der Ausdruck *wert.nachname* steht beispielsweise für das Element *nachname* des Wertes.

Analog wird für den Zugriff auf die Elemente einer Liste die Notation eines Indexoperators([]) vereinbart. Der Index des gewünschten Elementes steht hierbei innerhalb den Klammern. Das erste Elemente erhält dabei den Index 0. Der Ausdruck *wert[4]* steht beispielsweise für den Wert des fünften Elementes der Liste.

### 5.2.3 Formularfelder

Mit den vorgestellten Datentypen ist es möglich, die Daten der Prozessschrittparameter zu interpretieren. Diese Parameter müssen grafisch im Formular dargestellt werden. Dafür werden *Composite* verwendet. Um Verwechslungen zu vermeiden, werden die *Composite*, welche bei SWT verwendet wer-

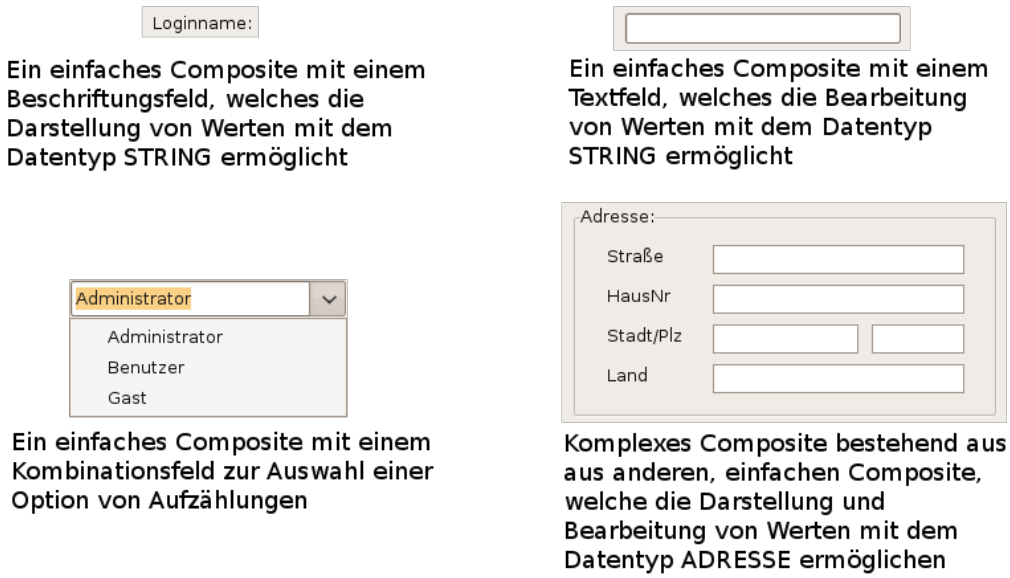


Abbildung 50: Beispiele für *Composite*

den, im folgenden als *SWT-Composite* bezeichnet. *Composite* sind Formularfelder, welche aus grafischen Elementen aufgebaut sind, wie beispielsweise Textfelder, Auswahlfelder oder Listen. Jedes *Composite* ist für Werte eines bestimmten Datentyps spezialisiert. Um zum Beispiel einen Wert des Typs STRING darzustellen wird ein *Composite* benötigt, welches ein Textfeld beinhaltet. In Abbildung 50 werden vier unterschiedliche *Composite* gezeigt. In Abbildung 51 ist ein Formular zu sehen, welches aus mehreren *Composite* besteht.

Für die primitiven Datentypen enthält die Entwicklungsumgebnug bereits passende *Composite*. Für Aufzählungsdatentypen ist ebenfalls ein *Composite* enthalten, dass jedoch nicht für eine Aufzählung spezialisiert ist, sondern mit allen Aufzählungen umgehen kann, da alle Aufzählungen die selbe Struktur haben. Für listenwertige Datentypen kann ein entsprechendes *Composite* automatisch generiert werden, solange bereits ein *Composite* für den Basistyp der Liste existiert. Für Listen von primitiven Datentypen und Aufzählungen ist dies daher der Fall.

Strukturierte Datentypen haben jeweils einen unterschiedlichen Aufbau. Daher müssen für solche Datentypen neue *Composite* erstellt werden. Um ein neues *Composite* für einen speziellen Datentyp zu definieren, werden *Composite Definition* verwendet. Bei einer *Composite Definition* handelt es sich um eine abstrakte Beschreibung des Aufbaus und des Verhaltens eines *Compo-*

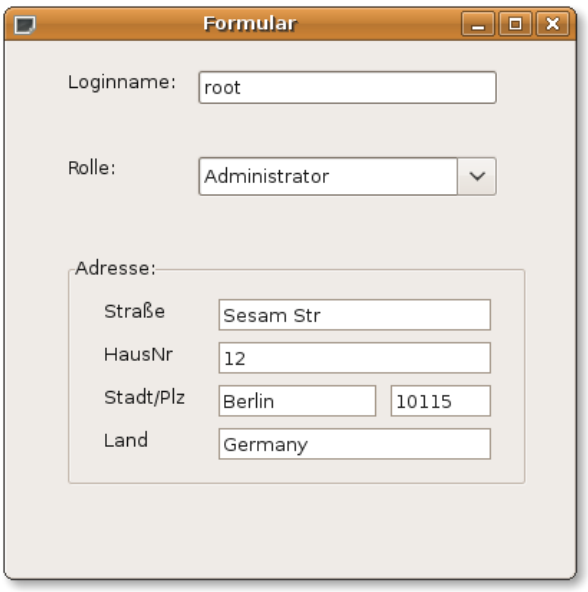


Abbildung 51: Formular bestehend aus mehreren *Composite*

*site*. Eine neue *Composite Definition* kann dabei aus bereits existierenden *Composite* aufgebaut werden, um mit Hilfe dieser *Composite* die Elemente eines strukturierten Datentyps darstellen zu können.

Um ein *Composite* zu erhalten, welches auf der neuen *Composite Definition* basiert, wird die *Composite Definition* instanziert. Das dadurch erzeugte *Composite* kann in einem Formular verwendet werden, um damit einen Wert des entsprechenden Datentyps darzustellen. Eine *Composite Definition* verhält sich zu einem *Composite* (Instanz einer *Composite Definition*) dabei ähnlich wie eine Klasse zu einem Objekt.

*Composite* und *Composite Definition* sind grundlegend für die Erstellung von Formularen. Daher werden diese in den folgenden Abschnitten weiter detailiert und voneinander abgegrenzt.

**5.2.3.1 Composite Definition** Eine *Composite Definition* beinhaltet mehrere Aspekte, welche ein *Composite* beschreiben. Der erste Aspekt ist das Layout (Layout), welches den Aufbau der Instanzen beschreibt. Des Weiteren gibt es Eigenschaften (Attributes), die für das Aussehen und teilweise für das Verhalten des *Composite* verantwortlich sind. Ein weiterer Aspekt sind Ereignisse (Events), welche durch Aktionen des Endanwenders ausgelöst werden. Die Eingabe von Text in ein Textfeld stellt beispielsweise ein Ereignis

dar. Der letzte Aspekt, der eng mit den Ereignissen verwandt ist, ist die Anwendungslogik (Logic), welche das Verhalten des *Composite* zur Laufzeit steuert.

**Layout** Jede *Composite Definition*, welche für die Darstellung eines strukturierten Datentyps zuständig ist, ist aus bereits existierenden *Composite* aufgebaut. Diese *Composite* müssen optisch angeordnet werden. Dafür ist das Layout verantwortlich. Das Layout legt die Positionierung der *Composite* innerhalb der neuen *Composite Definition* fest. Jedes *Composite*, die aus dieser *Composite Definition* abgelietet wird, hat daher den selben Aufbau. Ein Ändern dieses Aufbaus zur Laufzeit ist nicht möglich.

**Eigenschaften** *Composite* besitzen Eigenschaften, welche das Verhalten und das Aussehen von Formularfeldern steuern zu können, beispielsweise um die Hintergrundfarbe festzulegen, oder einzustellen, ob Werte bearbeitet oder nur dargestellt werden können. Jede Eigenschaft besitzt einen Namen, einen Wert und einen Datentyp. Die Eigenschaft mit dem Namen *editierbar* könnte beispielsweise einen booleschen Datentyp haben. Damit sind die Werte TRUE und FALSE möglich. Diese Eigenschaft legt fest, ob der Inhalt eines *Composite* nur angezeigt werden kann, oder ob ein Bearbeiten des Inhaltes durch den Endanwender möglich ist.

Der Name muss innerhalb der Eigenschaften einer *Composite Definition* eindeutig sein, da die Eigenschaft über den Namen identifiziert wird. Der Datentyp ist wichtig, um bei der Instanzierung der *Composite Definition* nur Werte für die Eigenschaften zuzulassen, welche für diese Eigenschaft semantisch sinnvoll sind. Für eine Eigenschaft *maximaleTextLänge*, welche zum Beispiel die Länge der Eingabe in einem Textfeld beschränkt, ist der Wert *"Automobil"* nicht sinnvoll.

Es müssen nicht alle *Composite* dieselben Werte für die Eigenschaften besitzen. Welche Eigenschaften ein *Composite* hat, wird in der entsprechenden *Composite Definition* bestimmt. Welche Werte diese Eigenschaften haben, wird dagegen erst bei der Erstellung eines entsprechenden *Composite* festgelegt. Dadurch haben alle *Composite*, welche auf derselben *Composite Definition* basieren, dieselben Eigenschaften. Die Werte können sich jedoch unterscheiden, was zu einem unterschiedlichen Verhalten und Aussehen der *Composite* zur Laufzeit führt. Um eine größtmögliche Flexibilität zu erreichen, lassen sich die Werte dieser Eigenschaften zur Laufzeit mit Hilfe von Anwendungslogik ändern. Ein *Composite*, welches nur das Anzeigen

eines Wertes erlaubt, kann auf diese Weise zur Laufzeit so geändert werden, dass auch ein Bearbeiten des Wertes möglich ist. Damit ist es beispielsweise möglich, die Hintergrundfarbe von Textfeldern zur Laufzeit zu ändern, etwa um den Endanwender auf fehlerhafte Eingaben hinzuweisen.

| Eigenschaft | Datentyp | Erläuterung |
|---|---|---|
| hintergrundfarbe | INTEGER | Beschreibt die Farbe, welche im *Composite* hinterlegt ist. |
| vordergrundfarbe | INTEGER | Beschreibt die Farbe der Schrift oder des Rahmens eines *Composite*. |
| editierbar | BOOLEAN | Legt fest, ob der Inhalt des *Composite* durch den Endanwender verändert werden kann. |
| tooltip | STRING | Zeichenfolge welche eingeblendet wird, wenn die Maus über das *Composite* bewegt wird. |
| maximaleLänge | INTEGER | Legt die maximale Eingabelänge bei einem Textfeld fest |
| inhalt | abhängig von *Composite Definition* | Besondere Eigenschaft, die den Wert enthält, der durch das *Composite* angezeigt werden soll. Der Datentyp ist abhängig davon, für welchen Datentyp die *Composite Definition* erstellt wurde, beispielsweise STRING bei einem *Composite* zur Darstellung von Texten. |

Tabelle 4: Mögliche Eigenschaften für *Composite* und deren möglichen Datentypen

Tabelle 4 zeigt eine beispielhafte Übersicht möglicher Eigenschaften einer *Composite Definition*. Eine besondere Eigenschaft einer *Composites Definition* ist *inhalt*. Der Wert dieser Eigenschaft stellt den Wert dar, den das *Composite* darstellen soll. Diese Eigenschaft gibt es daher in jeder *Composite Definition*. Der Datentyp dieser Eigenschaft unterscheidet sich jedoch

in jeder *Composite Definition* und ist davon abhängig, welche Werte das *Composite* darstellen soll. Bei einem Textfeld hat diese Eigenschaft zum Beispiel den Datentyp STRING. Bei einem Kontrollkästchen wiederum hat die Eigenschaft den Datentyp BOOLEAN.

**Anwendungslogik und Ereignisse** Eine weitere Möglichkeit, das Verhalten eines *Composite* zu steuern, stellt die Verwendung von Anwendungslogik dar. Jeder *Composite Definition* kann hierfür Anwendungslogik in Form von JavaScript hinzugefügt werden. Für jede *Composite Definition* lassen sich dabei Methoden mit einer Liste von sequentiell auszuführenden Befehlen anlegen. Hierbei gibt es beispielsweise Befehle um die Eigenschaften von *Composite* auszulesen oder zu verändern. Damit lassen sich, etwa in einem Onlinehsop, Preise aus unterschiedlichen Felder auslesen, addieren und in einem anderen Felder wieder anzeigen.

Um zur Laufzeit Methoden aus der Anwendungslogik auszuführen, werden Ereignisse verwendet. Ereignisse werden durch die *Composite* ausgelöst. *Composite Definition*, welche *Composite* verwenden, können deren Ereignisse abfangen, um darauf zu reagieren. Welche Ereignisse ausgelöst werden können, wird dabei in der *Composite Definition* festgelegt, auf denen verwendeten *Composite* basieren. Tabelle 5 zeigt Beispiele für solche Ereignisse. Die Instanzen einer neun *Composite Definition* sind selbst wieder *Composite* sind, welche ebenfalls Ereignisse verursachen können. Um diese Ereignisse auszulösen, können in der Anwendungslogik der entsprechenden *Composite Definition* spezielle Befehle verwendet werden.

#### 5.2.3.2 Composite

Ein *Composite* ist eine Instanz einer *Composite Definition* dar. Bei Instanzierung müssen die abstrakten Informationen der *Composite Definition* mit konkreten Werten belegt werden. Hierzu gehören Angaben über die Positionierung, die Eigenschaften und das Verhalten beim Eintreten von Ereignissen. Ausserdem hat ein *Composite* einen Namen, der die Instanz identifiziert. Über diesen Namen kann in der Anwendungslogik auf das *Composite* zugegriffen werden, um beispielsweise dessen Eigenschaften zu ändern.

Es gibt zwei Formen der Instanzierungen einer *Composite Definition*. Zum Einen werden *Composite Definition* innerhalb anderer *Composite Definition* instanziert. Die daraus resultierenden *Composite* werden im Layout der umschliessenden *Composite Definition* genutzt. Zum Anderen können *Composite Definition* innerhalb eines Formulares instanziert werden. Diese *Composite*

| Ereignis | Erläuterung |
|---|---|
| inhaltGeändert | Tritt ein, sobald der Wert, den das *Composite* darstellt, durch den Anwender geändert wurde. |
| erhalteFokus | Tritt ein, sobald der Anwender das *Composite* ausgewählt hat. |
| verliereFokus | Tritt ein, sobald der Anwender ein anderes *Composite* ausgewählt hat. |
| aktiviert | Tritt ein, sobald der Anwender die Schaltfläche eines entsprechenden *Composite* betätigt hat. |

Tabelle 5: Mögliche Ereignisse von *Composite*

werden für das Layout des Formulares verwendet, um die Parameter des Prozessschrittes in dem Formular darzustellen. Diese beiden Formen der Instanzierung sind sehr ähnlich. Allerdings gibt es einige Unterschiede in den Details bei der Belegung von Eigenschaften mit Werten.

**Belegung von Eigenschaften** Für die Belegung der Eigenschaften mit Werten, können bereits zur Entwicklungszeit konkrete Werte angegeben werden. Ein Beispiel verdeutlicht das:

Ein Formularentwickler entwirft eine *Composite Definition* mit mehreren Textfeldern. Die *Composite Definition* der Textfelder legt fest, dass die Textfelder die Eigenschaft *hintergrundfarbe* haben. Der Formularentwickler hat nun die Möglichkeit festzulegen, welche Hintergrundfarben die einzelnen Textfelder zur Laufzeit haben sollen. Dafür ordnet der Formularentwickler der Eigenschaft *hintergrundfarbe* jedes Textfelds einen entsprechenden Wert zu.

Alternativ dazu kann der Wert für eine Eigenschaft an einen Parameter des Formulars gebunden werden, der den selben Datentyp hat, wie die Eigenschaft. In diesem Fall wird zur Laufzeit der Wert des Parameters als Wert für die Eigenschaft eingesetzt. Dieser Wert ist zur Entwicklungszeit noch nicht bekannt. Zur Laufzeit wird der Wert des Parameters bei der Übersetzung des Formularmodells eingebunden. Handelt es sich bei diesem Parameter um

einen Ausgabeparameter, ist dafür kein Wert verfügbar. In diesem Fall ist der Wert für die Eigenschaft zu Beginn NULL. Während der Endanwender das Formular bearbeitet, kann sich der Wert der Eigenschaft ändern. Dies ist beispielsweise durch Anwendungslogik oder bei der Eigenschaft *inhalt* durch den Endanwender selbst möglich. Nachdem der Endanwender das Formular bearbeitet hat, wird der letzte Wert der Eigenschaft ermittelt und an das BPM-System als Wert für den Ausgabeparameter übergeben. Diese Bindung an einen Parameter ist nur bei einer Instanzierung innerhalb eines Formulars möglich.

Bei der Instanzierung innerhalb eines anderen *Composite Definition* ist dies nicht möglich. Jedoch können hier die Eigenschaften der verwendeten *Composite* an die Eigenschaften der umschliessenden *Composite Definition* gebunden werden. Der Wert für die Eigenschaft einer Instanz der *Composite Definition* wird dabei ebenfalls zur Laufzeit ermittelt. Auf diese Weise können Eigenschaften in der hierarchischen *Composite*-Struktur durchgereicht werden.

Im oben erläuterten Beispiel hat der Formularentwickler die Möglichkeit der neuen *Composite Definition* eine Eigenschaft mit dem Namen *hintergrund* zu geben. Der Name könnte auch *hintergrundfarbe* lauten, aber zur besseren Unterscheidung wurde ein anderer Name gewählt als *hintergrundfarbe*. Die Eigenschaft *hintergrundfarbe* von einem oder mehreren Textfelder können nun an die Eigenschaft *hintergrund* der *Composite Definition* gebunden werden, um eine einheitliche Hintergrundfarbe zu erhalten. Auch hier ist zu beachten, dass die Eigenschaften der *Composite* nur an Eigenschaften der *Composite Definition* gebunden werden können, wenn diese den selben Datentyp haben.

Es gibt noch eine weitere Möglichkeit, die Eigenschaften bei einer Instanzierung innerhalb einer *Composite Definition* zu belegen. Besitzt die umschliessende *Composite Definition* Eigenschaften mit einem strukturierten Datentyp, können auch die Elemente des Wertes dieser Eigenschaft an die Eigenschaften der verwendeten *Composite* gebunden werden. Ein weiteres Beispiel soll dies verdeutlichen: Eine neue *Composite Definition* soll für die Darstellung von Werten des Datentyps ADRESSE entwickelt werden. Dies bedeutet, dass die Eigenschaft *inhalt* der neuen *Composite Definition* den Datentyp ADRESSE hat. Der Aufbau dieses Datentyps ist in Tabelle 6 dargestellt. Für die Darstellung der einzelnen Elemente einer ADRESSE verwendet die *Composite Definition* unter anderem mehrere *Composite* mit Textfeldern. Diese *Composite* besitzen ebenfalls die Eigenschaft *inhalt*. Da die Elemente von ADRESSE jeweils den Datentyp STRING (bzw. INTEGER für

| Element | Datentyp |
|---|---|
| straße | STRING |
| hausNr | STRING |
| plz | INTEGER |
| stadt | STRING |
| land | STRING |

Tabelle 6: Elemente einer Adresse

die Postleitzahl) haben, erhält diese Eigenschaft bei den verwendeten *Composite* auch den Datentyp STRING (bzw. INTEGER für das *Composite*, welches die Postleitzahl darstellen soll). Diese Eigenschaft kann nun bei den verwendeten *Composite* an die jeweiligen Elemente der Eigenschaft *inhalt* der umschliessenden *Composite Definition* gebunden werden (Abbildung 52).

**Lokalisierbarkeit und Barrierefreiheit** Eine Besonderheit bei der Belegung der Eigenschaften ist, dass für jede Eigenschaft mehrere Werte angegeben werden. Dies kann wird genutzt, um Formulare zu lokalisieren und barrierefrei zu gestalten.

Um die *Composite* an unterschiedliche Sprachen anpassen zu können, müssen alle Texte, welche später in diesem *Composite* zu sehen sind, in allen zu unterstützenden Sprachen angegeben werden. Da alle Texte in Form von Eigenschaften hinterlegt werden, ist es bei der Belegung von Eigenschaften möglich, für jede Sprache eine andere Belegung zu wählen. Dies ist jedoch optional, da bei manchen Eigenschaften die Belegung unabhängig von der Sprache ist, beispielsweise eine Eigenschaft, welche die maximal Textlänge in einem Textfeld darstellt. Später zur Laufzeit wird vom BPM-System die Sprache für das Formular ermittelt und die entsprechende Belegung für die Eigenschaft ausgewählt.

Um die Endanwendern, welche auf Barrierefreiheit angwiesen sind, zu unterstützen ist es auch möglich, den Eigenschaften Werte für eine entsprechende Benachteiligung zuzuweisen. Zum Beispiel kann einer Eigenschaft, welche die

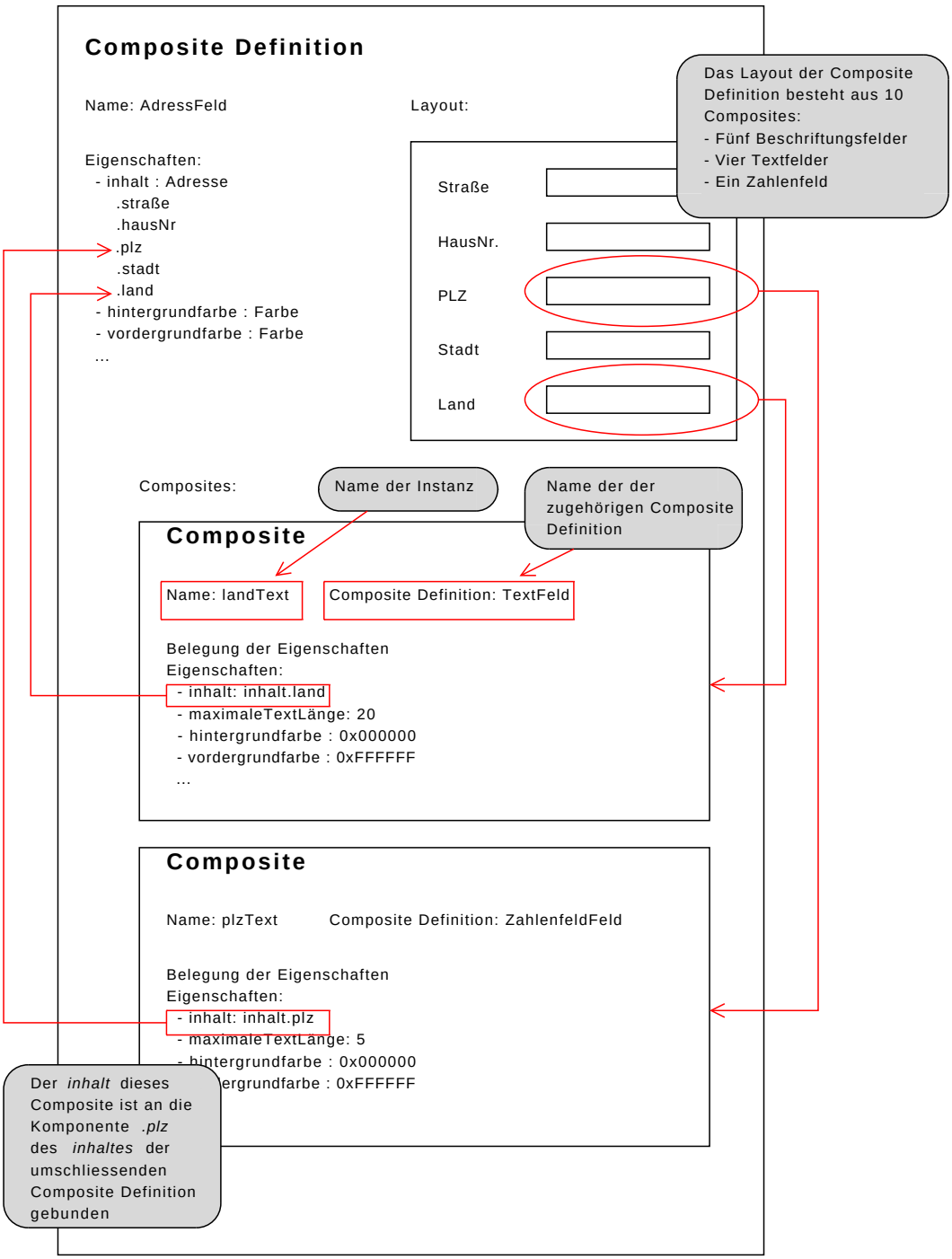


Abbildung 52: *Composite Definition* mit Eigenschaften, einem Layout und *Composite*

Hintergrundfarbe eines *Composite* im Falle einer Fehleingabe darstellt, bei Endanwendern mit einer Rot/Grün-Sehschwäche eine andere Farbe als Rot gewählt werden.

**Reagieren auf Ereignisse** Eine weitere abstrakte Information der *Composite Definition* sind die Ereignisse. In einer *Comopsite Definition* wird festgelegt, welche Ereignisse die *Composite* später auslösen können. Bei der Instanzierung muss angegeben werden, wie auf diese Ereignisse reagiert werden soll. Hierfür können die definierten Ereignisse an die Methoden der Anwendungslogik gebunden werden, welche in der umschliessenden *Composite Definition* definiert sind. Die Ausführungsumgebung sorgt später dafür, dass bei dem Eintreten eines Ereignisses bei einem *Composite* die zur Entwicklungszeit gebundene Methode aufgerufen wird.

### 5.2.4 Formularmodell

Das eigentliche Modell für ein Formular ist sehr einfach aufgebaut. Es besitzt wie eine *Composite Definition*, ein Layout, welches festlegt aus welchen *Composite* das Formular aufgebaut ist und wie diese *Composite* positioniert werden. Bei der Belegung der Eigenschaften dieser *Composite* gibt es mehrere Möglichkeit. Beispielsweise können diese mit den Parametern des Formulars verknüpft werden. Damit kann ein Eingabeparameter mit der Eigenschaft *inhalt* mit einem *Composite* verknüpft werden, um zu erreichen, dass dieses *Composite* den Parameter grafisch darstellt. Eine weitere Möglichkeit besteht darin, mit den Parametern das Aussehen des Formulars zu beeinflußen. Es ist möglich, einen Eingabeparameter den Datentyp *Farbe* zu geben und diesen mit der Eigenschaft *Hintergrundfarbe* eines (oder mehrerer) *Composite* zu verknüpfen. Zur Laufzeit des Formulars wird der Wert des Parameters dann dazu genutzt um die Hintergrundfarbe des (oder mehrerer) *Composite* festzulegen.

Neben den Ein- und Ausgabeparametern der Aktivitäten kann das Formular darüber hinaus noch weiter parametrisiert werden. Es können zusätzliche virtuelle Parameter für externe Datenanbindungen angelegt werden. Diese Parameter tauchen nicht im Datenfluß des Prozesses auf, sondern existieren nur im Kontext des Formulars. Durch diese Parameter können dem Formular zusätzlich Informationen zur Verfügung gestellt werden. Die Verwendung der virtuellen Parameter unterscheidet sich nicht von den normalen Parametert. Sie besitzen ebenfalls einen Datentyp und können daher auch an die Eigenschaften von Formularfeldern mit dem selben Datentyp gebunden werden.

Als Quelle für die externe Daten können Verbindungen zu Datenbanken und ein Abfragekommando für diese Datenabank angegeben werden. Diese Abfragekommando muss durch einen korrekten Standard-SQL-Befehl [Unt03] beschrieben werden. Der Aufbau dieses Befehls lässt bereits zur Entwicklungszeit Rückschlüsse über die zu erwartende Ergebnissmenge des Befehls zu. Eine Ergebnissmenge ist eine Tabelle, welche aus einer oder mehreren Spalten, sowie beliebig vielen Zeilen bestehen kann. Die Anzahl der Zeilen ist abhängig vonm Inhalt der Felder in der Datenbank. Welche Spalten in der Ergebnissmenge vorkommen ist abhängig vom verwendeten Abfragekommando. Dadurch wird der Datentyp für den virtuellen Parameter festgelegt. Wird ein Befehl verwendet, der ein einspaltiges Ergebnis liefert, wird ein primitiver Datentyp verwendet. Da der Befehl keine Aussagen über den Datentyp macht, der für diese Spalte in der Datenbank verwendet wird, muss der Formularentwickler den entsprechenden primitiven Datentyp selbst auswählen. Bei einem Befehl, der mehrere Spalten zurückgibt, wird ein strukturierter Datentyp für den virtuellen Parameter verwendet. Der Formularentwickler muss hierbei wieder die passenden primitiven Datentypen für die einzelnen Spalten angeben. Diese Datentypen werden für die Elemente des strukturieten Datentyps verwendet. Da zur Entwurszeit auch keine Aussage über die Anzahl der Ergebnisse des Abfragekommandos gemacht werden kann, ist es dem Formularentwickler ausserdem überlassen, ob der Datentyp für den virtuellen Parameter eine Liste des primitiven oder strukturierten Datentyps sein soll oder nicht. Wird keine Liste gewählt, wird bei einer Rückgabe mit mehreren Ergebnissen nur das erste Ergebnis verwendet. Bei einer Rückgabe mit ohne ein Ergebniss wird NULL für den Wert verwendet.

Eine weitere Möglichkeit von virtuellen Parametern ist die Einbeziehung von historischen Formulardaten. Dadurch können vom Formularentwickler die Ausgabeparameter von Aktivitäten mit Formularen gewählt werden, welche ebenfalls im aktuellen Formualr dargestellt werden sollen. Es können dabei nur Aktivitäten von Formularen gewählt werden, welche im Kontrollflußes vor dem aktuellen Formular liegen.

Des Weiteren enthält das Formularmodell ebenfalls einen Bereich für die Anwendungslogik. In diesem Bereich können, wie bei der *Composite Definition* Methoden definiert werden, welche mit den Ereignissen der *Composite* verknüpft und so zur Laufzeit beim Eintreten eines Ereignisses ausgeführt werden. Bei der Anwendungslogik für Formulare gibt es einige spezielle Befehle, die es erlauben, das Bearbeiten des Formulars zu beenden. Das fertige Formular kann hierbei abgesendet, abgebrochen oder unterbrochen werden.

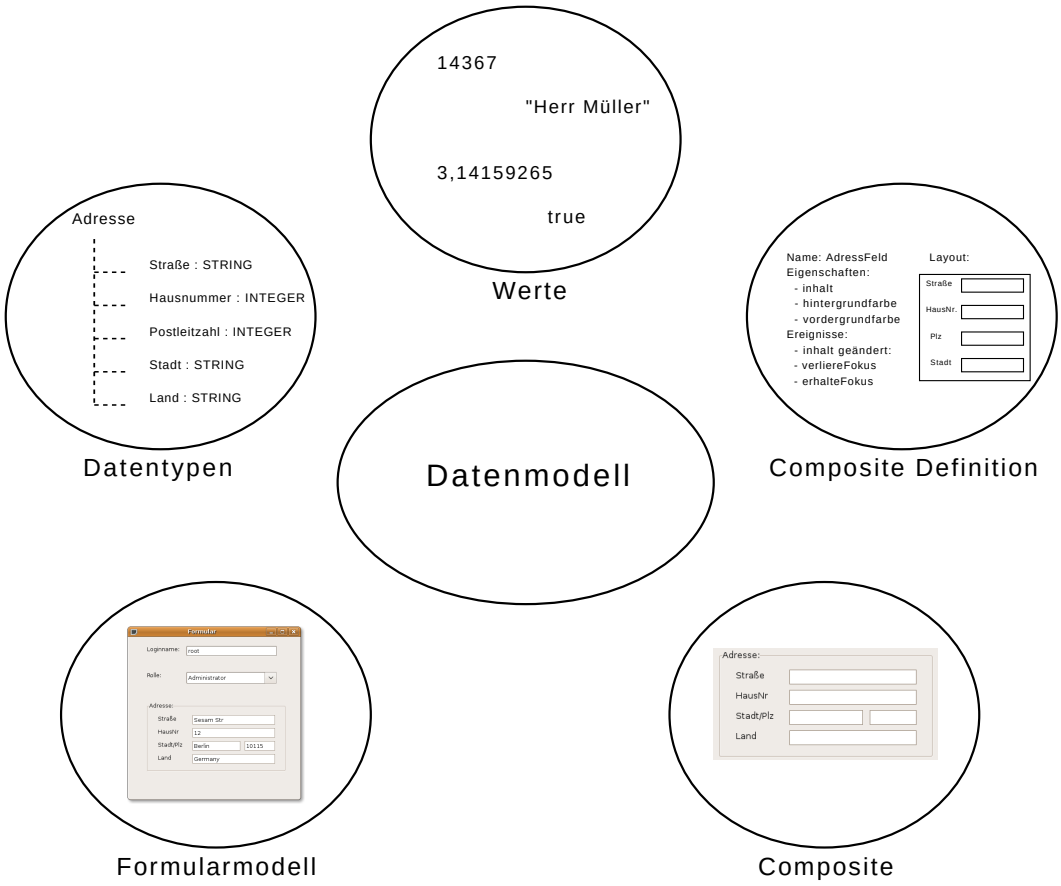


Abbildung 53: Übersicht über das Datenmodell

Dieses Datenmodell beinhaltet alle Aspekte die nötig sind um Formulare zu beschreiben. Abbildung 53 zeigt alle Elemente des Datenmodell im Überblick.

## 5.3 Entwurf der Entwicklungsumgebung

Der Formularentwickler hat die Aufgabe, mit der Entwicklungsumgebung das Formularmodell zu erstellen. Diese Umgebung bietet hierfür alle Werkzeuge für die verschiedenen Aspekte des Datenmodells eines Formulars.

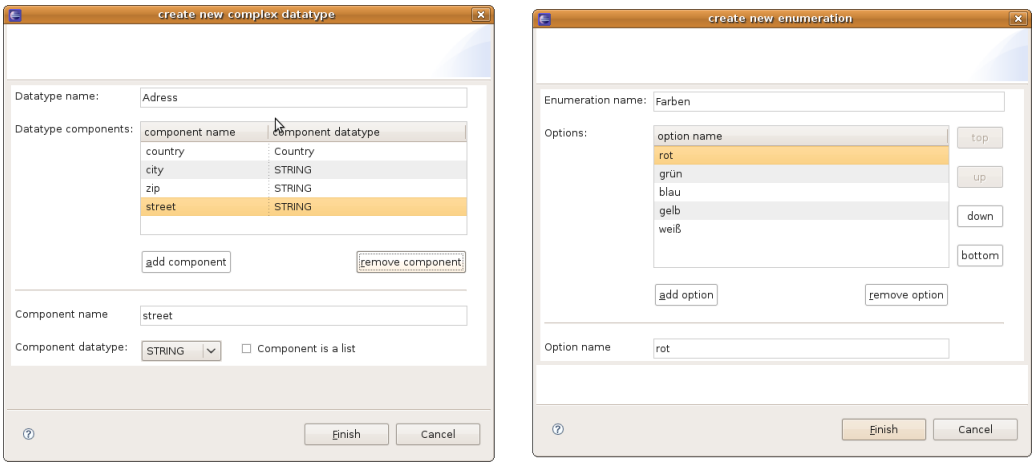


Abbildung 54: Editor für strukturierte Datentypen (links) und Aufzählungen (rechts)

### 5.3.1 Editoren für Datentypen

Um strukturierte Datentypen oder Aufzählungen in Formularen verwenden zu können, müssen diese zunächt angelegt werden. Hierfür bietet die Entwicklungsumgebung eine Übersicht über vorhandene Datentypen (Abbildung 55), sowie zwei Editoren für Datentypen (Abbildung 54).

Die Editoren ermöglichen das Erstellen von Aufzählungen und strukturierten Datentypen. Für listenwertige Datentypen gibt es keinen eigenen Editor, da Listen ausgehend von ihrem Basistyp automatisch generiert werden können. Dazu wird bei der Verwendung einer Liste nur der Basistyp für die Liste angegeben. Der Datentyp, welcher die Liste repräsentiert, wird implizit angelegt.

Für die Erstellung von strukturierten Datentypen stehen sämtliche primitiven Datentypen, Aufzählungen und bereits existierende strukturierte Datentypen zur Verfügung. Auch Listen dieser Datentypen sind erlaubt. Zirkuläre Abhängigkeiten zwischen strukturierten Datentypen sind jedoch nicht erlaubt. Auch beim Bearbeiten eines strukturierten Datentyps zu einem späteren Zeitpunkt ist dies nicht zulässig.

Für das Erstellen von Aufzählungen können beliebige Namen für die Optionen vergeben werden. Die Namen müssen sich voneinander unterscheiden. Die Reihenfolge der Optionen kann festgelegt werden. Für jede Option lassen sich zusätzlich weitere Namen für unterschiedliche Sprachen angeben. Der richtige Name für eine Option wird zur Laufzeit ausgewählt.

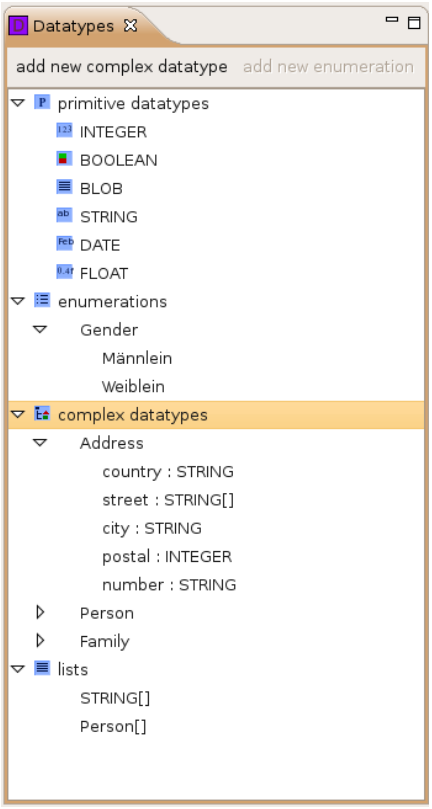


Abbildung 55: Baumansicht der vorhandenen Datentypen

Die Datentypübersicht aus Abbildung 55 zeigt alle verfügbaren Datentypen. Dies beinhaltet auch alle bereits angelegten strukturierten Datentypen und Aufzählungen. Alle listenwertigen Datentypen, welche als Elemente eines strukturierten Datentyps verwendet werden, sind ebenfalls zu sehen.

### 5.3.2 Editor für Werte

Diese Datentypen können beispielsweise verwendet werden, um sie den Eigenschaften von *Composite* zuzuweisen. Da diese Eigenschaften bereits zur Entwicklungszeit mit einem Standardwert versehen werden können, ist ein weiterer Editor nötig, der das Anlegen und Bearbeiten von Werten für beliebige Datentypen ermöglicht.

Aufgrund der Komplexität von strukturierten Datentypen ist ein einfacher Editor mit einer Textzeile nicht ausreichend, um einen Wert zu definieren. Der

Editor untersucht daher den Datentyp, für den ein Wert angegeben werden soll.

Wenn es sich um einen einfachen Datentypen oder eine Aufzählung handelt, wird der Formularentwickler aufgefordert einen entsprechenden Wert anzugeben oder eine Option auszuwählen. Bei einem strukturierten Datentyp wird der Formularentwickler nacheinander für jedes Element aufgefordert einen Wert anzugeben. Hierfür wird auch der Datentyp des Elements untersucht. Handelt es sich ebenfalls um einen strukturierten Datentyp, wird die Struktur solange rekursiv untersucht, bis ein Element mit einem nichtstrukturierten Datentyp erreicht wird. Bei einer Liste muss der Formularentwickler zunächst entscheiden, wieviele Elemente die Liste enthalten soll und kann dann für jedes Elemente einen Wert vom Basistyp der Liste angeben.

### 5.3.3 Editor für Anwendungslogik

Um die Anwendungslogik für *Composite Definition* und ein Formularmodell zu erstellen, steht ein Texteditor zur Verfügung, welcher das Anlegen von JavaScript-Methoden erlaubt. Innerhalb dieser Methoden kann die Logik implementiert werden. Abhängig davon, ob es sich um Anwendungslogik für Formulare oder *Composite* handelt stehen teilweise unterschiedliche Befehle zur Verfügung. Bei Anwendungslogik für Formulare stehen die Befehle für das Beenden der Formularbearbeitung bereit. Bei *Composite* gibt es Befehle, welche die Ereignisse des *Composite* auslösen. Ausserdem ist in beiden Fällen der Zugriff auf die verwendeten *Composite* durch dessen Instanznamen möglich.

Um den Formularentwickler bei der Erstellung der Anwendungslogik zu unterstützen, werden die Eingaben des Formularentwicklers permanent auf syntaktische und semantische Korrektheit geprüft. Die semantische Korrektheit bezieht sich dabei auf den Zugriff auf die Eigenschaften von *Composite*. Da diesen Eigenschaften Datentypen zugewiesen sind, darf innerhalb der Anwendungslogik keiner Eigenschaft ein Wert mit einem anderen Datentyp zugewiesen werden. Die semantische Analyse informiert den Formularentwickler, wenn die Logik fehlerhafte Zuweisungen enthält.

Die syntaktische Analyse der Anwendungslogik stellt sicher, dass die eingegebenen Befehle den syntaktischen Spezifikationen von JavaScript entspricht. Dies ist wichtig, um zur Laufzeit eine fehlerfreie Ausführung der Anwendungslogik zu gewährleisten.

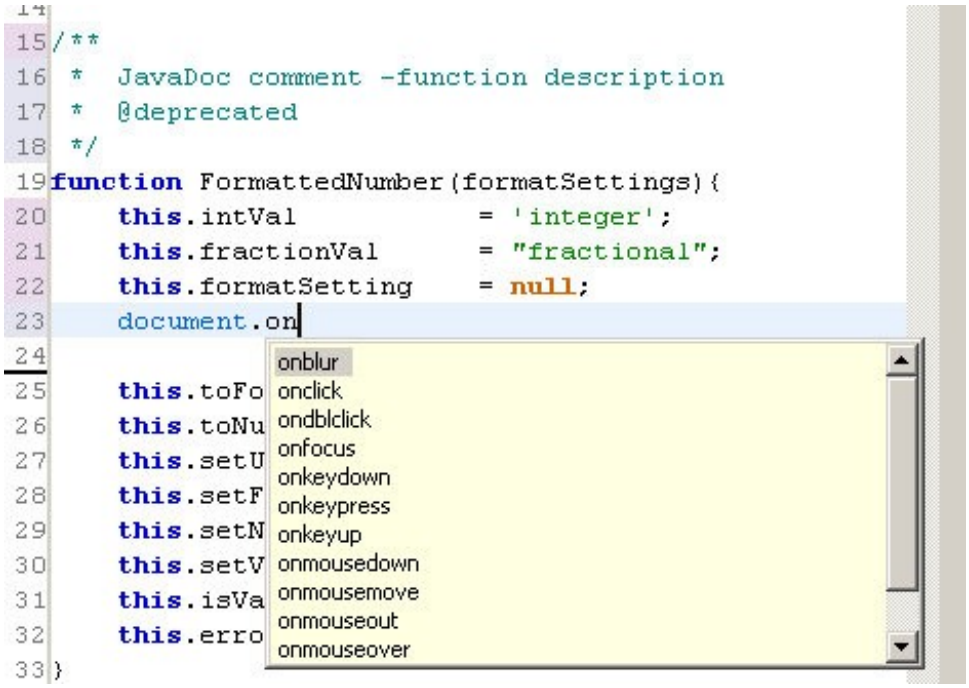


Abbildung 56: JSEclipse: Editor für JavaScript-Code [Aja09]

Um die Arbeit mit dem Editor weiter zu vereinfachen, bietet der Editor eine Funktion für dir automatische Vervollständigung von Eingaben an. Dabei werden Präfixe von möglichen Befehlen, Schlüsselwörtern oder Variblen erkannt und auf Basis dieses Präfixes eine Liste möglicher Eingaben angeboten. Abbildung 56 zeigt JSEclipse [Aja09], das in der Lage ist, die Eingaben des Entwicklers zu vervollständigen.

### 5.3.4 Editor für *Composite Definition* und Formularmodelle

Nachdem die Datentypen angelegt wurden, kann der Formularentwickler mit Hilfe eines weiteren Editors die *Composite Definition* erstellen, welche für Darstellung von Werten mit den neuen Datentypen geeignet sind.

Dabei wird zunächst entschieden, welche Datentypen das neue *Composite* darstellen soll. Anschließend werden die Eigenschaften für die *Composite Definition* festgelegt und mit Datentypen versehen. Ausserdem werden Ereignisse festgelegt, welche zur Laufzeit durch die Instanzen der neuen *Composite Definition* ausgelöst werden können.

Anschließend wird das Layout der *Composite Definition* festgelegt. Hier steht dem Formularentwickler eine Oberfläche zur Verfügung, die es erlaubt *Composite* zu platzieren.

Die Positionierung der *Composite* wird durch Angabe der X- und Y-Koordinaten der *Composite* relativ zur umschliessenden *Composite Definition* oder zum umschliessenden Formular festgelegt. Diese ist aufgrund der Unterschiede bei der Positonierung grafischer Elemente in den verschiedenen grafischen Kontexten notwendig. Der HTML-Kontext stellt durch CSS [W3C98] eine sehr elegante Möglichkeit der Positionierung für grafische Elemente dar, da der Formularentwickler hierbei keine Angaben zur exakten Positionierung machen muss. Dies wird durch den darstellenden Web-Browser übernommen. Für den SWT-Kontext kann diese jedoch nicht adäquat umgesetzt werden.

Dafür bietet SWT das Konzept der *Layouts*, womit der Formularentwickler ebenfalls keine konkreten Angaben zur Positionierung machen muss. Dieses Konzept wiederum lässt sich nicht problemlos im HTML-Kontext umsetzen. Daher muss für die Erstellung eines Formularmodells eine Möglichkeit angeboten werden, welche in beiden Kontexten umsetzbar ist. Die Festlegung der Position durch Angabe von Koordinaten ist sowohl im HTML-Kontext als auch im SWT-Kontext möglich.

Wie auch bei den Datentypen stehen hier zunächst nur einige *Composite* für die Darstellung primitiver Datentypen zur Verfügung, wie beispielsweise ein Textfeld für die Darstellung von STRING-Werten. Ausserdem stehen Schaltlächen zur Verfügung, die genutzt werden können, um mit deren Ereignissen Methoden der Anwendungslogik auszuführen. Auch bei der Erstellung von *Composite Definition* darf es keine zirkulären Abhängigkeiten geben.

Mit Hilfe des Werteeditors können anschließend die Werte für die Eigenschaften, der im Layout verwendeten *Composite*, festgelgt werden. Alternativ können diese Werte auch von den Eigenschaften der neuen *Composite Definition* durchgereicht werden. Wenn eine Eigenschaft der neuen *Composite Definition* einen strukturierten Datentyp hat, ist es auch möglich, ein Element dieser Eigenschaft durchzureichen und mit einer Eigenschaft eines verwendeten *Composite* zu verknüpfen.

Des weiteren wird mit Hilfe des Logikeditors die Anwendungslogik für die neue *Composite Definition* erstellt. Die Methoden dieser Logik können dann mit den Ereignissen der verwendeten *Composite* verknüpft werden.

Für das Anlegen eines Formularmodells gibt es einen Editor, welcher dem Editor für *Composite Definition* ähnlich ist. Es ist möglich über eine Ober-

fläche das Layout aus den vorhandenen *Composite Definition* zu gestalten und deren Eigenschaften mit Werten oder Formularparameter zu verknüfen. Anwendungslogik kann ebenfalls angegeben und mit den Ereignissen der im Layout vorkommenden *Composite* verbunden werden.

### 5.3.5 Assistent

Viele Arbeitsschritte beim Erstellen von Formularen und *Composite Definition* sind automatisierbar. Um den Formularentwickler zu entlasten und ihm die Arbeit zu erleichtern, steht in der Entwicklungsumgebung ein Assistent zur Verfügung, welcher einen Teil der Arbeit automatisch durchführt oder durch gezielte Rückfragen an den Formularentwickler sogar vollständig abschliesst.

Bei der Erstellung von Formularen untersucht der Assistent die Parameter des Formulars und deren Datentypen. Anschließend sucht er nach *Composite Definition*, welche für die Darstellung von Werten dieser Datentypen geeignet sind. Diese werden anschließend durch den Assistent in dem Formular platziert. Werden mehrere brauchbare *Composite Definition* gefunden, wird der Formularentwickler aufgefordert, das für diesen Fall zu verwendende *Composite* auszuwählen. Für den Fall, dass keine passende *Composite Definition* zur Verfügung steht, bietet der Assistent dem Formularentwickler mehrere Möglichkeiten an. Der automatische Erstellvorgang kann abgebrochen werden. Der Assistent kann für den Datentyp automatisch eine neue *Composite Definition* erstellen und verwenden. Oder der Assistent untersucht die einzelnen Elemente des Datentypes und sucht nach passenden *Composite Definition*, für die Datentypen der Elemente, um diese einzeln in das Formular einzufügen.

Beim Erstellen von *Composite Definition* wird ähnlich vorgegangen. Der Assistent untersucht zunächst die Elemente des Datentyps, für den das neue *Composite* die Darstellung übernehmen soll, und generiert dann das Layout mit Hilfe von passenden vorhandenen *Composite Definition*. Anschließend werden automatisch die Elemente des Datentypes der neuen *Composite Definition* mit den Inhalten der entsprechend erzeugten *Composite* verknüpft. Sollten die verwendeten *Composite* und die neue *Composite Definition* Eigenschaften mit gemeinsamen Namen und Datentyp haben, schlägt der Assistent vor, diese zu verbinden. Damit wird erreicht, dass universelle Eigenschaften zur Laufzeit automatisch einen sinnvollen Wert erhalten. Beispielsweise können viele *Composite Definition* die Eigenschaft *hintergrundfarbe* haben. Wenn eine neue *Composite Definition* ebenfalls eine solche Eigen-

schaft enthält ist es sinnvoll, den Wert dieser Eigenschaft weiterzuleiten. Sollte ein anderes Verhalten erwünscht sein, kann dies manuell geändert werden.

## 5.4 Entwurf der Laufzeitumgebung

Während der Ausführung eines Prozesses wird die Ausführungsumgebung für die Formulare aufgerufen, sobald ein Prozessschritt mit einem hinterlegten Formular erreicht wird. Die Ausführungsumgebung für die Formulare hat die Aufgabe, das Formular für den Endanwender vorzubereiten und dem BPM-System zu übergeben. Dies geschieht in zwei Schritten: Zunächst wird das bereits erstellte Formularmodell mit den Werten für die Prozessschrittparameter versorgt und angepasst. Anschließend wird aus dem Formularmodell ein kontextabhängiges Formular erstellt. Zur Verdeutlichung wird dieser Vorgang anhand des HTML- und des SWT-Kontextes besprochen.

### 5.4.1 Modellanpassung

Bei der Anpassung müssen zwei Aspekte des Modells berücksichtigt werden, welche zur Entwurfszeit nicht bekannt sind. Hierbei handelt es sich um die Werte der Formularparameter und die Lokalisierung des Formulars.

Bisher enthält das Formularmodell nur eine abstrakte Beschreibung über das Aussehen, das Verhalten und Angaben darüber, welche Prozessschrittparameter von welchen *Composite* dargestellt werden. Welche Werte diese Prozessschrittparameter haben, war bisher nicht bekannt. Diese Werte müssen dem Formularmodell hinzugefügt werden, bevor dieses übersetzt werden kann. Hierfür muss die Ausführungsumgebung die Werte zunächst aus dem BPM-System abfragen und anschließend den entsprechenden Prozessschrittparameter aus dem Modell zuweisen.

Neben den Prozessschrittparameter verwendet das Formularmodell auch virtuelle Parameter, deren Werte ebenfalls erst jetzt ermittelt werden können. Für virtuelle Parameter, welche historische Formulardaten darstellen, werden die entsprechenden Werte ebenfalls aus dem BPM-System abgefragt. Bei virtuellen Parametern mit einer externen Datenanbindung wird die Datenbankverbindung aufgebaut und das Abfragekommando versendet. Aus der erhaltenen Antwort wird ein Wert mit dem angegebenen Datentyp erstellt und dem Formularmodell hinzugefügt.

Um das Formular zu lokalisieren, muss ebenfalls das BPM-System befragt werden, um beispielsweise Informationen über die benötigte Sprache zu erhalten. Diese Informationen werden dem Modell hinzugefügt. Eine Auswertung dieser Information findet erst bei der Übersetzung des Modells statt.

### 5.4.2 Modellübersetzung

Bei der Modellübersetzung muss zwischen den Kontexten unterschieden werden, für die das Modell übersetzt werden soll. Welche Kontexte angeboten werden können. hängt vom BPM-System ab. Für jeden Kontext, welches vom BPM-System angeboten wird und von der neuen Komponente auch unterstützt werden soll, wird ein eigener Übersetzer benötigt. Daher ist es für den Entwurf der neuen Komponente wichtig, zu entscheiden, welche grafischen Kontexte unterstützt werden sollen.

Um bereits von Anfang an ein großes Einsatzfeld der Formulare abzudecken, soll im Rahmen dieser Arbeit je ein Übersetzungsmodul für einen auf HTML und einen auf SWT basierenden Kontext entwickelt werden. Die Unterschiede in beiden Kontexten liegen in der Darstellung. Für den HTML-Kontext wird ein HTML-Dokument erzeugt, welches später von einem Web-Browser dargestellt wird. Für den SWT-Kontexte hingegen wird ein *SWT-Composite* erzeugt, welches die Oberflächenbibliotheken des Betriebssystems nutzt, um das Formular darzustellen.

Für die Übersetzung in einen bestimmten Kontext müssen jeweils zwei Aspekte berücksichtigt werden. Zunächst wird erläutert, wie das Layout der *Composite* und Formulare übersetzt wird. Anschließend wird gezeigt, wie die Anwendungslogik umgesetzt wird, damit diese sich in beiden Kontexten identisch verhält.

**5.4.2.1 Übersetzung des Layouts** Bei der Übersetzung des Layouts müssen die *Composite Definition* in ein Format übersetzt werden, welches das Aussehen der *Composite Definition* im Kontext der grafischen Umgebung beschreibt. Für den SWT-Kontext handelt es sich dabei um eine Java-Klasse, während für den HTML-Kontext ein HTML-Dokument verwendet wird.

**Übersetzung für den SWT-Kontext** Um ein Formular für den SWT-Kontext zu übersetzen, erwartet das BPM-System die Instanz einer Java-Klasse, welche auf einem *SWT-Composite* basiert. Diese Klasse muss jedoch

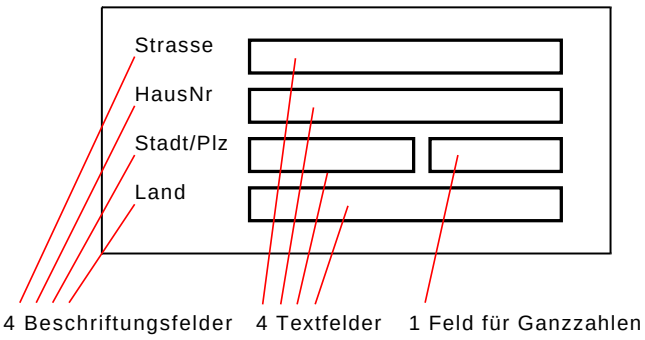


Abbildung 57: Modell eines *Composite* für die Darstellung einer Adresse

| Eigenschaft | Datentyp | Beschreibung |
|---|---|---|
| hintergrundFarbe | INTEGER | Legt die Hintergrundfarben der Text- und Zahlenfelder fest |
| inhalt | ADRESSE | Legt die Adresse fest, welche dargestellt werden soll |

Tabelle 7: Eigenschaften des Adress-*Composite*

erst generiert werden. Das bedeutet, dass zunächst Java-Code erzeugt werden muss und anschließend mit einem Java-Übersetzer in Java-Byte-Code übersetzt werden muss. Dieser Byte-Code kann danach durch den Java-Classloader in die laufende Java-Umgebung integriert werden. Damit steht die Klasse zur Verfügung und kann zu einem Java-Objekt instanziert und dem BPM-System übergeben werden. Dieses stellt mit Hilfe des erhaltenen Objekts das Formular beim Endanwender dar.

Um das komplette Formularmodell zu übersetzten reicht eine Klasse nicht aus. Sowohl für das eigentliche Formular, als auch für jede *Composite Definition*, die verwendet wird, wird Java-Code für eine separate Klasse erzeugt. Innerhalb der Klasse für das Formular werden die Klassen der *Composites Definition* instanziert, um daraus die Formularfelder zu erzeugen.

Bei der Übersetzung der *Composite Definition* werden die Eigenschaften auf die Attribute der Klasse abgebildet. Für die Belegung der Eigenschaften der *Composite Definition* mit Werten, werden entsprechende Methoden generiert.

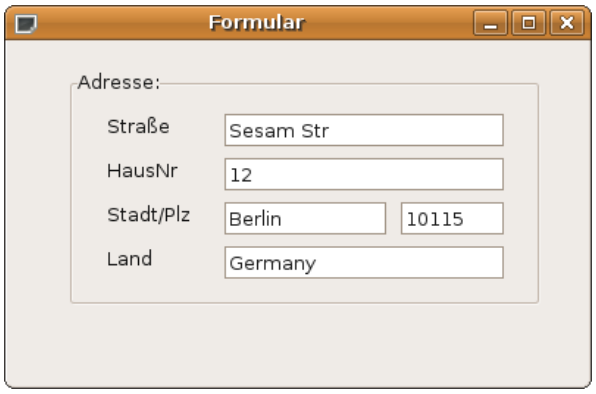


Abbildung 58: *SWT-Composite* für die Darstellung eines Adress-*Composite*

Da die Eigenschaften einen Datentyp haben, müssen auch alle im Formularmodell verwendeten strukturierten Datentypen mit übersetzt werden. Für die primitiven Datentypen ist keine Übersetzung erforderlich, da hierfür äquivalente Java-Datentypen verwendet werden. Bei der Übersetzung der strukturierten Datentypen wird für jeden Datentyp eine eigene Klasse generiert. Die Elemente des Datentyps werden als Attribute der Klasse übersetzt. Diese Attribute sind öffentlich sichtbar und können direkt manipuliert werden, ohne auf die Verwendung von Methoden zurückzugreifen. Der direkte Zugriff ist für die Anwendungslogik wichtig, da innerhalb der Anwendungslogik ebenfalls direkt auf die Elemente von strukturierten Datentypen zugegriffen wird und diese Semantik durch die ScriptEngine nicht verändert werden kann.

Für die Erstellung des Layouts von *Composite Definition* werden die Informationen über die Positionierung des *Composite* bei der Instanzierung der Klasse für die *Composite Definition* mit übergeben. Die Klasse nutzt dann die Methoden von SWT, um diese *Composite* an den richtigen Positionen darzustellen.

Für die primitiven *Composite*, welche für die Darstellung primitiver Datentypen bereits vorhanden sind, werden entsprechende Klassen aus den SWT-Bibliotheken verwendet. Hier gibt es beispielsweise Klassen, welche ein Textfeld oder ein Auswahlfeld darstellen. Für das Zeichnen dieser Elemente ist SWT verantwortlich. Diese SWT-Klassen werden durch das Übersetzungsmodul ebenfalls in Klassen gekapselt, welche eine *Composite Definition* repräsentieren. Dadurch wird gewährleistet, dass diese primitiven *Composite* transparent genutzt werden können, d. h. ihre Schnittstelle unterscheidet sich nicht von den Schnittstellen der *Composite Definition*, welche durch den Formularentwickler erstellt wurden.

```
1  public class SWT_AdressComposite extends Composite
2  {
3    SWT_BeschriftungComposite beschriftung1;
4    SWT_BeschriftungComposite beschriftung2;
5    SWT_TextComposite strComposite;
6    SWT_TextComposite hausNrComposite;
7
8    public SWT_AdressComposite(Composite parent, int x, int y,
9                 int width, int height)
10   {
11     super(parent, SWT.None);
12     setBounds(x, y, width, height);
13
14     beschriftung1 =
15       new SWT_BeschriftungComposite(this, 10, 10, 60, 22);
16     beschriftung2 =
17       new SWT_BeschriftungComposite(this, 10, 40, 60, 22);
18     strComposite =
19       new SWT_TextComposite(this, 90, 10, 190, 22);
20     hausNrComposite =
21       new SWT_TextComposite(this, 90, 40, 190, 22);
22
23     beschriftung1.setzeEigenschaft("inhalt", "Straße");
24     beschriftung2.setzeEigenschaft("inhalt", "HausNr");
25   }
26
27   public void setzeEigenschaft(String eigenschaft, object wert)
28   {
29     if(eigenschaft.equals("inhalt"))
30     {
31       strComposite.setzeEigenschaft("inhalt",
32                       ((Adresse)wert).str);
33       hausNrComposite.setzeEigenschaft("inhalt",
34                       ((Adresse)wert),hausNr");
35     }
36
37     else if(eigenschaft.equals("hintergrundFarbe"))
38     {
39       strComposite.setzeEigenschaft("hintergrundFarbe", wert);
40       hausNrComposite.setzeEigenschaft("hintergrundFarbe", wert);
41     }
42   }
43 }
```

Das neue Composite basiert auf den SWT-Composite

Deklaration der Composite für die Darstellung.

Beim Aufruf des Konstruktors werden die Informationen über die Positionierung übergeben. Diese werden an SWT weitergereicht.

Erzeugen der Composite für die Darstellung. Dabei werden die Daten über die Positionierung innerhalb des Composite angegeben

Initiales Setzen des Inhaltes der Beschriftungsfelder. Der Inhalt gibt an, welcher Text in dem Feld dargestellt werden soll.

Die Eigenschaft "inhalt" hat den Datentyp "Adresse". Wenn diese Eigenschaft gesetzt wird, muss auch der Inhalt der einzelnen Elemente neu gesetzt werden.

Beim Setzen der Hintergrundfarbe sollen die Farben der Text- und Zahlenfelder angepasst werden.

```
1  public class SWT_Formular extends Composite
2  {
3    SWT_AdressComposite adressComposite;
4
5    public SWT_Formular(Composite parent)
6    {
7      super(parent, SWT.None);
8
9      Adresse adresse;
10     adresse.str = "Sesam Str";
11     adresse.hausNr = 12;
12     adresse.stadt = "Berlin";
13     adresse.plz = "10115";
14
15     adressComposite = new SWT_AdressComposite(this, 10, 10, 300, 400);
16     adressComposite.setzeEigenschaft("inhalt", adresse);
17   }
18 }
```

Das Formular basiert ebenfalls auf einem SWT-Composite

Das Formular besteht lediglich aus dem oben beschriebenem Composite

Vorbereiten der Adresse für das Composite. Die Daten dafür kommen beispielsweise aus den Prozessschritt-parametern

Erzeugen des Composite und Setzen der vorbereiteten Adresse als Inhalt.

Abbildung 59: (Vereinfachter) Java-Code für die Erstellung des Adress-*Composite* im SWT-Kontext

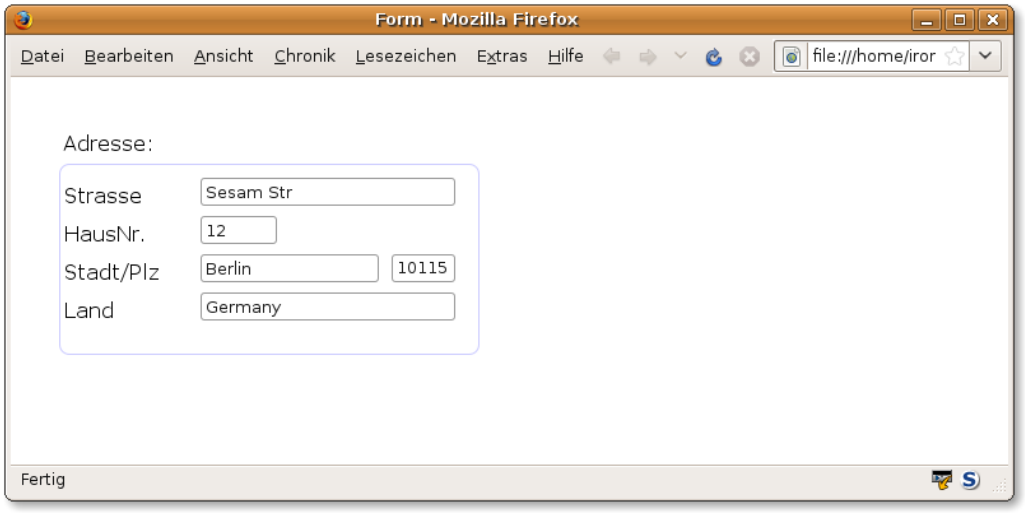


Abbildung 60: HTML-Dokument für die Darstellung eines Adress *Composite*

Wie eine Klasse für eine *Composite Definition* und ihre Verwendung aussieht, demonstriert ein Beispiel. Abbildung 57 zeigt das Modell einer *Composites Definition*, welches eine Adresse anzeigen soll. Diese *Composite Definition* besteht aus mehreren Beschriftungsfeldern, Textfeldern, sowie einem Feld für Ganzzahlen, welches für die Darstellung der Postleitzahl der Adresse zuständig ist. Ausserdem besitzt die *Composite Definition* zwei Eigenschaften, welche in Tabelle 7 gezeigt werden. Die Eigenschaft *hintergrundFarbe*, soll eine Farbe repräsentieren, welche bei den Text- und Zahlenfelder hinterlegt sind. Die Beschriftungsfelder sollen nicht hinterlegt werden. Die Eigenschaft *inhalt* ist die Eigenschaft für den Wert der Adresse. Abbildung 59 zeigt den erzeugten Java-Code. In Abbildung 58 ist die grafische Darstellung des *Composite* als *SWT-Composite* zu sehen.

**Übersetzung für den HTML-Kontext** Innerhalb des HTML-Kontextes ist das Übersetzen des Layouts mit einem größeren Aufwand verbunden. HTML bietet zwar ebenfalls bereits grafische Elemente wie Textfelder oder Auswahlfelder an. Jedoch sind diese statisch im HTML-Dokument integriert. Die Eigenschaften dieser Elemente müssen jedoch während der Berarbeitungszeit dynamisch geändert werden.

Um dieses Problem zu lösen, greift das Übersetzungsmodul für den HTML-Kontext auf JavaScript zurück, da hiermit die dynamische Veränderung des HTML-Dokumentes zur Ausführungszeit möglich ist. In JavaScript ist die Erstellung und Verwendung von Klassen ebenfalls möglich. Daher kann bei

```
1function HTML_AddressComposite(oben, links)
2{
3    this.setzeEigenschaft = HTML_AddressComposite_setzeEigenschaft;
4
5    this.beschriftung1 =
6        new HTML_BeschriftungComposite(oben + 10, links + 0, 100, 22);
7    this.beschriftung2 =
8        new HTML_Beschriftung(oben + 40, links + 0, 100, 22);
9    this.strComposite =
10       new HTML_TextComposite(oben + 10, links + 110, 200, 22);
11   this.hausNrComposite =
12       new HTML_TextComposite(oben + 40, links + 110, 60, 22);
13
14   this.beschriftung1.setzeEigenschaft("inhalt", "Str");
15   this.beschriftung2.setzeEigenschaft("inhalt", "HausNr.");
16}
17
18function HTML_AddressComposite_setzeEigenschaft(eigenschaft, wert)
19{
20   if(eigenschaft == "inhalt")
21   {
22       this.strComposite.setzeEigenschaft("inhalt", wert.str);
23       this.hausNrComposite.setzeEigenschaft("inhalt", wert.hausNr);
24   }
25   else if(eigenschaft == "hintergrundFarbe")
26   {
27       this.strComposite.setzeEigenschaft("hintergrundFarbe", wert);
28       this.hausNrComposite.setzeEigenschaft("hintergrundFarbe", wert);
29   }
30}
```

Beim Aufrufen des Konstruktors werden die Informationen über die Positionierung des Composite mit übergeben

Im Konstruktor werden die Composite erzeugt, die für das Adress-Composite benötigt werden. Dabei wird die relative Positionierung der Composite berechnet und mit übergeben

Die Eigenschaft "inhalt" beschreibt bei Beschriftungsfeldern, den Text der angezeigt werden soll. Dieser wurde zur Entwicklungszeit festgelegt

Die Eigenschaft "inhalt" hat den Datentyp "Adresse". Wenn diese Eigenschaft gesetzt wird, müssen die Composite, die für die Darstellung der einzelnen Elemente zuständig sind informiert werden

Beim Setzen der Hintergrundfarbe sollen nur die Farben der Text- und Zahlenfelder geändert werden

```
1function initiiereFormular()
2{
3    beschriftung = new HTML_BeschriftungComposite(30, 30, 190, 22);
4    beschriftung.setAttribute("inhalt", "Hier ist eine Adresse:");
5
6    var adresse = new Address();
7    adresse.land = "Germany";
8    adresse.strasse = "Sesam Str";
9    adresse.stadt = "Berlin";
10   adresse.plz = 10115;
11   adresse.hausNr = "12";
12
13   adressComposite = new HTML_AddressComposite(60, 30);
14   adressComposite.setzeEigenschaft("inhalt", adresse);
15   adressComposite.setzeEigenschaft("hintergrundFarbe", 0);
16}
```

Vorbereiten der Adresse, welche durch das Composite dargestellt werden soll

Anlegen des Composite und setzen der Eigenschaften. Der Wert 0 für die Hintergrundfarbe führt zu einem weißen Hintergrund

Abbildung 61: (Vereinfachter) JavaScript-Code für die Erstellung des Adress-*Composite* im HTML-Kontext

der Übersetzung der *Composite Definition* und deren Eigenschaften ähnlich vorgegangen werden, wie bei der Übersetzung für den SWT-Kontext. Bei der Übersetzung der *Composite Definition* für primitive Datentypen wird auf die Verwendung der bereits vorhandenen HTML-Elemente für Formulare zurückgegriffen. Diese können durch JavaScript zur Laufzeit dynamisch im DOM-Modell der HTML-Dokumente eingefügt werden.

Um das Vorgehen zu verdeutlichen, wird wieder das Beispiel aus Abbildung 57 verwendet. Die Eigenschaften aus Tabelle 7 werden ebenfalls wieder übersetzt. In Abbildung 60 ist ein Browser zu sehen, der das erzeugte HTML-Dokument zeigt. Abbildung 61 zeigt den Code für eine JavaScript Klasse, welche das *Composite* beim Aufruf des Konstruktors anlegt und eine Metho-

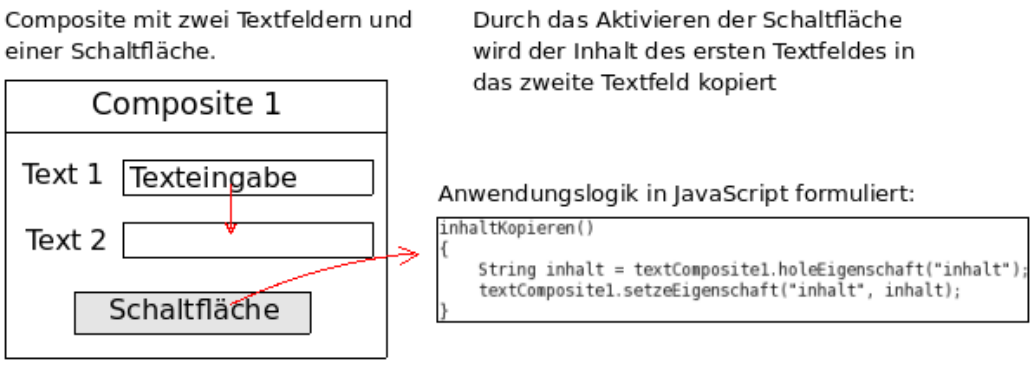


Abbildung 62: *Composite* mit beispielhafter Anwendungslogik

de zum Setzen der Eigenschaften bereitstellt. Ausserdem ist der Code zum Anlegen des *Composite* zu sehen. Dieser Code wird beim Initialisieren des Formulars ausgeführt. In diesem Beispiel werden im *Composite* nur zwei der Elemente einer Adresse verwendet, um den Code übersichtlich zu halten.

**5.4.2.2 Übersetzen der Anwendungslogik** Dadurch, dass bei der Übersetzung des Layouts im HTML-Kontext JavaScript verwendet wird, gestaltet sich die Übersetzung der Anwendungslogik recht einfach, da diese bereits in JavaScript formuliert ist. Abbildung 62 zeigt ein Beispiel für ein *Composite* mit Anwendungslogik. Um die Methoden, welche in der Anwendungslogik definiert sind, nutzen zu können, ist es ausreichend, den Code dieser Methoden in die JavaScript-Klassen der *Composite Definition* einzufügen. Diese können dann durch die Methoden für die Behandlung von Ereignissen aufgerufen. Abbildung 63 zeigt die Integration der Anwendungslogik aus dem Beispiel in die *Composite*-Klassen für den HTML-Kontext.

**Anwendungslogik im SWT-Kontext** Die Übersetzung der Anwendungslogik im SWT-Kontext gestaltet sich etwas umfangreicher. Da die *Composite Definition* mit Hilfe von Java-Code umgesetzt ist, ist es nicht möglich, die in JavaScript formulierte Anwendungslogik direkt als Methoden in die Klassen zu integrieren. Um den JavaScript-Code dennoch verwenden zu können, wird auf die Java-ScriptEngine [Ull09, Kapitel 10.6.3] zurückgegriffen.

Mit Hilfe dieser ScriptEngine kann zur Ausführungszeit des Formulars in einem SWT-Kontext, bei einem Auftreten eines Ereignisses die in JavaScript formulierte Anwendungslogik ausgeführt werden. Dazu wird beim Eintreten

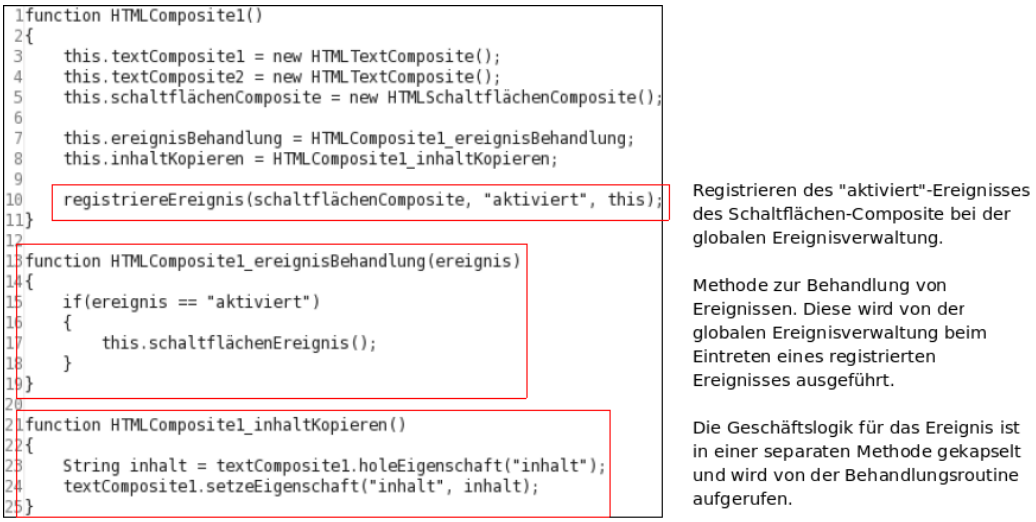


Abbildung 63: Integration der Anwendungslogik in die *Composite*-Klassen für den HTML-Kontext

des Ereignisses die ScriptEngine mit dem Code der Methode geladen, welche diese Ereignisse behandeln. Ausserdem werden der ScriptEngine alle Java-Objekte übergeben, welche für die Ausführung relevant sind. Innerhalb eines Ereignisses in einer *Composite* sind dies die Instanzen der *Composite*, aus denen das *Composite* aufgebaut ist. Die Namen, mit denen auf diese Instanzen zugegriffen werden, sind die Namen, welche für die *Composite* im Layout des *Componsite Definition* vergeben wurden. Daher werden diese Namen auch verwendet, um die Java-Objekte zu binden. Innerhalb des Skripts können auch Zugriffe auf die Methoden der *Composite* vorkommen. Daher müssen bei den Java-Klassen, gleichnamige Methoden existieren. Abbildung 64 zeigt die Integration der in JavaScript formulierten Anwendungslogik in eine *Composite*-Klasse für den SWT-Kontext.

## 5.5 Zusammenfassung

Dieses Kapitel hat einen Entwurf vorgestellt, der die Anforderungen aus Kapitel 3 vollständig umsetzt. Der Entwurf besteht aus einer Entwicklungs- und einer Ausführungsumgebung und einem gemeinsamen Datenmodell.

Das Datenmodell besteht aus mehreren Aspekten die zur Beschreibung von Formularmodellen benötigt werden:

- Datentypen, um die Daten des BPM-System interpretieren zu können


```
1public class SWTComposite1 extends SWTComposite
2{
3    private SWTTextComposite swtTextComposite1;
4    private SWTTextComposite swtTextComposite2;
5    private SWTSchaltflächenComposite swtSchaltflächenComposite;
6
7    public SWTComposite1()
8    {
9        swtSchaltflächenComposite.registriereEreignis("aktiviert", this);
10   }
11
12   public void ereignisBehandlung(String ereignis, SWTComposite composite)
13   {
14       if(ereignis.equals("aktiviert"))
15       {
16           inhaltKopieren();
17       }
18   }
19
20   private void inhaltKopieren()
21   {
22       String script =
23           "String inhalt = textComposite1.holeEigenschaft(\"inhalt\");" +
24           "textComposite1.setzeEigenschaft(\"inhalt\", inhalt);";
25
26       ScriptEngine engine =
27           new ScriptEngineManager().getEngineByName("JavaScript");
28
29       engine.put("textComposite1", swtTextComposite1);
30       engine.put("textComposite2", swtTextComposite2);
31       engine.eval(script);
32   }
33}
```

Abbildung 64: Integration der Anwendungslogik in die *Composite*-Klassen für den SWT-Kontext

- Werte, um Informationen zu repräsentieren
- *Composite* zur Darstellung von Werten in grafischen Elementen
- *Composite Definition* zur Beschreibung von *Composite*
- Formularmodelle zur Beschreibung des Aussehens und des Verhaltens von Formularen

Mit der Entwicklungsumgebung ist es möglich neue strukturierte Datentypen und Aufzählungen zu erstellen, sowie *Composite Definition* und Formularmodelle zu gestalten. Hierfür werden Editoren angeboten, welche auf die jeweilige Aufgabe spezialisert sind. Mit einem Assistenten ist es ausserdem möglich, Formularmodelle und *Composite Definition* größtenteils automatisch generieren zu lassen.

Die Aufgabe der Ausführungsumgebung ist es, aus der abstrakten Beschreibung des Formulars ein konkretes Dokument für den Endanwender zu erzeugen. Hierfür muss zum Einen das erstellte Formularmodell mit den Werten der Prozessschrittparameter, Daten aus externen Quellen und Informationen zur Sprache und Einschränkungen des Endanwenders vervollständigt werden.

Zum Anderen muss das nun vollständige Formularmodell für den grafischen Kontext des Endanwenders übersetzt werden. Dies wurde für den HTML- und den SWT-Kontext erläutert.

Dieser Entwurf ist größtenteils unabhängig von konkreten BPM-Systemen. In einigen Bereichen jedoch, beispielsweise die bei der Übersetzung für einen grafischen Kontext, musste eine Anpassung an ADEPT2 vorgenommen werden, um den Entwurf vollständig gestalten zu können. Bei einer Verwendung in einem anderen BPM-System müssen diese Bereiche erneut angepasst werden.

# 6 Umsetzung der Konzepte

Die vorgestellten Konzepte wurden im Rahmen dieser Arbeit in Form eines Prototypen implementiert. Dafür wurden geeignete Schnittstellen entworfen:

- Datatype: Schnittstelle für Datentypen
- Value: Schittstelle für Repräsentation von Werten
- Script: Schnittstelle für Anwendungslogik
- FormCompositeDefinition: Schnittstelle für *Composite Definition*
- FormComposite: Schnittstelle für *Composite*
- Reference: Schnittstelle für virtuelle Parameter, basierend auf Value
- Form: Schnittstelle für Formularmodelle

Im folgenden Abschnitt werden die aufgezählten Schnittstellen im Detail erläutert.

## 6.1 Datatyp

Quellcode 6 zeigt die Schnittstelle *Datatype*. Mit dieser Schnittstelle werden die Datentypen umgesetzt. Mit den Methoden *isPrimitive()*, *isPrimitive(Primitives type)*, *isList()*, *isComplex()*, *isEnumeration()* kann überprüft werden, um welche Art von Datentyp es sich handelt. Handelt es sich um einen listenwertigen Datentyp, kann mit Hilfe der Methode *getListedDatatype()* der Basistyp der Liste ermittelt werden. Wenn es sich bei dem Datentyp um eine Aufzählung handelt, kann die Methode *getEnumerationOption()* dazu genutzt werden, um die Option der Aufzählung zu ermitteln. Handelt es sich hingegen um einen strukturierten Datentyp, lassen sich mit Hilfe der Methode *getElementNames()* zunächst die Elemente der Struktur ermitteln. Durch die Methode *getElementDatatype()* lässt sich der Datentyp eines Elements herausfinden. Da diese Elemente selbst wieder einen strukturierten Datentyp haben können, lassen sich dadurch komplexe Datentyphierarchien erstellen. Diese dürfen wie bereits angesprochen, keine zyklischen Abhängigkeiten besitzen. Mit der Methode *getDependingDatatypes()* können Datentypen ermittelt werden, zu denen eine Abhängigkeit besteht. Bei listenwertigen Datentypen besteht die einzige Abhängigkeit zu dem entsprechenden Basistyp. Bei strukturierten Datentypen

werden die Abängigkeiten rekursiv für jedes Element ermittelt und verknüpft. Primitive Datentypen und Aufzählungen haben keine Abhängigkeiten.

```
1  public interface Datatype {
2    enum Primitives {
3      BOOLEAN,
4      INTEGER,
5      FLOAT,
6      STRING,
7      BLOB,
8      DATE;
9    }
10   String getName();
11   Datatype getRelativeDatatype(String relElementName)
12   boolean isPrimitive(Primitives type);
13   boolean isPrimitive();
14   boolean isList();
15   Datatype getListedDatatype();
16   boolean isComplex();
17   Set<String> getElementNames();
18   Datatype getElementDatatype(String elementName)
19   boolean isEnumeration();
20   List<String> getEnumerationOptions()
21   Set<Datatype> getDependingDatatypes();
22   void addListener(DatatypeListener listener);
23   void removeListener(DatatypeListener listener);
24 }
```

Quellcode 6: *Datatype*-Schnittstelle

## 6.2 Value

Um einen Wert mit einem speziellen Datentyp anzulegen, wird die Schnittstelle *Value* (Quellcode 7) benötigt, beispielsweise um beim Erstellen eines Formularmodells Standardwerte für Eigenschaften der *Composite* anzugeben. Hat ein Wert einen primitiven Datentyp, können mehrere Methoden genutzt werden, um den Wert unterschiedlich zu interpretieren:

- Long getPrimitiveValueAsInteger()
- String getPrimitiveValueAsString()
- Boolean getPrimitiveValueAsBool()

- Double getPrimitiveValueAsFloat()
- byte[] getPrimitiveValueAsBlob()
- Date getPrimitiveValueAsDate()

Es stehen auch korrespondierende Methoden bereit, um den Wert neu zu setzen.

Handelt es sich bei dem Datentyp des Wertes um eine Liste, stehen andere Methoden bereit um auf dessen Elemente zuzugreifen. Mit *getListLength()* wird die Anzahl der Elemente in der Liste ermittelt. Durch die Methode *getListIndex(int index)* wird auf ein bestimmtes Element der Liste zugegriffen. Mit Hilfe der Methoden *addListItem(Value value)*, *removeListIndex(int index)* und *setListItem(int index, Value value)* wird der Inhalt der Liste entsprechend manipuliert.

Hat der Wert hingegen einen strukturierten Datentyp, stehen die Methoden *getElementValue(String elementName)* und *setElementValue(String elementName, Value value)* zur Verfügung. Damit kann der Wert eines Elements ermittelt oder neu gesetzt werden.

Repräsentiert der Wert die Auswahl einer Aufzählung, ermittelt die Methode *getEnumOption()* die ausgewählte Option. Mit *setEnumOption(Strin optionName)* kann die aktuelle Auswahl geändert werden.

Einen besonderen Wert, der für alle Datentypen möglich ist, ist der NULL-Wert. Da durch die bisher beschriebenen Methoden nicht feststellbar ist, ob ein Wert NULL ist, enthält die Schnittstelle die Methode *isNull()*, mit der dies möglich ist.

```
1  public interface Value {
2    public Datatype getDatatype();
3    public boolean isNull();
4    public boolean isReadOnly();
5    public Value getRelativeValue(String relativeElement);
6    public int getListLength();
7    public Value getListIndex(int index);
8    public void addListItem(Value value);
9    public void removeListIndex(int index);
10   public void setListItem(int index, Value value);
11   public Value getElementValue(String elementName);
12   public void setElementValue(String elementName, Value
       value);
13   public Long getPrimitiveValueAsInteger();
14   public void setPrimitiveInteger(Long value);
```

```
15    public String getPrimitiveValueAsString();
16    public void setPrimitiveString(String value);
17    public Boolean getPrimitiveValueAsBool();
18    public void setPrimitiveBool(Boolean value)
19    public Double getPrimitiveValueAsFloat();
20    public void setPrimitiveFloat(Double value);
21    public byte[] getPrimitiveValueAsBlob();
22    public void setPrimitiveBlob(byte[] value);
23    public Date getPrimitiveValueAsDate();
24    public void setPrimitiveDate(Date value);
25    public String getEnumOption();
26    public void setEnumOption(String optionName);
27  }
```

Quellcode 7: *Value*-Schnittstelle

## 6.3 Script

Für die Angabe von Anwendungslogik wird die Schnittstelle *Script* (Quellcode 8) verwendet. Die Anwendungslogik wird durch JavaScript-Code beschrieben, der in JavaScript-Funktionen gekapselt wird. Diese Funktionen können, wie in JavaScript üblich, mit Parametern versehen werden. Im JavaScript-Code stehen diese Parameter als Objekte zur Vefügung. Eine Instanz der Schnittstelle *Script* kann beliebig viele Funktionen verwalten. Um sämtliche Funktion zu erhalten, kann die Methode *getFunctions()* verwendet werden. Damit erhält man die Funktionsnamen. Die dazugehörigen Parameter können durch die Methode *getFunctionParameters(String methodName)* ermittelt werden. Um den JavaScript-Code zu erhalten, steht die Methode *getFunctionCode(String methodName)* bereit. Um den JavaScript-Code oder die Parameter einer Funktion zu manipulieren, gibt es ebenfalls entsprechende Methoden.

```
1  public interface Script {
2    Set<String> getFunctions();
3    List<String> getFunctionParameters(String methodName);
4    String getFunctionCode(String methodName);
5    void addFunction(String functionName,
6    List<String> functionParameter, String fnuctionCode);
7    void addFunctionParameter(String functionName, String
       functionName);
8    void setFunctionCode(String functionname, String
       functionCode);
```

```
9     void removeFunction(String functionName);
10    void removeFunctionParameter(String functionName,
          String parameterName);
11 }
```

Quellcode 8: *Script*-Schnittstelle

## 6.4 FormCompositeDefinition

Um eine *Composite Definition* anzulegen, wird die Schnittstelle *FormCompositeDefinition* (Quellcode 9) verwendet. Jedes grafische Element, welches durch eine *Composite Definition* beschrieben wird, kann nur Werte mit einem bestimmten Datentyp darstellen. Dieser Datentyp wird durch die Methode *getSupportedDatatype()* ermittelt und ist auch implizit der Eigenschaft *inhalt* zugeordnet.

Um Informationen über die Eigenschaften einer *Composite Definition* zu erhalten, stehen die Methoden *getAttributes()*, *getAttributeDatatype(String attribute)*, *getAttributeDefaultValue(String attribute)* und *getAttributeDescription(String attribute)* bereit. Damit können die Namen der Eigenschaften ermittelt werden, die zugeordneten Datentypen, die Standardwerte und eine Beschreibung für den Formularentwickler.

Für mögliche Ereignisse, die ausgelöst werden können, gibt es zwei weitere Methoden: Die Methode *getEvents()* gibt die Namen aller Ereignisse zurück. Um eine Beschreibung eines Ereignisses für den Formularentwickler zu erhalten, steht die Methode *getEventDescription(String event)* bereit.

Es gibt spezielle *FormCompositedefinition*, welche nicht durch den Formularentwickler angelegt werden können. Dabei handelt es sich um primitive *FormCompositedefinition*, welche grundlegende grafische Elemente darstellen, wie zum Beispiel ein Textfeld, ein Beschriftungsfeld oder eine Schaltfläche. Solche *FormComopositeDefinintions* stehen von Anfang an zur Verfügung. Mit Hilfe der Methode *isPrimitive()* ist es möglich zu ermitteln, ob es sich um eine primitive *FormCompositeDefinition* handelt.

Stellt eine *FormCompositeDefinition* kein primitives grafisches Element dar, ist es aus *Composite* aufgebaut. Diese können durch die Methode *getUsedComposites()* ermittelt werden.

Um das Verhalten der *FormCompositeDefinition* beim Eintreten von Ereignissen zu steuern, enthält diese ein *Script*. Darauf kann durch die Methode *getScript()* zugegriffen werden.

```
1  public interface FormCompositeDefinition {
2    String getName();
3    Datatype getSupportedDatatype();
4    Set<String> getAttributes();
5    Datatype getAttributeDatatype(String attribute);
6    Value getAttribDefaultValue(String attribute);
7    String getAttribDescription(String attribute);
8    Set<String> getEvents();
9    String getEventDescription(String event);
10   Set<FormComposite> getUsedComposites();
11   Script getScript();
12   boolean isPrimitive();
13 }
```

Quellcode 9: *FormCompositeDefinition*-Schnittstelle

## 6.5 FormComposites

*FormCompositeDefinition* stellen grafische Elemente dar, die in einem Formular oder einer weiteren *FormCompositeDefinition* verwendet werden können. Dafür muss jedoch zunächst ein *Composite* erstellt werden. Hierzu dient die Schnittstelle *FormComposite*. Diese Schnittstelle dient zur Speicherung von Instanzspezifischen Aspekten, wie zum Beispiel der Name der Instanz, die relative Positionierung oder die Bindung von Ereignissen an Anwendungslogik.

Jedes *FormComposite* basiert auf einer *FormCompositeDefinition*. Diese kann durch die Methode *getDefinition()* ermittelt werden. Der Name der Instanz wird durch die Methode *getInstanceName()* zurückgegeben. Dieser Name kann dazu genutzt werden, um in der Anwendungslogik eines Formulars oder einer umschließenden *Composite Definition* auf das *Composite* zuzugreifen. Folgende Methoden erlauben den Zugriff auf die Daten über die relative Positionierung:

- *getInstanceX()*
- *getInstanceY()*
- *getInstanceWidth()*

- *getInstanceHeight()*

Um die Werte für die Eigenschaften zu ermitteln, stehen ebenfalls mehrere Methoden zur Verfügung. Durch *getUsedAttributes()* erhält man eine Liste der Eigenschaften, für welche Werte angegeben sind. Für Eigenschaften, für die kein expliziter Wert angegeben wurde, gilt der Standardwert, der in der zugehörigen *FormCompositeDefinition* definiert wurde. Falls jedoch ein Wert gesetzt wurde, kann dieser durch die Methode *getInitialAttributeValue(String attrib)* ermittelt werden.

Ein analoges Schema wird für die Behandlung von Ereignissen verwendet. Durch die Methode *getUsedEvents()* erhält man eine Liste von Ereignissen, welche durch die umschließende *Composite Definition* oder ein Formular abgefangen werden. Die Methode *getEventHandler(String event)* ermittelt den Namen der Funktion, die zur Behandlung des Ereignisses verwendet wird. Der Name bezieht sich dabei auf die verfügbaren Funktionen aus dem *Script* der umschließenden *Composite Definition* oder des umschließenden Formulars.

```
1  public interface FormComposite {
2    public FormCompositeDefinition getDefinition();
3    public String getInstanceName();
4    public int getInstanceRow();
5    public int getInstanceColumn();
6    public int getInstanceWidth();
7    public int getInstanceHeight();
8    public Set<String> getUsedAttributes();
9    public Value getInitialAttributeValue(String attrib);
10   public void setInitialAttributeValue(String attribute,
       Value value);
11   public Set<String> getUsedEvents();
12   public String getEventHandler(String event);
13   public void setEventHandler(String event, String
       function);
14 }
```

Quellcode 10: *FormComposites*-Schnittstelle

## 6.6 Reference

Die Werte für die Eigenschaften von *Composite* können auch an die Parameter des umschließenden Formulars gebunden werden, bzw. an die Eigen-

schaften der umschließenden *Composite Definition*. Um dies zu realisieren, wird eine weitere Schnittstelle benötigt. Quellcode 11 zeigt die Schnittstelle *Reference*, welche die Schnittstelle *Value* erweitert. Mit einer *Reference* ist es möglich, auf Werte zu verweisen, die noch nicht zur Verfügung stehen. Durch die neue Methode *isValueReady()* kann geprüft werden, ob der Wert bereits verfügbar ist oder nicht. Wenn der Wert verfügbar ist, kann die übliche Schnittstelle von *Value* verwendet werden, um diesen Wert abzufragen. Ist der Wert noch nicht verfügbar, ist dies nicht möglich. Es gibt 4 Implementierungen der Schnittstelle:

- *ParameterReference*
- *HistoricalParameterReference*
- *ContainerValueReference*
- *ExternalReference*

Wird ein Wert für eine Eigenschaft in einer *Composite* innerhalb eines Formulars angegeben, ist es mit einer *ParameterReference* möglich auf einen Ein- oder einen Ausgabeparameter des Formulars zu verweisen. Möchte der Formularentwickler die Ausgabeparameter einer vergangenen Aktivität mit in das Formular einbinden, kann dieser eine *HistoricalParameterReference* verwenden. Mit der *ContainerValueReference* ist es möglich, auf den Wert einer Eigenschaft von einer umschließenden *Composite Definition* zu verweisen. Durch eine *ExternalReference* lassen sich Werte aus einer externen Datenbank beziehen. In allen vier Fällen ist die Angabe eines Elements möglich, falls der referenzierte Wert einen strukturierten Datentyp besitzt.

```
1 public interface Reference extends Value {
2   boolean isValueReady();
3 }
```

Quellcode 11: *Reference*-Schnittstelle

## 6.7 Form

Für die Erstellung eines Formularsmodells wird die Schnittstelle *Form* verwendet (Quellcode 12). Sie bietet ähnlich wie die *FormCompositeDefinition* ein *Script* an, um die Anwendungslogik des Formulars festzulegen. Diese Anwendungslogik kann durch die Methode *getScript()* ermittelt werden. Um das Layout des Formularmodells festlegen zu können, verfügt eine *Form* auch

über *FormComposites*. Die Methode *getUsedComposites()* gibt eine Liste aller verwendeten *FormComposites* zurück.

```
1  public interface Form {
2    String getName();
3    Set<FormComposite> getUsedComposites();
4    Script getScript();
5    void setName(String name);
6    void addComposite(FormComposite composite);
7    void removeComposite(FormComposite composite);
8    void moveComposite(FormComposite composite, int row,
       int col);
9    void setScript(Script script);
10 }
```

Quellcode 12: *Form*-Schnittstelle

# 7 Zusammenfassung und Ausblick

## 7.1 Zusammenfassung

Die vorliegende Arbeit befasste sich mit der automatischen Erstellung von Oberflächen für Benutzerinteraktionen in BPM-Systemen. Im Rahmen dieser Arbeit wurden intelligente Formulare behandelt, die dem Austausch von Informationen mit dem Benutzer dienen und ein flexibles Verhalten ermöglichen. Das Ziel war es, Lösungen zu erarbeiten, welche die Erstellung und Integration solcher Formulare in vorhandene BPM-Systeme ermöglicht. Um die theoretischen Überlegungen auch praktisch umzusetzen, sollte ein Prototyp in Form einer Softwarekomponente entwickelt werden, welcher in das ADEPT2-System integriert werden konnte.

Nach der Diskussion der Grundlagen wurden die Anforderungen analysiert, welche sich für eine neue Softwarekomponente ergeben. Dabei wurde unter anderem gezeigt, dass die Komponente aus zwei Bereichen besteht: Einer Entwicklungsumgebung, mit der die Formulare erstellt werden und einer Laufzeitumgebung, welche diese Formulare dem Endanwender präsentiert. Für die Präsentation spielt der verwendete grafische Kontext ein wichtige Rolle. Nicht bei jedem Endanwender stehen alle grafischen Kontexte bereit, die vom BPM-System angeboten werden. Es kann daher nicht mit Sicherheit gesagt werden, in welchem grafischen Kontext das Formular dargestellt werden muss. Daher muss die Entwicklungsumgebung die Formulare in einem unabhängigen Format speichern. Die Laufzeitumgebung muss dieses dann entsprechend, der zur Verfügung stehenden Kontexte interpretieren.

Nach der Erläuterung der Anforderungen wurden darauf aufbauend Kriterien für eine Evaluierung entwickelt. In dieser Evaluierung wurden Produkte und Technologien untersucht, die sich ebenfalls mit der Gestaltung und Darstellung von grafischen Benutzeroberflächen befassen. Untersucht wurden dabei Inubit, Adobe Lifecycle Designer, Microsoft Infopath, JAXFront, WJP Form und XForms. Die Kriterien für die Evaluierung waren eine einfach Bedienbarkeit, Lizenz und Integrationsfähigkeit, unterstützte Datentypen, die Möglichkeit externe Daten einzubinden, die verfügbaren grafische Elemente, die Integrationsfähigkeit von Anwendungslogik, die möglichen grafische Kontexte sowie die Möglichkeit Formulare zu lokalisieren. Bei der eigentlichen Evaluierung hat sich herausgestellt, dass die Verwendung von Formularen bereits gut etabliert ist. Ein Produkt, welches alle Anforderungen erfüllen kann, gab es jedoch nicht.

Aus diesem Grund mussten eigene Konzepte untersucht werden, welche die Anforderungen umsetzen. Neben der Entwicklungsumgebung und der Laufzeitumgebung wurde auch ein gemeinsames Datenmodell besprochen. Dieses Datenmodell beinhaltet mehrere Konzepte: Neben primitiven Datentypen wurden Aufzählungen, listenwertige und strukturierte Datentypen eingeführt. Damit können die Ein- und Ausgabeparameter von Aktivitäten mit Formularen genauer spezifiziert werden, als mit den bisherigen Datentypen. Für die Repräsentation von Daten mit einem bestimmten Datentyp wurde ein Konzept für Werte vorgestellt. Um diese Werte grafisch darzstellen, werden grafische Elemente in Formularen verwendet. Um langfristig flexible Darstellungsmöglichkeiten zu haben wurden dafür *Composite Definition* eingeführt. Diese sind aus bereits vorhandenen grafischen Elemente zusammengestellt und verfügen über eine eigene Anwendungslogik. Dadurch sind sie in der Lage auch Werte mit sehr komplex strukturierten Datentypen darzustellen und den Endanwender beispielsweise bei falschen Eingaben auf die Fehler hinzuweisen. Um diese *Composite Definition* für die Erstellung von Formularen oder in weiteren *Composite Definition* zu verwenden, wurden zusätzlichen noch *Composite* eingeführt. Diese bilden die Instanz einer *Composite Definition* und beinhalten nur Instanzspezifische Aspekte, wie zum Beispiel Angaben über Positionierung oder den Instanznamen, der in der Anwendungslogik dazu verwendet wird, um auf das *Composite* zuzugreifen.

Den Abschluss der Arbeit stellte eine prototypische Implementierung des Entwurfs dar. Für Implementierung wurden Schnittstellen entwickelt, mit denen sich die vorgestellten Konzepte umsetzen konnten. Diese Schnittstellen wurden besprochen.

## 7.2 Ausblick

Während Formulare früher lediglich zum Austauschen von Daten zwischen System und Endanwender genutzt wurden, ist es nun auch möglich Formularen mit einem komplexen Verhalten zu verwenden. Diese hat den Vorteil, dass triviale Prüfungen von Eingaben und das Aufbereiten von Daten bereits im Formular durchgeführt werden können. Dadurch werden beim Modellieren von Geschäftsprozessen Aktivitäten eingespart, welche diese Arbeiten sonst in einem separatem Prozessschritt durchführen müssen. Die Prozesses werden dadurch übersichtlicher. Unnötige Komplexität wird vermieden. Dieser Vorteil wird noch deutlicher, wenn in einem Prozess das selbe Formular mehrmals benötigt wird. In diesem Fall kann nicht nur das Layout des Formulars wiederverwendet werden. Da die Anwendungslogik, die das Ver-

halten festlegt im Formular integriert ist, muss diese Anwendungslogik nicht erneut entworfen werden.

Ein weiterer Aspekt der neuen Formulare ist, dass diese größtenteils automatisch generiert werden können. Für die Gestaltung von Formularen sind somit keine teuren Softwareingenieure mehr nötig. Spezialisierte Mitarbeiter eines Unternehmens sind ausreichend um die Benutzerinteraktionen in einem Geschäftsprozess umzusetzen.

Intelligente Formulare sind daher ein wichtiger Schritt, wenn es darum geht die Gestaltung von Geschäftsprozessen durch die Entwickler zu vereinfachen und die Endanwender bei ihrer Arbeit zu unterstützen.

# A Abbildungsverzeichnis

# B Quellcodeverzeichnis

# C Tabellenverzeichnis

# D Literaturverzeichnis


[Ado09a] ADOBE: *FormCalc*, 2009. http://help.adobe.com/de_DE/livecycle/8.2/acrobat_designer/wwhelp/wwhimpl/common/html/wwhelp.htm?context=Adobe_LiveCycle_Designer-Hilfe\&file=Overview.112.1.html Abruf: 02.06.2009.

[Ado09b] ADOBE: *Lifecycle Designer*, 2009. http://www.adobe.com/de/products/livecycle/designer/ Abruf: 23.05.2009.

[Ado09c] ADOBE: *Lifecycle Enterprise Suite*, 2009. http://www.adobe.com/de/products/livecycle/ Abruf: 23.05.2009.

[Ado09d] ADOBE: *Reader*, 2009. http://www.adobe.com/de/products/reader/ Abruf: 23.5.2009.

[Aja09] AJAXIAN: *JSEclipse*, 2009. http://ajaxian.com/archives/jseclipse-javascript-editor-with-code-completion Abruf: 03.06.2009.

[Alb07] ALBY, TOM: *Web 2.0. Konzepte, Anwendungen, Technologien.* Hanser Verlag, 2007.

[App07] APPLE: *MAC OS X Leopard*, Oktober 2007. http://www.apple.com/de/macosx/techspecs/ Abruf: 27.05.2009.

[Ari09] ARISTAFLOW: *AristFlow BPM Suite*, 2009. http://aristaflow.com/ Abruf: 03.06.2009.

[BWF09] BWF VIRTUELL INSIDE: *E-Learning für Blinde und Sehbehinderte*, 2009. http://inside.bfwvirtuell.de/blind_und_pc Abruf: 06.06.2009.

[Dad06] DADAM, PETER: *ADEPT2 - Ein adaptives Prozess-Management-System der nächsten Generation.* Tagungsband Stuttgarter Softwaretechnik Forum, 2006.

[Dad09] DADAM, PETER: *Von ADEPT zur AristaFlow BPM Suite.* Koellen-Verlag, 2009.

[Ecl09a] ECLIPSE: *Standard Widget Toolkit*, Mai 2009. http://www.eclipse.org/swt/ Abruf: 08.05.2009.

[Ecl09b] ECLIPSE: *Understanding Layouts in SWT*, 2009. http://www.eclipse.org/articles/article.php?file=

Article-Understanding-Layouts/index.html Abruf: 25.05.2009.

[Ein09] *Einkaufsplaner Online*, Mai 2009. http://www.einkaufsplaner-online.de Abruf: 10.03.2009.

[FID09] *Fiducia*, 2009. http://www.fiducia.de/ Abruf: 01.06.2009.

[GNU91] GNU: *General Public License, version 2*, Juni 1991. http://www.gnu.org/licenses/gpl-2.0.html Abruf: 16.05.2009.

[GTK09] *Gimp Toolkit*, Mai 2009. http://www.gtk.org/ Abruf: 09.05.2009.

[Int93] International Organization for Standardization: *ISO/IEC 2382 - Information technology – Vocabulary – Part 1: Fundamental terms*, 1993.

[Inu09a] *Inubit BPM-Suite*, 2009. http://www.inubit.com/index.php Abruf: 23.05.2009.

[Inu09b] *Inubit BPM-Suite: Modulhandbuch*, 2009.

[ISO02] ISO (International Organization for Standardization): *ISO/IEC 16262 - ECMAScript language specification*, Juni 2002.

[ISO09] *International Organization for Standardization*, 2009. http://iso.org Abruf: 27.05.2009.

[JAX] *JAXFront: Smart GUI Solutions.* http://www.jaxfront.com Abruf: 17.05.2009.

[Joh95] Johnson, Erich Gamma; Richard Helm; Ralph E.: *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, März 1995.

[Jur06] Jurisch, Martin: *Konzeption eines Rahmenwerkes zur Erstellung und Modifikation von Prozessvorlagen und -instanzen.* Universität Ulm, März 2006.

[Mic09a] Microsoft: *Microsoft Foundation Classes*, Mai 2009. http://msdn.microsoft.com/de-de/library/d06h2x6e(VS.80).aspx Abruf: 09.05.2009.

[Mic09b] Microsoft: *.NET-Framework*, 2009. http://www.microsoft.com/NET/ Abruf: 24.05.2009.

[Mic09c] MICROSOFT: *SQL-Server*, 2009. http://www.microsoft.com/germany/sql/2008/default.mspx Abruf: 10.06.2009.

[Mic09d] MICROSOFT: *Visual Basic*, Mai 2009. http://www.microsoft.com/germany/express/product/visualbasicexpress.aspx Abruf: 19.05.2009.

[Mic09e] MICROSOFT: *Visual Basic for Application*, 2009. http://msdn.microsoft.com/de-de/isv/bb190538(en-us).aspx Abruf: 24.05.2009.

[Mic09f] MICROSOFT: *Windows Forms*, Mai 2009. http://windowsclient.net/ Abruf: 09.05.2009.

[Mic09g] MICROSOFT: *Windows SharePoint Services Overview*, 2009. http://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/%5BMS-WSSO%5D.pdf Abruf: 02.06.2009.

[Moz09] MOZILLA: *Firefox*, 2009. http://www.mozilla.com/en-US/firefox/personal.html Abruf: 24.05.2009.

[Mü08a] MÜNZ, STEFAN: *Einführung in JavaScript und DOM*, September 2008. http://de.selfhtml.org/javascript/intro.htm Abruf: 07.05.2009.

[Mü08b] MÜNZ, STEFAN: *HTML: Felder für Datei-Upload*, September 2008. http://de.selfhtml.org/html/formulare/datei_upload.htm Abruf: 11.05.2009.

[Ope09] OPENGROUP: *Motif*, Mai 2009. http://www.opengroup.org/motif/ Abruf: 09.05.2009.

[Pal09] PALM: *Palm OS*, 2009. http://www.palm.com/de/de/ Abruf: 27.05.2009.

[Ree08] REED, DORON ROSENBERG; AARON: *Mozilla XForms Plugin Version 0.8.6ff3*, Oktober 2008. https://addons.mozilla.org/de/firefox/addon/824 Abruf: 20.01.2009.

[Rei00] REICHERT, MANFRED: *Dissertation: Dynamische Ablaufänderungen in Workflow-Management-Systemen.* Universität Ulm, 2000.

[Rei06] REICHERT, MANFRED; DADAM, PETER: *ADEPTflex—Supporting Dynamic Changes of Workflows Without Losing Control.* Kluwer Academic Publishers, März 2006.

[RFC05] *RFC: Common Format and MIME Type for Comma-Separated Values (CSV) Files*, Oktober 2005. http://tools.ietf.org/html/rfc4180 Abruf: 24.05.2009.

[Sun06] Sun Microsystem: *Java Platform Enterprise Edition*, Mai 2006. http://java.sun.com/javaee/technologies/javaee5.jsp Abruf: 24.05.2009.

[Sun09a] Sun Microsystem: *Abstract Window Toolkit*, Mai 2009. http://java.sun.com/products/jdk/awt/ Abruf: 09.05.2009.

[Sun09b] Sun Microsystem: *Swing*, Mai 2009. http://java.sun.com/javase/technologies/desktop/ Abruf: 09.05.2009.

[Sun09c] Sun Microsystem: *The Java Database Connectivity (JDBC)*, 2009. http://java.sun.com/javase/technologies/database/index.jsp Abruf: 03.06.2009.

[Tro09] Trolltech: *QT*, Mai 2009. http://www.qtsoftware.com/ Abruf: 09.05.2009.

[Ull09] Ullenboom, Christian: *Java ist auch eine Insel*. Galileo Computing, 2009.

[Ulm09] *Universität Ulm*, 2009. http://www.uni-ulm.de Abruf: 23.05.2009.

[Unt03] Unterstein, Günter Matthiessen; Michael: *Relationale Datenbanken und SQL – Konzepte der Entwicklung und Anwendung*. Addison-Wesley, Mai 2003.

[W3C98] W3C: *Cascading Style Sheets 2*, Mai 1998. http://www.w3.org/Style/CSS/ Abruf: 31.05.2009.

[W3C07] W3C: *XForms 1.1*, November 2007. http://www.w3.org/MarkUp/Forms/ Abruf: 17.05.2009.

[W3C08] W3C: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, November 2008. http://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals Abruf: 23.05.2009.

[W3C09a] *W3C: The World Wide Web Consortium*, 2009. http://www.w3.org/ Abruf: 23.05.2009.

[W3C09b] W3C: *XForms - Introductory Example No. 1*, 2009. http://ubiquity-xforms.googlecode.com/svn/trunk/

testsuite/W3C-XForms-1.1/Edition1/Chapt02/2.1.a.xhtml Abruf: 07.06.2009.

[Wen06] Wenz, Christian: *AJAX. schnell + kompakt.* Entwickler.Press, April 2006.

[Wik09a] Wikipedia: *Liste von GUI-Bibliotheken*, Mai 2009. http://de.wikipedia.org/wiki/Liste_von_GUI-Bibliotheken Abruf: 08.05.2009.

[Wik09b] Wikipedia: *What you see is what you get (WYSIWYG)*, Mai 2009. http://de.wikipedia.org/wiki/WYSIWYG Abruf: 19.05.2009.

[WJP09] *WJ&P Systemhaus AG*, 2009. http://www.wjp.de/ Abruf: 23.05.2009.

[WX09] *WXWidgets*, Mai 2009. http://wxwidgets.org/ Abruf: 09.05.2009.

# Ehrenwörtliche Erklärung

Hiermit versichere ich, Florian Micheler, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ulm, 9. Juni 2009

Ort, Datum

Florian Micheler

# Danksagung

An dieser Stelle möchte ich mich bei einigen Personen bedanken, die mich während meiner Arbeit besonders unterstützt haben.

An erster Stelle danke ich meinen Eletern, Heinz und Karin Micheler. Ohne euch wäre mein Studium nicht möglich gewesen. Euer Vertrauen und eure Geduld haben mir immer Zuversicht gegeben.

Für die Betreung dieser Arbeit danke ich Ulrich Kreher. Du hast mich nicht nur unterstützt, sondern durch Deine hohen Ansprüche auch stets motiviert. Deine Vorschläge und Erfahrung waren es, die es mir ermöglichten die Arbeit Schritt für Schritt zu verbessern.

Neben Ulrich möchte ich mich auch bei seinen Kollegen bei AristaFlow für die hilfreichen Tipps bdedanken, die mir sehr geholfen haben. Zudem waren die täglichen Tischkickerspiele mit euch immer eine wilkommene Abwechselung und halfen oft dabei, wieder einen klaren Kopf zu bekommen.

Zuletzt danke ich noch besonders Bastian Glöckle, der sich sehr oft Zeit für Korrekturen an der Arbeit genommen hat. Deine zahlreichen Anregungen waren eine große Hilfe und Erleichterung für mich.