ulm university universität
uulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Implementierung einer Visualisierungskomponente für Prozesssichten

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Johannes Hofmann
johannes.hofmann@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Jens Kolb

2012

Fassung 21. Juni 2012

Satz: PDF-LaTeX 2ε

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Heutzutage werden in vielen Unternehmen Prozessmodelle verwendet um die internen Abläufe und Aufgaben darzustellen. Primäres Ziel solcher Prozessmodelle ist immer die Dokumentation der Prozesse, jedoch dienen sie auch dazu Kosten zu reduzieren und Abläufe zu optimieren bzw. zu automatisieren. Dazu gibt es zum Erstellen und Ausführen von Prozessmodellen unterschiedliche Modellierungsprogramme. Deren graphische Darstellung variiert zwar untereinander, dennoch besteht bei allen das Problem, dass sie auf Grund ihrer Darstellung als gerichteter Graph für einen unerfahrenenen Nutzer schwer zu verstehen oder gar zu modellieren sind. Jedoch ist das intuitive Verständnis eines Prozessmodells ein wichtiger Aspekt in der Praxis, da die Personen, die sich mit dem Prozessmodell befassen bzw. nach diesem arbeiten sollen, oft wenig Erfahrung mit dem Verstehen von graphenbasierten Darstellungen haben. Somit sind spezielle Einweisungen oder Schulungen in vielen Fällen notwendig, da das Prozessmodell sonst falsch verstanden werden könnte, was in der Regel hohe Kosten verursacht. Deshalb sind intuitivere Darstellungen von Prozessmodellen notwendig, um Zeit und Geld zu sparen, indem unerfahrenen Nutzern das Verständnis von Prozessmodellen erleichtert wird. Speziell große Prozessmodelle überfordern den normalen Nutzer bei der Betrachtung. In vielen Fällen sind in diesen Prozessmodellen nicht unbedingt alle Schritte für den Nutzer relevant. Deswegen ist es von großem Vorteil, wenn jeder Nutzer sein Prozessmodell auf die für ihn relevanten Arbeitsschritte reduzieren kann, weshalb eine leicht verständliche Bearbeitung und individuelle Anpassung eines Prozessmodells sinnvoll wäre.

1.2 Zielsetzung

Ziel der Arbeit ist es, eine intuitive Darstellung eines Prozessmodells in Form eines Formularen zu implementieren, welches auch von unerfahrenen Nutzern leicht verstanden und bearbeitet werden kann. Dazu sollen Prozessmodelle auf Basis von AristaFlow für die Entwicklung verwendet werden. Das zu entwickelnde Programm, das Prozessmodelle als Formular darstellt, nennt sich *proViewForm*.

Die Darstellung des Prozessmodells soll so sein, dass das Prozessmodell intuitiv und einfach zu verstehen ist. Alle verfügbaren Funktionen sollen klar ersichtlich sein und intuitiv verstanden werden. Zudem soll der Nutzer die Möglichkeit haben die formularbasierte Darstellung, durch ein-

und ausblende Funktionen, nach den für ihn wichtigen Prozessschritten auszurichten. Ebenso soll es dem Nutzer möglich sein mehrere Sichten eines Prozessmodells zu erstellen und diese verändern zu können. Dabei soll der Nutzer bestimmte Aktivitäten oder gar ganze Blöcke löschen können. Desweiteren soll er auch neue Aktivitäten hinzufügen und bestehende Aktivitäten umbenennen können, sowie mehrere Aktivitäten oder ganze Blöcke zusammen fassen.

Allgemein ist aber deutlich zu sagen, dass es hierbei primär um die intuitive Darstellung eines Prozessmodells geht und nicht darum einen vollständigen Editor für Prozessmodelle zu entwickeln. Zwar soll es auch möglich sein, die dargestellten Prozessmodelle zu bearbeiten und individuell anzupassen, jedoch nicht von Grund auf zu modellieren. Allerdings wird dieser Aspekt am Ende der Arbeit nochmal genauer betrachtet und diskutiert.

In Kapitel 2 wird analysiert, welche Elemente und Funktionen eines Prozessmodells für die formularbasierte Darstellung benötigt werden. In Kapitel 3 wird erläutert, wie die einzelnen Elemente der formularbasierten Darstellung aufgebaut sein sollen und wie die Funktionen der formularbasierten Darstellung umgesetzt werden sollen. Zudem wird erläutert, wie die formularbasierte Darstellung genau arbeitet und welche Abläufe und Zugriffe dafür im Hintergrund nötig sind. Kapitel 4 stellt die Implementierung der Visualisierungskomponente für Prozesssichten, mit allen Elementen und Funktionen, vor. Zum Schluss wird in Kapitel 5 mögliche Erweiterungen und Weiterentwicklungen des Programms diskutiert und die Arbeit zusammen gefasst.

2 Grundlagen

In diesem Kapitel wird die grundlegende Struktur eines Prozessmodells vorgestellt und die Bedeutung der einzelnen Elemente im Bezug auf eine formularbasierte Darstellung diskutiert. Ein Prozessmodell lässt sich in einen Kontroll- und einen Datenfluss untergliedern. Im Kontrollfluss werden Ausführungsreihenfolgen und -bedingungen festgelegt (siehe Kapitel 2.1), während im Datenfluss die Ein- und Ausgabeparameter der einzelnen Aktivitäten erfasst werden (siehe Kapitel 2.2). Aktivitäten sind die einzelnen Prozessschritte eines Prozessmodells, sie stellen die eigentlichen Arbeitsschritte in einem Prozess dar.

2.1 Kontrollfluss

Der Kontrollfluss beschreibt die zeitliche Ausführungsreihenfolge der Aktivitäten, innerhalb eines Prozessmodells. Dieser spielt bei der formularbasierten Darstellung die Hauptrolle, während der Datenfluss eine untergeordnetere Rolle spielt.

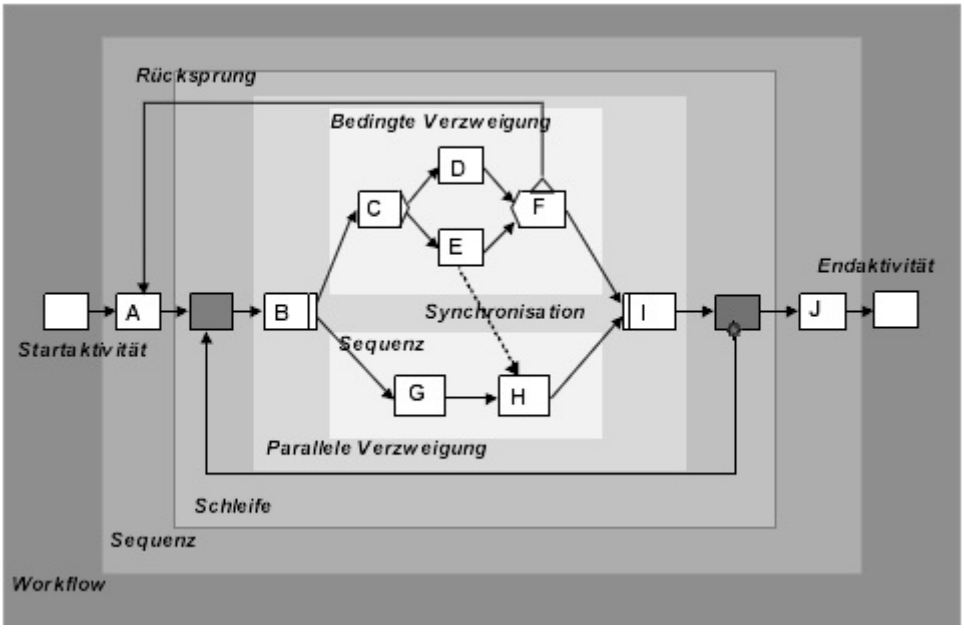

Abbildung 2.1: Kontrollfluss eines Prozessmodells [2]

Der Kontrollfluss besteht aus Aktivitäten, Schleifen und Verzweigungen, welche mit gerichteten Kontrollfluss- und Synchronisationskanten verbunden sind. Abbildung 2.1 zeigt, dass der

Kontrollfluss einer *Blockstrukturierung* unterliegt [2], wobei Kontrollflusselemente, wie Verzweigungen und Schleifen, jeweils einzelne Blöcke bilden, mit einer eindeutigen Eingangs- und Ausgangskante. Diese Blöcke werden auch *Kontrollblöcke* genannt und können beliebig ineinander geschachtelt werden; sich jedoch niemals überschneiden.

Im Folgenden werden die unterschiedlichen Kontrollflusselemente detailliert betrachtet, welche die Bausteine des Kontrollflusses darstellen.

2.1.1 Aktivität

Aktivitäten sind elementare Bausteine eines Prozessmodells, da sie in jedem Prozessmodell enthalten sind und die eigentlichen Prozessschritte darstellen. In graphenbasierten Darstellungen werden sie typischerweise als Knoten dargestellt. Außer der *Start-* und *Endaktivität* besitzen alle Aktivitäten immer genau einen Vorgänger- und einen Nachfolgerknoten, wobei in diesem Zusammenhang Knoten neben Aktivitäten auch Verzweigungen und Schleifen sein können.

2.1.2 Kontroll- und Synchronisationskanten

Die *Kontrollkanten* verbinden die einzelnen Kontrollelemente miteinander und sind gerichtete Kanten. Dadurch wird angezeigt, welcher Knoten nach welchem folgt bzw. folgen kann.

Synchronisationskanten verbinden ebenfalls zwei Knoten im Prozessmodell und sind ebenfalls gerichtet. Sie geben allerdings nicht den direkten Kontrollfluss an, sondern zeigen an, ob ein bestimmter Knoten nur zur Ausführung kommen darf, nachdem ein anderer bereits ausgeführt wurde. Dies ist bei AND-Verzweigungen notwendig, wenn z.B. Verzweigungspfad X und Y parallel ausgeführt werden und in Verzweigungspfad X eine Aktivität A ein Datenelement M schreibt und in Verzweigungspfad Y eine Aktivität B das selbe Datenelement M liest. Denn dann muss klar definiert sein, welche Aktivität zuerst zur Ausführung kommt, bevor die andere Aktivität auf das selbe Datenelement zugreift. In diesem Fall würde eine Synchronisationskante, welche von Aktivität A zu Aktivität B verläuft, verhindern, dass B ausgeführt wird bevor A ausgeführt wurde.

2.1.3 AND-Verzweigung

Verzweigungen bilden immer einen Kontrollblock, welcher mit einem *Split-Knoten* initialisiert und einem *Join-Knoten* geschlossen wird.

Bei einer *AND-Verzweigung* werden diese durch einen *AND-Split-Knoten* und einen *AND-Join-Knoten* dargestellt (siehe Abbildung 2.2). Split-Knoten A initialisiert den *AND-Kontrollblock* und stellt wie Aktivitäten ein aktives Element dar. Join-Knoten E hingegen beendet lediglich den AND-Kontrollblock, stellt aber keinen aktiven Prozessschritt dar.

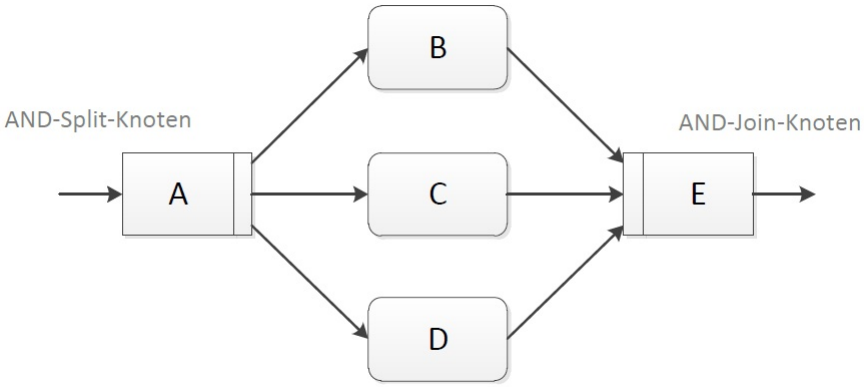

Abbildung 2.2: AND-Kontrollblock [1]

Die Aktivitäten B, C und D liegen auf den drei Verzweigungspfaden des AND-Kontrollblocks, welche alle parallel ausgeführt werden. Das heißt, dass erst alle drei Aktivitäten beendet werden müssen, bevor der AND-Kontrollblock beendet und der Prozess fortgesetzt werden kann.

2.1.4 XOR-Verzweigung

Die *XOR-Verzweigung* ist grundlegend aufgebaut wie die AND-Verzweigung (siehe Abbildung 2.3).

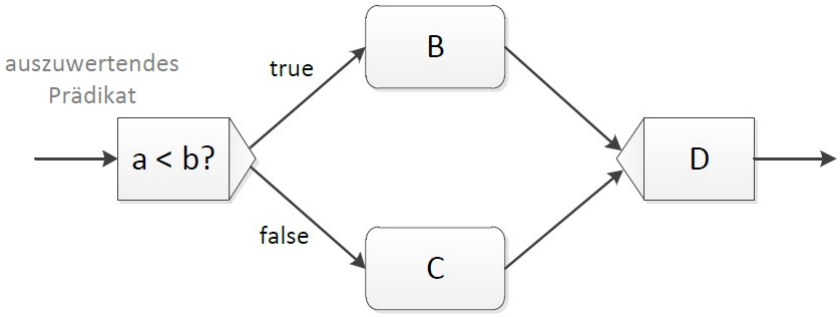

Abbildung 2.3: XOR-Kontrollblock [1]

Der entscheidende Unterschied ist allerdings, dass die Aktivitäten B und C nicht parallel ausgeführt werden, sondern immer nur genau ein Verzweigungspfad kommt zur Ausführung. Somit wird immer nur B oder C ausgeführt und niemals beide. Welcher der beiden Verzweigungspfade zur Ausführung kommt, wird bei der Auswertung des *XOR-Split-Knotens* entschieden. Diese Entscheidung trifft während der Ausführung des Prozesses entweder der Nutzer manuell oder der XOR-Split-Knoten anhand einer vordefinierten Formel und eingelesenen Daten.

2.1.5 Schleife

Schleifen bilden ebenfalls einen Kontrollblock und sind der XOR-Verzweigung ähnlich. Der *Schleifen-End-Knoten* muss die Entscheidung treffen, bei der exakt ein Verzweigungspfad gewählt werden muss. Es stehen jedoch immer nur zwei Verzweigungspfade zur Auswahl und der zurückspringende enthält keine Aktivitäten.

Im Unterschied zum XOR-Split-Knoten wird die Entscheidung allerdings immer anhand einer Formel und ausgelesenen Daten getroffen. Wie in Abbildung 2.4 zu sehen ist, liegt der am Ende der Schleife.

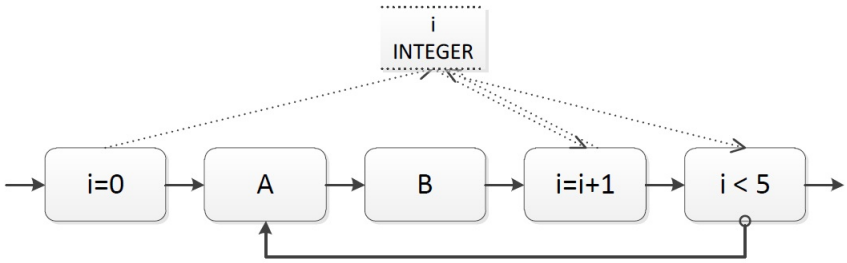

Abbildung 2.4: Schleifen-Kontrollblock [2]

2.2 Datenfluss

Der Datenfluss enthält *Datenelemente* und deren zugehörigen *Datenkanten*, womit er alle im Prozessmodell verfügbaren Daten und deren Zugriffe darstellt.

2.2.1 Datenelement

Datenelemente enthalten die Daten bzw. Variablen, welche in dem Prozessmodell verwendet werden. Sie bestehen immer aus einem *Namen* und einem dazugehörenden *Datentyp*. Ihren Wert bekommen sie erst im *Prozessablauf* zugewiesen, weshalb darauf geachtet werden muss, dass ein Datenelement nicht gelesen wird, bevor es geschrieben wurde [2] [3].

2.2.2 Ein- und ausgehende Datenkante

Da Datenelemente in einem Prozessmodell gelesen und geschrieben werden können, gibt es *ein-* und *ausgehende Datenkanten*. Bei Datenkanten, die von einem Kontrollelement zu einem Datenelement führen, handelt es sich um einen *Schreibzugriff* und umgekehrt um einen *Lesezugriff*. Ein Kontrollelement kann ein Datenelement auch lesen und danach wieder schreiben, wie z.B. in Abbildung 2.4 zu sehen ist.

3 Abbildungsmodell für formularbasierte Prozesse

Dieses Kapitel beschreibt, wie die *formularbasierte Darstellung* aufgebaut ist und Prozessmodelle auf diese Darstellung abgebildet werden [1].

3.1 Darstellung und Abbildung

Die formularbasierte Darstellung wird horizontal ausgerichtet, so dass sich das erste *Formularelement* oben und das letzte unten befindet. Zudem unterliegt die Darstellung einer Blockstrukturierung, ähnlich zu Prozessmodellen, bei der Verzweigungen und Schleifen als Blöcke dargestellt werden. Aktivitäten bilden immer einzelne Formularelemente, genauso wie die einleitenden Knoten von Verzweigungen und Schleifen. Der Start- und End-Knoten eines Prozessmodells wird nie dargestellt, da er keinen eigenen Prozessschritt darstellt.

3.1.1 Sequenz

Mehrere Aktivitäten, die aufeinander folgen, werden Sequenz genannt, wobei die Aktivitäten hintereinander ausgeführt werden. Die Abbildung 3.1 zeigt, wie eine Sequenz im Fomular dargestellt wird.

3.1.2 AND-Verzweigung

Blöcke besitzen im Formular ein Verzweigungs- bzw. Schleifen-Formularelement, welches den Split-Knoten des Blocks abbildet und alle auf den Block anwendbaren Funktionen bereitstellt. Bei einer AND-Verzweigung im Formular befinden sich darunter mehrere Spalten, wobei jede Spalte einen Verzweigungspfad darstellt (siehe Abbildung 3.2). Diese Spalten bestehen wiederum aus Aktivitäts-Formularelementen und/oder weiteren Verzweigungs- bzw. Schleifen-Blöcken.

Der abschließende Knoten eines Verzweigungs- bzw. Schleifen-Blocks wird nicht dargestellt, da aus der Struktur des Formulars das Ende des Blocks automatisch ersichtlich wird. Um dem Nutzer die Blockstruktur optisch zu verdeutlichen, werden die Spalten ein Stück eingerückt und der Blockhintergrund farblich abgesetzt. Dadurch bleibt das Formular trotz mehrerer Verschachtlungen übersichtlich und intuitiv verständlich.

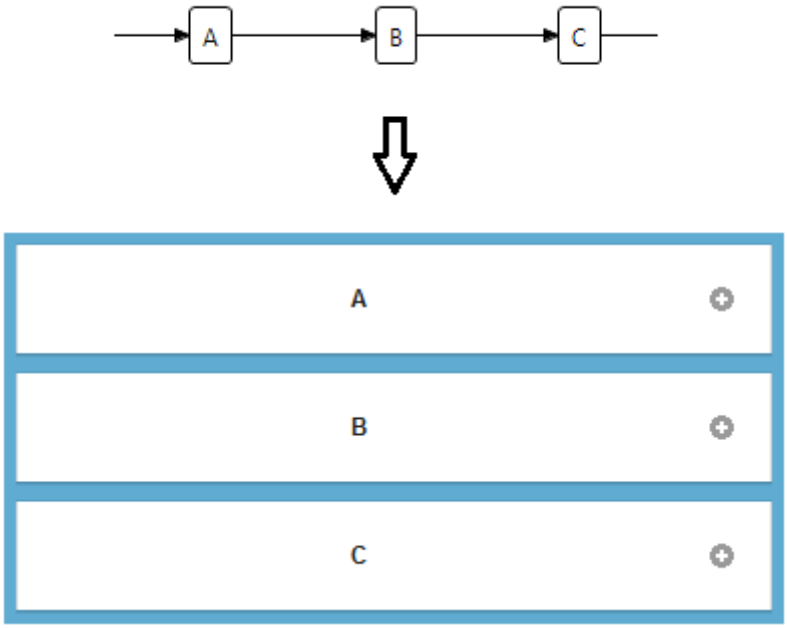

Abbildung 3.1: Sequenz in Formular-Darstellung

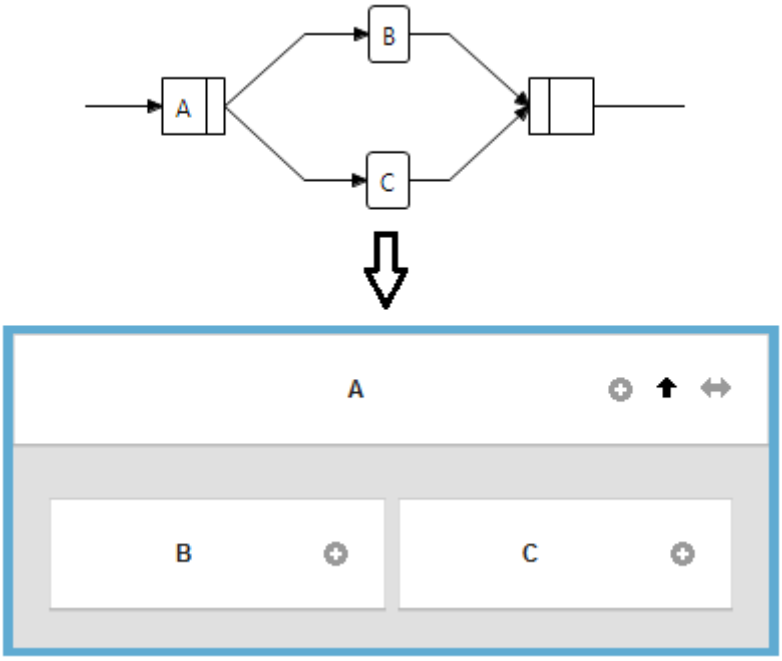

Abbildung 3.2: AND-Verzweigung in Formular-Darstellung

3.1.3 XOR-Verzweigung

Bei einer XOR-Verzweigung im Formular werden die Verzweigungspfade genauso dargestellt, wie bei einer AND-Verzweigung. Jedoch besitzt das Schleifen-Formularelement einer XOR-Verzweigung, im Gegensatz zu einer AND-Verzweigung, für jeden Verzweigungspfad einen But-

ton, mit dem der entsprechende Pfad hervorgehoben werden kann (siehe Abbildung 3.3). Das macht dem Nutzer den Unterschied zwischen einem AND- und XOR-Block visuell klar.

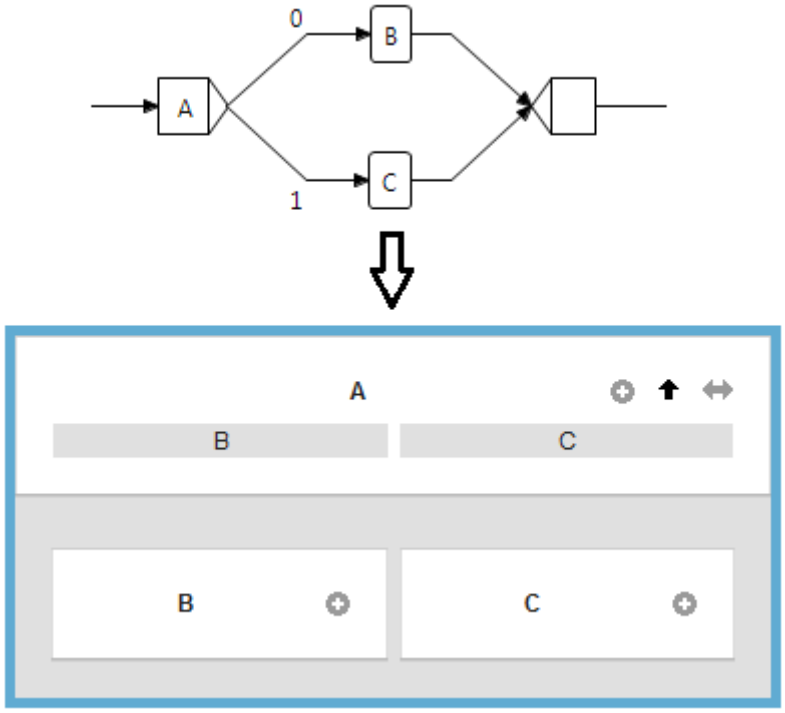

Abbildung 3.3: XOR-Verzweigung in Formular-Darstellung

3.1.4 Schleife

Bei einem Schleifen-Block im Formular existiert immer nur eine der Spalten und diese stellt den Schleifenkörper dar (siehe Abbildung 3.4). Der Schleifenkörper wird bei jeder Schleife mindestens einmal durchlaufen, da die Bedingung immer erst am Ende überprüft wird.

3.1.5 Datenelement

In einer graphenbasierten Darstellung werden die Datenelemente typischerweise oberhalb des Kontrollflusses in einer Reihe dargestellt und die Datenkanten verbinden sie mit den Kontrollelementen, von denen sie geschrieben bzw. gelesen werden. Diese Art der Darstellung ist für die formularbasierte Darstellung nicht geeignet, da man aus oben genannten Gründen keine Kanten in einem Formular haben will. Außerdem wäre diese Darstellung bei vielen Datenelementen und langen Prozessen völlig unübersichtlich. Da es aber keine gute Alternative gibt, um einzeln dargestellte Datenelemente intuitiv den Kontrollelementen zuzuordnen und dabei auch noch zwischen lesendem und/oder schreibendem Zugriff zu unterscheiden, werden die interagierenden Datenelemente bei der formularbasierten Darstellung direkt in den Formularelementen angezeigt (siehe Abbildung 3.5).

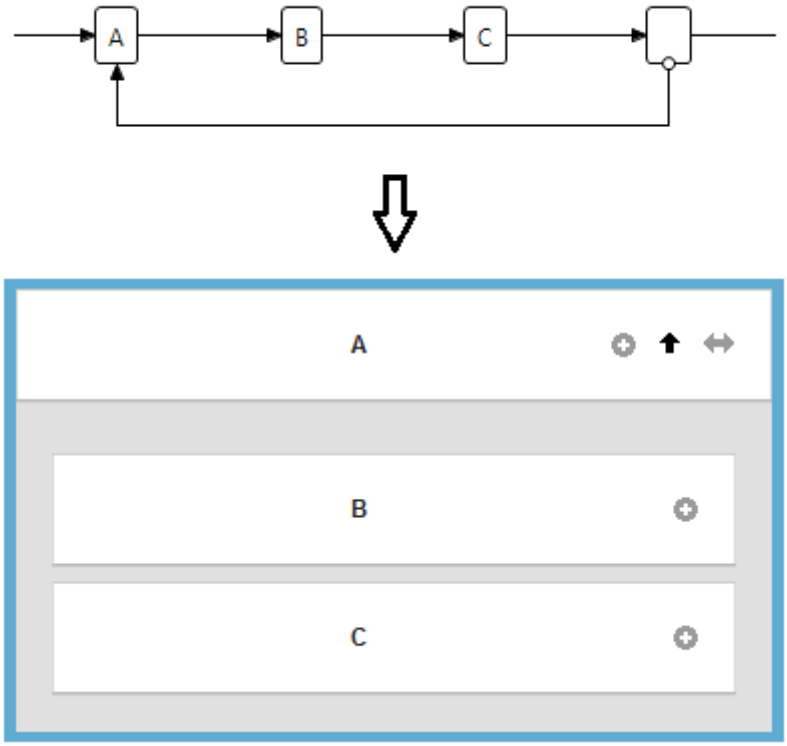

Abbildung 3.4: Schleife in Formular-Darstellung

Auf Grund der Unterscheidung zwischen Lese- und Schreibzugriffen muss man in der formularbasierten Darstellung auch zwischen Daten, die vom Kontrollelement gelesen und geschrieben werden, unterscheiden. Dabei stellt es kein Problem dar, dass mehrere Kontrollelemente auf ein Datenelement zugreifen, weil das selbe Datenelemente bei mehreren Kontrollelementen angezeigt werden kann. Die Zugriffsreihenfolge ist ebenfalls klar, da sie sich aus dem Kontrollfluss und den Synchronisationskanten ergibt.

Abbildung 3.5 zeigt auch, dass Datenelemente als ein- und ausgebende Formularfelder in den Formularelementen dargestellt werden. Dabei sind eingebende Formularfelder diese, welche von der entsprechenden Aktivität gelesen werden, und ausgebende Formularfelder diese, welche von der entsprechenden Aktivität geschrieben werden.

3.1.6 Kanten

Kontroll- und Datenkanten werden nicht explizit dargestellt, da die beschriebene Visualisierung des Formulars deren Existenz überflüssig macht. Synchronisationskanten hingegen werden im Formular in den Formularelementen dargestellt, weil sie aus den ein- und ausgebenden Formularfeldern nicht zwangsläufig ersichtlich sind. Diese Darstellung erleichtert es dem Nutzer zusätzlich das Formular zu verstehen. In diesem Zusammenhang wird auch zwischen ein- und/oder ausgehenden Synchronisationskanten unterschieden (siehe Abbildung 3.6).

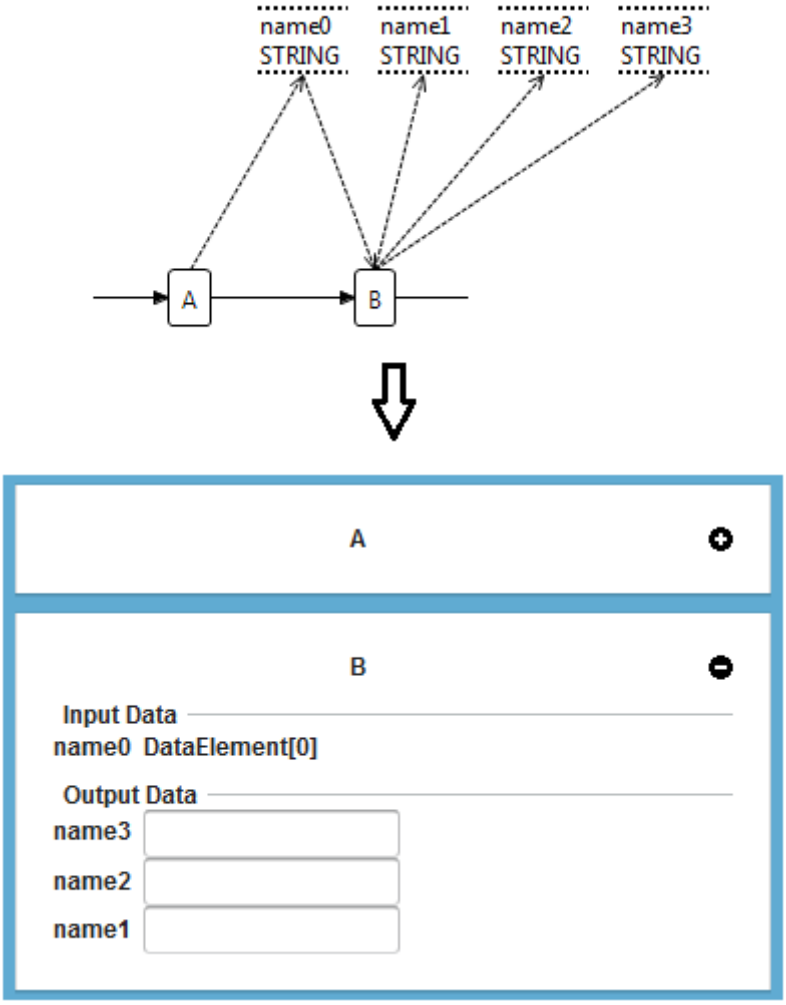

Abbildung 3.5: Datenelemente in Formular-Darstellung

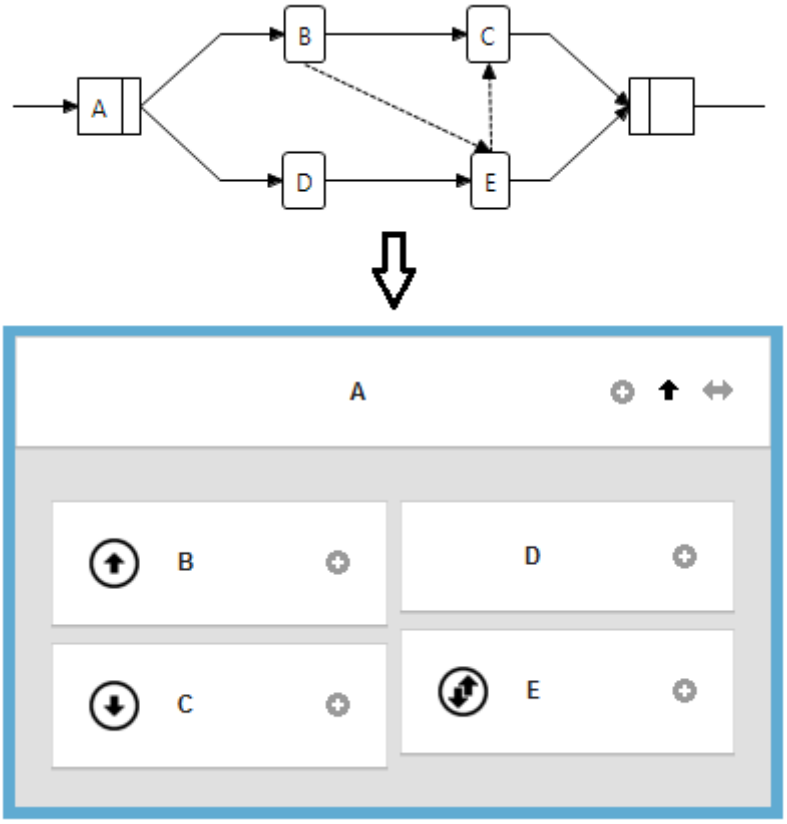

Abbildung 3.6: Synchronisationskanten in Formular-Darstellung

4 Realisierung

Dieses Kapitel erläutert die genaue Arbeitsweise von proViewForm und welche Zugriffe im Hintergrund nötig sind. Ebenso wird der generelle Aufbau der Anwendung proViewForm gezeigt. Desweiteren wird auf die einzelnen Funktionen der formularbasierten Darstellung eingegangen und deren Zusammenhang mit verschiedenen Formularelementen. Dabei werden zunächst die visuellen Änderungen vorgestellt und dann folgen die reellen.

4.1 Abläufe und Zugriffe

Bevor ein Prozessmodell als Formular visualisiert werden kann, muss es allerdings erst mal von einem Server abgerufen werden. Dazu wird hier auf den proView-Prototyp aufgebaut, der nutzergerechte Prozessmodelle durch Abstraktions- und Interaktionstechniken sowie alternative Visualisierungen realisiert [4] [5] [6]. Um während des Betriebs die Wartezeit für den Nutzer möglichst gering zu halten, werden beim Start von proViewForm alle aktuellen Prozessmodelle heruntergeladen. Sobald der Nutzer ein Prozessmodell auswählt, wird eine entsprechende Methode aufgerufen, welche das Prozessmodell analysiert, die formularbasierte Darstellung stückweise aufbaut und dem Nutzer anzeigt. Auf Grund der Blockstrukturierung von Prozessmodellen kann die grundlegende Struktur der Darstellung, mittels rekursiver Aufrufe, einfach erfasst werden. Beim Analysieren des Modells orientiert sich die Visualisierungsmethode am Kontrollfluss und betrachtet den Datenfluss immer nur eingeschränkt für jedes damit interagierende Kontrollelement. Das bedeutet, dass die Korrektheit des Datenflusses nicht überprüft wird, was aber deshalb kein Problem darstellt, da seine Korrektheit vom proView-Prototyp überprüft wird. Zudem ist der Laufzeitaspekt eines Prozessmodells, bei dem die Korrektheit des Datenflusses wichtig ist, nicht Teil dieser Arbeit.

Nachdem das Prozessmodell in der formularbasierten Darstellung dargestellt wurde, kann der Nutzer Veränderungen daran vornehmen, wie z.B. Datenelemente einblenden oder Formularelemente löschen. Bei diesen Veränderungen wird zwischen rein *visuellen* und *reellen Veränderungen* unterschieden. Bei rein *visuellen Veränderungen* wird die Anzeige des Prozessmodells verändert, wohingegen bei den *reellen Veränderungen* das Prozessmodell in seiner Struktur selbst verändert wird. Da man aber nicht will, dass jeder Nutzer das “Original“-Prozessmodell verändern kann, wird bei Prozessmodellen zwischen einem CPM und einer View unterschieden [6]. Ein CPM ist ein Prozessmodell im “Originalzustand“, also so wie es erstellt wurde. Eine View hingegen ist eine Abstraktion einer CPM, die beliebig verändert werden kann. Daher sind visu-

elle Änderungen sowohl auf CPMs als auch auf Views möglich und reelle Änderungen nur auf Views.

Im Gegensatz zu visuellen Änderungen muss bei reellen Änderungen eine Interaktion mit dem proViewServer stattfinden, da das Prozessmodell nicht nur lokal, sondern auch auf dem proViewServer verändert werden muss, damit andere Nuter, welche ebenfalls auf den proViewServer zugreifen, die Änderungen abrufen können und nicht mit veralteten Prozessmodellen arbeiten.

4.2 Aufbau von proViewForm

Die Anwendung proViewForm basiert auf dem Model-View-Controller-Prinziep, weshalb die Anwendung in die drei Pakete *Model*, *View* und *Ctrl* untergliedert ist (siehe Abbildung 4.1).

Alle graphischen Elemente und die allgemeine Darstellung der Anwendung proViewForm werden durch die Klassen im Paket *view* realisiert. Die Funktionen der graphischen Elemente von proViewForm werden alle durch die Klassen im Paket *ctrl* implementiert. Die Klassen im Paket *model* sind für die Komunikation mit dem proView-Prototyp und dem proViewServer zuständig und stellen speziefische Objekte für die Anwendung proViewForm dar. Die seperate Klasse *ProViewFormAppearanceApplication* startet die gesamte Anwendung proViewForm.

4.3 Funktionen für visuelle Änderungen

Die Funktionen für visuelle Änderungen am Prozessmodell sind fast alle über Buttons ausführbar, welche sich entweder direkt auf den Formularelementen oder oberhalb des Formulars befinden. Ansonsten gibt es für visuelle Änderungen noch eine Scrollbar und ein Dropdown-Menü, die sich auf einem separaten Formularelement befinden, welches als Verzweigungs-Wechsler bezeichnet wird. Dieses Element ist allerdings nur unter bestimmten Voraussetzungen sichtbar und somit verwendbar.

4.3.1 Blockspalten ein-/ausblenden

Mit *Blockspalten* sind die Spalten von Verzweigungs- und Schleifen-Blöcken gemeint, in denen die untergeordneten Formularelemente der Blöcke zu finden sind. Deshalb ist die Funktion, diese Elemente ein- und auszublenden, bei Aktivitätsblöcken nicht vorhanden. Der Button dieser Funktion wird durch einen einfachen Pfeil dargestellt (siehe Abbildung 4.2. Bei der Schleife *Loop-Split-Node* und der AND-Verzweigung *AND-Split-Node* zeigt er nach oben, weil bei diesen beiden Blöcken die dazu gehörenden Blockspalten eingeblendet sind. Bei der XOR-Verzweigung *XOR-Split-Node* hingegen sind die Blockspalten nicht eingeblendet, weshalb der Pfeil nach unten zeigt.

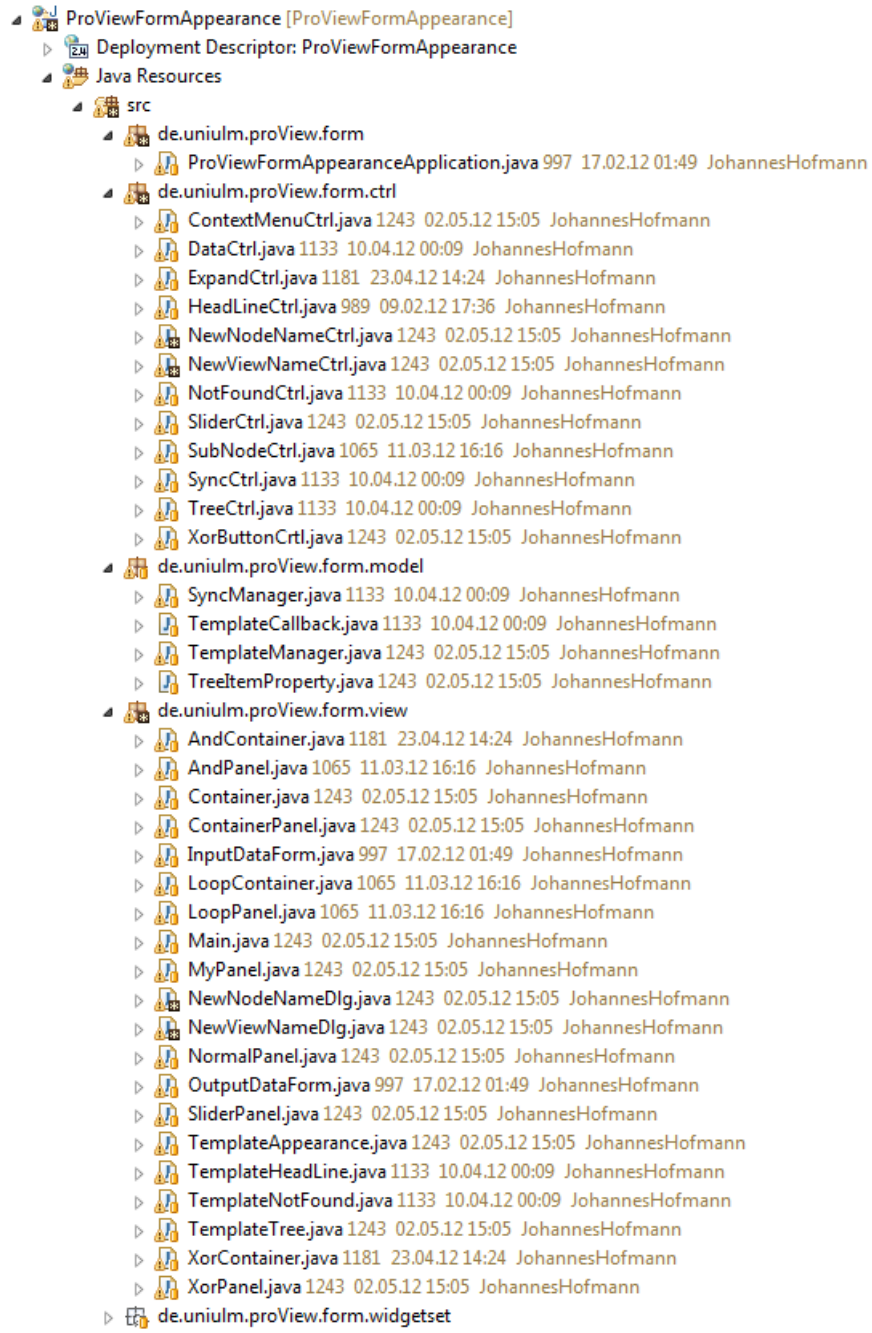

Abbildung 4.1: Paketdiagramm von proViewForm

Bei dieser Abbildung sind die Blockspalten des XOR-Verzweigungsblocks nicht eingeblendet, weil es sich dabei um die “Standard-Anzeige“ eines Prozessmodells handelt und dort immer nur die Blockspalten der Blöcke angezeigt werden, welche sich auf der obersten Ebene der Blockstruktur befinden. Das hat den Grund, dass man den Nutzer nicht überfordern will, da große Prozessmodelle auf den ersten Blick unübersichtlich wirken können. Alle Formularelemente, die sich auf der obersten Ebene der Blockstruktur befinden, werden *Hauptelemente* genannt. In Abbildung 4.2 stellen die Formularelemente *Loop-Split-Node*, *Activity 3* und *AND-Split-Node* die Hauptelemente dar. Falls der Nutzer jetzt alle oder nur die Hauptelemente des Prozessmodells

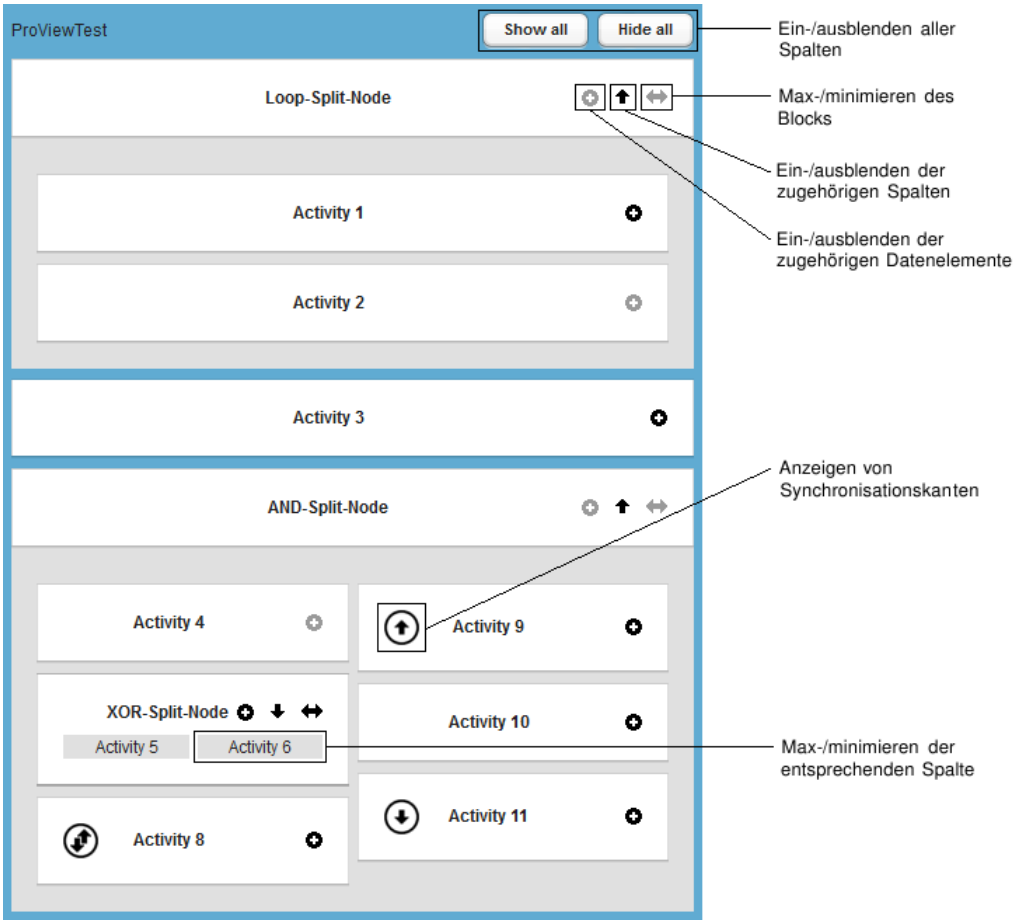

Abbildung 4.2: Formularbasierte Darstellung eines Prozessmodells

sehen möchte, ist er unter Umständen lange damit beschäftigt, da mit den Pfeilen immer nur die direkt untergeordneten Blockspalten ein- bzw. ausgeblendet werden. Daher befinden sich oberhalb des Formulars zwei Buttons *Show all* und *Hide all*, mit denen alle Formularelemente eingeblendet bzw. alle außer den Hauptelementen ausgeblendet werden können.

4.3.2 Maximieren von Blockspalten

In einem Prozessmodell können Verzweigungsblöcke beliebig viele Verzweigungspfade besitzen oder können Verzweigungen beliebig oft ineinander verschachtelt sein. Das führt bei der formularbasierten Darstellung zu dem Problem, dass die Blockspalten der Verzweigungen sehr schmal werden können, was das Formular schnell unleserlich für den Nutzer macht.

Um dem entgegen zu wirken, gibt es bei Formularelementen, die den Kopf einer Verzweigung darstellen, wie z.B. *XOR-Split-Node* (siehe Abbildung 4.2), einen Button, mit dem man diesen Baustein und den dazu gehörenden Block maximieren kann. Dieser Button wird durch zwei entgegengesetzte Pfeile dargestellt, wie bei *XOR-Split-Node* zu sehen ist.

Abbildung 4.3 zeigt, dass beim Maximieren von *XOR-Split-Node* auch alle Formularelemente maximiert werden, welche sich in der selben Blockspalte befinden, womit genau genommen die

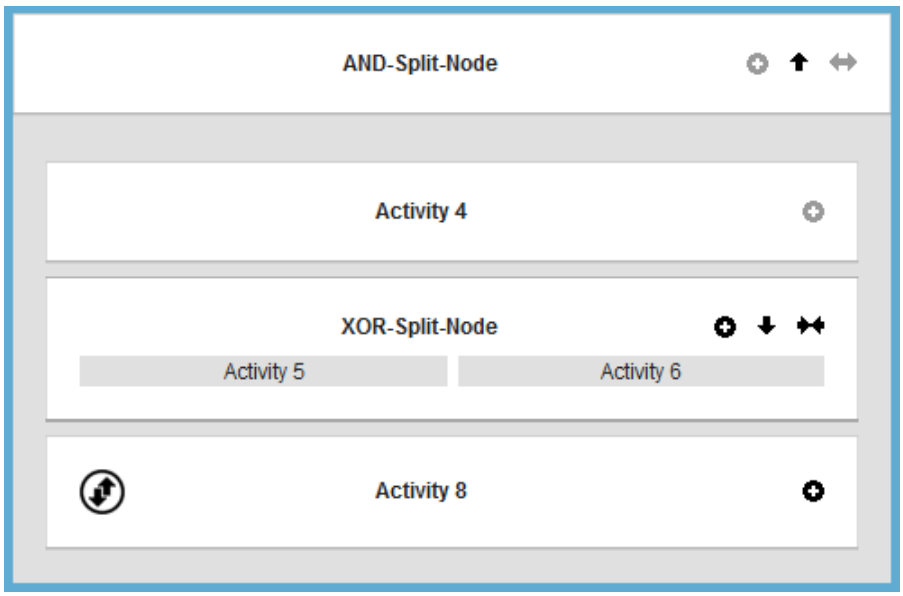

Abbildung 4.3: Maximierte Darstellung eines Verzweigungsblocks

Blockspalte maximiert wird und nicht speziell das aufrufende Formularelement bzw. der zugehörige Block. Falls ein solches Formularelement maximiert wurde, wird der Button durch zwei gegeneinander gerichtete Pfeile dargestellt und der Block kann mit diesem wieder auf seine ursprüngliche Größe gebracht werden. Dabei wird auch die ganze Blockspalte wieder auf die ursprüngliche Größe gebracht. Bei allen Verzweigungs-Formularelementen, die standardmäßig die maximale Größe besitzen, sich also nicht in dem Block einer Verzweigung befinden, ist der Button dieser Funktion deaktiviert. Das ist z.B. beim Formularelement *AND-Split-Node* in Abbildung 4.2 zu sehen.

Bei den Formularelementen von XOR-Verzweigungsblöcken gibt es zusätzlich für jede Blockspalte einen Button, welcher die entsprechende Blockspalte maximiert und beim erneuten Aktivieren auch wieder auf die normale Größe bringt. Falls die zum XOR-Verzweigungsblock gehörenden Blockspalten noch nicht angezeigt werden, geschieht dies ebenfalls. Diese speziell den Spalten zugeordneten Buttons sind in Abbildung 4.4 zu sehen und bei XOR-Verzweigungsblöcken vorhanden, da dort immer nur ein Verzweigungspfad durchlaufen wird und der Nutzer somit den gewünschten Verzweigungspfad hervorheben kann.

Wenn eine Blockspalte maximiert wird, werden alle nebenstehenden Blockspalte ausgeblendet um für die ausgewählte Blockspalte Platz zu schaffen. Bei XOR-Verzweigungsblöcken wird zudem der Verzweigungs-Wechsler am Ende des Blocks eingeblendet, wie in Abbildung 4.5 zu sehen ist. Dieser wird auch angezeigt, falls eine Verzweigung mehr als vier Blockspalten besitzt. Dann werden nur die ersten vier angezeigt, um das Formular übersichtlich zu halten. Mit dem Verzweigungs-Wechsler können die weiteren Spalten eingeblendet werden.

Der Verzweigungs-Wechsler besitzt eine Scrollbar und ein Dropdown-Menü und stellt zwei Funktionen zur Verfügung. Mit der Scrollbar kann zwischen allen Blockspalten der Verzweigung hin

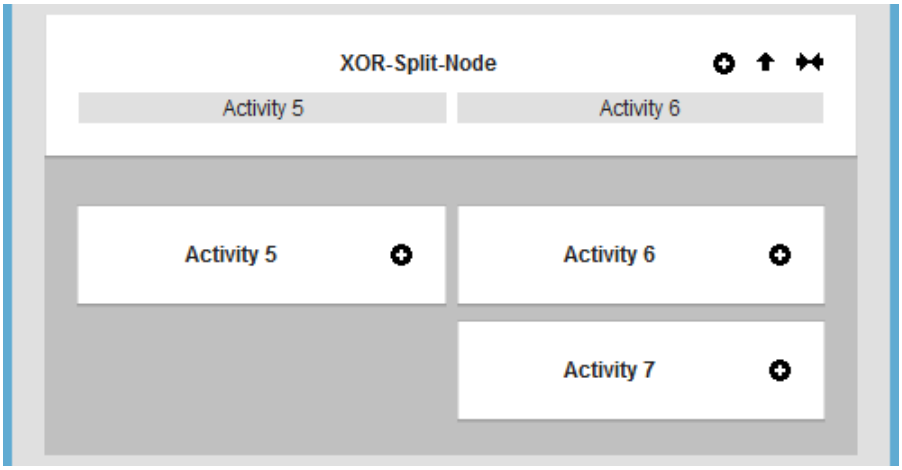

Abbildung 4.4: Standard-Darstellung einer XOR-Verzweigung

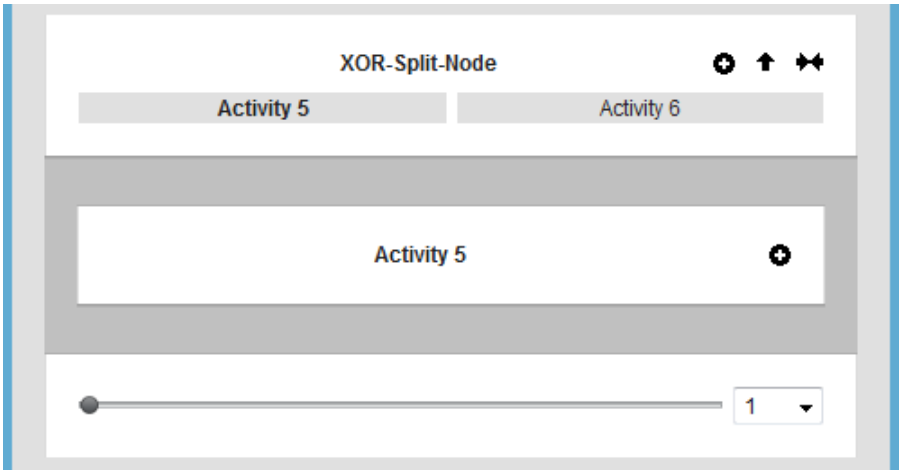

Abbildung 4.5: XOR-Verzweigung, mit der Blockspalte “Activity 5“ maximiert

und her gewechselt werden. Das heißt, wenn wie in Abbildung 4.5 die erste Blockspalte maximiert ist, kann man mit der Scrollbar nach rechts zur zweiten Blockspalte scrollen, was den Effekt hat, dass die zweite statt der ersten Blockspalte maximiert dargestellt wird. Das Dropdown-Menü hingegen ermöglicht es dem Nutzer die Anzahl der angezeigten bzw. maximierten Spalten zu bestimmen. Dabei wird immer von der aktuellen Spalte ausgegangen. Das bedeutet, dass der Nutzer die Anzahl der angezeigten Spalten nur so hoch setzen kann, dass sie die Anzahl der aktuell angezeigten Spalten plus aller rechts davon gelegenen Spalten nicht überschreitet. In Abbildung 4.5 bedeutet das, dass das Dropdown-Menü maximal auf den Wert *2* gesetzt werden kann. Ebenso würde der Wert maximal auf den Wert *1* gesetzt werden können, falls die zweite statt der ersten Spalte maximiert ist. Allgemein können nie mehr als sechs Spalten angezeigt werden, auch wenn die Verzweigung mehr Verzweigungspfade besitzen würde. Das liegt daran, dass es das Formular, selbst bei großen Monitoren, unübersichtlich machen würde.

4.3.3 Datenelemente ein-/ausblenden

Da neben Aktivitäten auch Verzweigungen und Schleifen in einem Prozessmodell Daten lesen und schreiben können, existiert bei allen Formularelementen ein Button, mit dem Datenelemente ein- bzw. ausgeblendet werden können. Wie in Abbildung 4.6 zu sehen ist, wird der Button mit einem Plus dargestellt, falls es möglich ist Daten einzublenden und mit einem Minus, falls sie bereits eingeblendet sind und wieder ausgeblendet werden können. Bei einem Formularelement, wie z.B. *Activity 4* in der Abbildung, bei dem weder Daten gelesen noch geschrieben werden, ist dieser Button jedoch inaktiv.

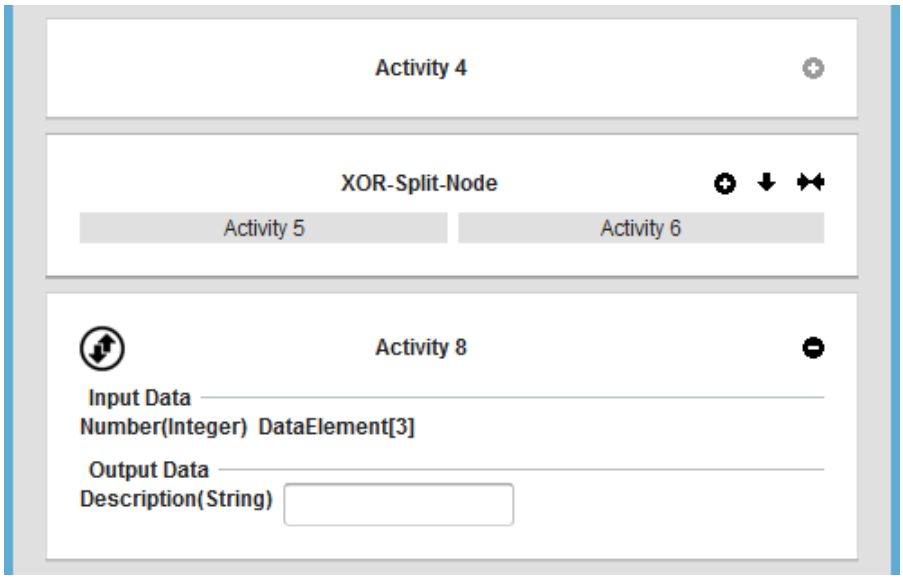

Abbildung 4.6: Darstellung von Datenelementen

Wie in Abbildung 4.6 ebenfalls zu sehen ist, wird bei Datenelementen auch immer zwischen ein- und ausgehenden Daten unterschieden, damit immer klar ersichtlich ist, welche Daten von dem Formularelement gelesen und/oder geschrieben werden. Falls bei einem Formularelement nur ein- oder nur ausgehende Datenelemente existieren, wird auch nur die entsprechende Kategorie eingeblendet. Allgemein sind bei der Anzeige eines Prozessmodells als Formular die Datenelemente standardmäßig immer ausgeblendet.

4.3.4 Synchronisationskanten anzeigen

Da alle Typen von Formularelementen Daten lesen bzw. schreiben können, können alle auch ein- und/oder ausgehende Synchronisationskanten besitzen. Bei Synchronisationskanten werden daher drei Fälle unterschieden, welche alle durch einen anderen *Synchronisationsbutton* im Formular dargestellt werden. Diese Fälle unterscheiden sich in nur ausgehenden, nur eingehenden oder ein- und ausgehenden Synchronisationskanten. In Abbildung 4.7 sind diese drei Fälle zu sehen. Das Formularelemente *Activity 9* besitzt nur ausgehende Synchronisationskanten, weshalb der Synchronisationsbutton als Kreis mit einem nach oben zeigenden Pfeil dargestellt

ist. Das Formularelement *Activity 11* besitzt nur eingehende Synchronisationskanten, weshalb der Synchronisationsbutton mit einem nach unten zeigenden Pfeil dargestellt ist. Bei der Darstellung des Synchronisationsbuttons vom Formularelement *Activity 8* befindet sich sowohl ein nach oben als auch nach unten zeigender Pfeil innerhalb des Kreises, da dieses Formularelement sowohl ein- als auch ausgehende Synchronisationskanten besitzt. Ob es sich bei den ein- bzw. ausgehenden Synchronisationskanten um eine oder mehrere handelt, hat keinen Einfluss auf die Darstellung des Synchronisationsbuttons.

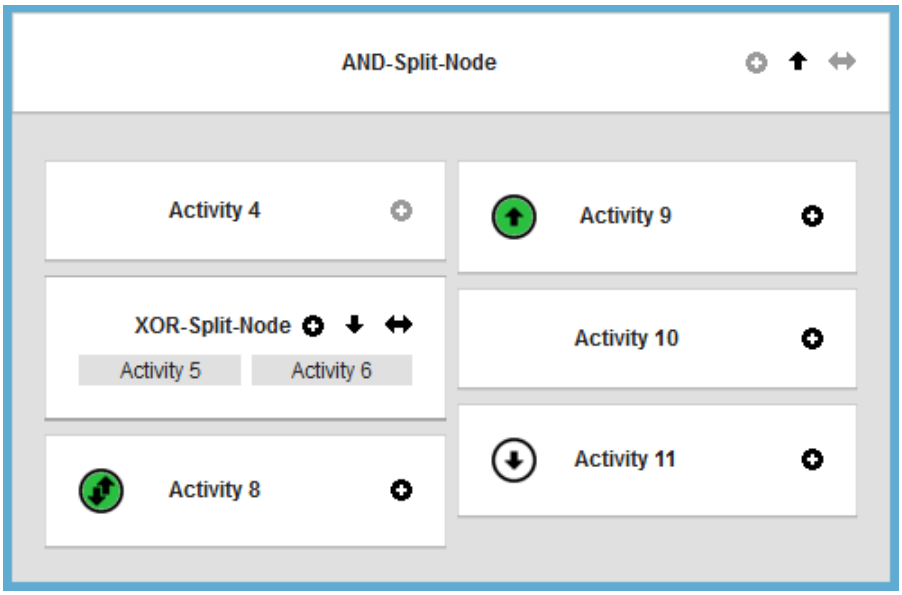

Abbildung 4.7: Anzeige von Synchronisationskanten

Der Synchronisationsbutton ist immer nur sichtbar, falls dieses Formularelement eine oder mehrere Synchronisationskanten besitzt. Falls man diesen Synchronisationsbutton anklickt, wird dieser und alle Synchronisationsbuttons grün markiert, zu deren Formularelement eine Synchronisationskante führt, welche von diesem Formularelement ausgeht. In Abbildung 4.7 ist der Synchronisationsbutton des Formularelementes *Activity 9* angeklickt worden, wodurch die Synchronisationsbuttons, von *Activity 9* und *Activity 8*, markiert wurden. Das bedeutet, dass vom Formularelement *Activity 9* eine Synchronisationskante zum Formularelement *Activity 8* führt. Würde man den Synchronisationsbutton des Formularelements *Activity 8* anklicken, würde dieser und der des Formularelements *Activity 11* markiert, da vom Formularelement *Activity 8* eine Synchronisationskante zum Formularelement *Activity 11* führt. Der Synchronisationsbutton des Formularelements *Activity 9* wäre dann nicht mehr markiert. Würde man den Synchronisationsbutton des Formularelements *Activity 11* anklicken, würde nichts passieren, da dieser keine ausgehenden Synchronisationskanten besitzt. Falls man einen Synchronisationsbutton anklickt, den man zuvor angeklickt hatte, wird die entstandene Markierung wieder aufgehoben.

4.4 Funktionen für reelle Änderungen

Die Funktionen für reelle Änderungen des Prozessmodells werden alle über ein Kontextmenü zur Verfügung gestellt. Dieses Kontextmenü wird per Rechtsklick auf ein Formularelement aufgerufen, das damit zu tun hat, dass die verfügbaren Funktionen vom Typ des Formularelements abhängen. Bei den zur Verfügung stehenden Funktionen für ein Formularelement wird darin unterschieden, ob es sich bei dem Formularelement um ein Aktivitäts-Formularelement handelt, oder ob das Formularelement einen Verzweigungs- oder Schleifenblock initialisiert. Die beiden entsprechenden Kontextmenüs sind in Abbildung 4.8a und 4.8b zu sehen.

Abbildung 4.8: Contextmenü eines Formularelements

4.4.1 Erstellen einer View

Wie bereits erwähnt, können reelle Änderungen eines Prozessmodells nur auf Views angewendet werden und nicht auf CPMs, weshalb es dem Nutzer möglich sein muss, von jedem CPM eine View zu erstellen. Um dies dem Nutzer möglichst intuitiv und einfach zu gestalten, kann der Nutzer die Funktionen für reelle Änderungen am Prozessmodell auch auf CPMs aufrufen. Jedoch erscheint dann ein Popup-Fenster, das ihm die Situation beschreibt und ihn auffordert einen Namen für eine View einzugeben (siehe Abbildung 4.9). Nachdem der Nutzer einen neuen Namen eingegeben und den *Ok* Button gedrückt hat, wird eine neue View des aktuellen CPMs auf dem proViewServer erstellt und die gewünschten Änderungen auf dieser ausgeführt.

Die neue View wird dann dem Nutzer direkt angezeigt, damit er zum einen die Änderungen sieht und zum anderen die neue View nicht extra selektieren muss, um weitere Änderungen daran vorzunehmen.

4.4.2 Formularelemente umbenennen

Mit der Funktion *Rename* kann der Nutzer den Namen eines Formularelements beliebig ändern. Wie in Abbildung 4.10 zu sehen ist, erscheint dazu ebenfalls ein Popup-Fenster, das dem Nutzer die Möglichkeit gibt, den gewünschten Namen einzugeben.

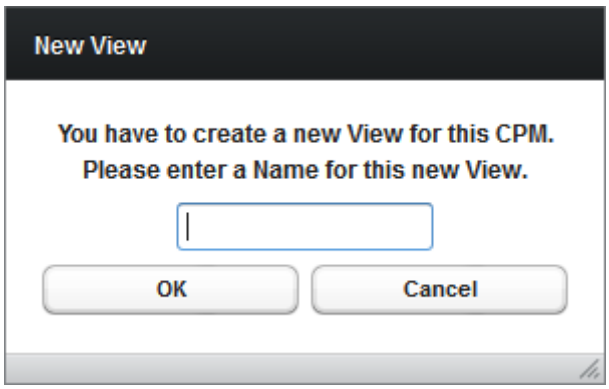

Abbildung 4.9: Popup-Fenster für eine neue View

Abbildung 4.10: Popup-Fenster, um den Namen eines Formularelements zu ändern

Das Umbenennen ist besonders dann notwendig, wenn man z.B. einen Block aggregiert hat [6], denn dann bekommt der daraus entstehende Baustein einen automatisch generierten Namen, welcher für den Nutzer in der Regel nicht brauchbar ist. Ebenso ist die Funktion notwendig, wenn bestimmte Formularelemente einen Namen besitzen, der für den Nutzer nicht intuitiv ist.

4.4.3 Formularelemente löschen

Die Funktion *Delete* ermöglicht es dem Nutzer, das entsprechende Formularelement aus dem Prozessmodell zu löschen. Falls es sich dabei um ein Formularelement handelt, welches einen Block, z.B. eine Schleife, initialisiert, wird nicht nur das Formularelement selbst aus dem Prozessmodell gelöscht, sondern der gesamte Verzweigungs- bzw. Schleifenblock mit allen darin enthaltenen Formularelementen. Beim Löschen von Formularelementen wird die Korrektheit der Datenelemente bzgl. der Lese- und Schreibzugriffe nicht berücksichtigt, da es sich lediglich um eine View handelt.

4.4.4 Aktivitäts-Formularelemente einfügen

Die Funktion *Insert Node after* fügt ein neues Aktivitäts-Formularelement in das Prozessmodell ein. Neue Verzweigungs- und Schleifenblöcke können damit nicht eingefügt werden. Falls es sich

bei dem aktuellen Formularelement um ein Aktivitäts-Formularelement handelt, wird das neue Formularelement direkt hinter diesem eingefügt. Bei einem Verzweigungs- oder Schleifenblock wird das neue Formularelemente direkt hinter dem Block eingefügt. Das neue Formularelement bekommt immer einen automatisch generierten Namen, welcher dann mit der Funktionl *Rename* angepasst werden kann.

4.4.5 Formularelemente markieren

Mit der Funktion *(Un-)Select* können Formularelemente markiert werden, wie man in Abbildung 4.11 sehen kann.

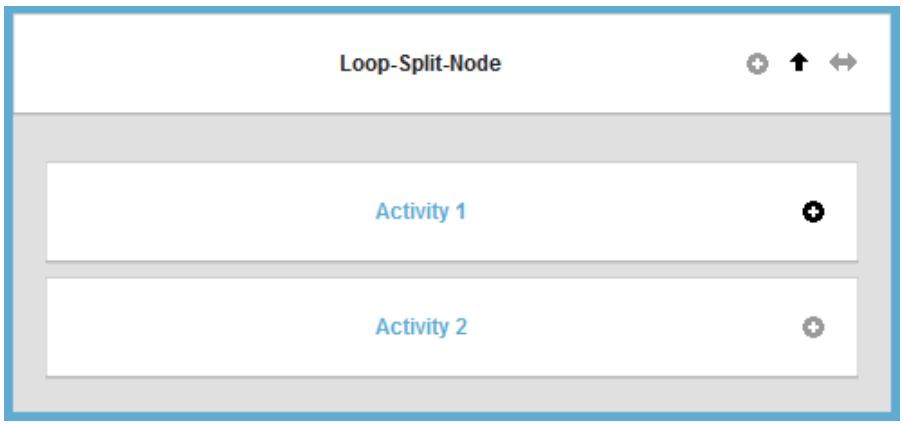

Abbildung 4.11: Markierte Formularelemente

Dabei werden die Formularelemente aber nicht nur farblich hervorgehoben, sondern auch in einer internen Liste gespeichert. Diese Liste wird dann als Grundlage für die Funktion *Aggregate Selected* verwendet, welche immer nur zur Verfügung steht, falls bereits Formularelemente markiert wurden.

4.4.6 Formularelemente, Verzweigungs- und Schleifenblöcke aggregieren

Die beiden Funktionen *Aggregate Container* und *Aggregate Selected* dienen dazu, einen ganzen Block oder mehrere Formularelemente zu aggregieren.

Um mehrere Formularelemente miteinander zu aggregieren, verwendet man die Funktion *Aggregate Selected*, welche alle markierten Formularelemente miteinander aggregiert. Falls das nicht möglich sein sollte, wird ein Fehler ausgegeben.

Wenn man nun z.B. versucht die beiden in Abbildung 4.11 markierten Formularelemente miteinander zu aggregieren, erhält man ein neues Aktivitäts-Formularelement, welches einen automatisch generierten Namen erhält und die beiden markierten Formularelemente ersetzt (siehe Abbildung 4.12).

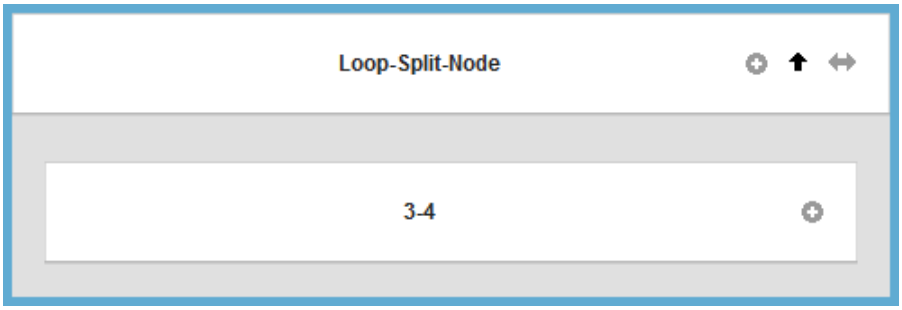

Abbildung 4.12: Neu entstandenes Formularelement, nach Aggregation

Das neue Formularelement ist ein normales Aktivitäts-Formularelement und stellt daher auch alle Funktionen zur Verfügung, die Aktivitäts-Formularelemente besitzen. Wie beim Vergleich der beiden Abbildungen 4.11 und 4.12 festzustellen ist, hatten die beiden markieren Formularelemente Datenelemente assoziiert, wohingegen das neu entstandene Formularelement keine Datenelemente besitzt. Das liegt daran, dass beim Aggregieren die Datenelemente nicht mitübernommen werden und somit auch nicht mehr angezeigt werden können.

Zum Aggregieren eines gesamten Verzweigungs- oder Schleifenblocks wird die Funktion *Aggregate Container* verwendet, weshalb diese auch nur beim Kontextmenü eines Formularelements verfügbar ist, welches einen Verzweigungs- oder Schleifenblock öffnet. Beim Aggregieren eines Blocks wird das initialisierende Formularelement zusammen mit allen anderen Formularelementen innerhalb des Blocks zu einem neuen Aktivitäts-Formularelement zusammengefasst (siehe Abbildung 4.13).

Abbildung 4.13: Neu entstandenes Formularelement, nachdem ein Schleifenblock aggregiert wurde

Hier wurde diese Funktion auf das Formularelement *Loop-Split-Node* angewendet, welches in Abbildung 4.12 zu sehen ist.

5 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Arbeit zusammen und stellt mögliche Erweiterungen und Weiterentwicklungen der formularbasierten Darstellung vor.

5.1 Zusammenfassung

In vielen Unternehmen werden heutzutage Prozessmodelle verwendet, um interne Abläufe darzustellen. Da die Prozessmodelle eine graphenbasierte Darstellung besitzen, die für den unerfahrenen Nutzer nur schwer zu verstehen ist, wurde proViewForm entwickelt. proViewForm dient dazu Prozessmodelle als formularbasierte Darstellung anzuzeigen, was für unerfahrene Nutzer leichter zu verstehen ist. Bei proViewForm ist es dem Nutzer möglich die formularbasierte Darstellung des Prozessmodells seinen Interessen entsprechend zu verändern, indem der Nutzer z.B. für ihn uninteressante Formularelemente ausblendet oder Datenelemente ein- bzw. ausblendet. Ebenso ist es dem Nutzer bei proViewForm möglich eine eigene Ansicht eines Prozessmodells zu erstellen und bei dieser z.B. für ihn unwichtige Formularelemente zu löschen oder zu einem neuen Formularelement zusammenzufassen.

5.2 Zusätzliche Funktionen

Sinnvolle und nützliche Funktionen, welche man der formularbasierten Darstellung noch hinzufügen könnte, werden im Folgenden diskutiert. Dabei ist vorweg aber zu sagen, dass diese Funktionen sich alle auf reelle Änderungen am Prozessmodell beziehen, da alle sinnvollen Funktionen für die reine Anpassung der Darstellung bei dieser Arbeit bereits implementiert sind.

Wenn man an die Verbesserung der bereits vorhandenen Funktionen denkt, sticht einem die Funktion *Insert Node after* ins Auge. Denn bei dieser Funktion wäre es gut, wenn man nicht nur ein neues Aktivitäts-Formularelement, sondern auch einen neuen Verzweigungs- und Schleifen-Block, einfügen könnte.

In diesem Zusammenhang wird automatisch noch eine andere Funktion wünschenswert. Wenn neue Formularelemente und ganze Verzweigungs- bzw. Schleifenblöcke eingefügt werden, wäre es auch wichtig, dass man diese Formularelemente mit Datenelementen verbinden kann. Was wiederum nur dann die volle Effizienz für den Nutzer mit sich bringen würde, wenn man dann

auch neue Datenelemente erstellen kann. An diesem Punkt wäre für den Nutzer auch eine Drag-and-Drop-Funktion der einzelnen Formularelemente wichtig.

Wenn man dem Nutzer all diese Funktionen zur Verfügung stellt, ist es auch sinnvoll ihm einen vollständigen Editor, zum Erstellen von Prozessmodellen, in formularbasierter Darstellung, bereit zu stellen.

Eine Laufzeitkomponente stellt wohl den attraktivsten Aspekt einer Erweiterung dar, da dadurch Firmen die Möglichkeit hätten das Prozessmodell aktiv für ihren Prozessablauf zu verwenden, da die formularbasierte Darstellung zeit- und kostenintensive Schulungen und Einweisungen der Mitarbeiter überflüssig machen könnte. Zudem könnten die Mitarbeiter die Darstellung auf die für sie relevanten Prozessschritte ausrichten, was selbst große Prozessmodelle für sie gut verständlich machen würde.

Jedoch wäre die Erweiterung der formularbasierten Darstellung um eine Laufzeitkomponente mit Sicherheit auch die mit Abstand umfangreichste, da dabei viele Aspekte eines Prozessmodells betrachtet werden müssen, welche für die reine Darstellung keine Rolle spielen. Zudem müssten sehr viele Abfragen und Kontrollelemente implementiert werden, um sicherzustellen, dass der Nutzer die Korrektheit des Kontroll- und Datenflusses beachtet. Ebenso müsste beim Durchlaufen eines Prozessmodells jeder Schritt mit dem proViewServer synchronisiert werden und genauso auch jede Änderung des proViewServers dem proViewForm mitgeteilt werden, was wiederum dem Aspekt der Performanz einen erheblich höheren Stellenwert geben und somit mehr Programmieraufwand erzeugen würde.

Literaturverzeichnis

[1] Barner, J.: *Formular-basierte Modellierung, Ausführung und Änderung von Prozessmodellen*. Bachelorarbeit, Universität Ulm, 2011

[2] Reichert, M.: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. Universität Ulm, Diss., 2000

[3] Reichert, M. ; Dadam, P. ; Rinderle-Ma, S. ; Lanz, A. ; Pryss, R. ; Predeschly, M. ; Kolb, J. ; Ly, L. ; Jurisch, M. ; Kreher, U. ; Goeser, K.: Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite. In: *Proc. BPM'09 Demonstration Track*. 2009, Ulm, Germany

[4] Kolb, J. ; Rudner, B. ; Reichert, M.: Towards Gesture-based Process Modeling on Multi-Touch Devices. In: *1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12)*. 2012, Gdansk, Poland

[5] Kolb, J. ; Reichert, M. ; Weber, B.: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: *S-BPM ONE*. 2012, Vienna

[6] Reichert, M. ; Kolb, J. ; Bobrik, R. ; Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. In: *27th ACM Symposium On Applied Computing (SAC'12), 9th Enterprise Engineering Track*. 2012, Trento, Italy

Abbildungsverzeichnis

Name: Johannes Hofmann Matrikelnummer: 683659

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Johannes Hofmann