Konzept und Implementierung von Swimlanes in Ereignisgesteuerten Prozessketten

Bachelorarbeit

im Studiengang Informatik

von

Tristan Wehrmaker

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Wolfgang Nejdl
Betreuer: Dipl.-Wirt.-Inform. Daniel Lübke

27. Februar 2007
Zusammenfassung

Abstract
Event-driven process chains can get very huge and unclear fastly. The more participants a process has the sensiblier it is to offer them their own tailored subprocess from the huge one. This work deals with viewing event-driven process chains in swimlane-diagrams. The swimlane-view can give significant ease to all involved usergroups.
Danksagung

Ich danke Herrn Daniel Lübke,
der mich bei meiner Bachelorarbeit
hervorragend betreut und unterstützt hat.
Inhaltsverzeichnis

6. Implementierung 44
   6.1. Zugrundeliegende Architekturen 44
   6.1.1. Eclipse und das Graphical Editing Framework (GEF) 44
   6.1.2. ProFLOW 44
   6.2. LaneEPC 44
   6.2.1. Verhältnis zwischen den Plug-Ins 45
   6.2.2. Interne Struktur 45
   6.2.3. Benutzerinteraktion 47
   6.3. Transformator 47
   6.4. Erweiterungsmöglichkeiten 48
   6.4.1. Weitere Diagrammtypen und weitere EPK-basierte Prozestypen 48
   6.4.2. Editorenfenster 49
   6.4.3. Verschiebung und Skalierung von Elementen 49
   6.4.4. EPK-Elemente 49
   6.4.5. EPML 50
   6.5. Zusammenfassung 50

7. Beispiel 51
   7.1. Organisations-Swimlane-Diagramm 52
   7.2. Informations-Swimlane-Diagramm 54

8. Zusammenfassung und Ausblick 56
   8.1. Ausblick 57

A. Fragebogen 62

B. Beiliegende CD 67
Akronyme

<table>
<thead>
<tr>
<th>Abkürzung</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPML</td>
<td>Business Process Modeling Language</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Modeling Notation</td>
</tr>
<tr>
<td>eEPK</td>
<td>erweiterte Ereignisgesteuerte Prozesskette</td>
</tr>
<tr>
<td>EPK</td>
<td>Ereignisgesteuerte Prozesskette</td>
</tr>
<tr>
<td>EPML</td>
<td>Event-Driven-Process-Chain-Markup-Language</td>
</tr>
<tr>
<td>GEF</td>
<td>Graphical Editing Framework</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>ISD</td>
<td>Informations-Swimlane-Diagramm</td>
</tr>
<tr>
<td>OSD</td>
<td>Organisations-Swimlane-Diagramm</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>SOA</td>
<td>Serviceorientierte Architektur</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1. Einleitung

1.1. Motivation

In Organisationen treffen immer wieder Mitglieder, die über fachliches Wissen, und Mitglieder, die über methodisches oder technisches Wissen verfügen, aufeinander. Um die fachlichen Arbeitsabläufe zu definieren und an beide Mitgliedergruppen zu kommunizieren, werden Geschäftsprozessmodelle aufgestellt. Durch die Unschärfe der natürlichen Sprache und der vermehrten Unbrauchbarkeit von mathematischen Formulierungen wird daher eine semiformale Beschreibungsform benötigt [ST05].

Als eine dieser Beschreibungsformen erfreut sich heute die Ereignisgesteuerte Prozesskette (EPK), besonders im deutschsprachigen Raum, einer breiten Akzeptanz. EPKs wurden 1992 in Zusammenarbeit mit der SAP AG an der Universität des Saarlandes in Saarbrücken unter der Leitung von Prof. Dr. Dr. h.c. August-Wilhelm Scheer entwickelt. Mit Hilfe von EPKs können Geschäftsprozesse leicht verständlich graphisch dargestellt werden. Mit steigender Komplexität der EPKs kann die Übersichtlichkeit immens sinken. Daher wäre es wünschenswert, eine andere Darstellungsform zu finden, mit der auch große EPKs leichter lesbar werden.

Swimlane-Diagramme, wie sie schon aus UML oder BPMN bekannt sind, könnten eine solche Darstellungsform bieten. Sie stellen eine Kombination aus Zuständigkeits- und Flussdiagramm dar. Dadurch ist es für den Betrachter einer solchen Grafik leicht möglich, seine Rolle ausfindig zu machen, seine Verantwortlichkeiten und Entscheidungen zu identifizieren und die zeitliche Abfolge der Prozesse abzulesen.

1.2. Ziel dieser Arbeit

In dieser Arbeit soll nun untersucht werden, in welcher Form Swimlanes für die Visualisierung von Ereignisgesteuerten Prozessketten geeignet sein können. Darüber hinaus wird ein Konzept aufgestellt, in dem die Untersuchungsergebnisse beschrieben und umgesetzt werden sollen.

Zur Validierung dieses Konzepts soll im Anschluss ein Werkzeug entwickelt werden, das diese Visualisierung leistet. Vom Fachgebiet Software Engineering der Leibniz Universität Hannover wird ein Plug-In für die Entwicklungsumgebung Eclipse namens ProFLOW zur Verfügung gestellt, auf welchem das Werkzeug dieser Arbeit basieren soll.

1.3. Struktur dieser Arbeit


Nach dieser Einführung folgt in Kapitel 3 eine Gegenüberstellung der Unified Modeling Language (UML) und der Ereignisgesteuerten Prozesskette (EPK). Hier wird UML und das dazugehörige Aktivitätsdiagramm
1. Einleitung

vorgestellt und dann mit der EPK verglichen.

Im Anschluss daran behandelt Kapitel 4 eine Einführung in das Konzept, gefolgt von modelltheoretischen Untersuchungen und einer empirischen Befragung zur Validierung von ersten Konzeptansätzen.

In Kapitel 5 werden dann mögliche Darstellungen von Swimlanes beschrieben, sowie Regeln und Vorgehensweisen zur Erstellung von Swimlane-Diagrammen vorgestellt.

Kapitel 6 behandelt die Implementierung dieses Konzepts in ein Plug-In für die Entwicklungsumgebung Eclipse. Dabei wird auf die zugrundeliegenden Architekturen eingegangen und der Aufbau des zu entwickelnden Plug-Ins LaneEPC erläutert.

Nachdem die Implementierung abgehandelt wurde, wird im siebten Kapitel jeweils ein Beispiel für ein Swimlane-Diagramm, das mittels Organisationseinheiten aufgebaut ist, und ein Swimlane-Diagramm, das mittels Informationsobjekten aufgebaut ist, in LaneEPC dargestellt.

Im Anschluss daran werden die durch diese Arbeit gewonnen Erkenntnisse zusammengefasst und ein Ausblick auf mögliche Erweiterungen des Konzeptes gegeben.
2. Grundlagen


Im darauf folgenden Abschnitt werden dann Swimlanes anhand eines Beispiels aus der Unified Modeling Language vorgestellt.

2.1. Geschäftsprozesse und serviceorientierte Architekturen

Werden in einem Unternehmen Leistungen erstellt, laufen eine Vielzahl von Arbeitsschritten ab. Der Ablauf dieser Arbeitsschritte wird als Geschäftsprozess bezeichnet. Dabei finden sich heute in jedem Unternehmen Geschäftsprozesse, auch wenn diese insbesondere in kleinen Unternehmen nicht formal beschrieben werden. Besonders zur Optimierung der Geschäftsprozesse ist es aber von Nutzen, diese formal zu spezifizieren [Int06].

Eine Definition der Geschäftsprozesse gibt Ulrich Frank in [FvL03]:

**Definition 1** (Geschäftsprozess): Ein Geschäftsprozess ist eine wiederkehrende Abfolge von Aktivitäten, die mehr oder weniger rigiden Regelungsmustern genügt. Er ist zielgerichtet und steht in einem direkten Zusammenhang mit der marktgerichteten Leistungserstellung eines Unternehmens. [...] Die Ausführung von Geschäftsprozessen erfordert den Einsatz knapper Ressourcen.

Da Geschäftsprozesse also festlegen, welche Vorgänge innerhalb eines Unternehmens durch welche Rollen ausgeführt werden sollen [LGS05], wird sich diese Arbeit besonders mit diesen Rollen beschäftigen.


In SOA müssen Business-Funktionen nicht neu programmiert, sondern nur orchestriert, also nur noch existierende Services neu zusammengestellt, werden. So ist es leicht möglich, Programmenteile auszutauschen, bzw. den gesamten Prozess umzustellen. Eine Möglichkeit, die Anforderungen für solche Orchestrierungen aufzustellen, sind die Ereignisgesteuerten Prozessketten, welche im folgenden Abschnitt erläutert werden.
2. Grundlagen

2.2. Ereignisgesteuerte Prozesskette (EPK)

Die Ereignisgesteuerte Prozesskette wurde 1992 am Institut für Wirtschaftsinformatik der Universität des Saarlandes in Saarbrücken, in Zusammenarbeit mit der SAP AG, entwickelt. Ihr Ziel ist die Dokumentation von Geschäftsprozessen [KNS92].


Abbildung 2.1.: Metamodell einer EPK in Anlehnung an [Lüb06]

2.2.1. Elemente

Funktion

Der Begriff der Funktion kann auf verschiedene Arten definiert werden. So wird der Begriff in der Mathematik als eine Vorschrift angesehen, in der jedem Element einer Menge genau ein Element einer zweiten Menge zugeordnet wird. Hier wird der Begriff Funktion vielmehr als eine Aufgabe, d.h. als physische oder geistige Aktivität zur Erfüllung eines definierten Soll-Zustands, verstanden.

Abbildung 2.2.: Funktion
2. Grundlagen


Ereignis

In Anlehnung an die DIN 69900-1 ist ein Ereignis das Eingetretensein eines definierten Zustands, der eine Folge von Aktivitäten bewirkt [KNS92].

Ereignisse werden dabei durch folgende Eigenschaften gekennzeichnet:

- Ereignisse repräsentieren einen eingetretenen Zustand.
- Ereignisse dienen zur Spezifikation betriebswirtschaftlicher Bedingungen.
- Ereignisse können Funktionen auslösen.
- Ereignisse können Prozesswegweiser auslösen.
- Funktionen werden durch Ereignisse ausgelöst.
- Prozesswegweiser werden durch Ereignisse ausgelöst.

Abbildung 2.3.: Ereignis

Abbildung 2.3 zeigt exemplarisch die Darstellung eines Ereignis-Elements als Hexagon.

Prozesswegweiser


Abbildung 2.4.: Prozesswegweiser

Als Symbol für einen Prozesswegweiser wird in der EPK ein Ereignissymbol und ein Funktionssymbol überlappend dargestellt (siehe Abbildung 2.4).

Verknüpfungen

2. Grundlagen

So muss zum Beispiel, wenn Funktionen miteinander verknüpft werden, ein Ereignis folgen. Die drei Verknüpfungsarten werden in Tabelle 2.1 zusammengefasst.


<table>
<thead>
<tr>
<th>Operator</th>
<th>Beschreibung</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>konjunktiv (AND)</td>
<td>Eine Aussage ist wahr, wenn alle Teilaussagen wahr sind.</td>
<td>(\land)</td>
</tr>
<tr>
<td>adjunktiv (OR)</td>
<td>Eine Aussage ist wahr, wenn min. eine Teilaussage wahr ist.</td>
<td>(\lor)</td>
</tr>
<tr>
<td>disjunktiv (XOR)</td>
<td>Eine Aussage ist wahr, wenn genau eine Teilaussage wahr ist.</td>
<td>XOR</td>
</tr>
</tbody>
</table>

Tabelle 2.1.: Verknüpfungen

Erweiterte Ereignisgesteuerte Prozesskette

Im Folgenden werden drei Arten von Objekten beschrieben, welche im Zuge der erweiterten Ereignisgesteuerten Prozesskette (eEPK) zur Definition von EPKs hinzugefügt worden sind. Diese Objekte stehen im Mittelpunkt der in dieser Arbeit aufgestellten Untersuchungen.

Abbildung 2.5.: eEPK-Elemente

Die in Abbildung 2.5 dargestellten Objekte können nur mit Funktionen, aber nicht mit Ereignissen verbunden sein. So ist es für den Geschäftsprozess nur relevant, welche Informationen für die Ausführung einer Funktion benötigt werden, wer diese Funktion ausführt und ob eine Anwendung dafür benötigt wird. Es ist aber nicht von Relevanz, wer z.B. ein Ereignis ausgelöst hat.

Organisationseinheiten stehen für Personen oder Abteilungen, die an einer Funktion beteiligt sind. Informationsobjekte stellen ein Datenobjekt dar. Dies können zum Beispiel Dokumente oder Datenbanken sein. Desweiteren können Anwendungen, also unterstützende Softwaresysteme, an Funktionen beteiligt sein.

2. Grundlagen

2.2.2. Hierarchie

Man unterscheidet zwischen flachen und hierarchischen Ereignisgesteuerten Prozessketten. Dabei bestehen flache EPKs nur aus einer Ebene und stellen damit den trivialen Fall einer EPK dar. In den meisten Fällen verhält es sich allerdings so, dass eine Funktion in einen bzw. mehrere Prozesse untergliedert werden kann.

Abbildung 2.6 zeigt exemplarisch diesen Sachverhalt. Die Funktion „Bestellung bearbeiten“ wird dabei in der nächsten Hierarchieebene weiter unterteilt und so detailliert. So könnte man sich zum Beispiel vorstellen, dass die Funktion „Artikel versenden“ auch noch weiter präzisiert wird.

Abbildung 2.6: Fallbeispiel „Bestellung“

2.3. Swimlanes


Dabei sei angemerkt, dass die in dieser Definition angegebene Ausrichtung der horizontalen und vertikalen Dimensionen in dieser Ausarbeitung vertauscht wird. Es ist in Ereignisgesteuerten Prozessketten allgemein üblich, dass die vertikale Achse die Zeit darstellt.

Abbildung 2.7.: Darstellung eines UML-Aktivitätsdiagramms in Swimlanes in Anlehnung an [BRJ99]

Abbildung 2.7 verdeutlicht den Vorteil einer Einteilung in Swimlanes am Beispiel eines UML-Aktivitätsdiagramms. Auf der linken Seite sieht man die übliche Notation eines Aktivitätsdiagramms, auf der rechten
Seite ist der Graph um Swimlanes für die Organisationseinheiten Kunde, Vertrieb und Lager erweitert. So-
mit können die Angehörigen dieser Organisationseinheiten auf einen Blick ihre Aufgaben und den zeitlichen
Ablauf ablesen. Dabei sind Swimlanes in UML-Aktivitätsdiagrammen die einzige Möglichkeit, Verantwort-
lichkeiten darzustellen.

2.4. XML

Die Extensible Markup Language (XML) wurde 1996 vom W3 Consortium entwickelt. Sie ist dabei abgeleitet
von SGML, auf welcher zum Beispiel auch die Hypertext Markup Language (HTML) basiert. XML wurde
mit den folgenden Zielen definiert [W3C06]:

- XML soll direkt über das Internet verwendbar sein.
- XML soll eine Vielzahl von Anwendungen unterstützen.
- Programme, die XML-Dokumente verarbeiten, sollen einfach zu schreiben sein.
- XML-Dokumente sollen für den Menschen lesbar und halbwegs anschaulich sein.
- Das Design von XML soll formal und präzise sein.
- XML-Dokumente sollen leicht erstellt werden können.

Auf Basis von XML sind mit der Zeit viele Formate für spezielle Anwendungsfälle entstanden. Bei diesem
unterscheidet man zwischen schwacher und starker Standardunterstützung [MN03].

Eine schwache Standardunterstützung ist gekennzeichnet durch ein produktabhängiges und damit proprietä-
res XML Schema. Dadurch kommt es zu einer Vielzahl von heterogenen Werkzeugen, die alle unterschiedli-
che XML-Formate unterstützen, unter welchen mittels Transformationsprogrammen vermittelt werden muss.
Starke Standardunterstützung hingegen setzt auf anerkannte XML-Austauschformate, so können Werkzeuge
durch andere ersetzt werden, ohne dass sich an den Daten etwas ändert.

Für diverse Modellierungskonzepte haben sich derartige Standards heute etabliert, so zum Beispiel XML
Metadata Interchange (XMI) für UML, die Business Process Modeling Language (BPML) für die Business
Process Modeling Notation (BPMN) und auch die Event-Driven-Process-Chain-Markup-Language (EPML)
für Ereignisgesteuerte Prozessketten, welche im nächsten Abschnitt beschrieben wird.

2.5. EPC Markup Language (EPML)

Listing 2.1 gibt einen Überblick über die Elemente von EPML. Dabei soll hier nur kurz auf die wichtigsten
Elemente eingegangen werden. Eine ausführliche Beschreibung findet sich in [MN05].

Das epml-Element stellt das Rootelement dar. Über das graphicsDefault-Element können verschiedene gra-
phische Richtlinien festgelegt werden, welche dann von Visualisierungswerkzeugen verwendet werden können.
Die Einstellungen gelten für alle EPK-Elemente, sofern für diese im graphics-Element keine Informationen
hinterlegt sind.

Das directory-Element kann die Ereignisgesteuerten Prozessketten als epc-Elemente, sowie weitere directory
-Elemente enthalten. Auf diese Weise werden in EPML hierarchische EPKs realisiert. Im epc-Element finden
sich die Ereignisse, Funktionen, Verknüpfungen usw.
2. Grundlagen


```
epml
documentation?
toolInfo?
graphicsDefault?
  fill? (color, image, gradient-color, gradient-rotation)
  line? (shape, width, color, style)
  font? (family, style, weight, size, decoration, color, verticalAlign, horizontalAlign, rotation)
coordinates (xOrigin, yOrigin)
definitions?
  definition* (defId)
    name
    description
attributeTypes?
  attributeType* (typeId)
    description
directory (name)
  directory* (name) ...
epc* (epcId, name)
  attribute* (typeRef, value)
  event* (id, defRef) ...
  function* (id, defRef)
    name
    description
  graphics?
    position? (x, y, width, height)
    fill? (color, image, gradient-color, gradient-rotation)
    line? (shape, width, color, style)
    font? (family, style, weight, size, decoration, color, verticalAlign, horizontalAlign, rotation)
syntaxInfo? (implicitType)
toProcess? (linkToEpcId)
  attribute* (typeRef, value)
processInterface* (id, defRef) ...
and* ...
or* ...
xor* ...
arc* ...
application* ...
participant* ....
dataField* ...
relation* ...
```

Listing 2.1: EPML-Elemente als Syntax-Baum

2.6. Zusammenfassung

In diesem Kapitel wurde in die für diese Arbeit wichtigen Grundlagen eingeführt. Dazu wurde der Begriff Geschäftsprozess und dessen Verbindung zu serviceorientierten Architekturen vorgestellt. Hier wurde gezeigt, welche Vorteile diese Architekturen bieten können.

Im Anschluss daran wurde die Ereignisgesteuerte Prozesskette, sowie deren Elemente und die Bedeutung der Hierarchien beschrieben. Danach wurde die Swimlane-Darstellung anhand eines UML-Aktivitätsdiagramms vorgestellt.

Zum Abschluss dieses Kapitels wurde dann die Extensible Markup Language (XML) in Zusammenhang mit schwacher und starker Standardunterstützung erläutert und mit dem im darauffolgenden Abschnitt erläuterten XML-Dialekt für Ereignisgesteuerte Prozessketten (EPML) in Verbindung gebracht.
3. Gegenüberstellung von EPK und UML

In diesem Kapitel sollen nun die Konzepte der Unified Modelling Language (UML) und der Ereignisgesteuerten Prozesskette gegenübergestellt werden. Dazu soll gezeigt werden, in welchen Fällen welches Konzept genutzt wird.

3.1. Unified Modeling Language (UML)


In der aktuellen Version 2.0 besteht die UML aus dreizehn Diagrammtypen, dabei unterscheidet man zwischen Strukturdiagrammen und Verhaltensdiagrammen [Obj05].

Strukturdiagramme:
- Klassendiagramm (class diagram)
- Kompositionstrukturdiagramm (composite structure diagram)
- Komponentendiagramm (component diagram)
- Verteilungsdiagramm (deployment diagram)
- Objektdiagramm (object diagram)
- Paketdiagramm (package diagram)

Verhaltensdiagramme:
- Aktivitätsdiagramm (activity diagram)
- Interaktionsdiagramme (interaction diagrams):
  - Sequenzdiagramm (sequence diagram)
  - Kommunikationsdiagramm (communication diagram)
  - Interaktionsüberblicksdiagramm (interaction overview diagram)
  - Zeitverlaufsdiagramm (timing diagram)
- Anwendungsfalldiagramm (use case diagram)
- Zustandsdiagramm (state machine diagram)

Aus dieser Vielzahl von Diagrammtypen kommen die Aktivitätsdiagramme den Ereignisgesteuerten Prozessketten am nächsten. Eine Gegenüberstellung dieser beiden Diagramme befindet sich im folgenden Abschnitt.
3.2. EPKs und UML-Aktivitätsdiagramme im Vergleich


Abbildung 3.1.: Gegenüberstellung - EPK (links) und UML 2.0 Aktivitätsdiagramm (rechts)

In Abbildung 3.1 wird ein Geschäftsprozess auf zwei verschiedene Weisen dargestellt. Links sieht man die Darstellung als Ereignisgesteuerte Prozesskette und rechts als UML-Aktivitätsdiagramm. Die Abbildung beschreibt einen Geschäftsprozess, in dem die Bearbeitung einer Bestellung formuliert ist. Dabei zeigen sich schon auf den ersten Blick mehrere Unterschiede in der Notation:
3. Gegenüberstellung von EPK und UML

- Die Organisationseinheiten werden in EPKs als Elemente an die Funktionen geschrieben. In UML hingegen ist nur eine Notation in Swimlanes möglich.

- Die in EPKs verwendeten Ereignisse treten in UML nicht auf. Sie werden hier indirekt durch Zustands- transitionen modelliert.

- XOR-Verknüpfungen werden in UML durch sogenannte Entscheidungskristalle realisiert. Eine Bedingung für die Entscheidung wird ggf. an der folgenden Transition vermerkt.

- AND-Verknüpfungen werden in UML als Parallelisierungs- bzw. Synchronisationsbalken dargestellt.

- Für OR-Verknüpfungen gibt es in der UML keine eigene Notation. Diese können lediglich durch Bedingungen an Synchronisationsbalken dargestellt werden.

- Informationsobjekte werden in UML nicht, wie in EPKs an die betreffende Funktion notiert, sondern stehen als eigenständige Objekte im Kontrollfluss.

Desweiteren sind in UML-Aktivitätsdiagrammen keine Zuordnungen von Daten zu Aktionen möglich, wodurch Geschäftsregeln nur eingeschränkt darstellbar sind [Dan99].


Der ursprüngliche Verwendungszweck der Ereignisgesteuerten Prozesskette und damit auch ihre Stärke liegt in der Modellierung von Geschäftsprozessen. Somit ist keine implementierungsnähe Modellierung vorgesehen, wie sie von UML forciert wird.


3.3. Zusammenfassung

4. Konzeptionelle Untersuchung

In diesem und dem darauffolgenden Kapitel soll nun ein Konzept entworfen werden, das es ermöglicht, Ereignisgesteuerte Prozessketten mit Hilfe von Swimlanes darzustellen. Dazu wird zuerst eine kurze Einführung in das Konzept gegeben, woraufhin dann im Folgenden der Nutzen von Swimlanes in EPKs beschrieben wird. Im Anschluss daran wird auf eine empirische Befragung, welche zu dieser Arbeit durchgeführt wurde, eingegangen.

4.1. Einführung in das Konzept


Abbildung 4.1.: Konzept
4. Konzeptionelle Untersuchung


4.2. Nutzen von Swimlane-Diagrammen

In diesem Abschnitt soll beantwortet werden, für wen die Modellierung von Swimlanes in Ereignisgesteuerten Prozessketten von Nutzen ist und welche Qualitätsmerkmale von Bedeutung sind.

In erster Linie kann man sich vorstellen, dass Swimlanes für die Betrachter von Geschäftsprozessdiagrammen eine erhebliche Erleichterung darstellen können. So müssen diese ihre Konzentration nur auf ihren Bereich richten. Damit ist für sie nur relevant, was außerhalb ihrer Bereiche passiert, wenn sie mit anderen gemeinsam an einer Aufgabe arbeiten.

Auf Seiten des Modellierers ist es natürlich sehr wünschenswert, den Beteiligten auf sie zugeschnittene Prozesse bieten zu können. Dadurch stellt sich eine erhebliche Kosten- und Zeitermäßigung heraus. Es kommt zwar entgegen der Zeitaufwand auf den Modellierer zu, allerdings können sich so die Beteiligten schneller auf den Prozess einstellen. Die Mehrkosten eines Modellierers bzw. einer Modellierergruppe stehen so den Minderkosten aller anderen Beteiligten gegenüber.

Von höchster Bedeutung für den Nutzen von Swimlanes in Ereignisgesteuerten Prozessketten ist, so wenig Veränderungen wie möglich an der Notation von EPKs vorzunehmen, um die Einstiegshürden möglichst gering zu halten. So sollte die Darstellung besonders für Nutzer, die keine Erfahrungen in der Modellierung von Geschäftsprozessen haben, keine große Umstellung sein.

4.3. Empirische Validierung

4. Konzeptionelle Untersuchung

4.3.1. Fragenblock 1: Swimlane-Darstellungen

Der erste Fragenblock behandelt eine Gegenüberstellung der möglichen Swimlane-Darstellungen. Abbildung 4.2 zeigt die für diesen Fragenblock zugrundegelegte Ereignisgesteuerte Prozesskette, in der das Eingehen eines Kundenauftrages bearbeitet wird. Dazu sollten die Befragten ihre Meinung zu den Bereichen Übersichtlichkeit, Brauchbarkeit (beim Betrachten) und mögliche Handhabbarkeit (beim Erstellen) in Schulnoten ausdrücken.

Variante 1: Anlehnung an UML-Swimlanes


Abbildung 4.2.: zugrundeliegende EPK

Anschließend war eine bevorzugte Variante zu wählen. Besonders interessant ist bei diesem Block, dass oftmals eine Variante als Bevorzugte ausgewählt wurde, welche bei der Bewertung nach Schulnoten schlechter ausgefallen ist. Weiter wurde in diesem Block danach gefragt, für wen das Konzept der Swimlanes interessant sein könnte.

Die in diesem Fragenblock behandelten Varianten sollen nun beschrieben werden:

Variante 1: Anlehnung an UML-Swimlanes

4. Konzeptionelle Untersuchung

Wie man in Abbildung 4.3 sieht, können hier, insbesondere im Falle von Verknüpfungen, semantische Unklarheiten auftreten. So lassen sich Verknüpfungen oftmals nicht eindeutig einem einzelnen Bereich zuordnen. Ein weiterer Nachteil dieser Darstellung ist, dass die Swimlane-Bereiche nicht unabhängig voneinander betrachtet werden können.

<table>
<thead>
<tr>
<th>Verhältnis</th>
<th>Fertigungsabteilung</th>
<th>Controlling</th>
<th>Beschaffung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kundenanfrage abnehmen</td>
<td>Kundenanfrage abnehmen</td>
<td>Kundenanfrage abnehmen</td>
<td>Kundenanfrage abnehmen</td>
</tr>
<tr>
<td>Kundenanfrage annehmen</td>
<td>Kundenanfrage annehmen</td>
<td>Kundenanfrage annehmen</td>
<td>Kundenanfrage annehmen</td>
</tr>
<tr>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
</tr>
<tr>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
</tr>
<tr>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
<td>Kundenanfrage ablehnen</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Abbildung 4.3.: Variante 1: Anlehnung an UML-Swimlanes

4. Konzeptionelle Untersuchung

Variante 2: passive Swimlanes

Eine weitere Möglichkeit wird in Abbildung 4.4 dargestellt. Hierbei handelt es sich um passive Swimlanes. In dieser Darstellung enthalten die Swimlanes keine aktiven Elemente (Funktionen etc.), sondern nur die Organisationseinheiten bzw. Informationsobjekte.

Abbildung 4.4.: Variante 2: passive Swimlanes

4. Konzeptionelle Untersuchung

Variante 3: Unabhängige Teilprozesse


Abbildung 4.5 zeigt exemplarisch eine solche Darstellung. Der Vorteil dieser Variante ist, dass in den Swimlanes vollwertige Prozesse stehen, die unabhängig von den Nachbarswimlanes betrachtet werden können.

Abbildung 4.5.: Variante 3: Unabhängige Teilprozesse

4. Konzeptionelle Untersuchung

Variante 4: Unabhängige Teilprozesse mit Wartezeitmarkierungen


Abbildung 4.6.: Variante 4: Unabhängige Teilprozesse mit Wartezeitmarkierungen


4.3.2. Fragenblock 2: Hierarchien

In einem zweiten Fragenblock wurden zwei Möglichkeiten zur Verarbeitung von Hierarchien in Swimlane-Diagrammen vorgestellt. Dabei wurde gefragt, ob Hierarchien entweder abgeflacht oder getrennt behandelt werden sollten. 70% der Befragten waren der Meinung, dass Hierarchien abgeflacht werden sollte, also dass die Hierarchie auf eine Ebene heruntergebrochen wird. Der Vorteil dieser Abflachung ist, dass Hierarchien nicht gesondert betrachtet werden müssen. Bei tiefen Hierarchien könnte dies aber auch zu sehr großen, teils unübersichtlichen Darstellungen führen.
4.3.3. Fragenblock 3: AND-Join-Verknüpfung

Ein dritter Fragenblock behandelte eine AND-Join-Verknüpfung, wie sie in Abbildung 4.7 dargestellt ist.

Abbildung 4.7.: AND-Join-Verknüpfung


Abbildung 4.8.: Möglichkeit 1

Für Möglichkeit 2, wie sie in Abbildung 4.9 dargestellt ist, haben sich 70% der Befragten entschieden. Eine genaue Beschreibung dieser Darstellung findet sich in Abschnitt 5.2.3.

Abbildung 4.9.: Möglichkeit 2
4.3.4. Fragenblock 4: Datenflüsse bei Informationsobjekten

Der letzte Fragenblock behandelte die Darstellung von Swimlane-Diagrammen, welche anhand von Informati-
onsojekt aufgebaut werden. Dabei ging es um die Frage, wie die Datenflüsse von einem Informationsobjekt zu einer Funktion bzw. andersherum dargestellt werden könnten.


Eine Möglichkeit zur Notation des Datenflusses ist, ihn als Text in die Beschriftung der Funktion einzufügen. In einer weiteren Möglichkeit wurde ein Textlabel neben der Funktion platziert (siehe Swimlane DB 2).

Für die dritte gegebene Möglichkeit entschieden sich 80% der Befragten. Dabei handelt es sich um eine Variante, in der Pfeile mit der Flussrichtung als Aufschrift zur Funktion, bzw. von der Funktion weg zeigen (siehe Swimlane DB 1). Diese Variante wird schlussendlich auch in der in Abschnitt 5.3 vorgestellten Transformation von EPK-Diagrammen anhand von Informationsobjekten verwendet.

4.4. Zusammenfassung

Das vorangegangenen Kapitel führte in das zu erarbeitende Konzept. Dazu stellte es den Nutzen für Be-
trachter und Modellierer von Ereignisgesteuerten Prozessketten heraus. Im Anschluss wurde eine empirische Befragung vorgestellt. Dabei wurden die Fragen detailliert beschrieben und die Ergebnisse erläutert.
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten


5.1. Verwendete Variante

Die Ergebnisse der empirischen Untersuchung haben gezeigt, dass Variante 3 und 4 (Abschnitt 4.3.1) eine große Akzeptanz gefunden haben. Aus diesem Grund und durch die semantischen Unklarheiten von Variante 1 wird daher Variante 3 als Basis für die hier beschriebene Swimlane-Darstellung genommen.

Abbildung 5.1: Verwendete Variante

Um die zeitliche Komponente und die Einhaltung der Syntax zu gewährleisten, werden die Ereignisse, welche zur gleichen Zeit auftreten - also z.B. gleiche Ereignisse, die in mehreren Swimlanes vorkommen -, mit einer Kante verbunden. Abbildung 5.1 soll dieses Vorgehen verdeutlichen.
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

5.2. Regeln

An dieser Stelle sollen nun einige Regeln aufgestellt werden, welche dann später in den Algorithmen zum Aufbau von Swimlanes angewandt werden.

Im Folgenden werden solche Swimlane-Diagramme, die nach Organisationseinheiten aufgebaut sind, als Organisations-Swimlane-Diagramme (kurz: OSD) bezeichnet. Hingegen werden Swimlane-Diagramme, die nach Informationsobjekten aufgebaut sind, Informations-Swimlane-Diagramme (kurz: ISD) genannt.

5.2.1. Grundlegende Regeln

- Anwendungen bzw. unterstützende Systeme werden als Organisationseinheiten betrachtet.
- Jeder Organisationseinheit (in Organisations-Swimlane-Diagrammen) bzw. jedem Informationsobjekt (in Informations-Swimlane-Diagrammen) wird eine Swimlane zugeordnet.
- Gibt es in OSDs Funktionen, bei denen keine Organisationseinheit angegeben ist, existiert eine Swimlane mit dem Namen „undefiniert“, zu welcher diese Funktionen zugeordnet werden.
- Gibt es in ISDs Funktionen, bei denen kein Informationsobjekt angegeben ist, existiert eine Swimlane mit dem Namen „unabhängig“, zu welcher diese Funktionen zugeordnet werden.
- In OSDs bleiben Symbole für Informationsobjekte erhalten.
- In ISDs bleiben Symbole für Organisationseinheiten erhalten.
- In ISDs werden die Richtungen der Datenflüsse mittels Pfeilen für Input und Output gekennzeichnet.

5.2.2. Einfache Transformationsschritte

Nun sollen einfache Transformationsschritte beschrieben werden.

**Funktion mit einer Organisationseinheit bzw. einem Informationsobjekt**

Ist an einer Funktion nur eine Organisationseinheit bzw. ein Informationsobjekt beteiligt, so wird diese Funktion mit ihrem Vorgänger- und Nachfolgerereignis in die jeweilige Swimlane verschoben. Diesen Sachverhalt zeigt Abbildung 5.2.

Abbildung 5.2.: Funktion mit einer Organisationseinheit
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

Funktion mit mehreren Organisationseinheiten

Abbildung 5.3 zeigt das Vorgehen bei Funktion mit mehreren Organisationseinheiten. Da mehrere Organisationseinheiten an einer Funktion bedeutet, dass sie parallel an einer Aufgabe arbeiten, wird die dargestellte Prozesskette in eine AND-Verknüpfung überführt. Nach dieser Transformation kann dann mit den in Abschnitt 5.2.3 vorgestellten Verknüpfungen fortgefahren werden.

![Abbildung 5.3: Funktion mit mehreren Organisationseinheiten](image)

Funktion mit mehreren Informationsobjekten

Da Informationsobjekte selbst keine aktiven Elemente sind, also nicht an einer Funktion teilnehmen, muss bei ihnen nicht darauf geachtet werden, ob ein anderes Informationsobjekt vorhanden ist. Daher wird hier die Verknüpfung, wie in Abbildung 5.4 dargestellt, unverändert in die Swimlanes übernommen.

![Abbildung 5.4: Funktion mit mehreren Informationsobjekten](image)

5.2.3. Verknüpfungen


Ereignis → Funktion AND-Split

Bei dieser Verknüpfung wird im Falle von verschiedenen Organisationseinheiten bzw. Informationsobjekten die AND-Verknüpfung aufgelöst und auf die jeweiligen Swimlanes verteilt. Dies kann geschehen, da beide
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

Funktionen ausgeführt werden müssen. Daher ist es an dieser Stelle für den Betrachter irrelevant, ob und wie der andere Teilnehmer reagiert. Ist nur eine Organisationseinheit bzw. ein Informationsobjekt beteiligt, so kann diese Verknüpfung exakt in die jeweilige Swimlane übernommen werden.

![Diagramm 5.5: Ereignis -> Funktion AND-Split](image1)

**Funktion -> Ereignis AND-Join**

Wenn an dieser AND-Verknüpfung in einem Organisations-Swimlane-Diagramm mehrere Organisationseinheiten beteiligt sind, so muss in jeder beteiligten Swimlane überprüft werden, ob der bzw. die anderen Beteiligten ihre Aufgaben ausgeführt haben. Aus diesem Grund wird die Verknüpfung wie folgt transformiert. Diese Transformation ist in Abbildung 5.6 dargestellt.


Zur Einhaltung der Syntax muss eine neue Funktion eingeführt werden. Diese Funktion ist dann eine Dummy-Funktion, das heißt in ihr wird nichts ausgeführt, sie stellt lediglich einen Wartezustand dar. Daher erhält sie die Aufschrift „wait for completion“. Auf diese Funktion folgt dann das Ereignis, das auf die ursprüngliche AND-Verknüpfung folgte.

![Diagramm 5.6: Funktion -> Ereignis AND-Join (OSD)](image2)

Wenn an dieser AND-Verknüpfung in einem Informations-Swimlane-Diagramm mehrere Informationsobjekte beteiligt sind (siehe Abbildung 5.7), so wird diese Verknüpfung getrennt und in die jeweiligen Swimlanes
verteilt. Da Informationsobjekte keine aktiven Elemente sind, sie also nicht selber den Erfolg der Funktion bestimmen können, ist hier keine weitere Behandlung notwendig.

Abbildung 5.7.: Funktion $\rightarrow$ Ereignis AND-Join (ISD)

Ist an der ursprünglichen AND-Verknüpfung nur eine Organisationseinheit bzw. ein Informationsobjekt beteiligt, so wird auch diese exakt in die jeweilige Swimlane übernommen.

**Funktion $\rightarrow$ Ereignis OR-Join**

Bei dieser Art von Verknüpfung wird die OR-Verknüpfung wie im Funktion $\rightarrow$ Ereignis AND-Join behandelt (siehe Abbildung 5.8). Da das folgende Ereignis eintritt wenn entweder eine oder alle Funktionen ausgeführt worden sind, kann es zu semantischen Problemen kommen. So wäre es zum Beispiel möglich, dass die folgende Funktion mehrfach ausgeführt wird, wenn die auslösenden Ereignisse zeitversetzt eintreten.

Abbildung 5.8.: Funktion $\rightarrow$ Ereignis OR-Join

Abbildung 5.9.: Funktion $\rightarrow$ Ereignis OR-Join (ISD)
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

Wie auch beim Funktion->Ereignis AND-Join, können solche Verknüpfungen in Informations-Swimlane-Diagrammen aufgespalten und auf die Swimlanes verteilt werden. Ist an dem OR-Join nur eine Organisationseinheit bzw. ein Informationsobjekt beteiligt, so kann er ohne weitere Behandlung in die Swimlane übernommen werden.

**Funktion -> Ereignis XOR-Join**


Dabei muss bei dieser XOR-Verknüpfung geprüft werden, ob die anderen Teilnehmer ihre Funktion nicht ausgeführt haben. Auch hier gilt, dass eine Dummy-Funktion eingeführt werden muss. Auf diese folgt dann das ursprüngliche Ereignis.

Abbildung 5.10.: Funktion -> Ereignis XOR-Join (OSD)

Tritt solch eine XOR-Verknüpfung in Informations-Swimlane-Diagrammen auf, so wird sie aufgeteilt und in die jeweiligen Swimlanes verschoben (siehe Abbildung 5.11).

Abbildung 5.11.: Funktion -> Ereignis XOR-Join (ISD)

Ist lediglich eine Organisationseinheit in OSDs bzw. ein Informationsobjekt in ISDs am dem Join beteiligt, wird dieser ohne Änderung in die jeweilige Swimlane übertragen.
weitere Verknüpfungen

Da an den weiteren Verknüpfungen, wie sie in Tabelle 5.1 dargestellt sind, immer nur eine Funktion beteiligt ist, können diese direkt in die jeweiligen Swimlanes übertragen werden.

<table>
<thead>
<tr>
<th>Ereignis $\rightarrow$ Funktion</th>
<th>Funktion $\rightarrow$ Ereignis</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagramm AND-Join]</td>
<td>![Diagramm AND-Split]</td>
</tr>
<tr>
<td>![Diagramm OR-Join]</td>
<td>![Diagramm OR-Split]</td>
</tr>
<tr>
<td>![Diagramm XOR-Join]</td>
<td>![Diagramm XOR-Split]</td>
</tr>
</tbody>
</table>

Tabelle 5.1: weitere Verknüpfungen

5.3. Vorgehen

Nun werden Vorgehensweisen vorgestellt, mit Hilfe derer vorhandene Ereignisgesteuerte Prozessketten als Swimlane-Diagramme dargestellt werden können. Im Anschluss wird gezeigt, wie neue Ereignisgesteuerte Prozessketten in Swimlanes aufgebaut werden.

Transformation von vorhandenen EPKs in Organisations-Swimlane-Diagramme

Im Folgenden wird davon ausgegangen, dass eine fertig modellierte Ereignisgesteuerte Prozesskette vorhanden ist, welche in ein Organisations-Swimlane-Diagramm transformiert werden soll. Das Vorgehen beim Erstellen von neuen EPKs wird in einem späteren Abschnitt beschrieben.

Zu Beginn wird für jede Organisationseinheit eine Swimlane angelegt. Existieren Funktionen, denen kei-

Wurden die Swimlanes angelegt, so kann begonnen werden die Elemente in die jeweiligen Swimlanes zu verschieben. Dazu werden alle Funktionen in die Swimlane zu ihrer Organisationseinheit verschoben bzw. - im Falle von mehreren teilnehmenden Organisationseinheiten - kopiert. Ist an einer Funktion keine Organisationseinheit angegeben, so wird diese in der Swimlane „undefiniert“ abgelegt. Zu diesen Funktionen werden dann die anderen zugehörigen Elemente anhand der Regeln aus Abschnitt 5.2 hinzugefügt.

Die Symbole für Organisationseinheiten entfallen in der Swimlane-Darstellung komplett. Symbole für Informationsobjekte bleiben allerdings weiterhin erhalten. Da die vertikale Achse der Swimlane die Zeit repräsentiert, werden alle Elemente, die gleich sind, und jene, die zur gleichen Zeit stattfinden, auf der gleichen gedachten horizontalen Linie gezeichnet. Ereignisse, auf die dies zutrifft, werden zusätzlich mit Linien verbunden. Dieses sind die einzigen Linien in EPK-Swimlane-Diagrammen, die die Grenzen einer Swimlane überschneiden können.

Transformation von vorhandenen EPKs in Informations-Swimlane-Diagramme


Erstellen von EPKs in Swimlane-Diagrammen

Will man Ereignisgesteuerte Prozessketten in einer Swimlane-Darstellung aufstellen, so hat man zwei Möglichkeiten. Entweder man baut sie in der klassischen Darstellung auf und geht dann mit den oben beschriebenen Methoden vor oder man stellt die EPKs direkt in Swimlanes dar.

Dazu beginnt man zeitlich mit der ersten Funktion und legt für die an dieser Funktion beteiligten Organisationseinheiten bzw. Informationsobjekte Swimlanes an. In diese Swimlanes wird die Funktion und das vorhergehende Ereignis und evtl. der vorhergehende Prozesswegweiser in die neu angelegten Swimlanes gezeichnet.

Nun kann nach den oben vorgestellten Regeln fortgeführt werden. Dabei ist wichtig, dass für jede neu hinzukommende Organisationseinheit bzw. jedes Informationsobjekt eine neue Swimlane erstellt wird.
5. Beispiel-Transformation


Abbildung 5.12.: Beispiel einer manuellen Transformation: EPK in klassischer Darstellung
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

Schritt 1

Abbildung 5.13.: Beispiel einer manuellen Transformation (Schritt 1)

Funktionen mit mehreren Organisationseinheiten werden aufteilen.
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

Schritt 2

Für jede Organisationseinheit wird eine Swimlane angelegt. Daraufhin werden die Funktionen mit Vorgänger- und Nachfolgerereignissen und eventuellen Verknüpfungen auf die jeweilige Swimlanes verteilt.

Abbildung 5.14.: Beispiel einer manuellen Transformation (Schritt 2)
Abbildung 5.15.: Beispiel einer manuellen Transformation (Schritt 3)

Die Verknüpfungen werden nach den in Abschnitt 5.2.3 dargestellten Regeln behandelt.
5. Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten

Schritt 4

Abbildung 5.16.: Beispiel einer manuellen Transformation (Schritt 4)

Im letzten Schritt werden gleiche Ereignisse mit einer Linie verbunden.
5. **Darstellung von Swimlanes in Ereignisgesteuerten Prozessketten**

5.5. **Zusammenfassung**

6. Implementierung

In diesem Kapitel soll nun die Implementierung einer Visualisierung von Swimlanes in Ereignisgesteuerten Prozessketten beschrieben werden. Zu diesem Zweck wurde ein Plug-In mit dem Namen LaneEPC für die Entwicklungsumgebung Eclipse entwickelt. Dieses Plug-In basiert wiederum auf dem vom Fachgebiet Software Engineering der Leibniz Universität Hannover zur Verfügung gestellten Plug-In ProFLOW. Im folgenden Abschnitt werden LaneEPC und die zugrundeliegenden Architekturen beschrieben.

6.1. Zugrundeliegende Architekturen

6.1.1. Eclipse und das Graphical Editing Framework (GEF)


6.1.2. ProFLOW


6.2. LaneEPC

Zunächst soll aber das Verhältnis der Plug-Ins zueinander und die Struktur eines Swimlane-Diagramms in LaneEPC beschrieben werden. Im Anschluss daran wird kurz auf Benutzerinteraktionen eingegangen.

6.2.1. Verhältnis zwischen den Plug-Ins

Abbildung 6.1.: Das Verhältnis der Plug-Ins untereinander

Abbildung 6.1 verdeutlicht das Verhältnis zwischen LaneEPC und den beteiligten ProFLOW-Plug-Ins. Ein ProFLOW-Prozesstyp-Plug-In, zum Beispiel gegeben durch das Plug-In ProFLOW.eepk, enthält Model, View und Controller aller dazugehörigen Elemente. Dieser Prozesstyp muss in der Form auf der Ereignisgesteuerten Prozesskette basieren, dass EPK-Elemente, wie Ereignisse und Funktionen, definiert werden. Dabei verwendet er die jeweiligen Mechanismen aus ProFLOW.core und ProFLOW.ui.

LaneEPC benutzt wiederum die Elemente und Funktionalitäten, die vom Prozesstyp zur Verfügung gestellt werden. Darüber hinaus werden neue Elemente auf Basis von ProFLOW.core und ProFLOW.ui implementiert, wie zum Beispiel Model und View einer Swimlane oder einer Synchronisationslinie.

6.2.2. Interne Struktur

Abbildung 6.2.: Plug-In-Struktur
6. Implementierung

Wie Abbildung 6.2 zeigt, wird beim Start von LaneEPC ein LaneEPCEditor geöffnet. Dieser erhält einen Prozess von dem mit ProFLOW gelieferten XMLImporter. Der XMLImporter liest eine Datei im ProFLOW-Format ein. In dieser Datei ist der Prozesstyp als ID gespeichert. Mittels dieser ID sucht der aus ProFLOW stammende ExtensionPointManager das zugehörige Prozessobjekt, in welches die in der Datei enthaltenen Daten gespeichert werden. Bis zu diesem Punkt ist der Ablauf in LaneEPC und ProFLOW gleich.

Wurde der Prozess fertig eingelesen, sucht der LaneEPCEditor durch den ExtensionPointManager die für diesen Prozess möglichen Diagrammtypen und bietet diese dem Benutzer als Auswahl an. Mit der Benutzerwahl bekommt der Editor wiederum aus dem ExtensionPointManager das zu diesem Diagrammtyp gehörende Transformatorobjekt.

An dieses Transformatorobjekt wird der Prozess übergeben und umgewandelt. Das Transformatorobjekt muss dabei von der Klasse CommonTransformator abgeleitet sein.

Im Plug-In LaneEPC sind bereits zwei Transformatoren für Ereignisgesteuerte Prozessketten aus dem Plug-In ProFLOW.cepk enthalten. Dabei implementiert ein Transformator die Transformation von Ereignisgesteuerten Prozessketten in Organisations-Swimlane-Diagramme (EEPKOSDTransformator) und ein weiterer in Informations-Swimlane-Diagramme (EEPKISDTransformator). Auf diese Transformatoren beziehen sich die folgenden Abschnitte.

Abbildung 6.3.: Metamodell eines Swimlane-Diagramms


6.2.3. Benutzerinteraktion


6.3. Transformator

Ein Transformator wandelt einen eingelestenen Prozess in eine Swimlane-Darstellung um und gibt ihn hinterher an den Editor zurück, der ihn dann anschließend in LaneEPC integriert. Transformatoren von ProFLOW.cepk-Prozessen in OSDs und ISDs vorgestellt. Wie Transformatoren für andere Prozess- oder Diagrammtypen implementiert werden können, wird in Abschnitt 6.4.1 beschrieben.


- Im nächsten Schritt werden alle Funktionen behandelt, an denen mehr als eine Organisationseinheit bzw. ein Informationsobjekt verzeichnet sind. Diese werden dann nach der Regel in Abschnitt 5.2.2 aufgespaltet. Der Einfachheit halber wird diese Regel auch bei ISDs angewandt, so müssen mehrere Informationsobjekte später nicht gesondert behandelt werden.

- Nachdem nun an jeder Funktion in OSDs genau eine Organisationseinheit bzw. in ISDs genau ein Informationsobjekt verzeichnet ist, können alle Beteiligten gesucht werden. Diese Liste aller Beteiligten wird nun benutzt, um die jeweiligen Swimlanes anzulegen. Dabei werden die Swimlanes so sortiert, dass die Swimlane „undefined“ in OSDs ganz links steht und die Swimlane „unabhängig“ in ISDs ganz rechts steht. Dies geschieht, damit in OSDs Inkonformitäten in der Modellierung sofort sichtbar sind, da, wie bereits beschrieben, in OSDs keine Funktionen ohne Organisationseinheiten existieren sollten.

- Im nächsten Schritt werden nun die Elemente auf die zugehörigen Swimlanes verteilt. Dabei wird jede Funktion einzeln behandelt. Sie und ihre Vorgänger- und Nachfolgerereignisse, inklusive eventueller Verknüpfungen, werden in die jeweilige Swimlane geschrieben.
6. Implementierung

- Sind nun alle Funktionen verteilt, werden auf die in den Swimlanes vorhandenen Verknüpfungen die in Abschnitt 5.2.3 vorgestellten Regeln angewandt.


- Im letzten Schritt werden dann die Maße der Swimlanes angepasst und diese nebeneinander platziert.

- Der so transformierte Prozess wird nun wieder an das Editorfenster übergeben und von GEF und ProFLOW gezeichnet.

6.4. Erweiterungsmöglichkeiten

Das Ziel der Implementierung dieser Bachelorarbeit ist eine Visualisierung von Swimlanes in Ereignisgesteuerten Prozessketten. Hier sollen nun Erweiterungsmöglichkeiten für LaneEPC vorgestellt werden, die über dieses Ziel hinausgehen, aber dennoch in Zukunft interessant sein könnten.

6.4.1. Weitere Diagrammtypen und weitere EPK-basierte Prozesstypen


LaneEPC bietet zu diesem Zweck die Möglichkeit, Plug-Ins zu schreiben, die neue Diagrammtypen definieren. Dazu muss in diesem lediglich eine Extension auf LaneEPC.diag erstellt werden. In dieser werden die folgenden Werte angegeben:

- processId - Die Typ-ID des Prozesses, der transformiert werden soll.
- id - Die ID des Diagramm-Typs
- name - Der Name des Diagramm-Typs. (Wird bei der Auswahl der Diagramm-Typen angezeigt.)
- class - Die Transformator-Klasse des Diagramm-Typs (erweitert CommonTransformator).
- undefinedName - Der Name der Swimlane, in die nicht zugewiesene Elemente eingefügt werden.
- undefinedClass - Die Klasse der Annotation, mit welcher undefinedName angefügt wird.

Desweiteren muss ein Transformator - als Singleton-Klasse - geschrieben werden, der CommonTransformator erweitert. In diesem werden Funktionen zur Verarbeitung von Verknüpfungen (handleConnectors()), zur Sortierung der Swimlanes bzw. Liste der Beteiligten (sortListOfParticipants()), falls notwendig zum Ersetzen von Annotationen (replaceAnnotation()) und folgende Getter und Funktionen zum Validieren und Erstellen von Elementen implementiert:

- getDiagramId() - Gibt die ID des Diagrammtypes zurück.
- getTransitionType() - Gibt den Transitionstyp zurück.
- isValidFunction(Element element) - Überprüft, ob element eine Funktion ist.
6. Implementierung

- `isValidEvent(Element element)` - Überprüft, ob `element` ein Ereignis ist.
- `isValidConnector(Element element)` - Überprüft, ob `element` eine Verknüpfung ist.
- `isValidAnnotation(Annotation annotation)` - Überprüft, ob `annotation` eine Annotation ist, nach der die Swimlanes aufgebaut werden.
- `createNewEvent(String name)` - Erstellt ein neues Ereignis mit Namen `name`.
- `createNewFunction(String name)` - Erstellt eine neue Funktion mit Namen `name`.
- `createNewANDConnector()` - Erstellt eine neue AND-Verknüpfung. (Benötigt für die Aufsplittung von mehreren Annotationen an einer Funktion.
- `createNewProcess()` - Erstellt einen neuen Prozess().

In den zuletzt genannten Funktionen werden die durch den Prozesstyp gegeben Elemente eingetragen.

Desweiteren wird die Möglichkeit gegeben, über den Extension Point `LaneEPC.action` Aktionen zu definieren, die speziell auf einen Diagrammtyp angewendet werden können. So können die Aktionen, wie in ProFLOW üblich, über `ProFLOW.ui.processActions` und `ProFLOW.ui.modelObjectActions` oder `LaneEPC.action`, unter der Angabe eines Diagrammtyps, eingetragen werden.

6.4.2. Editorenfenster

In Eclipse gibt es die Möglichkeit, Editorenfenster mit mehreren Arbeitsflächen zu öffnen. Zwischen diesen Arbeitsflächen kann dann mittels Registerkarten gewechselt werden. So könnte man sich vorstellen, dass, wenn eine ProFLOW-Datei geöffnet wird, auf einer Registerkarte der Standard-ProFLOW-Editor und auf weiteren Registerkarten die Darstellungen für die Diagrammtypen, die für diesen Prozess verfügbar sind, angezeigt wird. Um dies zu realisieren, muss die Editor-Klasse in `ProFLOW.ui` angepasst, bzw. ersetzt werden. Weitere Informationen hierzu finden sich in der API von Eclipse [Theb].

6.4.3. Verschiebung und Skalierung von Elementen

ProFLOW unterstützt in der vorliegenden Version keine Containerelemente, welche allerdings für die Swimlanes zwingend notwendig sind. Containerelemente sind Elemente, die andere Elemente beinhalten können. Die Swimlanes wurden hier provisorisch als Containerelemente implementiert.

Allerdings sind im Moment noch keine Benutzerinteraktionen möglich, die sich auf das Verschieben und Skalieren von Elementen innerhalb einer Swimlane beziehen. Aus diesem Grund wurde eine automatische Positionierung der Elemente implementiert.


6.4.4. EPK-Elemente

`ProFLOW.cepk` unterstützt aktuell nicht alle Elemente einer Ereignisgesteuerten Prozesskette. So sind im Moment noch keine Prozesswegweiser und keine Anwendungssysteme implementiert. Wenn diese fertiggestellt sind, müssen in den jeweiligen Transformatoren nur die beiden Funktionen `isValidAnnotation()` und `isValidEvent()` angepasst werden.
6. Implementierung

6.4.5. EPML

In Abschnitt 2.5 wurde der XML-Dialekt EPML vorgestellt. Man könnte sich nun gut vorstellen, EPML als alternative Dateiquelle für Ereignisgesteuerte Prozessketten zu verwenden. Dazu würde, parallel zu dem in ProFLOW integrierten XMLImporter, ein EPML-Importer implementiert werden, der EPML-Dateien ausliest und daraus Prozesse erstellt. EPML und das ProFlow-Dateiformat sind sich sehr ähnlich, daher sollten keine umfangreichen Änderungen gegenüber dem XMLImporter vorgenommen werden müssen. Durch das EPML-Format wären auch Hierarchien in Ereignisgesteuerten Prozessketten (siehe Abschnitt 2.2.2) möglich.

6.5. Zusammenfassung

In diesem Kapitel wurde die Implementierung einer Visualisierung von Swimlanes in Ereignisgesteuerten Prozessketten vorgestellt. Dazu wurden die zugrundeliegenden Architekturen beschrieben. Darauf folgend wurde LaneEPC erläutert. Es wurde das Verhältnis zu den ProFLOW-Plug-Ins dargestellt und auf die interne Struktur von LaneEPC, sowie die Benutzerinteraktionen eingegangen. Im Anschluss wurden die Transformatoren für ProFLOW.cepk vorgestellt. Zum Ende dieses Kapitels wurden dann ausführlich diverse Erweiterungsmöglichkeiten beschrieben.
7. Beispiel

In diesem Kapitel werden zwei Beispiele für Swimlane-Diagramme in ProFLOW gegeben. Dazu wird jeweils zuerst die Darstellung in ProFLOW und danach in LaneEPC gezeigt. Die Ereignisgesteuerte Prozesskette welche hier als Organisations-Swimlane-Diagramm dargestellt ist, handelt von der Leergutannahme in einem Supermarkt mit den Beteiligten \textit{Kunde} und \textit{Mitarbeiter}. Das Beispiel für Informations-Swimlane-Diagramme zeigt wiederum das aus Abschnitt 4.3.4 bekannte Beispiel eines Kopiervorgangs von Datenbanktabellen.
7. Beispiel

7.1. Organisations-Swimlane-Diagramm

Beispiel OSD - ProFLOW

Abbildung 7.1.: Beispiel für OSD: EPK in ProFLOW
Beispiel OSD - LaneEPC

Abbildung 7.2.: Beispiel für OSD: EPK in LaneEPC
7. Beispiel

7.2. Informations-Swimlane-Diagramm

Beispiel ISD - ProFLOW

Abbildung 7.3.: Beispiel für ISD: EPK in ProFLOW
### Beispiel ISD - LaneEPC

Abbildung 7.4.: Beispiel für ISD: EPK in LaneEPC
8. Zusammenfassung und Ausblick


Nachdem in Kapitel 6 die Implementierung dargestellt wurde, wurden in Kapitel 7 jeweils ein Beispiel für ein Organisations-Swimlane-Diagramm und ein Informations-Swimlane-Diagramm in LaneEPC vorgestellt.

8. Zusammenfassung und Ausblick

Je nach Komplexität des Prozesses können aber durchaus große Diagramme entstehen. Werden relativ kleine Prozesse mit wenigen beteiligten Organisationseinheiten oder Informationsobjekten verwendet, ist zu erwägen, ob die klassische Darstellung nicht die bessere Wahl wäre. Sind allerdings viele Organisationseinheiten bzw. Informationsobjekte in einem Prozess beteiligt, so sollte die Swimlane-Darstellung bevorzugt werden.

8.1. Ausblick


Literaturverzeichnis


Abbildungsverzeichnis

2.1. Metamodell einer EPK in Anlehnung an [Lüb06] ................................. 11
2.2. Funktion .................................................. 11
2.3. Ereignis ................................................. 12
2.4. Prozesswegweiser ...................................... 12
2.5. eEPK-Elemente ......................................... 13
2.6. Fallbeispiel „Bestellung“ .............................. 14
2.7. Darstellung eines UML-Aktivitätsdiagramms in Swimlanes in Anlehnung an [BRJ99] .................. 15

3.1. Gegenüberstellung - EPK (links) und UML 2.0 Aktivitätsdiagramm (rechts) .................. 19

4.1. Konzept .................................................... 21
4.2. zugrundeliegende EPK .................................. 23
4.3. Variante 1: Anlehnung an UML-Swimlanes ................................. 24
4.4. Variante 2: passive Swimlanes ......................................... 25
4.5. Variante 3: Unabhängige Teilprozesse ..................................... 26
4.7. AND-Join-Verknüpfung ..................................... 28
4.8. Möglichkeit 1 .............................................. 28
4.9. Möglichkeit 2 .............................................. 28
4.10. EPK (links) und Swimlane-Diagramm (rechts) .......................... 29

5.1. Verwendete Variante ....................................... 30
5.2. Funktion mit einer Organisationseinheit ................................. 31
5.3. Funktion mit mehreren Organisationseinheiten ......................... 32
5.4. Funktion mit mehreren Informationsobjekten ............................ 32
5.5. Ereignis ® Funktion AND-Split .................................. 33
5.6. Funktion ® Ereignis AND-Join (OSD) .................................. 33
5.7. Funktion ® Ereignis AND-Join (ISD) .................................. 34
5.8. Funktion ® Ereignis OR-Join ...................................... 34
5.9. Funktion ® Ereignis OR-Join (ISD) .................................. 34
5.10. Funktion ® Ereignis XOR-Join (OSD) .................................. 35
5.11. Funktion ® Ereignis XOR-Join (ISD) .................................. 35
5.12. Beispiel einer manuellen Transformation: EPK in klassischer Darstellung ... 38
5.13. Beispiel einer manuellen Transformation (Schritt 1) ................. 39
5.14. Beispiel einer manuellen Transformation (Schritt 2) ................. 40
5.15. Beispiel einer manuellen Transformation (Schritt 3) ................. 41
5.16. Beispiel einer manuellen Transformation (Schritt 4) ................. 42

6.1. Das Verhältnis der Plug-Ins untereinander .................................. 45
6.2. Plug-In-Struktur ............................................ 45
6.3. Metamodell eines Swimlane-Diagramms .................................. 46
Abbildungsverzeichnis

7.1. Beispiel für OSD: EPK in ProFLOW .......................... 52
7.2. Beispiel für OSD: EPK in LaneEPC .......................... 53
7.3. Beispiel für ISD: EPK in ProFLOW .......................... 54
7.4. Beispiel für ISD: EPK in LaneEPC .......................... 55

Tabellenverzeichnis

2.1. Verknüpfungen .................................................. 13
5.1. weitere Verknüpfungen ........................................ 36

Listings

2.1. EPML-Elemente als Syntax-Baum ............................ 17
A. Fragebogen

Die folgenden Seiten enthalten den Fragebogen, welcher zur Validierung des Konzeptes dieser Bachelorarbeit verwendet worden ist.
Fragebogen zur Validierung von Swimlanes in EPK


1 generelle Swimlane-Darstellung

Im Folgenden werden 4 mögliche Varianten zur Darstellung der in Abbildung 1 gezeigten EPK in Swimlanes vorgestellt. In allen Varianten stellt die vertikale Achse die Zeit dar.

Variante 1: Keine Überschneidungen, Swimlanes können einzeln betrachtet werden.

Variante 2: wie Variante 1, mit Verbindungspfeilen zur Darstellung von Weiterführungen (Wartezustand)


Variante 4: Überschneidungen zwischen Swimlanes möglich. Semantisch unklar.
1.1 Bewerten Sie bitte die 4 Varianten: (Schulnoten von 1 bis 6)

Übersichtlichkeit:
Variante 1: _____  Variante 2: _____
Variante 3: _____  Variante 4: _____

Brauchbarkeit (im Betrachten):
Variante 1: _____  Variante 2: _____
Variante 3: _____  Variante 4: _____

Handhabbarkeit (im Erstellen):
Variante 1: _____  Variante 2: _____
Variante 3: _____  Variante 4: _____

1.2 Alles in Allem: Welche Variante würden Sie bevorzugen? __________

1.3 Für wen könnte das Konzept der Swimlanes interessant sein?

2 Hierarchie

In Abbildung 6 sehen Sie ein Beispiel für eine hierarchische EPK. Nun ist zu untersuchen, auf welche Art solche Hierarchien in Swimlanes dargestellt werden können.

In Abbildung 7 sehen Sie eine Möglichkeit mit Hierarchien umzugehen. Dabei wird die Hierarchie abgeflacht, so dass es nur noch eine Ebene gibt, die dargestellt wird. Es ist allerdings zu beachten, dass es sich hierbei um ein einfaches Beispiel mit nur einer Unterebene handelt. Sobald es mehrere Funktionen gibt die u.U. mehrere Unterebenen haben, wird diese Variante sicherlich nicht mehr so übersichtlich sein. Daneben findet sich in Abbildung 8 eine mögliche Darstellung in Swimlanes.

Konzept und Implementierung von Swimlanes in Ereignisgesteuerten Prozessketten
Eine weitere Möglichkeit ist in Abbildung 9 und 10 dargestellt. Hierbei bleiben die Hierarchien getrennt und werden einzeln in Swimlanes abgebildet.

2.1 Welche dieser beiden Varianten würden Sie bevorzugen?
- Abgeflacht
- Getrennt

2.2 Wie könnte man Hierarchien in Swimlanes noch darstellen?

3 AND-Verknüpfung (Join)

Nun soll Darstellungsmöglichkeiten für eine AND-Join-Verknüpfung aufgezeigt werden. Diese Verknüpfung ist besonders interessant, da innerhalb einer Swimlane geprüft werden muss, ob eine Funktion einer anderen Swimlane ausgeführt wurde.


Variante 2: Anstatt Ereignisse einzuführen, wird hier die Funktion der Nachbarswimlane eingeführt. Allerdings hätte man in diesem Fall Organisationseinheiten innerhalb einer Swimlane, was eigentlich dem Prinzip der Swimlanes widerspricht.

3.1 Welche Variante würden sie präferieren?
- Variante 1
- Variante 2

3.2 Wie könnte man eine solche AND-Verknüpfung noch realisieren?
4 Dokumente / Informationsobjekte

Im Folgenden soll beispielhaft eine EPK betrachtet werden, in der es darum geht den Inhalt einer Datenbanktabelle in eine neue Datenbanktabelle einer anderen Datenbank zu kopieren.

Besonderes Augenmerk soll hier auf die Notation der Ein- und Ausgaben gelegt werden, welche in der klassischen EPK-Darstellung durch die Pfeile an den Ressourcen gekennzeichnet wird.

In Swimlane DB 1 ist hier eine Variante dargestellt, in der der Datenfluss mittels Pfeilen gekennzeichnet wird.

Eine andere Variante ist in Swimlane DB 2 dargestellt. Hier wird neben den Funktionen der Datenfluss als Textlabel dargestellt.

Eine dritte Variante wäre es, den Datenfluss direkt in das Label der Funktion einzufügen.

4.1 Welche Variante finden sie am ansprechendsten?

☐ Variante 1 (Pfeile)
☐ Variante 2 (Textlabel)
☐ Variante 3 (Einfügen in Funktion)

5 Anmerkungen (zur freien Verfügung)

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

Vielen Dank für ihre Unterstützung

Konzept und Implementierung von Swimlanes in Ereignisgesteuerten Prozessketten
B. Beiliegende CD

Auf der am hinteren Einband dieser Arbeit befestigten CD findet sich die folgende Struktur:

<table>
<thead>
<tr>
<th>Verzeichnis</th>
<th>Inhalt</th>
</tr>
</thead>
<tbody>
<tr>
<td>/thesis</td>
<td>Diese Arbeit in elektronischer Form als PDF-Datei</td>
</tr>
<tr>
<td>/LaneEPC/source</td>
<td>Der Quellcode von LaneEPC</td>
</tr>
<tr>
<td>/LaneEPC/binary</td>
<td>LaneEPC im Binär-Format als JAR-Datei</td>
</tr>
<tr>
<td>/LaneEPC/doc</td>
<td>Die JavaDoc-API zu LaneEPC</td>
</tr>
<tr>
<td>/ProFLOW</td>
<td>ProFLOW in einer leicht abgeänderten Version (Erläuterung: siehe unten)</td>
</tr>
</tbody>
</table>

Erläuterungen zur beigefügten ProFLOW-Version

Um Zugriff auf das EPK-Modell von ProFLOW zu gelangen, musste im Plug-In ProFLOW.eepk das Paket proFlow.eepk.model zum Export freigegeben werden. Damit LaneEPC reibungslos verwendet werden kann, wurde diese Anpassung hier vorgenommen. Der Maintainer des Plug-Ins ist benachrichtigt, daher wird diese Änderung im nächsten Release von ProFLOW wahrscheinlich vorhanden sein.
Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe.

Tristan Wehrmaker
Bad Salzdetfurth, den 27. Februar 2007