View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Qucosa - Publikationsserver der Universität Leipzig

Musterbasierte Überprüfung der Qualitätseigenschaften von Geschäftsprozessmodellen

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION
zur Erlangung des akademischen Grades
DOCTOR RERUM NATURALIUM
(Dr. rer. nat)

im Fachgebiet
Informatik

vorgelegt von Dipl.-Math. Ralf Laue
geboren am 25. April 1968 in Leipzig

Die Annahme der Dissertation wurde empfohlen von:
1. Professor Dr. Volker Gruhn, Universität Leipzig
2. Professor Dr. Markus Nüttgens, Universität Hamburg

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am
2. Februar 2010 mit dem Gesamtprädikat *summa cum laude*.

Zusammenfassung

Geschäftsprozessmodelle, die mit graphischen Modellierungssprachen erstellt werden, spielen eine bedeutende Rolle sowohl im betriebswirtschaftlichen Kontext als auch als frühe Artefakte der Softwareerstellung. Ihr Haupteinsatzzweck besteht darin, die Kommunikation zwischen Fachexperten, Betriebswirten und Softwareentwicklern zu vereinfachen. Um diesen Zweck zu erfüllen, sollten solche Modelle fehlerfrei und leicht verständlich sein.

In dieser Arbeit wird die in der Softwaretechnik seit langem etablierte Erkenntnis, dass gut strukturierte Programme leichter verständlich und einfacher zu ändern sind, aufgegriffen und auf Geschäftsprozessmodelle übertragen.

Hierzu wird zunächst der Begriff der Strukturiertheit eines Geschäftsprozessmodells (am Beispiel der Modellierungssprache *ereignisgesteuerte Prozessketten*) definiert. Auf dieser Definition aufbauend wird eine Metrik entwickelt, die den Grad von (Un-)strukturiertheit in einem Geschäftsprozessmodell misst. Es wird gezeigt, dass zwischen den nach dieser Metrik ermittelten Werten und tatsächlich im Modell vorhandenen Fehlern ein starker Zusammenhang besteht.

Im Anschluss daran werden Modellstrukturen, die zur Verletzung der Strukturiertheits-Eigenschaft führen, systematisch katalogisiert. Auf diese Art wird ein Katalog von „Problemmustern“ entwickelt und die Auswirkungen der einzelnen Muster werden diskutiert. Anschließend wird ein heuristisches Prolog-basiertes Verfahren vorgestellt, mit dem die Muster, die Anlass zu einer Warnung oder einer Fehlermeldung geben, erkannt werden. Ziel ist es, dass dieses heuristische Verfahren (integriert in ein Modellierungswerkzeug) dem Modellierer während des Modellierens sofort eine Rückmeldung über mögliche Modellverbesserungen liefert.

Zur Validierung des Ansatzes wurde das Verfahren an fast 1000 Modellen ausgeführt. Im Vergleich dazu wurde eine Korrektheitsanalyse mit drei bekannten Werkzeugen vorgenommen. Es zeigte sich, dass der heuristische Ansatz nahezu gleichwertige Ergebnisse liefert wie die Analysen dieser Werkzeuge, die auf einer Untersuchung des gesamten Zustandsraumes basieren. Er hat jedoch erhebliche Vorteile bezüglich Zeitdauer und Speicherverbrauch.

Anders als bei bisher etablierten Verfahren wird es so möglich, Tests während des Modellierens im Hintergrund ablaufen zu lassen, um dem Modellierer sofortige Rückmeldungen zu liefern. Durch eine Erweiterung der Heuristiken auf Indizien für mögliche Verständnisprobleme und inhaltliche Fehler ist es außerdem möglich, weitere Probleme in Modellen zu finden, die bei bekannten Ansätzen noch keine Beachtung fanden.

Inhaltsverzeichnis

Symbole und Abkürzungen

$\triangleright k$	Menge der Eingangskanten des Knotens k, vgl. Def. 10 auf Seite 20
$k\triangleleft$	Menge der Ausgangskanten des Knotens k, vgl. Def. 10 auf Seite 20
$\wp(M)$	Potenzmenge einer Menge M
$\mathbf{B}(s,j)$	ein Split-Join-Block (s-j-Block), vgl. Def. 29 auf Seite 101
$card\ M$	Zahl der Elemente einer endlichen Menge M
$\mathbf{N}_j(s)$	die durch j begrenzte Nachfolgermenge von s, vgl. Def. 27 auf Seite 101
$\mathbf{M}(P)$	Menge der in einem Pfad P vorkommenden Knoten, vgl. Def. 4 auf Seite 16
$match(x,y)$	Match-Relation, vgl. Def. 9 auf Seite 18
$pred(\overline{K})$	Vorgängerknoten einer Menge reduzierbarer Knoten, vgl. Def. 20 auf Seite 78
$succ(\overline{K})$	Nachfolgeknoten einer Menge reduzierbarer Knoten, vgl. Def. 20 auf Seite 78
$type(x)$	Typ eines Konnektors, vgl. Def. 2.2 auf Seite 13
$\mathbf{V}_s(j)$	die durch s begrenzte Vorgängermenge von j, vgl. Def. 28 auf Seite 101
BPEL	Business Process Execution Language
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
BPMN-Q	Business Process Modeling Notation Query Language
DoS	Degree of Structuredness
EPK	ereignisgesteuerte Prozesskette
EPML	Event-Driven-Process-Chain-Markup-Language
GPM	Geschäftsprozessmodell
MOF	Meta Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
UCC	Unmatched Connector Count
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation
YAWL	Yet Another Workflow Language

Verzeichnis der eingeführten Begriffe und Definitionen

1 Einleitung

1.1 Geschäftsprozessmodellierung mit graphischen Modellen

Geschäftsprozessmodellierung dient dazu, Geschäftsprozesse, d.h. betriebliche Abläufe in Unternehmen oder Organisationen, abstrakt darzustellen. Hierzu werden meist graphische Modellierungssprachen verwendet, da eine graphische Darstellung im Vergleich zu einer text- oder formelbasierten Schreibweise als leichter verständlich für den Anwender gilt.

Sehr prägnant erklärt Hübscher [74] den Begriff eines Geschäftsprozesses wie folgt: „Ein Geschäftsprozess besteht aus einer zusammenhängenden abgeschlossenen Folge von Tätigkeiten, die zur Erfüllung einer betrieblichen Aufgabe notwendig sind." Gausmeier und Fahrwinkel [53] verstehen unter einem Geschäftsprozess eine „Menge von unternehmensspezifischen Aktivitäten, die in einem logischen sowie zeitlichen Zusammenhang zueinander stehen und inhaltlich abgeschlossen sind".

Beiden Definitionen[1] ist gemein, dass sie einen **Geschäftsprozess** erklären als eine organisatorisch zusammenhängende abgeschlossene Folge von Schritten (Tätigkeiten), die zur Erlangung eines Geschäftsresultats nötig sind. Im Gegensatz etwa zu einem Projekt, werden Geschäftsprozesse in der Regel wiederholt durchlaufen. Das bedeutet, dass bei der Beschreibung solcher Prozesse eine Abstraktion erfolgen muss, so dass eine Beschreibung mehrere konkrete Instanzen des Geschäftsprozesses darstellt.

Ein Beispiel für einen Geschäftsprozess ist ein Prozess „Verkauf eines Neuwagens". Hier wird von konkreten Details des Verkaufsvorgangs abstrahiert. Kein Geschäftsprozess wäre dagegen der - nur einmal durchgeführte - Ablauf „Kreditfinanzierter Verkauf eines Neuwagens vom Typ *Opel Corsa* in der Farbe schwarz an Bertram Hinze". Ein solcher konkreter Ablauf sollte (bei korrekter Modellierung) durch ein Modell des abstrakten Geschäftsprozesses beschrieben sein. Der konkrete Ablauf bildet eine **Instanz** des abstrakten Geschäftsprozesses.

[1] Einen Vergleich weiterer Definitionen ähnlichen Inhalts findet man in [198] sowie [102].

Geschäftsprozessmodelle (im Folgenden: GPM) sind Beschreibungen eines Geschäftsprozesses. Diese Beschreibungen erfolgen oft in einer graphischen Notation. GPM können verschiedene Aspekte von Geschäftsprozessen darstellen. Dies sind im Wesentlichen:

- der **Kontrollfluss**, d.h. der Ablauf der Prozesse (Reihenfolge der Tätigkeiten),
- der **Datenfluss**, der Erzeugung und Austausch von Daten und Dokumenten beschreibt,
- die **betriebliche Organisation**, um zu beschreiben, welche Organisationseinheiten und Rollen existieren und welche Organisationseinheit bzw. Rolle für welche Tätigkeit zuständig ist,
- die für die Durchführung des Geschäftsprozesses verwendeten **Betriebsmittel** (Ressourcen).

1.1.1 Anwendung von Geschäftsprozessmodellen

Es gibt verschiedene Gründe, Geschäftsprozessmodelle zu erstellen. Ein solcher Grund ist die **Dokumentation existierender Geschäftsprozesse**, zum Beispiel in Form von Prozesshandbüchern. Ein Nutzen einer solchen Dokumentation liegt darin, neuen Mitarbeitern die Einarbeitung in die existierenden Prozesse zu erleichtern. Werden die im Geschäftsprozessmodell beschriebenen Vorgaben in der täglichen betrieblichen Arbeit tatsächlich umgesetzt, so ist weiterhin eine garantierte Qualität der Arbeit zu erwarten. Mehrfacharbeit und Unklarheiten über Verantwortlichkeiten werden vermieden, und die Prozessbeteiligten kennen die sinnvollen Reihenfolgen der zur Erreichung eines Geschäftsresultats nötigen Aktivitäten. Aus den genannten Gründen verlangt die weltweit akzeptierte Qualitätsmanagement-Norm ISO 9001:2000 [75] auch eine Dokumentation der Soll-Geschäftsprozesse und eine ständige, dokumentierte Kontrolle, inwiefern die tatsächlichen betrieblichen Abläufe mit den modellierten übereinstimmen. Schließlich ist eine Dokumentation von Geschäftsprozessen auch Voraussetzung für den Aufbau organisationsübergreifender Geschäftsprozesse, wie er etwa durch die Auslagerung von Teilprozessen an Zulieferer geschieht.

Ein weiterer wichtiger Grund für den Einsatz von GPM ist die **Optimierung von Geschäftsprozessen** (engl. Business Process Optimization). Dabei werden die vorhandenen Prozesse zunächst strukturiert und analysiert, bevor sie auf Optimierungsmöglichkeiten hin untersucht werden. Das Aufstellen formaler GPM hilft bei allen genannten Tätigkeiten: Die Notwendigkeit, ein GPM zu modellieren, zwingt zur

Strukturierung. Die dann erstellten Modelle helfen bei der Geschäftsprozessanalyse sowie bei der anschließenden Prozessoptimierung. Meist wird diese Prozessoptimierung in Sitzungen mit mehreren Beteiligten aus verschiedenen Verantwortungsbereichen durchgeführt. Dabei sorgen graphische Modelle für ein gemeinsames Verständnis der Prozesse bei den Beteiligten. In bestimmten Fällen ist es auch möglich, Hinweise zur Prozessoptimierung automatisch aus den Modellen zu gewinnen. Automatisch erkannt werden können zum Beispiel Medienbrüche (wenn beispielsweise Ausgaben eines Computersystems von Hand in ein anderes System übernommen werden), Verstöße gegen Sicherheitsrichtlinien [161] oder das Erstellen von Dokumenten, die im weiteren Verlauf des Geschäftsprozesses überhaupt nicht mehr benötigt werden [164].

Häufig ist das Erstellen von GPM sogar der entscheidende Schritt bei der Geschäftsprozessoptimierung. Üblicherweise verfügen die Prozessbeteiligten nämlich über eine auf ihren Aufgabenbereich eingeschränkte Sicht auf die betrieblichen Prozesse. Sobald aber die vorhandenen Prozesse formal dokumentiert werden müssen, wächst dann das Verständnis des Gesamtprozesses. Unklarheiten werden beseitigt, Ausnahmen erkannt und standardisiert („Prozessglättung“). Kawalek und Kueng beobachteten diese positiven Effekte in einer Fallstudie [83] und kamen zu der Aussage, dass „der Prozess der Erstellung eines Modells ebenso wichtig sein kann wie das Modell selbst“.

Ähnlich ist die Situation bei der **Neugestaltung von Geschäftsprozessen** (engl. Business Process Reengineering) [71]. Hier werden nicht bereits vorhandene Prozesse optimiert, sondern Geschäftsprozesse grundlegend neugestaltet. Graphische Modelle unterstützen die Kommunikation über die zu schaffenden Prozesse.

Neben der Dokumentation, Umgestaltung und Optimierung von Geschäftsprozessen spielen GPM auch eine wichtige Rolle bei der Unterstützung betrieblicher Abläufe mit IT-Systemen. Etwa seit der Jahrtausendwende ist bei Unternehmensanwendungen ein Trend zu servicebasierten Architekturen zu beobachten, bei denen einzelne Geschäftsprozesse in betrieblichen Informationssystemen durch Services (insbesondere Webservices) implementiert sind. Hier dienen die Modelle vor allem zur Beschreibung von Schnittstellen zwischen den an einem Geschäftsprozess beteiligten Services. Beschrieben wird insbesondere, in welcher Form Nachrichten zwischen den Services ausgetauscht werden. Als Standard für solche Beschreibungen hat sich in den vergangenen Jahren die Sprache BPEL [2] hervorgetan. Sie erfährt eine breite Unterstützung durch namhafte Anbieter wie IBM, Oracle, SAP oder Microsoft. Die genannten Unternehmen (und viele weitere) bieten Worfkflow-Engines an, mit denen es möglich

ist, in BPEL geschriebene Anwendungen direkt auszuführen. BPEL ist allerdings selbst keine graphische Modellierungssprache, sondern eine maschinell ausführbare Sprache. Für menschliche Leser ist das XML-basierte BPEL schwer lesbar.

Um einen Übergang von graphischen GPM zu maschinell ausführbaren BPEL-Anwendungen zu ermöglichen, wurden Transformationen von graphischen Modellierungssprachen nach BPEL vorgeschlagen [174, 128].

1.2 Geschäftsprozessmodellierungssprachen

Zur Spezifikation von Abläufen in Computerprogrammen stehen formal fundierte Spezifikationssprachen wie Z [154] oder VDM [80] zur Verfügung. Diese stark formalen Sprachen haben den Vorzug einer wohldefinierten Semantik. Für Geschäftsprozessanalysten und Experten der Anwendungsdomäne, für die man keine tieferen Informatikkenntnisse voraussetzen kann, sind solche Sprachen wegen ihres hohen Abstraktionsgrades jedoch ungeeignet. Statt dessen werden für die Geschäftsprozessmodellierung graphische Modellierungssprachen bevorzugt. Für diese Sprachen wird dann in Kauf genommen, dass ihre Semantik nur unvollständig formal definiert ist.

Die im deutschsprachigen Raum am häufigsten verwendete GPM-Sprache sind **ereignisgesteuerte Prozessketten** (EPKs). Diese Sprache wurden 1992 an der Universität des Saarlandes entwickelt und stellt eine semiformale Notation zur Modellierung von Geschäftsprozessen dar. Semiformal bedeutet hier, dass Syntax und Semantik in der ursprünglichen Veröffentlichung der Modellierungsmethode [84] nicht formal-exakt definiert wurden. Lehrbücher und Kurse zur Einführung in die Modellierung mit EPKs erwähnen auch heute noch nur in den seltensten Fällen exakte Definitionen für Syntax und Semantik.

Die Verwendung der EPK-Notation im Referenzmodell der Firma SAP hatte einen entscheidenden Einfluss auf die Verbreitung dieser Modellierungssprache. Diese weite Verbreitung ist auch ein Hauptgrund dafür, dass die in dieser Arbeit vorgestellten Ansätze hauptsächlich an EPK-Modellen dargestellt und validiert werden. Syntax und Semantik von ereignisgesteuerten Prozessketten werden in Kapitel 2 beschrieben.

Neben EPKs wird auch die Business Process Modeling Notation (BPMN) sehr häufig eingesetzt. Diese 2002 entwickelte Notation wurde von der Business Process Management Initiative (BPMI) propagiert. Diese fusionierte 2005 mit der Object Management Group (OMG), einer herstellerneutralen Organisation, die auch die Unified Modelling Language (UML) standardisiert.

In Marketingaussagen der BPMI wurde BPMN häufig als bequeme graphische

Darstellung der XML-basierten Business Process Execution Language (BPEL) dargestellt. Eine Abbildung von BPMN auf BPEL wurde von der BPMI jedoch nur anhand von Beispielen angegeben [191]. Da BPMN im Gegensatz zu BPEL, das eine strenge Blockstruktur voraussetzt, eine beliebig unstrukturierte Modellierung erlaubt, kann es eine solche Übersetzung im allgemeinen Falle auch nicht geben [136]. Für BPMN-Modelle mit eingeschränktem Sprachumfang wurden BPMN-BPEL-Transformationen angegeben [174, 128].

Im Vergleich zu EPKs verfügt BPMN über eine größere Zahl von Modellierungselementen. In der Praxis gut verwendbar sind beispielsweise eigene Symbole für die Behandlung von Ausnahmen (engl. Exceptions) und das Zurücknehmen von bereits ausgeführten Aktivitäten (Kompensierung). Auf Grund der hohen Zahl an Modellierungselementen gibt es noch keine Veröffentlichung, die die Semantik von BPMN in ihrem Gesamtumfang formal definiert. Existierende Ansätze beschreiben jeweils eine Untermenge der BPMN-Notation [36, 193, 19].

Aktivitätsdiagramme der Unified Modeling Notation (UML) eignen sich ebenso zur Darstellung von GPM. Sie werden allerdings für diesen Einsatzzweck in der Praxis eher wenig genutzt. Verbreitung haben Aktivitätsdiagramme zur Modellierung von GPM vor allem dort gefunden, wo eine enge Beziehung zwischen dem GPM und der Entwicklung von geschäftsprozessbegleitender Software existiert. Ebenso wie BPMN unterstützen UML Aktivitätsdiagramme eine recht große Zahl von Notationselementen. Das führt dazu, dass der Standard einige Fragen zur Semantik von Aktivitätsdiagrammen unbeantwortet lässt. Veröffentlichungen, die UML-Aktivitätsdiagrammen eine formale Semantik geben (etwa [45, 157]) beschreiben wiederum nicht die Gesamtheit der möglichen Modellierungselemente. Als Vorzug der UML-Aktivitätsdiagramme im Vergleich zu anderen GPM-Modellierungssprachen ist zu erwähnen, dass ihre Syntax formal durch ein Metamodell (MOF) definiert ist. Das hat u.a. zur Folge, dass es mit XMI ein anerkanntes Dateiformat zum Austausch solcher Modelle gibt. Ein weiterer Vorzug besteht in der großen Zahl an vorhandenen Modellierungswerkzeugen.

Aus wissenschaftlicher Sicht interessant ist die Modellierungssprache YAWL („Yet Another Workflow Language“) [175], die allerdings bisher in der Praxis wenig Verbreitung gefunden hat. Sie zeichnet sich durch eine formal wohldefinierte Semantik aus und unterstützt alle in [176] identifizierten Workflow-Patterns. Diese Workflow-Patterns stellen Ablaufbeziehungen zwischen Aktivitäten innerhalb eines Geschäftsprozesses dar. Unter Ablaufbeziehungen verstehen wir hierbei Dinge wie „Parallele Ausführung“, „Abbrechen eines laufenden Prozesses“ oder „gleichzeitige

Ausführung eines Prozesses in mehreren Instanzen“. Workflow-Patterns wurden durch die Untersuchung von Geschäftsprozessen aus der Praxis sowie den Modellierungsmöglichkeiten in Software, die die Ausführung von Geschäftsprozessen unterstützen soll (sog. Workflow Engines), gewonnen. Keine der zuvor besprochenen GPM-Sprachen unterstützt so wie YAWL alle dieser Workflow-Patterns [176].

Schließlich ist zu erwähnen, dass zahlreiche weitere graphische Modellierungssprachen als nichtstandardisierte Darstellungen jeweils eines Werkzeuganbieters existieren. Beispiele hierfür sind die Modellierungssprachen des *IBM WebSphere BusinessModellers* oder des *JBoss jBPM Visual Designers*.

Für die weiteren Untersuchungen in dieser Arbeit wird die Notation der ereignisgesteuerten Prozessketten (EPK) verwendet. Ein Grund hierfür ist die große Verbreitung dieser Sprache und damit die Verfügbarkeit vieler Modelle, an Hand derer die Ergebnisse der Arbeit validiert werden können. Der zweite Grund für die Wahl besteht darin, dass die später in Kapitel 2 beschriebenen Grundelemente in allen anderen GPM-Sprachen wiederzufinden sind. Daher sind die in dieser Arbeit am Beispiel von EPK gewonnenen Erkenntnisse zum größten Teil auch auf die anderen GPM-Sprachen übertragbar.

1.3 Ziele dieser Arbeit

In dieser Arbeit soll die in der Softwaretechnik seit langem etablierte Erkenntnis, dass wohlstrukturierte Programme leichter zu verstehen und leichter zu warten sind, auf GPM übertragen werden. Zu diesem Zweck soll - am Beispiel der Modellierungssprache EPK - zunächst der Begriff der Wohlstrukturiertheit definiert werden. Auf dieser Definition aufbauend soll eine Komplexitätsmetrik entwickelt werden, die den Grad der Unstrukturiertheit eines Modells misst. Die Tauglichkeit dieser Metrik soll so untersucht werden, dass die „Güte“ der Metrik mit anderen in der Literatur eingeführten Komplexitätsmetriken verglichen werden kann.

Weiterhin soll der aus der Softwaretechnik bekannte Ansatz von *bad code smells* bzw. Anti-Entwurfsmustern (häufig anzutreffende schlechte Lösungen für oft wiederkehrende Probleme) auf GPM angewendet werden. Typische Modellierungsfehler oder Modellbestandteile, die auf solche Fehler hindeuten, sollen in einem Musterkatalog zusammengestellt werden. Insbesondere sollen Verletzungen der definierten Wohlstrukturiertheits-Eigenschaft als Muster erfasst werden.

Darauf aufbauend soll ein Verfahren beschrieben werden, mit dem die ermittelten Muster effizient in vorhandenen Modellen gefunden werden können. Dieses Verfahren

soll schnell genug arbeiten, um in einem Modellierungswerkzeug ständig im Hintergrund zu laufen und so dem Modellierer eine sofortige Rückmeldung über mögliche Modellierungsprobleme zu geben. Die Tauglichkeit dieses musterbasierten Verfahrens soll validiert werden, indem die Ergebnisse mit den Resultaten vorhandener Werkzeuge verglichen wird.

1.4 Prämissen für diese Arbeit

Für die Untersuchungen in dieser Arbeit werden einige Annahmen getroffen, die in diesem Abschnitt beschrieben werden und bestimmend für die Zielrichtung dieser Arbeit sind. Der Entwicklung der in dieser Arbeit vorgestellten Validierungsmethoden für GPM liegen die in diesem Abschnitt aufgeführten Prämissen zugrunde.

1.4.1 Prämisse 1: GPM sollen von Menschen gelesen werden

Wie in den einleitenden Abschnitten ausgeführt wurde, ist eine wesentliche Aufgabe von GPM, die Kommunikation zwischen Prozessbeteiligten zu unterstützen. In dieser Arbeit soll der Schwerpunkt auf die Betrachtung von GPM gelegt werden, die zu Zwecken wie der Prozessdokumentation oder der Prozessverbesserung erstellt wurden, also von Menschen gelesen werden sollen. Der in verschiedenen Ansätzen ebenfalls propagierte Anwendungsbereich, GPM in eine von Workflow-Engines ausführbare Sprache (wie BPEL) zu übersetzen, steht nicht im Vordergrund der Betrachtungen.

Schlussfolgerungen aus Prämisse 1:

- Neben der formalen Korrektheit von Modellen ist auch deren Verständlichkeit für den menschlichen Leser eine Qualitätsanforderung, die an GPM zu stellen ist.
- Es ist zu tolerieren, dass Modelle existieren, die zwar von einem menschlichen Leser verstanden werden können, die jedoch nicht unmittelbar in eine maschinell ausführbare Sprache übersetzbar sind. Ein Beispiel für ein solches Modell wäre ein GPM, das mit einem der Ereignisse „Rechnung ist bezahlt“ und „Mahnung ist zu erstellen“ beginnt. Auch ohne dass dies ausdrücklich im Modell vermerkt wird, weiß der menschliche Betrachter eines solchen Modells, dass nur eines der beiden Ereignisse eintritt. Während also das Modell für den menschlichen Leser genügend Informationen enthält, müsste die Ausschluss-Beziehung zwischen beiden Ereignissen zusätzlich beschrieben werden, falls der Prozess in ausführbaren Code übersetzt werden soll.

1.4.2 Prämisse 2: Es erfolgt keine Einschränkung auf eine bestimmte Anwendungsdomäne

Die in dieser Arbeit angestrebten Untersuchungen der Qualitätseigenschaften von GPM sollen ohne Einschränkung auf eine bestimmte Anwendungsdomäne erfolgen.

Schlussfolgerungen aus Prämisse 2:

- Für die Untersuchungen stehen keine formal-ontologischen Beschreibungen der Begriffe aus einer bestimmten Anwendungsdomäne zur Verfügung. Zur Analyse kommen daher nur domänenunabhängige Methoden in Frage.
- Die Validierung der vorgeschlagenen Methoden soll an Modellen aus möglichst vielen verschiedenen Anwendungsdomänen erfolgen.

1.4.3 Prämisse 3: Ein Bestand an vorhandenen GPM ist zu unterstützen.

Wir gehen davon aus, dass in einer Organisation bereits eine Menge von GPM vorliegt, die weiter zu bearbeiten bzw. zu benutzen sind. Dies ist heute in der Regel insbesondere in großen Unternehmen der Fall. Oft wurden die vorhandenen Modelle in verschiedenen Projekten von verschiedenen Arbeitsgruppen und mit verschiedenen Werkzeugen erstellt. Über die Qualität dieser Modelle kann keine Annahme getroffen werden.

Schlussfolgerungen aus Prämisse 3:

- Wir können keine Einschränkungen treffen, nur nach bestimmten Modellierungskonventionen erstellte Modelle zu betrachten. Ebenso wenig können wir voraussetzen, dass bei der Erstellung der Modelle ein Werkzeug benutzt wurde, dass die Einhaltung bestimmter Qualitätsmerkmale erzwingt.
- Die vorhandenen Modelle wurden unter Umständen mit Werkzeugen erstellt, über deren Eigenschaften und Qualität nichts bekannt ist. Vor der Weiterverarbeitung der GPM in einem eigenen Werkzeug muss also zunächst mindestens sichergestellt werden, dass das vorliegende Modell syntaktisch korrekt ist.

2 Ereignisgesteuerte Prozessketten

Die Untersuchungen in dieser Arbeit erfolgen an Modellen, die in der Modellierungssprache der ereignisgesteuerten Prozessketten (im Folgenden: EPKs) vorliegen. Die Hauptgründe für diese Wahl sind:

- die große Verbreitung dieser Sprache im deutschsprachigen Raum, u.a. aufgrund der Tatsache, dass EPKs von der Firma SAP zur Beschreibung der Geschäftsprozesse in den SAP R/3-Standardanwendungen verwendet wurden,
- das Vorliegen einer großer Menge von Modellen, die für die Validierung der entwickelten Ansätze genutzt werden können,
- die Tatsache, dass EPKs die wichtigsten Konstrukte enthalten, die sich auch in anderen Modellierungssprachen (z.B. BPMN) finden. Dadurch sind die Ergebnisse der Arbeit auf diese Sprachen übertragbar.
- der relativ geringe Umfang an Modellierungselementen (wiederum im Vergleich etwa zu BPMN), wodurch die Formalisierung der Sprache nicht unnötig erschwert wird.

2.1 Informelle Beschreibung

Grundlegende Elemente von EPKs sind als Rechteck dargestellte **Funktionen** sowie als Sechseck dargestellte **Ereignisse**. Funktionen sind Aktivitäten innerhalb des Geschäftsprozesses. Ereignisse stehen für Bedingungen, genauer gesagt, Vor- und Nachbedingungen. Vorbedingungen sind Voraussetzungen, die erfüllt sein müssen, um eine bestimmte Aktivität zu starten. Nachbedingungen sind Aussagen, die (als Ergebnis der ausgeführten Aktivitäten) zu einem bestimmten Zeitpunkt im Ablauf des Geschäftsprozesses erfüllt sind. Der Kontrollfluss wird durch Pfeile (auch **Kontrollflusskanten** genannt) dargestellt. Zeigt ein Pfeil von einem Modellelement A zu einem Modellelement B, bedeutet dies, dass nach Ausführung von Element A das Element B auszuführen ist.

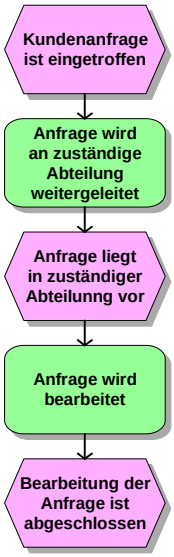


Abbildung 2.1: Beispiel für ein sehr einfaches EPK-Modell

Abb. 2.1 zeigt ein sehr einfaches EPK-Modell: Das Ereignis „Kundenanfrage ist eingetroffen“ ist Auslöser für die Bearbeitung des Prozesses. Als erste Aktivität wird die eingetroffene Anfrage an die zuständige Abteilung weitergeleitet. Dies hat das Ereignis „Anfrage liegt in zuständiger Abteilung vor“ zur Folge (Nachbedingung). Gleichzeitig ist dieses Ereignis Vorbedingung für die folgende Aktivität „Anfrage wird bearbeitet“. Schließlich ist der gesamte Geschäftsprozess mit dem Ereignis „Bearbeitung der Anfrage ist abgeschlossen“ beendet.

Solche einfachen Modelle, bei denen Aktivitäten einfach sequentiell ausgeführt werden, sind allerdings in der Praxis eher selten. In der Realität erfolgen im Verlaufe eines Geschäftsprozesses oft Entscheidungen. Durch eine solche Entscheidung wird eine von mehreren denkbaren Fortsetzungen des Geschäftsprozesses ausgewählt. Diese Entscheidungen müssen ebenso wie die möglichen Alternativen für den weiteren Verlauf modelliert werden. Ebenso muss ein GPM Situationen berücksichtigen, in denen mehrere Aktivitäten zeitgleich („parallel“) ausgeführt werden.

Zur Modellierung alternativer und paralleler Abläufe dienen in EPKs **Konnektoren**, dargestellt als Kreise mit einem Symbol darin. Wir unterscheiden zwei Arten von Konnektoren: **Splits** verzweigen den Kontrollfluss. Sie haben also eine ein- und mehrere ausgehende Kontrollflusskanten. **Joins** führen mehrere Zweige zusammen, sie haben also mehrere ein- und nur eine ausgehende Kontrollflusskante.

AND-Konnektoren (dargestellt mit dem Symbol ⓐ) modellieren die parallele Ausführung: Nach einem AND-Split werden die Kontrollflüsse der abgehenden Kontrollflusskanten parallel ausgeführt. Die parallelen Zweige werden an einem AND-Join wieder zusammengeführt. Analog modellieren XOR-Konnektoren (dargestellt als ⓧ) die Ausführung genau eines von mehreren alternativ möglichen Kontrollflüssen. OR-Konnektoren (dargestellt als ⓥ) modellieren die Ausführung von einem oder mehreren möglichen Kontrollflüssen. Nach einem OR-Join mit n Ausgängen wird also mindestens einer und höchstens alle n dieser Pfade parallel durchlaufen. An einem OR-Split wird gewartet, bis alle zuvor begonnenen Kontrollflüsse, die diesen OR-Split erreichen können, abgeschlossen sind.

Eine etwas komplexere Modellierung unseres Beispiels „Kundenanfrage“ zeigt Abb. 2.2. Nach Eintreffen der Anfrage wird hier zunächst die Aktivität (bzw. in EPK-Sprechweise: die Funktion) „Kundendaten im Bestandssystem suchen“ ausgeführt. Diese kann zu zwei möglichen Ereignissen (Nachbedingungen) führen, je nachdem, ob der Kunde Neukunde ist oder nicht. Diese Auswahl ist durch den XOR-Konnektor dargestellt. Falls es sich um einen Neukunden handelt, ist die Ausführung zweier Aktivitäten nötig: Dem Kunden ist ein Betreuer zuzuteilen und die Kundendaten müssen im Bestandssystem erfasst werden. Beide Aktivitäten sollen (um Zeitverlust zu vermeiden) in paralleler Ausführung erfolgen können, was durch den AND-Konnektor symbolisiert wird. Die letzten Schritte des Geschäftsprozesses sind für Neu- und Bestandskunden wieder einheitlich, weshalb der Kontrollfluss durch die entsprechenden Joins wieder zusammengeführt wird.

Mit den bisher genannten Elementen sind bereits die wesentlichen zur Modellierung des Kontrollflusses nötigen Bestandteile einer EPK beschrieben.

Weitere Modellierungselemente, die zur Zerlegung großer EPKs in mehrere Teilmodelle dienen, sind Prozesswegweiser und hierarchisierte Funktionen.

Prozesswegweiser dienen dazu, große Modelle in mehrere Teilmodelle zu zerlegen. In unserem Beispiel könnte etwa die Darstellung des Prozesses nach dem Ereignis „Anfrage liegt beim zuständigen Betreuer vor“ in eine andere EPK ausgelagert werden. Dann wäre nach diesem Ereignis ein Prozesswegweiser zu modellieren, der auf eine separate EPK „Auftragsverarbeitung“ verweist. Eine Verkettung von EPKs durch Prozesswegweiser ist vergleichbar mit einer `GOTO`-Anweisung einer iterativen Programmiersprache.

Demgegenüber bieten **hierarchisierte Funktionen** die Möglichkeit, verschiedene Abstraktionsstufen in einem Modell zu vereinen. In unserem Beispiel genügt es für einen groben Überblick, zu wissen, dass die Daten eines Neukunden im Bestands-

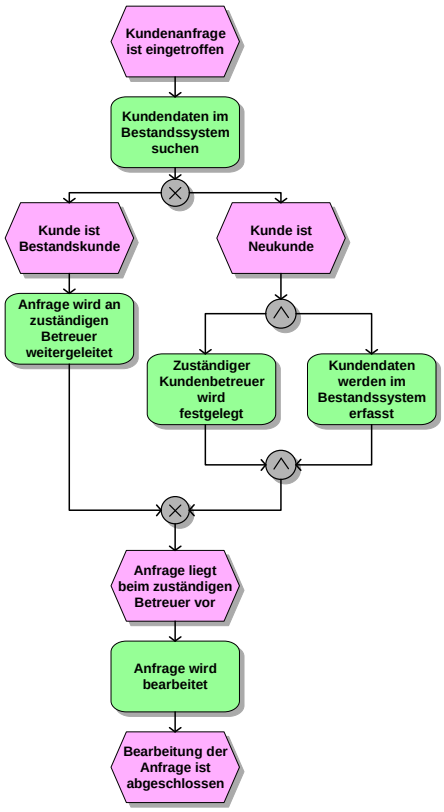


Abbildung 2.2: Einfaches EPK-Modell mit Konnektoren

system anzulegen sind. Wie das im Detail geschieht, wird auf dieser Abstraktionsstufe nicht betrachtet. Es ist aber möglich, den Prozess „Kundendaten werden im Bestandssystem erfasst“ in einer eigenen EPK zu modellieren. Eine solche Funktion, deren Details in einer separaten EPK modelliert sind, heißt hierarchisierte Funktion. Bei Benutzung eines geeigneten Modellierungswerkzeuges ist es leicht, zwischen den verschiedenen Abstraktionsebenen zu wechseln, was die Arbeit mit komplexen Modellen sehr erleichtert. Hierarchisierte Funktionen sind vergleichbar mit dem Aufruf eines Unterprogramms in einer Programmiersprache. Bei guter Anwendung des Prinzips der Modularisierung ist die genaue Kenntnis der detaillierten EPK nicht nötig, um den Ablauf in der übergeordneten EPK zu verstehen.

Prinzipiell ist es möglich, mehrere über Prozesswegweiser und hierarchisierte Funktionen miteinander verbundene EPK-Modelle zu einem einzigen EPK-Modell zusammenzufassen. Hierfür hat sich die anschauliche Bezeichnung „flachklopfen“ eingebürgert. In wissenschaftlichen Arbeiten über die Analyse von EPKs wird in

der Regel zur Vereinfachung angenommen, dass eventuell vorhandene Teilmodelle schon zu einem Gesamtmodell zusammengefügt wurden (siehe etwa [145]). Diese in praktisch relevanten Fällen meist zutreffende Annahme soll in dieser Arbeit ebenfalls getroffen werden. Im Folgenden werden wir also Prozesswegweiser und hierarchisierte Funktionen nicht weiter betrachten. Genaueres zu den Bedingungen, unter denen das Zusammenfügen von EPKs zu einem Gesamtmodell möglich ist, haben wir in [64] dargestellt.

In sogenannten **erweiterten EPK-Modellen** ist es möglich, zusätzliche Angaben zu notieren. Diese betreffen beispielsweise organisatorische Aspekte (*Wer führt eine Funktion aus?*) oder die Datenmodellierung (*In welchen Datenbanken sind Informationen abgelegt?*). Diese zusätzlichen Informationen werden im weiteren Verlauf dieser Arbeit ebenfalls nicht betrachtet.

2.2 Formale Syntax

Nachdem die Modellierungssprache EPK im vorigen Abschnitt zunächst informell eingeführt wurde, soll in diesem und den folgenden Abschnitten die Syntax und Semantik eines EPK-Modells formal diskutiert werden.

Die folgende Definition der Syntax orientiert sich an [145], [85] und [123]:

Definition 1 (EPK-Modell)
Ein EPK-Modell ist ein endlicher gerichteter Graph $\Lambda = (K, A)$. Dabei bezeichnet K die Knotenmenge und $A \subset K \times K$ die Kantenmenge von Λ.

Die Knotenmenge K ist die Vereinigung der folgenden fünf paarweise disjunkten Mengen:

- *F (Menge der Funktionen)*
- *E (Menge der Ereignisse)*
- *C_{xor} (Menge der XOR-Konnektoren)*
- *C_{or} (Menge der OR-Konnektoren)*
- *C_{and} (Menge der AND-Konnektoren)*

Weiter nennen wir $C = C_{xor} \cup C_{or} \cup C_{and}$ die Menge der Konnektoren und definieren für $c \in C$ die Funktion type derart, dass $type(c) = XOR$, falls $c \in C_{xor}$, $type(c) = OR$, falls $c \in C_{or}$ und $type(c) = AND$, falls $c \in C_{and}$.[1]

[1] Die Bezeichnung A für die Menge der Kanten ist vom englischen *arc* (Pfeil) und die Bezeichnung C für die Menge der Konnektoren vom englischen *connector* abgeleitet.

Ferner müssen die folgenden Eigenschaften erfüllt sein:

1. *Der Graph Λ ist zusammenhängend.*
2. *Der Graph Λ ist antisymmetrisch und einfach, d.h. enthält die Kantenmenge A eine Kante (a, b), so gibt es in A keine Kante (b, a) und auch keine weitere Kante (a, b).*
3. *Die Mengen E und F sind nicht leer, d.h. es gibt mindestens ein Ereignis und mindestens eine Funktion.*
4. *Funktionen besitzen genau eine eingehende und genau eine ausgehende Kante.*
5. *Ereignisse besitzen höchstens eine eingehende und höchstens eine ausgehende Kante. (Besitzt ein Ereignis genau eine eingehende und keine ausgehende Kante, nennen wir es **Endereignis**. Besitzt ein Ereignis keine eingehende und genau eine ausgehende Kante, nennen wir es **Startereignis**.)*
6. *Konnektoren haben entweder genau eine eingehende und mehr als eine ausgehende Kante (dann heißen sie **Split**) oder mehr als eine eingehende und genau eine ausgehenden Kante (dann heißen sie **Join**).*
7. *Der Graph Λ enthält keinen gerichteten Zyklus, der nur aus Konnektoren besteht.*
8. *Sind e_1 und e_2 Ereignisse, so gibt es keine Kante der Form (e_1, e_2) sowie keine Folge von Kanten der Form (e_1, k_1), (k_1, k_2), $\ldots$, (k_{n-1}, k_n), (k_n, e_2) mit $k_i \in C$ für alle $i = 1, \ldots n$ (wobei der Fall $n = 1$ zugelassen sein soll).*
9. *Sind f_1 und f_2 Funktionen, so gibt es keine Kante der Form (f_1, f_2) sowie keine Folge von Kanten der Form (f_1, k_1), (k_1, k_2), $\ldots$, (k_{n-1}, k_n), (k_n, f_2) mit $k_i \in C$ für alle $i = 1, \ldots n$ (wobei der Fall $n = 1$ zugelassen sein soll).*
10. *Ist e ein Ereignis und s ein Split mit $type(s) \in \{XOR, OR\}$, so gibt es keine Kante der Form (e, s) sowie keine Folge von Kanten der Form (e, k_1), (k_1, k_2), $\ldots$, (k_{n-1}, k_n), (k_n, s) mit $k_i \notin F$ für alle $i = 1, \ldots n$ (wobei der Fall $n = 1$ zugelassen sein soll).*
11. *Es gibt mindestens ein Start- und mindestens ein Endereignis.*
12. *Für jeden Knoten im Graphen gibt es einen gerichteten Pfad von einem Startereignis zu diesem Knoten.*
13. *Für jeden Knoten im Graphen gibt es einen gerichteten Pfad von diesem Knoten zu einem Endereignis.*

2.2.1 Anmerkungen zur Definition 1

Die Eigenschaften 1 bis 11 sind im Wesentlichen [145], [85] und [123] entnommen. Die Eigenschaften 12 und 13 werden in [85], nicht aber in [145] und [123] erwähnt. Da diese Eigenschaften keineswegs, wie man vielleicht vermuten könnte, aus den anderen Eigenschaften folgen (siehe Gegenbeispiel in Abb. 2.3), müssen sie als separate Forderungen genannt werden.

Der Sinn der meisten der oben geforderten Eigenschaften sollte nach der informellen Einführung von EPKs in Abschnitt 2.1 unmittelbar verständlich sein. Die Eigenschaften 8 und 9 sorgen dafür, dass sich Ereignisse und Funktionen im Kontrollfluss immer „abwechseln“, d.h. jede Funktion ist im Graphen (ggf. über Konnektoren) mit einem Ereignis verbunden und umgekehrt. Diese Forderung resultiert aus der Interpretation der Ereignisse als Vor- und Nachbedingungen der Funktionen. Eigenschaft 10 entspricht der in [84] formulierten Regel, dass „Ereignisse keine Entscheidungskompetenz“ haben sollen. Bevor nämlich eine Aufsplittung des Kontrollflusses mittels eines XOR-Splits oder eines OR-Splits erfolgt, muss eine Entscheidung gefällt werden, welche der Alternativen gewählt werden soll. Das Fällen einer solchen Entscheidung ist ein aktiver Prozess und muss daher ausdrücklich als Funktion modelliert werden.

In der Praxis werden die Eigenschaften 8 bis 10 mitunter weniger strikt gehandhabt. Insbesondere ist eine Aufeinanderfolge mehrerer Funktionen ohne dazwischenliegende Ereignisse durchaus üblich und in vielen Fällen auch unproblematisch. Im Gegenteil: Der Verzicht auf die Notation solcher Ereignisse kann der in den „Grundsätzen ordnungsgemäßer Modellierung“ [11] aufgestellten Forderung entsprechen, dass „das Modell nicht mehr Elemente beinhalten soll, als zum Verständnis und zur Wiedergabe der Intention notwendig sind“ (wörtlich zitiert aus [11]).

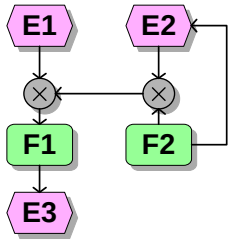


Abbildung 2.3: EPK, für die nur Eigenschaft 12 verletzt ist

2.2.2 Einfache Folgerungen aus Definition 1

Aus den in Definition 1 angegebenen Forderungen folgen unmittelbar die folgenden Eigenschaften:

1. Der Graph Λ enthält keine reflexive Kante (also eine Kante, die einen Knoten mit sich selbst verbindet), was unmittelbar aus den Forderungen 7, 8 und 9 in Def. 1 folgt.

2. Ein Knoten in Λ ohne eingehende Kante oder ohne ausgehende Kante muss stets ein Ereignis sein, da die Eigenschaft, keine ein- bzw. ausgehende Kante zu haben, in den Forderungen 4 bzw. 6 für Funktionen und Konnektoren ausgeschlossen wird.

2.3 Einige Definitionen

Eine EPK $\Lambda = (K, A)$ ist laut Definition ein gerichteter Graph. Wir definieren in diesem Abschnitt einige Begriffe im Zusammenhang mit Pfaden in Graphen.

Definition 2 (Vorgänger- und Nachfolgeknoten)
*Existiert in einer EPK $\Lambda = (K, A)$ eine Kante $(a, b) \in A$, so heißt a der **Vorgängerknoten** von b und b der **Nachfolgeknoten** von a.*

Definition 3 (Pfad)
In einer EPK $\Lambda = (K, A)$ verstehen wir unter einem Pfad von $a \in K$ nach $b \in K$

- *die Folge (a, b), falls $(a, b) \in A$*
- *eine Folge von Knoten $(a, k_1, \ldots, k_n, b)$ mit $k_i \in K$ für $i = 1, \ldots, n$ (wobei der Fall n=1 zugelassen sein soll), so dass gilt: $(a, k_1) \in A$, $(k_n, b) \in A$ sowie $(k_i, k_{i+1}) \in A$ für alle $i = 1, \ldots n - 1$.*

Wollen wir die Menge der Knoten in einem Pfad betrachten, benutzen wir das Symbol **M** wie folgt:

Definition 4 (Menge der Knoten eines Pfades)
Sei $P = (a, k_1, \ldots, k_n, b)$ wie oben definiert ein Pfad von a nach b. Mit dem Symbol $\boldsymbol{M}(P)$ bezeichnen wir dann die Menge der in P vorkommenden Knoten.

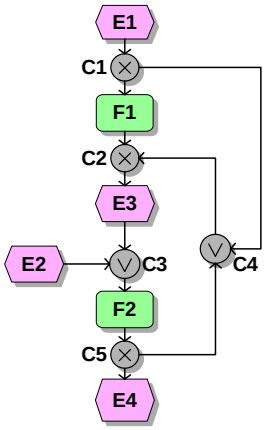


Abbildung 2.4: EPK mit Zyklus

Weiter definieren wir:

Definition 5 (schnittfreie Pfade)
Sei P_1 ein Pfad von a_1 nach b_1 sowie P_2 ein Pfad von a_2 nach b_2, so heißen P_1 und P_2 schnittfrei, wenn $\boldsymbol{M}(P_1) \cap \boldsymbol{M}(P_2) = \{a_1, b_1\} \cup \{a_2, b_2\}$.

Zwei schnittfreie Pfade dürfen also ihre Start- und Endknoten gemeinsam haben, jedoch keine weiteren Knoten.

Definition 6 (Zyklus)
Sei $\Lambda = (K, A)$ eine EPK. Ein Pfad Z von $a \in K$ nach $b \in K$ heißt Zyklus, falls $a = b$.

Definition 7 (Zykleneinstiegs- und -ausstiegsknoten)
Sei Z ein Zyklus in einer EPK $\Lambda = (K, A)$ und $e \in \boldsymbol{M}(Z)$. e heißt Zykleneinstiegsknoten, falls es ein Startereignis e_{start} und einen Pfad P von e_{start} nach e gibt, so dass $\boldsymbol{M}(P) \cap \boldsymbol{M}(Z) = \{e\}$.

e heißt Zyklenausstiegsknoten, falls es ein Endereignis e_{end} und einen Pfad P von e nach e_{end} gibt, so dass $\boldsymbol{M}(P) \cap \boldsymbol{M}(Z) = \{e\}$.

Beispiel:
Die EPK von Abb. 2.4 hat einen Zyklus (C2,E3,C3,F2,C5,C4,C2). Zykleneinstiegsknoten sind C2, C3 und C4; einziger Zyklenausstiegsknoten ist C5.

Definition 8 (Kanten und Pfade zwischen Mengen)
Sei $\Lambda = (K, A)$ eine EPK und $K_1 \subset K$ sowie $K_2 \subset K$ zwei echte Untermengen von K. Wir sagen dann, dass

- *eine Kante von K_1 nach K_2 existiert, falls es ein $s \in K_1$ und ein $t \in K_2$ gibt, so dass $(s, t) \in A$.*
- *ein Pfad von K_1 nach K_2 existiert, falls es ein $s \in K_1$ und ein $t \in K_2$ gibt, so dass ein Pfad von s nach t existiert.*

Unter den in der Praxis anzutreffenden EPKs gibt es solche, bei denen stets ein Split und ein Join vom gleichen Typ als zusammengehörig betrachtet werden können. Wir sprechen dann von einer wohlstrukturierten EPK; eine exakte Definition dieses Begriffs folgt in Abschnitt 7.1.

Es gibt jedoch ebenso EPKs, in dem Splits und Joins nicht in der beschriebenen Weise paarweise auftreten. Im weiteren Verlauf der Arbeit benötigen wir eine Definition, wann wir in einer (im allgemeinen Falle nicht wohlstrukturierten EPK) ein Paar aus Split und Join als zusammengehörig betrachten wollen. Dies führt uns zu

Definition 9 (match-Relation)
Einem Split s ist ein Join j zugeordnet (ausgedrückt durch die Relation $match(s, j)$), genau dann, wenn es zwei schnittfreie Pfade von s nach j gibt.

Diese Definition ist angelehnt an den in [103] eingeführten Begriff der *corresponding control elements*. Es sei an dieser Stelle darauf hingewiesen, dass der durch die match-Relation einem Split zugeordnete Join keinesfalls eindeutig bestimmt sein muss und dass umgekehrt durch die match-Relation mehrere verschiedene Splits einem Join zugeordnet sein können.

Für die EPK aus Abb. 2.4 etwa gilt sowohl $match(C1, C4)$ (die beiden schnittfreien Pfade sind hierbei $(C1, F1, C2, E3, C3, F2, C5, C4)$ und $(C1, C4)$) als auch $match(C1, C2)$ (hier sind die schnittfreien Pfade $(C1, F1, C2)$ und $(C1, C4, C2)$).

2.4 Das Austauschformat EPML

Das XML-basierte Austauschformat EPML [111] für EPK-Modelle wurde entwickelt, um eine Möglichkeit des Datenaustausches zwischen Modellierungswerkzeugen zu schaffen.

Im Folgenden wird eine kurze skizzenhafte Einführung in EPML gegeben. Diese ist bei weitem nicht vollständig, beschreibt aber die wichtigsten Elemente von EPML und reicht zum Verständnis der in dieser Arbeit beschriebenen Verfahren und Werkzeuge aus. Eine vollständige Einführung in EPML findet sich in [111] sowie auf der Website `www.epml.de`.

Das unten gezeigte EPML-Fragment beschreibt den oberen Teil der in Abb. 2.2 dargestellten EPK. Jeder Funktion der EPK entspricht ein function-Element, jedem Ereignis ein event-Element und jedem Konnektor ein and-, or- oder xor-Element. Jedes der genannten Elemente hat ein Attribut namens id, durch das es eindeutig identifiziert ist. Der Kontrollfluss wird durch arc-Elemente ausgedrückt. In unserem Beispiel bezeichnet das arc-Element mit dem Attribut id=11 die Kontrollflusskante zwischen dem Element mit id=1 und dem Element mit id=2, also zwischen dem Ereignis „Kundenanfrage eingetroffen“ und der Funktion „Kundendaten im Bestandssystem suchen“.

```
<?xml version="1.0" encoding="UTF-8"?>
<epml:epml xmlns:epml="http://www.epml.de"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="epml_1_draft.xsd">
  <epc EpcId="1" Name="Beispiel-EPK">
    <event id="1">
    <name>Kundenanfrage ist eingetroffen</name>
    </event>
    <function id="2">
      <name>Kundendaten im Bestandssystem suchen</name>
    </function>
    <xor id="3">
      <name/>
    </xor>
    <event id="4">
      <name>Kunde ist Bestandskunde</name>
    </event>
    <event id="5">
      <name>Kunde ist Neukunde</name>
    </event>
...weitere Funktionen, Ereignisse und Konnektoren zur Vereinfachung weggelassen
    <arc id="11">
      <flow source="1" target="2"/>
    </arc>
... sowie je ein weiteres <arc>-Element für jeden Pfeil im Modell
  </epc>
</epml:epml>
```

Die in dieser Arbeit durchgeführten Analysen an EPK-Modellen wurden an Modellen vorgenommen, die in im EPML-Format vorlagen.

2.5 Formale Semantik

Nachdem im vorangegangenen Abschnitt die Syntax von EPK-Modellen definiert wurde, stellt dieser Abschnitt aus der Literatur bekannte Definitionen ihrer Semantik vor. Eine formale Semantik von EPK-Modellen ist weder in der ursprünglichen Veröffentlichung, mit der EPKs eingeführt wurden [84], noch in anerkannten Standards zu finden. Um EPK-Modelle einer formalen Analyse zugänglich zu machen, wurden verschiedene wissenschaftliche Ansätze entwickelt, die in diesem Abschnitt dargestellt werden.

Durch die Semantikdefinition wird formal festgelegt, welche möglichen Abläufe eines Geschäftsprozesses eine EPK beschreibt.

2.5.1 Zustand und Übergangsrelation

Wie schon in der informellen Beschreibung in Abschnitt 2.1 dargestellt, besteht ein Ablauf eines Geschäftsprozesses in einem EPK-Modell darin, dass Funktionen (Aktivitäten) ausgeführt werden bzw. Ereignisse eintreten. Die zeitliche Abfolge entspricht dabei (in einem einfachen Modell ohne Konnektoren) der Richtung der Pfeile im Modell.

Formal lässt sich der augenblickliche Zustand der Instanz eines Geschäftsprozesses beschreiben, indem die zum aktuellen Zeitpunkt gerade ausgeführten Funktionen bzw. eingetretenen Ereignisse „markiert“ werden. Die Verwendung von AND- und OR-Konnektoren sowie der parallele Start der Abarbeitung bei mehreren möglichen Startereignissen haben zur Folge, dass zu einem Zeitpunkt mehr als ein Knoten einer EPK markiert sein kann.

Aus technischen Gründen und in Einklang mit anderen Autoren (Details siehe [87, 123]) wird in dieser Arbeit der Zustand durch eine Markierung von Kanten statt durch eine Markierung von Knoten beschrieben. Dabei bedeutet die Markierung einer Kante, dass die der Kante unmittelbar vorangehenden Funktionen ausgeführt wurden bzw. die der Kante unmittelbar vorangehenden Ereignisse eingetreten sind. Für die spätere Formalisierung nutzen wir die folgenden Bezeichnungen:

Definition 10 (Eingangs- und Ausgangskanten)
Sei K die Knotenmenge und $A \subset K \times K$ die Kantenmenge einer EPK. Für einen

*Knoten $k \in K$ nennen wir die Menge aller $a \in A$, die die Form $a = (x, k)$ (mit $x \in K$) haben, die Menge der **Eingangskanten** von k. Wir bezeichnen diese Menge mit dem Symbol $\triangleright k$. Analog nennen wir die Menge aller $a \in A$, die die Form $a = (k, x)$ (mit $x \in K$) haben, die Menge der **Ausgangskanten** von k und bezeichnen diese Menge mit $k\triangleleft$.*

Den Zustand einer EPK definieren wir als die Menge ihrer markierten Kanten:

Definition 11 (Zustand einer EPK)
Für eine EPK mit der Kantenmenge A heißt eine Untermenge $Z \subseteq A$ der Zustand der EPK.

Jeder Ablauf eines durch eine EPK modellierten Geschäftsprozesses beginnt damit, dass gewisse Startereignisse der EPK eintreten. Wir definieren einen Startzustand einer EPK wie folgt:

Definition 12 (Startzustand)
Ein Startzustand einer EPK ist ein Zustand, der ausschließlich Kanten enthält, die Ausgangskanten eines Startereignisses sind.

Ein Ablauf eines durch eine EPK modellierten Geschäftsprozesses (im Folgenden werden wir auch kurz vom Ablauf einer EPK sprechen) wird nun durch eine Folge von Zuständen beschrieben, wobei diese Folge mit einem Startzustand beginnen muss. Welche Zustände in einem Ablauf aufeinander folgen dürfen, wird durch eine Übergangsrelation R bestimmt. Die Übergangsrelation R beschreibt somit die formale Semantik von EPK-Modellen. Sie enthält Paare der Form (alter Zustand, neuer Zustand) und legt somit fest, welche Zustandsänderungen erlaubt sind.

Definition 13 (Ablauf einer EPK)
*Ein Ablauf einer EPK ist eine (endliche oder unendliche) Folge von Zuständen $Z_0, Z_1, \ldots$, wobei Z_0 ein Startzustand ist und zwei aufeinanderfolgende Zustände Z_n und Z_{n+1} in einer **Übergangsrelation** R enthalten sind, d.h. für alle $n = 0, 1, \ldots$ gilt $R(Z_n, Z_{n+1})$.*

Eine wünschenswerte Eigenschaft ist es, dass ein solcher Ablauf immer mit einem Zustand enden kann, in dem nur die Eingangskanten von Endereignissen markiert sind. Wir nennen einen solchen Zustand einen **Endzustand**.

Weiterhin definieren wir die Erreichbarkeit eines Zustandes Z_a aus einem Zustand Z_0 wie folgt:

Definition 14 (Erreichbarkeit)
Ein Zustand Z_a ist aus einem Zustand Z_0 erreichbar, wenn es eine Folge von Zuständen $Z_0, Z_1, \ldots, Z_n = Z_a$ gibt, in der jeweils zwei aufeinanderfolgende Zustände Z_n und Z_{n+1} in der Übergangsrelation R enthalten sind, wenn also für alle $i = 0 \ldots n$ gilt: $R(Z_i, Z_{i+1})$.

In den Definitionen 13 und 14 wird noch nichts darüber ausgesagt, wie die Übergangsrelation R definiert ist. Dies soll in den kommenden Abschnitten besprochen werden.

2.5.2 Übergangsrelation für lokal entscheidbare Übergänge

Für die EPK-Elemente Funktion, Ereignis, AND-Join sowie für alle Arten von Splits lässt sich das „Abarbeiten“ dieses Elements (d.h. das Weiterleiten der Markierungen an die im Graph folgenden Elemente im Einklang mit der in Abschnitt 2.1 beschriebenen informellen Semantik) leicht beschreiben:

Ist eine Eingangskante einer Funktion markiert, kann diese Funktion ausgeführt werden. Nach ihrer Ausführung ist die Ausgangskante der Funktion markiert. Analog kann, wenn die Eingangskante eines Zustandes markiert ist, dieser Zustand eintreten. Anschließend kann die Markierung auf die Ausgangskante des betreffenden Zustandes weitergeleitet werden.

Sind *alle* Eingangskanten eines AND-Joins markiert, so kann dieser eine Markierung an seine Ausgangskante weitergeben.

Ist die Eingangskante eines Splits markiert, so hängt das Verhalten vom Typ des Splits ab: Ein XOR-Split leitet die Markierung an genau eine seiner Ausgangskanten weiter, während ein AND-Split eine Markierung an *alle* Ausgangskanten weiterleitet. Ein OR-Split leitet eine Markierung derart an die Ausgangskanten weiter, dass mindestens eine der Ausgangskanten markiert ist.

Voraussetzung für das Weiterleiten einer Markierung auf eine Ausgangskante eines Elements ist jeweils, dass diese Ausgangskante noch nicht markiert ist.

Der bisher beschriebene Teil der Übergangsrelation ist in der folgenden Definition beschrieben, die sich an [123] orientiert.

Wir bezeichnen im Folgenden mit *card* M die Zahl der Elemente einer Menge. $card \triangleright k = 1$ bedeutet also beispielsweise, dass k genau eine Eingangskante hat.

Definition 15 (Übergangsrelation für lokal entscheidbare Elemente)
Sei Z ein Zustand einer EPK Λ. Dann enthält R die folgenden Zustandsübergänge:

- *Für das Abarbeiten einer Funktion, eines Ereignisses oder eines AND-Konnektors:*
  Sei $k \in (F \cup E \cup C_{and})$, $\triangleright k \subseteq Z$ und für alle $x \in k\triangleleft$ gelte $x \notin Z$. Dann ist $(Z, Z') \in R$, wenn $Z' = (Z \setminus \triangleright k) \cup k\triangleleft$.[2]

- *Für das Abarbeiten eines XOR-Splits:*
  Sei $k \in C_{xor}$, $card \triangleright k = 1$, $\triangleright k \subseteq Z$ und für alle $x \in k\triangleleft$ gelte $x \notin Z$. Für alle $a \in k\triangleleft$ gilt dann $(Z, Z') \in R$ falls $Z' = (Z \setminus \triangleright k) \cup \{a\}$.

- *Für das Abarbeiten eines OR-Splits:*
  Sei $k \in C_{or}$, $card \triangleright k = 1$, $\triangleright k \subseteq Z$ und für alle $x \in k\triangleleft$ gelte $x \notin Z$. Für alle nichtleeren $T \subseteq k\triangleleft$ gilt dann $(Z, Z') \in R$ falls $Z' = (Z \setminus \triangleright k) \cup T$.

2.5.3 Probleme mit nichtlokalen Konnektoren

Wie gesehen, ist der „Beitrag“ von Funktionen, Ereignissen, Splits und AND-Joins zur Übergangsrelation leicht zu formalisieren. Demgegenüber bereitet die Formalisierung der Übergänge, die durch die Abarbeitung von OR- und XOR-Joins entstehen, einige Schwierigkeiten [139, 172]. Um zu entscheiden, ob ein Element abgearbeitet werden kann, verlangt die in Abschnitt 2.1 beschriebene informelle Definition der Semantik dieser Elemente nämlich nicht nur eine Überprüfung, ob die entsprechende Eingangskante markiert ist. Stattdessen wird zusätzlich gefordert, dass es nicht möglich sein soll, durch wiederholte Anwendung der Übergangsrelation weitere Eingangskanten zu markieren, bevor der XOR- bzw. OR-Join durchlaufen wird. Die Entscheidung, ob ein XOR- bzw. OR-Join durchlaufen werden kann, hängt also nicht nur vom Zustand seiner lokalen Umgebung (also seiner Eingangskanten) ab. Daher werden diese Konnektoren als nichtlokale Konnektoren bezeichnet.

Die Formulierung, dass es „nicht möglich ist, dass weitere Eingangskanten markiert werden“ ist in dieser Form noch vage und von einer exakten Definition weit entfernt[3]. Das Problem liegt darin, dass in der Definition der Übergangsrelation auf ebendiese

[2] Für $k \in F$ und $k \in E$ (d.h. für Funktionen und Ereignisse) haben $k\triangleleft$ und $\triangleright k$ genau ein Element. Für $k \in C_{and}$ ist k entweder ein AND-Split (dann hat $k\triangleleft$ mehrere Elemente und $\triangleright k$ genau ein Element) oder ein AND-Join (dann hat $k\triangleleft$ genau ein Element und $\triangleright k$ mehrere Elemente).

[3] Übrigens finden sich ähnlich vage Formulierungen auch in den Beschreibungen der Semantik analoger Modellierungselemente in den Standards der BPMN [24] und von UML 2.0 Aktivitätsdiagrammen [125].

zu definierende Übergangsrelation Bezug genommen wird. Der Versuch, in [123], die Übergangsrelation entsprechend der informellen Beschreibung zu formalisieren, führte tatsächlich zu einer Definition mit Zirkelschluss. Diese beschreibt (wie in [172] nachgewiesen) nicht für alle syntaktisch korrekten EPKs eine wohldefinierte Übergangsrelation.

Zur Illustration des Problems wurde in [172] das in Abb. 2.5 gezeigte Modell unter dem Namen „vicious circle“ (Teufelskreis) angeführt:

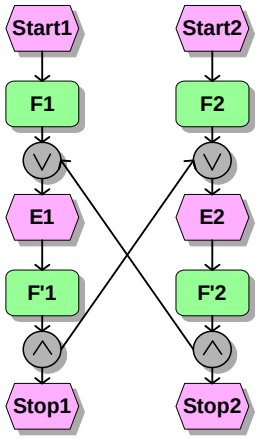


Abbildung 2.5: „Teufelskreis“ aus [172]

Sind beide Ereignisse Start1 und Start2 im Startzustand enthalten, so kann die Frage, ob nach Durchlaufen von F1 bzw. F2 die OR-Joins durchlaufen werden können, nicht zufriedenstellend beantwortet werden. Wie erwähnt, lautet die informelle Forderung: „Ein OR-Join wird durchlaufen, wenn mindestens eine seiner Eingangskanten markiert ist und vor Durchlaufen des OR-Joins keine weiteren Eingangskante markiert werden kann.“ Dadurch soll gewährleistet sein, dass der OR-Join auch tatsächlich wartet, bis *alle* parallel gestarteten Kontrollflüsse, die er abschließen soll, zu ihrem Ende gekommen sind.

Nimmt man nun für das Modell in Abb. 2.5 an, im Anfangszustand könne der linke OR-Join durchlaufen werden. Dann folgt daraus, dass das Ereignis E1 und die Funktion F’1 durchlaufen werden und danach über den AND-Split der linke Eingang des rechten OR-Joins erreicht wird. Das bedeutet aber, dass der rechte OR-Join im Anfangszustand nicht durchlaufen werden kann (da ja später noch ein weiterer Eingang aktiviert wird).

Nimmt man umgekehrt an, im Anfangszustand könne der linke OR-Join nicht

durchlaufen werden. Dann folgt umgekehrt, dass am rechten OR-Join kein weiterer Kontrollfluss ankommt, so dass der rechte OR-Join durchlaufen werden kann.

Wegen der Symmetrie von Abb. 2.5 sind jedoch beide Annahmen und die sich daraus ergebenden Schlussfolgerungen nicht zufriedenstellend.

Eine Folge der Nichtlokalitäts-Eigenschaft ist, dass sich EPKs ohne zusätzliche Einschränkungen nicht auf klassische Petrinetze abbilden lassen, da für diese das Schalten von Transitionen nur von der unmittelbaren (lokalen) Nachbarschaft dieser Transitionen abhängt.

2.5.4 Übergangsrelation für nicht lokal entscheidbare Übergänge

Um die in Abschnitt 2.5.3 beschriebenen Problem zu umgehen, wurden verschiedene Wege vorgeschlagen.

Ein Verzicht auf nichtlokale Konnektoren (wie bei van der Aalst in [169]) umgeht mögliche Probleme, schränkt die Modellierungsmächtigkeit von EPKs jedoch auch deutlich ein.

Weniger einschränkend sind die Ansätze von Chen/Scheer [27] sowie von Langner, Schneider und Wehler [95, 96]. Diese stellen zusätzliche strenge Forderungen an die Struktur der Modelle (u.a. muss es zu jedem öffnenden Split einen typgleichen Join geben, der die vom Split geöffneten Pfade wieder zusammenführt). Für derart eingeschränkte Modelle lässt sich zeigen, dass die Semantik der nichtlokalen Konnektoren widerspruchsfrei definiert werden kann [95, 96]. Zur formalen Definition der Semantik von EPKs nutzen die genannten Arbeiten dann gefärbte Petrinetze, in denen „schwarze“ Marken die tatsächliche Abarbeitung von Modellelementen darstellen, während „weiße“ Marken aussagen, dass der betreffende Pfad nach einer OR- bzw. XOR-Entscheidung gerade nicht durchlaufen wurde.

Solche Ansätze, die von allen Modellen zusätzliche strukturelle Eigenschaften fordern, können gute Dienste leisten, wenn ein neues Modellierungsprojekt begonnen wird. Dann nämlich lassen sich die vorgeschlagenen strengen Syntaxregeln als unternehmensweite Modellierungskonventionen durchsetzen, womit die Modellierer als Nebeneffekt gleich noch zum Erstellen gut strukturierter Modelle angehalten werden. Wenn allerdings (wie wir es als eine der Prämissen in Abschnitt 1.4 angenommen haben) bereits Prozessmodelle existieren, ist die Wahrscheinlichkeit groß, dass diese nicht den sehr restriktiven strukturellen Anforderungen von [27] bzw. [95, 96] folgen. Diese Modelle können dann nicht ohne Zusatzaufwand für eine Transformation weiterverwendet werden.

Kindler und Cuntz [87, 28, 30] kommen in ihrer Definition einer Übergangsrelation ohne zusätzliche Anforderungen an die Struktur von Modellen aus. Sie verwendeten die Idee der Fixpunktsemantik, die in ähnlicher Form bei formalen Semantikdefinitionen von gewissen Konstrukten (z.B. `while`-Schleifen) in Programmiersprachen verwendet wird.

Um Selbstbezüglichkeit in der Definition der Übergangsrelation R zu vermeiden, wird zunächst vorausgesetzt, dass bereits eine (zunächst nicht weiter beschriebene) Übergangsrelation P vorliegt. Die Semantik von XOR- und OR-Joins wird dann so festgelegt, dass die betreffenden Joins die Markierung an die Ausgangskante weitergeben dürfen, wenn nach der als bekannt vorausgesetzten Übergangsrelation P keine weitere Eingangskante markiert werden kann. Def. 15 wird also um die beiden folgenden Punkte erweitert:

Sei Z ein Zustand einer EPK Λ. Dann enthält R die folgenden Zustandsübergänge:

- *Für das Abarbeiten eines XOR-Joins:*
  Sei $k \in C_{xor}$, card $k\triangleleft = 1$. Es gebe genau ein $x_{in} \in \triangleright k$, für das $x_{in} \in Z$ gilt, und für alle $x \in k\triangleleft$ sei $x \notin Z$. Weiterhin gebe es laut Übergangsrelation P keinen von Z aus erreichbaren Zustand Z^* , für den mehr als ein Element von $\triangleright k$ in Z^* enthalten ist. Dann gilt $(Z, Z') \in R$ falls $Z' = (Z \setminus \triangleright k) \cup k\triangleleft$.

- *Für das Abarbeiten eines OR-Joins:*
  Sei $k \in C_{or}$, card $k\triangleleft = 1$. Für eine nichtleere Teilmenge $I \subseteq \triangleright k$ gelte $I \subseteq Z$, und für alle $x \in k\triangleleft$ sei $x \notin Z$. Weiterhin gebe es laut Übergangsrelation P keinen von Z aus erreichbaren Zustand Z^* , für den für eine Menge J mit $I \subset J \subseteq \triangleright k$ Teilmenge von Z^* ist. Dann gilt $(Z, Z') \in R$ falls $Z' = (Z \setminus \triangleright k) \cup k\triangleleft$.

Die Tatsache, dass die Übergangsrelation R von der als feststehend vorausgesetzten Übergangsrelation P abhängig ist, soll im Folgenden durch die Verwendung des Symbols $R(P)$ für die Übergangsrelation R ausgedrückt werden.

Gibt es ein P mit $R(P) = P$, so erfüllt die Übergangsrelation P offenbar gerade die Anforderungen, die in Abschnitt 2.1 als informelle Semantik einer EPK beschrieben wurden. In [87, 28, 30] wird in diesem Falle von einer idealen Semantik gesprochen. Es wurde in [87] gezeigt, dass sich eine ideale Semantik, falls sie existiert, durch eine Fixpunktiteration berechnen lässt. Diese wird gestartet mit einer Übergangsrelation $P_0^{pess} = \emptyset$ (in dieser Übergangsrelation darf sich also nie eine Markierung ändern), sowie einer Übergangsrelation $P_0^{opt} = Z \times Z$ (in dieser Übergangsrelation kann jede Markierung von jeder gefolgt werden). Im Folgenden wird dann iterativ berechnet:

$P_1^{pess} = R(R(P_0^{pess}))$, $P_2^{pess} = R(R(P_1^{pess}))$, ...
sowie
$P_1^{opt} = R(R(P_0^{opt}))$, $P_2^{opt} = R(R(P_1^{opt}))$, ...

In [87] wird mit Hilfe des Fixpunktsatzes von Kleene gezeigt, dass beide Iterationen gegen jeweils einen Fixpunkt konvergieren, d.h. es gibt eine Zahl n, für die $P_{n+1}^{opt} = R(R(P_n^{opt}))$ und $P_{n+1}^{pess} = R(R(P_n^{pess}))$. Diese Fixpunkte werden als optimistische Semantik P^{opt} und pessimistische Semantik P^{pess} bezeichnet. Falls $P^{opt} = P^{pess}$, hat die EPK eine ideale Semantik. EPKs mit idealer Semantik sind gerade die, für die es keine durch Selbstbezüglichkeit von nichtlokalen Konnektoren hervorgerufene Widersprüche wie im Beispiel von Abb. 2.5 gibt. In der Praxis zeigt sich, dass die übergroße Mehrzahl der EPKs eine ideale Semantik hat.

Bei der Definition der Semantik von OR-Joins in der Sprache YAWL wurde zunächst einfacher vorgegangen [175]: Als Relation P wurde der „nichtlokale Teil" der Übergangsrelation angesehen, d.h. es wurde für P angenommen, dass OR-Joins nie eine Markierung weiterleiten können[4]. Als Semantik der OR-Joins wurde dann $R(P)$ wie oben beschrieben definiert. Da diese Definition jedoch Nachteile mit sich bringt, (Details siehe [196]), wurde in [196] eine neue Semantikdefinition vorgestellt. Hier wurde P so gewählt, dass OR-Joins wie XOR-Joins mit lokaler Semantik behandelt werden. Diese lokale Semantik von XOR-Joins nimmt an, dass ein XOR-Join stets durchlaufen wird, sobald eine seiner Eingangskanten markiert ist, die Ausgangskante jedoch nicht.

Der Beitrag von XOR-Joins zur Übergangsrelation R lässt sich dann wie folgt beschreiben, ohne auf eine weitere Übergangsrelation P Bezug zu nehmen:

Definition 16 (XOR-Join mit lokaler Semantik)

- *Abarbeiten eines XOR-Joins (lokale Semantik):*
  Mit den in Def. 15 eingeführten Bezeichnungen sei $k \in C_{xor}$, card $k^\triangleleft = 1$. Gibt es dann ein $x \in {}^\triangleright k$ mit $x \in Z$ und gilt für alle $x \in k^\triangleleft$ zusätzlich $x \notin Z$, so ist $(Z, Z') \in R$, wenn $Z' = (Z \setminus \{x\}) \cup k^\triangleleft$.

Inspiriert von den Semantikdefinitionen mit farbigen Petrinetzen von Langner, Schneider und Wehler schlägt Mendling in [119, 113] eine weitere EPK-Semantik vor, in der die Markierung eines EPK-Elements aus einem Paar (s, c) besteht. Dabei sagt s (der Status) aus, ob ein Knoten gerade durchlaufen wird bzw. ob an einem

[4] XOR-Joins haben in YAWL die weiter unten in Def. 16 beschriebene lokale Semantik.

vorangegangenen OR-Split die Entscheidung gefällt wurde, den Zweig, in dem der Knoten liegt, nicht zu durchlaufen. c (der Kontext) soll die Information abbilden, ob es möglich ist, dass an diesem Knoten weitere s-Markierungen eintreffen. Die Übergangsrelation beschreibt dann, wie sich der Inhalt der Markierungen abhängig vom aktuellen Zustand der EPK ändern kann. Damit erhält jedes EPK-Modell eine Semantik. Das gilt auch für solche Modelle, die nach [87] keine ideale Semantik besitzen wie den Teufelskreis aus Abb. 2.5.

In Abschnitt 11.3 wird das Ergebnis von Auswertungen der Soundness-Eigenschaft zahlreicher Modelle unter Zugrundelegung einer Fixpunktsemantik [87], der YAWL-Semantik [196] und Mendlings Semantik [119, 113] vorgestellt, wodurch einige Vor- und Nachteile der verschiedenen Semantiken deutlich werden.

3 Qualitätskriterien für Geschäftsprozessmodelle

Im einführenden Abschnitt 1.1.1 wurde bereits deutlich gemacht, dass das Modellieren von Geschäftsprozessen in den meisten Fällen die *Kommunikation zwischen Personen* unterstützen soll. Erstellen und Lesen von GPM bedeutet also im Grunde nichts anderes als in einer bestimmten (Modellierungs)-Sprache miteinander zu kommunizieren.

Bei der Aufstellung von Qualitätskriterien für Geschäftsprozessmodelle folgen wir Lindland, Sindre und Sølvberg, die in [101] schrieben, dass „Modellieren im Wesentlichen bedeutet, Aussagen in einer Sprache zu formulieren“. Als Folge dessen nutzten die Autoren in der Sprachwissenschaft eingeführte Theorien, um Anforderungen an die Qualität von Modellen zu beschreiben.

Sie unterteilen die Anforderungen in die drei Kategorien:

- *Syntax* – die Regeln, nach denen aus einem gegebenen Vorrat an Zeichen erlaubte („wohlgeformte“) Modelle erstellt werden,
- *Semantik* – die Bedeutung eines Modells (worunter wir die möglichen Abläufe des Modells verstehen wollen),
- *Pragmatik* – die Interpretation eines Modells durch dessen Leser

Für diese drei Kategorien werden im Folgenden Qualitätsanforderungen untersucht. Darauf aufbauend werden in späteren Kapiteln dann die derzeit bekannten Verfahren zur Analyse und Verbesserung der Modellqualität in den drei Kategorien vorgestellt sowie Verbesserungsmöglichkeiten aufgezeigt.

3.1 Syntaktische Qualität

Die grundlegendste Forderung, die an ein Modell gestellt werden muss, ist, dass es sich tatsächlich um ein syntaktisch korrektes Modell der jeweiligen Modellierungssprache handelt.

Voraussetzung dafür ist, dass die Syntax der Modellierungssprache formal definiert ist. Für die in dieser Arbeit betrachtete Sprache EPK leistet dies die in Abschnitt 2.2 eingeführte Definition 1. In der Praxis allerdings wird eine Modellierungsmethode in den seltensten Fällen mittels abstrakter Definitionen vermittelt. Das hat zur Folge, dass in der betrieblichen Praxis nicht selten Modelle anzutreffen sind, die tatsächlich keine EPK-Modelle (nach Definition 1) sind. Eine häufige Abweichung sind z.B. Funktionen oder Ereignisse mit mehreren ein- oder ausgehenden Kanten. Diese werden sowohl in der ursprünglichen Beschreibung der EPK-Modellierung [84] als auch in gängigen Lehrbüchern [189, 85] oder wissenschaftlichen Veröffentlichungen [97, 139, 169, 123] nicht zugelassen. Eine Syntaxüberprüfung sollte also solche Modelle als syntaktisch unkorrekt zurückweisen.

3.2 Semantische Qualität

Der Begriff der „semantischen Eigenschaften eines GPM“ wird nicht einheitlich genutzt [146]. Wir verstehen im Folgenden unter den semantischen Eigenschaften eines Modells die Eigenschaften seiner möglichen Abläufe.

Dies unterscheidet sich klar von dem, was im Modellierungswerkzeug *ARIS Toolset* der Firma IDS Scheer als Funktion „Semantikcheck“ für EPK-Modelle implementiert ist: Diese Funktion prüft Modelleigenschaften wie „Alle Funktionen und Ereignisse haben nur eine ein- bzw. ausgehende Kante“, die im Rahmen dieser Arbeit den syntaktischen Eigenschaften eines Modells hinzugerechnet werden.

Ebenso unterscheidet sich die hier benutzte Bedeutung des Begriffs einer semantischen Eigenschaft von ontologiebasierten Ansätzen wie z.B. [18], wo unter der Semantik einer EPK die Bedeutung ihrer Elemente und Abläufe in der realen Anwendungsdomäne verstanden wird.

Die in dieser Arbeit verwendete Interpretation des Begriffs „Semantik“ abstrahiert von der Bedeutung, die Funktionen und Ereignisse in der betrieblichen Realität haben. Damit stimmt die Verwendung der Bezeichnung „Semantik“ in dieser Arbeit mit dem überein, was das UML 2 Semantics Project [20] formulierte: Die Semantik wird dargestellt durch ein mathematisches Modell, durch das alle Abläufe eines Modells exakt beschrieben werden können.

3.2.1 Soundness

Das in der wissenschaftlichen Literatur am häufigsten verwendete Korrektheitskriterium für GPM ist das der **Soundness**. Dieses Kriterium wurde ursprünglich von van

der Aalst für Workflow-Netze [167, 166] eingeführt und später auf andere Modellierungssprachen übertragen. Entsprechende Definitionen existieren unter anderem für EPKs [169, 113], den Pi-Kalkül [133], BPMN [129, 36] und YAWL [195].

Die ursprüngliche Definition des Begriffes Soundness aus [167] verlangt von einem modellierten Geschäftsprozess die drei folgenden Eigenschaften:

1. Jeder gestartete Geschäftsprozess kann nach endlich langer Zeit zum Ende gelangen.

2. Wenn der Geschäftsprozess zu seinem Ende gelangt ist, gibt es keine verbleibenden Aktivitäten, die noch in Ausführung sind bzw. auf ihre Ausführung warten.

3. Für jeden Bestandteil des Modells gibt es einen Ablauf des Geschäftsprozesses, in dem dieser Bestandteil durchlaufen wird.

Wie bereits erwähnt, bezieht sich diese ursprüngliche Definition auf Workflow-Netze. Dies ist eine spezielle Form von Petrinetzen, nämlich Stellen-/Transitions-Netze mit der Eigenschaft, dass es genau eine ausgezeichnete Stelle i gibt, deren Vorgängermenge leer ist (Quelle) und eine weitere ausgezeichnete Stelle o, deren Nachfolgermenge leer ist (Senke). Als Anfangsmarkierung eines Workflow-Netzes wird die Markierung bezeichnet, bei der lediglich auf i eine Marke liegt. Dann lassen sich die drei oben genannten Eigenschaften wie folgt formal definieren [167]:

1. Für jeden Zustand des Workflow-Netzes, der von der Anfangsmarkierung aus erreichbar ist, existiert eine Schaltfolge zu einem Zustand, in dem o markiert ist.

2. Es gibt ausgehend von der Anfangsmarkierung keine Schaltfolge zu einem Zustand, in dem sowohl o als auch noch weitere Stellen eine Marke enthalten.

3. Das Workflow-Netz enthält keine toten Transitionen, d.h. für jede im Netz enthaltene Transition gibt es eine Folgemarkierung der Anfangsmarkierung, unter der diese Transition schalten kann.

Um diese Definition auf EPKs übertragen zu können, muss man der Tatsache Rechnung tragen, dass es für ein EPK-Modell anders als im Workflow-Netz im Allgemeinen mehrere denkbare Startzustände und analog auch mehrere Endzustände (d.h. Zustände, in denen nur Endereignisse markiert sind) geben kann. Van der Aalst führt in [169] aus diesem Grund den Begriff der regulären EPK ein. Diese erhält man

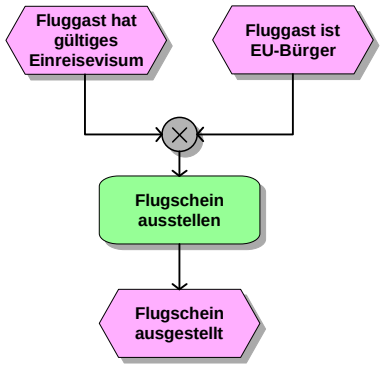


Abbildung 3.1: Nicht alle Kombinationen von Startereignissen sind erlaubt

aus einer beliebigen gegebenen EPK Λ, indem man ein künstliches Startereignis i und ein künstliches Endereignis o zur Menge der Ereignisse von Λ hinzunimmt. Wird dann i über einen OR-Split mit allen Startereignissen von Λ verbunden und werden weiterhin alle Endereignisse von Λ über einen OR-Join mit o verbunden, so erhält man die zu Λ zugehörige reguläre EPK, eine EPK mit genau einem Start- und einem Endereignis.

Die Betrachtung einer regulären EPK würde aber dazu führen, dass ein fehlerfreier Ablauf der EPK für jede denkbare Kombination von Startereignissen zu fordern wäre. Das ist aber in der Praxis in der Regel nicht erwünscht. Als Illustration hierzu soll die EPK von Abbildung 3.1 dienen. In dieser Abbildung ist stark vereinfacht ein Ablauf an einem Schalter eines Luftfahrtunternehmens modelliert. Eine Geschäftsregel besagt, dass ein Fluggast nur dann den Flugschein ausgehändigt bekommt, wenn er ein gültiges Einreisevisum für das Zielland vorweist oder EU-Bürger ist. Da es nach den Ausführungen von Abschnitt 2.5 als Fehler betrachtet wird, wenn mehrere Eingangskanten eines XOR-Joins markiert sind, bietet dieses Modell die theoretische Möglichkeit, dass bei gleichzeitigem Eintreten beider Startereignisse ein Fehlerzustand eintritt. Die Intention des XOR-Joins im Modell ist jedoch gerade, deutlich zu machen, dass sich die beiden Startereignisse ausschließen, so dass dieser Fall nicht betrachtet werden muss. Bei einer Umformung in eine reguläre EPK geht diese Information verloren.

Es reicht also, wenn es überhaupt mindestens einen Startzustand (also eine Kombinationen von Startereignissen, die gemeinsam auftreten können) gibt, der einen korrekten Ablauf garantiert. Welche Kombinationen von Startereignissen zulässig sind, wird in der Praxis im Modell meist nicht explizit angegeben, kann aber oft aus dem Sachzusammenhang erschlossen werden. Die etwa von Rump [145] aufgestellte

Forderung, die Menge der erlaubten Startzustände ausdrücklich zu benennen (also diese Information Teil des Modells werden zu lassen) hat sich in der Praxis nicht durchgesetzt.

Bezeichnen wir mit A_s diejenigen Kanten einer EPK Λ, die Ausgangskanten eines Startereignisses sind, so ist die Menge aller möglicher Startzustände $\wp(A_s) \setminus \emptyset$.

Unter Beibehaltung dieser Bezeichnungen definieren wir so wie von Mendling in [113] eingeführt:

Definition 17 (Soundness)
Eine EPK Λ heißt sound, wenn es eine nichtleere Teilmenge von Startzuständen $I \subseteq \wp(A_s) \setminus \emptyset$ gibt, so dass gilt:

1. *Jede Kante $x \in A_s$ ist in einem der Startzustände aus I enthalten.*
2. *Von jeder Markierung aus, die von einem Startzustand $i \in I$ aus erreichbar ist, ist ein Endzustand erreichbar.*
3. *Ist von einem Startzustand $i \in I$ aus ein Zustand z erreichbar, von dem aus laut Übergangsrelation keine weiteren Zustände erreichbar sind, so muss z ein Endzustand sein.*

Forderung (1) besagt, dass es im Modell keine „unnötigen“ Startzustände (also solche, die nicht zu einem korrekten Ablauf beitragen können) gibt. Forderung (2) besagt, dass man von jedem in I enthaltenen Startzustand einen Endzustand erreichen kann. Forderung (3) schreibt fest, dass ein Zustand, von dem aus keine weiteren Zustände erreichbar sind, immer ein Endzustand sein muss. Durch diese Forderung wird der Fall ausgeschlossen, dass von einem Zustand aus zwar keine weiteren Zustände mehr erreicht werden können, aber noch ein AND-Join auf die Markierung aller Eingangskanten wartet. Solche Fehler werden als **Deadlock** bezeichnet. Formal kann man einen Deadlock wie folgt definieren:

Definition 18 (Deadlock)
Ein Zustand z einer EPK heißt Deadlock, falls es entsprechend der Übergangsrelation R kein z' mit $(z, z') \in R$ gibt und außerdem in z mindestens eine Eingangskante eines AND-Joins enthalten ist.

Ein Deadlock beschreibt somit die Situation, dass an einem AND-Join zu wenige markierte Eingangskanten vorhanden sind. Die umgekehrte Situation, dass an einem XOR-Join zu viele (also mehr als eine) Eingangskanten markiert werden können, wird

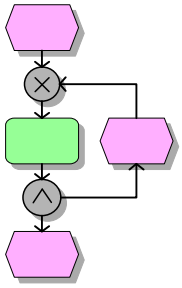


Abbildung 3.2: Mehrfache Beendigung (Endlosschleife)

von Sadiq und Orlowska [147] als **Synchronisationsfehler** *(Lack of Synchronisation)* bezeichnet. Bei dieser Interpretation wird davon ausgegangen, dass ein XOR-Join, bei dem mehr als eine Eingangskante markiert ist „blockiert". Es kann dann keine Markierung an die Ausgangskante weitergegeben werden.

Eine andere Betrachtungsweise besteht darin, dass die Definition eines Zustands derart erweitert wird, dass auch mehrfach markierte Kanten erlaubt sind. Nach dieser Betrachtungsweise kann ein XOR-Join mit mehreren markierten Eingangskanten von einem Zustand gefolgt werden, in dem seine Ausgangskante mehrfach markiert ist. In der weiteren Folge des Ablaufs können schließlich Eingangskanten von Endereignissen mehrfach markiert werden. Diese Klasse von Fehlern wird daher als **mehrfache Beendigung** (*Multiple Termination*) bezeichnet [37]. „Mehrfache Beendigung" findet sich auch in Fällen wie dem in Abb. 3.2 gezeigt. Hier führt ein AND-Split als Zyklenausstiegsknoten dazu, dass der Ablauf nie endet und die Eingangskante eines Endereignisses unendlich viele Markierungen enthält.

Während also die Bezeichnung „Deadlock" Fehler beschreibt, bei denen nicht genug Eingangskanten eines AND-Joins markiert werden können, steht „Synchronisationsfehler" dafür, dass zu viele Eingangskanten eines XOR-Joins eine Markierung erhalten können. „Mehrfache Beendigung" liegt vor, wenn ein Endereignis mehrfach erreicht werden kann. Wir fassen die genannten Fehlerarten unter dem Überbegriff **Kontrollflussfehler** zusammen.

Ein korrektes Modell muss frei von Kontrollflussfehlern sein. Ist ein Modell sound, so folgt, dass es frei von diesen Fehlern ist.

Da Soundness eine sehr starke Forderung ist, schlugen verschiedene Autoren Abschwächungen vor. Die schwächste Forderung ist die der Relaxed Soundness [34], bei der Deadlocks und Synchronisationsfehler erlaubt sind, solange es nur überhaupt einen Ablauf gibt, der zu einem Endzustand führt. Dies erlaubt z.B. Modellfragmente

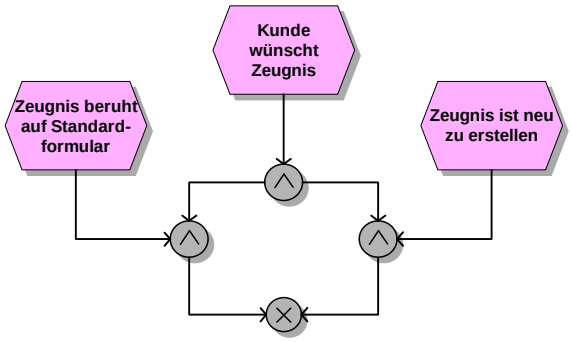


Abbildung 3.3: Modell ist relaxed sound trotz Deadlock

wie das aus Abb. 3.3[1], bei dem der Autor offenkundig davon ausgeht, dass genau eines der beiden Startereignisse „Zeugnis beruht auf Standardformular“ und „Zeugnis ist neu zu erstellen“ eintritt. Der Deadlock am nicht eintretenden Startereignis wird als unproblematisch angesehen.

Martens [104] definierte Weak Soundness für Workflow-Netze. Seine Definition umfasst die beiden ersten Punkte der o.g. Soundness-Definition von van der Aalst [167]. Verzichtet wird also auf die Forderung, dass es keine toten Transitionen im Workflow-Netz gibt.

Puhlmann und Weske [134] verzichten dagegen bei dem von ihnen definierten Begriff der Lazy Soundness auf die zweite Forderung. Lazy Soundness erlaubt also, dass gleichzeitig mit der Senke im Workflow-Netz weitere Stellen markiert sind. Als sinnvollen Anwendungsfall führen Puhlmann und Weske einen Prozess an, in dem drei Experten um ein Gutachten gebeten werden und der Prozess fortgeführt und schließlich beendet werden kann, falls zwei Gutachten eingetroffen sind. Das Warten auf das dritte Gutachten ist somit für die korrekte Abarbeitung des Prozesses ohne Belang.

3.3 Pragmatische Qualität (Lesbarkeit und Verständlichkeit)

In der kognitionswissenschaftlichen Forschung, speziell der Cognitive Load Theory, konnte belegt werden, dass Betrachter komplexer graphischer Modelle die Aussage solcher Modelle erschließen, indem kleinere (lokale) Modellabschnitte als eine Einheit

[1] entnommen der EPK *Qualitätsmanagement im Vertrieb - Zeugniserstellung* des SAP R/3-Referenzmodells

	Soundness [169]	Weak Soundness [104]	Relaxed Soundness [34]	Lazy Soundness [134]
Keine Deadlocks	√	√	muss nur für einen Ablauf gelten	√
Ist kein neuer Zustand erreichbar, sind nur Eingangskanten von Endereignissen markiert.	√	√	muss nur für einen Ablauf gelten	
keine „toten Teile“ im Modell	√		√	√

Tabelle 3.1: Abgeschwächte Soundness-Eigenschaften

wahrgenommen werden und in der Folge Zusammenhänge zwischen solchen Modellabschnitten hergestellt werden [159, 40, 122].

Daraus ergeben sich zwei Forderungen: Zum einen soll eine einfache gedankliche Unterteilung des GPM in solche Teileinheiten möglich sein. Zum anderen sollen die Teilmodelle selbst möglichst leicht verständlich sein.

Die erste Forderung führt zu dem Wunsch, dass das Modell zu großen Teilen aus dem Leser von der Struktur her bekannten Teilmodellen besteht, die gedanklich klar voneinander getrennt werden können. Dies führt zu dem Begriff der Wohlstrukturiertheit, der zunächst in Abschnitt 3.3.1 informell vorgestellt und später in Abschnitt 7.1 formal definiert wird.

Aus der zweiten Forderung ergibt sich der Wunsch nach Teilmodellen, die mit möglichst leicht verständlichen Notationselementen auskommen.

3.3.1 Wohlstrukturiertheit

In der Softwareentwicklung herrscht seit langer Zeit Übereinstimmung darüber, dass strukturierte Programme deutlich besser verständlich und wartbar sind als unstrukturierter Code mit zahlreichen `GOTO`-Anweisungen. Es wurde gezeigt, dass sog. „Spaghetti-Programme“ schwerer zu verstehen und schwerer zu warten sind als Programme mit strukturierten Kontrollblöcken (wie z.B. `while-do`) [142, 55].

Bei der Modellierung von GPM sind dagegen auch heute noch Verzweigungen zwischen verschiedenen Modellteilen anzutreffen, die an `GOTO`-lastige Programme aus der Frühzeit der Softwareentwicklung erinnern.

Holl schreibt in [72], dass „der vorherrschende unstrukturierte Stil der Geschäftsprozessmodellierung, den wir Spaghetti-GPM nennen können, zu ähnlichen Problemen führt wie Spaghetti-Programmierung."

Die Aussage von Holl ist, dass die Lesbarkeit von GPM verbessert werden kann, wenn die Modelle weitgehend wohlstrukturiert erstellt werden. Das bedeutet, dass von einem Split abgehende Pfade später von einem Join gleichen Typs zusammengeführt werden und dass die so gebildeten Kontrollflussblöcke sauber verschachtelt sind. Eine formelle Definition des Begriffs der Wohlstrukturiertheit von EPKs wird in Abschnitt 7.1 gegeben.

Van der Aalst empfiehlt in [169] für die Erstellung von GPM: „Wenn möglich, sollten nicht wohlstrukturierte Konstrukte vermieden werden. Ist ein solches Konstrukt tatsächlich notwendig, dann sollte seine Korrektheit nochmals überprüft werden, da es eine potentielle Fehlerquelle darstellt."

Ebenso gibt es jedoch Erfahrungen aus der Software-Entwicklung, nach denen ein gelegentliches wohlüberlegtes Abweichen von strenger Strukturierung (z.B. das vorzeitige Verlassen von Schleifen mittels des `break`-Befehls) die Qualität und Lesbarkeit von Programmen erhöhen kann [141]. Für die Übertragung auf GPM heißt das, dass mehr zu tun ist, als lediglich generell Wohlstrukturiertheit von Modellen zu fordern. Statt dessen ist es sinnvoll, verschiedene Muster, in denen die Wohlstrukturiertheits-Eigenschaft verletzt wird, situationsbedingt zu bewerten.

3.3.2 Wahl des inhaltlich passendsten Notationselements

Oft gibt es verschiedene Möglichkeiten, einen Sachverhalt im Modell auszudrücken. Wenn möglich, sollte diejenige bevorzugt werden, von der zu erwarten ist, dass sie vom Leser am besten verstanden werden kann. Insbesondere sollte, falls die verwendete Modellierungssprache Alternativen zulässt, immer das Notationselement gewählt werden, das das Verhalten des Modells am genauesten beschreibt. Die EPK-Modellierungssprache lässt hier (im Gegensatz etwa zu umfangreicheren Modellierungssprachen wie BPMN oder UML Aktivitätsdiagrammen) auf Grund ihres sehr geringen Notationsumfanges nur wenig Spielraum. Es gibt nur einen denkbaren Fall, in dem ein Notationselement unter Beibehaltung der Semantik des Modells durch ein anderes, die Intention des Modells besser beschreibendes, ersetzt werden kann: die Ersetzung von OR-Konnektoren durch XOR- oder AND-Konnektoren. Ist eine solche

Ersetzung möglich, so sollte sie auch durchgeführt werden: Wie Sarshar und Loos in einem Laborexperiment [149] zeigten, werden OR-Joins in einem Modell schlechter verstanden als XOR- oder AND-Joins.

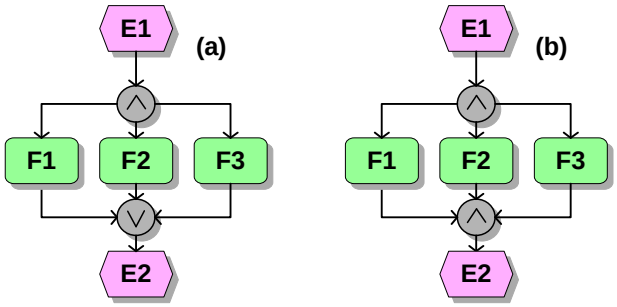


Abbildung 3.4: Ersetzen des OR-Joins durch einen AND-Join

Abb. 3.4 (a) zeigt ein Modell, in dem durch den AND-Split drei Abläufe parallel gestartet werden. Am OR-Join wird darauf gewartet, dass alle Eingangskanten, an denen eine Markierung möglich ist, markiert werden. Im speziellen Falle sind dies alle drei Eingangskanten. Der OR-Join leitet also nur dann eine Markierung an seine Ausgangskante weiter, wenn alle Eingangskanten markiert sind, und verhält sich somit wie ein AND-Join. Folglich sollte eine solche Situation auch durch einen AND-Join modelliert werden, wie in Abb. 3.4 (b) gezeigt. Dem Leser des Modells hilft dies beim schnellen Verstehen des Modells. Um etwa zu erkennen, dass vor Eintreten des Ereignisses E2 *alle* Funktionen F1-3 ausgeführt werden müssen, reicht es, die lokale Umgebung des Ereignisses zu betrachten. In Abb. 3.4 (a) dagegen ist dies nur durch Betrachtung des gesamten Modells erkennbar. Bei so kleinen Modellen wie in Abb. 3.4(a) mag sich daraus noch keine Schwierigkeit ergeben, bei größeren und komplexeren Modellen kann aber die Lesbarkeit durchaus leiden.

Weitere Fälle, in denen ein OR-Konnektor durch einen anderen Konnektor ersetzt werden sollte, werden später in Abschnitt 8.2.6 vorgestellt.

4 Verfügbare Techniken zur Überprüfung der Qualität von Geschäftsprozessmodellen

Nachdem im vorangegangenen Kapitel Qualitätskriterien für GPM beschrieben wurden, gibt dieses Kapitel eine Übersicht über die in der Literatur vorgeschlagenen Techniken, die eine Analyse der Qualität von GPM gestatten. Wie bereits bei der Beschreibung von Qualitätskriterien im vorherigen Kapitel, werden wieder Techniken zur Prüfung der syntaktischen, der semantischen und der pragmatischen Qualität unterschieden.

4.1 Techniken zur Prüfung der syntaktischen Qualität

Unter den im vorangegangenen Kapitel genannten Qualitätseigenschaften von Modellen ist die syntaktische Korrektheit die am leichtesten zu prüfende Eigenschaft.

Der am häufigsten gebräuchliche Ansatz zur Gewährleistung der syntaktischen Korrektheit eines Modells besteht darin, die im Modell erlaubten Konstrukte in einem Metamodell zu beschreiben. Dann kann leicht geprüft werden, ob ein betrachtetes Modell eine Instanz dieses Metamodells ist. Tatsächlich wurden für die EPK-Sprache formale Metamodelle publiziert. Rosemann [143] nutzte hierzu Entity-Relationship-Diagramme, Fillies und Weichhardt [51] die Metamodellierungssprache des Werkzeuges *Semtalk*, Kornherr und List [91] das Metamodell der UML. Kern und Kühne [86] schlugen ein im Eclipse Modeling Framework beschriebenes Metamodell vor. Allerdings enthalten diese Metamodelle nur Aussagen über erlaubte Modellelemente, ihre Multiplizität sowie Beziehungen zueinander. Damit lassen sich die Forderungen 2-6 sowie 8-11 aus Def. 1 problemlos beschreiben und für ein gegebenes Modell automatisch prüfen. Nicht geeignet sind die genannten Metamodelle jedoch, um Forderungen wie Eigenschaft 1 (Zusammenhang des Graphen), 7 (Verbot eines Zyklus aus Konnektoren) sowie 12 und 13 (Existenz eines Pfades vom Startereignis zum

Modellelement bzw. vom Modellelement zum Endereignis) zu definieren bzw. die Einhaltung dieser Forderungen zu überprüfen.

Eine Überprüfung aller syntaktischen Forderungen ist jedoch sehr wünschenswert. So sollte beispielsweise, wenn Modelle zwischen verschiedenen Werkzeugen und möglicherweise sogar zwischen verschiedenen Organisationen ausgetauscht werden, vor dem Import des Modells in das Ziel-Werkzeug eine Überprüfung auf syntaktische Korrektheit erfolgen.

Mendling und Nüttgens befassten sich in [115] mit der Frage, wie die in Def. 1 aufgestellten Syntaxanforderungen für ein in der Austauschsprache EPML vorliegendes EPK-Modell[1] überprüft werden können. Sie untersuchten hierfür verschiedene XML-Schemavalidierungssprachen. Mendling und Nüttgens stellten fest, dass die Ansätze *XML Schema* des World Wide Web Consortiums sowie *Relax NG* für diesen Zweck ungeeignet sind, da nur die einfachsten der in Def. 1 aufgestellten Forderungen überprüft werden können. Mendling und Nüttgens schlugen die Verwendung der Sprache Schematron [78] vor. Damit kann die Mehrzahl der in Def. 1 aufgestellten Syntaxanforderungen validiert werden. Diese Aussage gilt auf Grund konzeptueller Gemeinsamkeiten mit Schematron auch für die Simple Guideline Specification Language (SGSL) [160], die nicht in die Betrachtungen von [115] einbezogen war.

[115] gibt an, dass sich mit Schematron keine Validierungsregeln für diejenigen Forderungen aus Def. 1 aufstellen lassen, deren Überprüfung ein Durchlaufen des Graphen des Modells erfordert, also die Forderungen 1, 7, 12 und 13[2].

Neben dieser Einschränkung hat das in [115] vorgestellte Verfahren zur Prüfung der syntaktischen Korrektheit einen weiteren deutlichen Nachteil: Zu den Elementen der EPML-Datei müssen zusätzliche Attribute notiert werden, die Aussagen über die Knoten der EPK treffen. Beispielsweise muss der Beginn der in Abb. 2.1 dargestellten EPK statt

```
<event id="1">
<name>Kundenanfrage ist eingetroffen</name>
</event>
<arc id="11">
  <flow source="1" target="2"/>
</arc>
```

[1] genauer formuliert: für ein Konstrukt, für das zu untersuchen ist, ob es sich um ein EPK-Modell handelt

[2] Diese Aussage wurde in [115] ohne Beweis angeführt. [99] beschreibt, wie bei Benutzung komplexer XPath-Ausdrücke auch eine Prüfung auf Zyklen möglich ist, was der genannten Aussage zumindest in einem Punkt widerspricht.

folgendermaßen notiert werden:

```
<event id="1" type="EventStart">
<name>Kundenanfrage ist eingetroffen</name>
</event>
<arc id="11" type="EventFunctionArc">
  <flow source="1" target="2"/>
</arc>
```

Damit wird angegeben, dass es sich beim Ereignis „Kundenanfrage ist eingetroffen“ um ein Startereignis handelt und die gerichtete Kante zwischen diesem Startereignis (mit der id=1) und der Funktion mit der id=2 den Typ „Kante von einem Ereignis zu einer Funktion“ hat [114].

Diese zusätzlichen Attribute sind im XML-Schema der Sprache EPML seit der Anfangsversion 1.0 vorhanden. Ihre Nutzung hat allerdings zwei Nachteile: Zum einen werden die EPML-Dokumente größer und komplexer. Zum anderen sind die Informationen, die aus diesen Attributen gewonnen werden können, ausnahmslos *redundant*. Die kurze Version der EPML-Datei (ohne zusätzliche Attribute) enthält nämlich bereits alle Informationen, um beispielsweise festzustellen, dass es sich beim Ereignis „Kundenanfrage ist eingetroffen“ um ein Startereignis handelt. Da eine Austauschsprache möglichst einfach und ohne redundante Informationen sein sollte, ist ein Verzicht auf die zusätzlichen Attribute wünschenswert.

Hinzu kommt, dass bei der in [114] vorgeschlagenen Nutzung zusätzlicher `type`-Attribute für die Elemente der EPML-Datei keineswegs auf das Durchlaufen des Graphen verzichtet werden kann. Dieses muss zwar dank der `type`-Attribute nicht mehr bei der Validierung der Syntaxregeln erfolgen, dafür aber beim Erzeugen der EPML-Datei, da schließlich zu diesem Zeitpunkt bestimmt werden muss, wie die `type`-Attribute zu belegen sind. Sollten beim Erzeugen der EPML-Datei gar falsche `type`-Attribute gesetzt werden, wird dies mit dem in [114] vorgeschlagenen Ansatz nicht erkannt – gerade das wäre aber Aufgabe einer Eingabevalidierung.

Neben den bereits genannten Lösungsansätzen, die bereits in [115] hinsichtlich ihrer Eignung zur Validierung von EPML-Dateien bewertet wurden, sind zwei weitere Ansätze erwähnenswert, die mit dem Ziel entwickelt wurden, komplexere Validierungen von XML-Dateien durchführen zu können: die Sprache Incox [127] und die Constraint Language in XML (CLiXML) [82]. Beide sind prinzipiell zur Validierung sämtlicher Forderungen aus Def. 1 geeignet.

Ebenso prinzipiell geeignet ist die Object Constraint Language (OCL) [188], mit der einschränkende Forderungen direkt im Metamodell notiert werden können.

Allen bisher in diesem Abschnitt besprochenen Ansätzen zur syntaktischen Validierung ist gemein, dass sie eine spezielle (meist auf die Eigenschaften des XML-Formats zugeschnittene) Sprache benutzen. Die syntaktische Validierung erfolgt somit in einer anderen Sprache als eine Überprüfung der semantischen oder pragmatischen Qualität.

4.2 Techniken zur Prüfung der semantischen Qualität

In den vergangenen Jahren wurden zahlreiche Ansätze zur Prüfung der semantischen Qualität von GPM im Allgemeinen und EPKs im Besonderen publiziert. Diese Ansätze lassen sich in dynamische Analyseverfahren und statische Analyseverfahren unterteilen. Dynamische Analyseverfahren beruhen darauf, dass GPM in einen Formalismus „übersetzt“ werden, der über eine formale Semantik verfügt. Am häufigsten werden hierzu Petrinetze genutzt, seltener werden andere Formalismen wie abstrakte Automaten oder der Pi-Kalkül gewählt. Im Anschluss daran wird dann der Zustandsraum des in den entsprechenden Formalismus übertragenen Modells untersucht, wozu die Menge aller möglichen endlichen Abläufe berechnet wird.

Demgegenüber werden bei statischen Analyseverfahren aus dem GPM Schlussfolgerungen über mögliche Abläufe gezogen, ohne den Zustandsraum explizit zu berechnen.

4.2.1 Graph-Reduktionsalgorithmen

Graph-Reduktionsalgorithmen stellen eine besondere Klasse der statischen Analyseverfahren dar. Hier wird das zu untersuchende GPM wiederholt reduziert, indem Modellteile, die nicht zu Kontrollflussfehlern beitragen können, entfernt werden. Das GPM wird somit in jedem Reduktionsschritt vereinfacht. Hilfreich sind Reduktionsalgorithmen, wenn gezeigt werden kann, dass das reduzierte Modell eine wünschenswerte Eigenschaft (z.B. Soundness) genau dann besitzt, wenn das Ausgangsmodell diese Eigenschaft aufweist. Statt das Ausgangsmodell auf die entsprechende Eigenschaft zu untersuchen, kann die Analyse dann an einem kleineren und somit einfacher zu analysierenden Modell durchgeführt werden. Man spricht in diesem Falle von einer **eigenschaftserhaltenden** Reduktion.

Gelingt es insbesondere, ein GPM durch wiederholtes Reduzieren zu einem Graphen mit leerer Knotenmenge (bzw. je nach Ansatz zu einem einzelnen Knoten) zu reduzieren, ist damit gezeigt, dass das Ausgangsmodell die gewünschte Eigenschaft besitzt.

Solche Reduktionsalgorithmen sind seit längerem für Petrinetze bekannt [14,

46, 81]. Van der Aalst benutzte Petrinetz-Reduktionsalgorithmen in [167], um als Workflow-Netze modellierte GPM zu validieren.

Sadiq und Orlowska stellten in [147] ein Reduktionsverfahren für GPM vor. Dabei wurde eine Modellierungssprache betrachtet, die über AND- und XOR-Konnektoren verfügt. Sadiq und Orlowska definierten fünf Reduktionsregeln und zeigten, dass deren Anwendung das Vorhandensein von Deadlocks und Synchronisationsfehlern im Modell nicht verändert. Ihre Aussage, dass mit den vorgestellten Reduktionsregeln jedes korrekte Modell zu einem leeren Graphen reduziert werden kann, erwies sich jedoch als falsch: Es wurden Gegenbeispiele von Modellen gefunden, die trotz ihrer Korrektheit nicht komplett reduziert werden können [173, 100].

Reduktionsalgorithmen für EPKs wurden von van Dongen et al. [179] und Mendling [113] veröffentlicht. In [179] werden sie genutzt, um Modelle vor der Analyse zu vereinfachen. Bei Mendling [113] können durch die Reduktion auch unmittelbar Fehler gefunden werden. Näheres hierzu ist im Abschnitt 8.1.4 dargestellt.

Im Kontext der Modellierungssprache YAWL untersuchte Wynn [195] Reduktionsregeln für YAWL-Modelle und für Petrinetze mit Reset-Kanten [38]. Das erlaubt es, auch die komplexeren Konstrukte der Modellierungssprache YAWL (wie das Abbrechen von Vorgängen) zu untersuchen.

Reduktionsalgorithmen werden nicht nur als alleinstehende Verfahren angewendet, sondern auch in Verbindung mit Verfahren zur dynamischen Modellanalyse, die eine Untersuchung des Zustandsraumes nötig machen. Auf solche Verfahren wird im folgenden Abschnitt eingegangen.

4.2.2 Dynamische Analyse (Untersuchung des Zustandsraumes)

Ansatz und vorhandene Werkzeuge

Zahlreiche Veröffentlichungen beschäftigen sich damit, wie GPM in formal ausführbare Systeme überführt werden können. Liegt ein solches formal ausführbares System vor, können Model-Checking-Werkzeuge benutzt werden, um Eigenschaften der Abläufe dieser GPM zu untersuchen. Solche Werkzeuge berechnen systematisch und effizient alle möglichen Abfolgen von Zuständen, die ein GPM im Laufe seiner Abarbeitung annehmen kann. Hierbei soll unter einem Zustand, wie schon in Abschnitt 2.5.1, die Menge der zu einem bestimmten Zeitpunkt „aktiven“ Modellelemente verstanden werden.

Mittels Model-Checking kann nun die Menge aller Abfolgen von Zuständen daraufhin untersucht werden, ob bestimmte Eigenschaften eingehalten werden. Für

EPKs wäre eine solche wünschenswerte Eigenschaft beispielsweise, dass von jedem Startzustand aus stets ein Endzustand erreicht werden kann.

Eines der ersten verfügbaren Werkzeuge dafür war das GPM-Modellierungswerkzeug *Testbed*. In [76] wurde gezeigt, wie ein in *Testbed* erstelltes GPM in die Eingabesprache eines Model-Checking-Werkzeugs überführt werden kann. Mit diesem können dann Eigenschaften des Modells überprüft werden. Der Nachteil, dass diese Überprüfung nicht vom Modellierer selbst erfolgen kann, wurde in einer Folgeveröffentlichung [77] beseitigt: Leicht verständliche Eigenschaftsmuster erlauben es auch Modellierern, die keine Fachleute für formale Verifikation sind, Modelle zu prüfen.

Matousek [106] übersetzt in der Sprache XPDL vorliegende GPM in die Eingabesprache des Model-Checkers *SPIN*. Koehler [88] modelliert den Geschäftsprozess direkt in der Eingabesprache des Model-Checkers *NuSMV*, während Eshuis [45] UML-Aktivitätsdiagramme in diese Eingabesprache überführt, um dann deren Eigenschaften zu überprüfen.

Ein Ansatz für EPKs wurde von Rump [145] vorgestellt. Rump berechnet mit dem Werkzeug *EGraph* den Zustandsraum einer EPK, der als Zustandsgraph dargestellt wird. Dieser Zustandsgraph wird dann auf Verletzungen erwünschter Eigenschaften untersucht (z.B. „Es existiert kein AND-Join, der in jedem möglichen Ablauf blockiert.“). Das Verfahren von Rump hat den großen Vorteil, dass nicht nur die Existenz von Fehlern festgestellt werden kann, sondern die Fehler auch benannt werden können. Der Modellierer wird somit darüber informiert, welche Änderungen am Modell vorgenommen werden können. Ähnlich wie Rump, arbeitet das Werkzeug *EPCTools* von Cuntz und Kindler [30] direkt mit dem Zustandsgraphen einer EPK. Um diesen berechnen zu können, wird eine Fixpunktiteration (wie in Abschnitt 2.5.3 beschrieben) durchgeführt. Ein weiteres Werkzeug, das ohne Umweg über Petrinetze oder andere Formalismen direkt den Zustandsgraphen einer EPK betrachtet, ist das *ProM EPC Soundness Analysis Plugin* [9], das auf der von Mendling vorgeschlagenen EPK-Semantik [113] aufbaut.

Die bisher genannten Ansätze überführen ein GPM in Zustandsgraphen bzw. Transitionssysteme abstrakter Automaten. Da für solche Automaten eine abstrakte Semantik definiert ist, können die Eigenschaften der Automaten überprüft werden, woraus auf die Eigenschaften des zugrundeliegenden GPM geschlossen werden kann.

In [59] zeigten wir, wie entsprechende Verfahren auch verwendet werden können, um zeitliche sowie ressourcenabhängige Eigenschaften von GPM (etwa die Tatsache, dass

bei jedem Ablauf ein Endzustand innerhalb einer bestimmten Zeitvorgabe erreicht wird) nachzuweisen.

Häufiger noch als abstrakte Automaten werden Petrinetze als formale Grundlage gewählt, um Eigenschaften von GPM zu überprüfen.

Einige der Ansätze bzw. Werkzeuge gehen davon aus, dass Workflownetze, eine besondere Form von Petrinetzen, als Sprache zur Modellierung eines Geschäftsprozesses genutzt werden. Ein Vertreter dieser Klasse ist das an der Universität Eidhoven entwickelte Werkzeug *Woflan* [170]. Es testet neben anderen Eigenschaften von Petrinetzen auch die Soundness des Modells. Van der Aalst zeigte, dass Soundness in Workflownetzen gleichbedeutend ist mit den Eigenschaften Lebendigkeit und Beschränktheit [168, 171]. Diese Eigenschaften können mit verschiedenen Analysewerkzeugen für Petrinetze effizient überprüft werden.

Sind Modelle in anderen Geschäftsprozess-Modellierungssprachen (etwa EPK oder BPMN) zu validieren, so ist es möglich, diese zunächst in ein Petrinetz zu überführen[3] und dann die Eigenschaften des erhaltenen Petrinetzes zu untersuchen.

Für EPKs veröffentlichte van der Aalst in [169] einen Ansatz, um EPK-Modelle in Workflownetze zu überführen. Wegen der in Abschnitt 2.5.3 beschriebenen Probleme, die die nichtlokale Semantik von Konnektoren mit sich bringt, verzichtete er dabei zunächst auf EPKs mit OR-Joins und gab XOR-Joins die lokale Semantik von Def. 16.

Dass die Abbildung von OR-Joins in Petrinetze nicht zufriedenstellend möglich ist, überrascht nicht: Ob sich der Zustand in einem Petrinetz ändert, d.h. ob eine Transition schaltet, hängt immer nur von der lokalen Umgebung der Transition ab. Ein nichtlokales Verhalten von OR-Joins („schalte, wenn ausgeschlossen ist, dass noch weitere Marken eintreffen“) kann somit von einem (klassischen) Petrinetz nicht adäquat abgebildet werden. Chen/Scheer [27] sowie Langner, Schneider und Wehler [95, 96] stellen daher zusätzliche Anforderungen an die Strukturiertheit von EPKs. Sie nutzen dann gefärbte Petrinetze für die Analyse der Modelle.

Van Dongen, van der Aalst und Verbeek [179] nutzen zur Analyse von EPKs einfache Petrinetze. Ihr Ansatz weist zwei bemerkenswerte Eigenschaften auf: Zum einen ist die Mitwirkung des Modellierers erforderlich, um Informationen zu erlaubten Startzuständen und fachlich erlaubten Kontrollflüssen nach einem OR-Split zu erhalten. Dies erhöht die Aussagekraft der Analyseergebnisse. Zum zweiten unterscheidet sich das Ergebnis der Analyse von der bei anderen Verfahren üblichen Rückmeldung „korrekt“ oder „nicht korrekt“: Das Verfahren nach [179] unterscheidet die Rückmeldungen „garantiert korrekt“, „möglicherweise korrekt“ und „garantiert

[3] Genau genommen wird durch diese Überführung die genaue Bedeutung eines GPM erst definiert.

fehlerhaft“. Dies entspricht annähernd der Klassifizierung „sound“ / „relaxed sound nach [34], jedoch nicht sound“ / „nicht relaxed sound“. Wie auf Seite 35 am Beispiel von Abb. 3.3 diskutiert wurde, liegt die Entscheidung über die Korrektheit von Modellen, die relaxed sound, jedoch nicht sound sind, durchaus im Ermessen des Modellierers.

Oanea [124] analysiert EPKs, die zusätzliche Informationen über Zeiten und Ressourcen enthalten. Hierfür werden zeitbehaftete gefärbte Petrinetze verwendet.

Auch für GPM-Sprachen mit größerer Ausdrucksmächtigkeit (etwa mit Sprachkonstrukten, die ein Abbrechen oder Rückgängigmachen von Aktivitäten modellieren) wurde gezeigt, dass eine Überprüfung von Modelleigenschaften nach einer Transformation des Modells in ein Petrinetz möglich ist. Für die Sprache BPMN wurden entsprechende Ansätze etwa in [135] und [36] beschrieben. Für die Sprache YAWL stellt [197] einen Ansatz vor, der auf Petrinetzen mit Reset-Kanten beruht.

Zustandsexplosion und Maßnahmen dagegen

Die in diesem Abschnitt betrachteten Ansätze haben gemeinsam, dass die Zahl der zu untersuchenden Zustände schnell auch für praxisübliche Modelle sehr groß werden kann. Zeit- und Arbeitsspeicherressourcen reichen dann nicht mehr aus, um die Analysen durchzuführen. Bekannt ist dieses Phänomen unter dem Namen Zustandsexplosion (siehe auch Abschnitt 5.3).

Insbesondere OR-Splits im Modell tragen dazu bei, dass der zu untersuchende Zustandsraum eines Modells sehr groß werden kann. Ist die eingehende Kante eines OR-Splits mit n Ausgangskanten markiert, so gibt es nach der in Def. 15 beschriebenen Übergangsrelation $2^n - 1$ Möglichkeiten, Markierungen weiterzuleiten. Bei mehreren OR-Splits im Modell führt dies schnell zu praktisch nicht mehr handhabbaren Zustandsräumen.

Um bei dynamischen Analyseverfahren die Zahl der zu untersuchenden Zustände zu verringern, ist es häufig möglich, vor deren Anwendung die zu untersuchenden Modelle mittels eigenschaftserhaltender Reduktionsregeln wie in Abschnitt 4.2.1 beschrieben zu vereinfachen.

Alle in der Literatur diskutierten Systeme von Reduktionsregeln, die zu diesem Zweck eingesetzt werden, beinhalten das Reduzieren von Sequenzen, von wohlstrukturierten Kontrollblöcken sowie von Iterationen, die mit einem XOR-Join begonnen und mit einem XOR-Split abgeschlossen werden. Rump zeigte in [145], dass diese Reduktionen eigenschaftserhaltend sind, d.h. mit dem von ihm betrachteten Analyseprogramm für die reduzierten Modelle dieselben Fehler und Warnungen wie

für die unreduzierten Originalmodelle geliefert werden. Der Vorteil der Verwendung reduzierter Modelle liegt darin, dass in vielen Fällen die erwähnte Zustandsexplosion vermieden werden kann.

Die Verwendung von Reduktionsregeln vor einer Untersuchung des Zustandsraumes der (reduzierten) Modelle wurde von verschiedenen Autoren unter unterschiedlichen Bezeichnungen eingeführt. Rump [145] spricht von der Generierung des Junktorennetzes, Cuntz und Kindler [28, 30] von Abstraktion von Ketten, van Dongen [179], Wynn [195] und Mendling [113] von Reduktion.

Vanhatalo et al. [182, 181] verfolgen einen weiteren Ansatz, um die Komplexität der Validierung von GPM zu verringern. Sie nutzen ein aus der Compilertheorie [79] bekanntes Verfahren, um ein GPM in Teilgraphen zu zerlegen, die genau eine eingehende und genau eine ausgehende Kante haben. Es lässt sich zeigen, dass das GPM genau dann sound ist, wenn alle durch einen solchen Teilgraphen gebildeten Teilmodelle sound sind [165]. Somit vereinfacht sich die Überprüfung der Soundness-Eigenschaft auf das Validieren der Teilmodelle. Allerdings wurde dieses Verfahren für Modelle entwickelt, die mit dem Werkzeug *IBM WebSphere Business Modeler* erstellt wurden. Solche Modelle haben immer genau einen Start- und genau einen Endpunkt. Für EPKs mit mehreren Start- und Endereignissen gelingt die Aufteilung in Teilmodelle weniger effektiv. Da nach dem Ansatz aus [182, 181] z.B. alle Modellfragmente mit einem Endereignis in dasselbe Teilmodell aufgenommen werden müssten, ergeben sich bei der Zerlegung von EPKs unter Umständen relativ große Teilmodelle, was den Effekt der Komplexitätsverbesserung deutlich verringern kann.

4.2.3 Weitere statische Analyseverfahren und heuristische Verfahren

Die im vorangegangenen Abschnitt besprochenen Ansätze untersuchen sämtliche möglichen Abläufe eines Modells, es wird also systematisch der Raum aller möglichen Zustände erzeugt. Demgegenüber prüfen statische Verfahren das Modell, ohne den Zustandsraum zu berechnen. Die im Abschnitt 4.2.1 vorgestellten Reduktionsalgorithmen stellen eine besondere Klasse der statischen Verfahren dar.

Auch unter den statischen Analyseverfahren existieren solche, die ein GPM in ein Petrinetz überführen und dann die aus der Theorie der Petrinetze bekannten statischen Verfahren verwenden, um Fehler im GPM zu erkennen. Van der Aalst benutzt zur Untersuchung von Workflow-Netzen in [168] neben Verfahren, die eine Untersuchung des gesamten Zustandsraums des Petrinetzes erfordern, auch statische Verfahren. Dabei wird die Tatsache genutzt, dass ein Modell, in dem ein Block von

einem AND-Split begonnen, jedoch von einem XOR-Join abgeschlossen wird (oder umgekehrt) immer in ein Petrinetz überführt wird, das Handles [47] aufweist.

Van Dongen [178] nutzt Invarianten in Petrinetzen, um die Berechnung des gesamten Zustandsraumes insbesondere für komplexe Modelle zu vermeiden. In [178] wird gezeigt, dass ein durch Petrinetz-Invarianten entdecktes Problem tatsächlich immer auf eine Verletzung der Relaxed Soundness-Eigenschaft einer EPK schließen lässt. Umgekehrt gibt es aber (in der Praxis allerdings nur selten zu erwartende) Modelle, deren Verletzung der Relaxed-Soundness-Eigenschaft sich nicht durch die Betrachtung von Invarianten finden lässt.

In [37] berechnen van Dongen et al. aus einem gegebenen GPM alle möglichen Relationen (sog. *causal footprints*) folgender Art:

- In jedem Ablauf folgt der Markierung von Knoten a immer eine Markierung eines Knotens aus der Menge M.
- In jedem Ablauf geht der Markierung von Knoten a immer eine Markierung eines Knotens aus der Menge M voraus.

[37] nennt verschiedene strukturelle Muster, nach denen sich von den genannten Relationen auf Kontrollflussfehler im Modell schließen lässt. In der in [37] vorgestellten Form ist das Verfahren für größere Modelle allerdings praktisch nicht anwendbar. Es wird nämlich kein Versuch unternommen, die in den Relationen vorkommenden Knoten(mengen) auf solche zu beschränken, die tatsächlich Aussagen über Kontrollflussfehler liefern. Daher ergibt sich aus kombinatorischen Gründen schnell eine sehr große Menge von Knoten(mengen), die in einer der Relationen zueinander stehen. Rechenzeit und Speichernutzung sind selbst für Modelle mittlerer Größe nicht mehr praktikabel.

Van Dongens in [37] beschriebener Ansatz ist heuristischer Natur. Bei Vorliegen eines der dort betrachteten Muster muss nicht stets ein Kontrollflussfehler auftreten[4]. Positiv ist dagegen bei van Dongens Ansatz, dass der Modellierer eine gute Rückmeldung darüber erhält, welche Konnektoren für einen Modellfehler verantwortlich sind. Weiterhin garantiert das Verfahren, dass Modellfehler wie „einem XOR-Split ist

[4] In [37] wurde ausgeführt, dass es noch eine offene Frage sei, ob das Vorliegen eines Fehlermusters in den causal footprints immer garantiert auf einen Kontrollflussfehler schließen lässt. Wir werden auf Seite 112 Beispiele dafür sehen, bei denen dies nicht der Fall ist (Abb. 9.1(d) und (e)). Nach Wissen des Autors sind diese Modelle die ersten publizierten Beispiele dafür, dass die mit causal footprints in [37] erhaltenen Vermutungen über Kontrollflussfehler nicht immer zutreffen. Da es sich hierbei nur um ein Randresultat dieser Arbeit handelt, wird auf eine ausführliche Diskussion verzichtet.

ein AND-Join zugeordnet“ unabhängig von den zwischen Split und Join liegenden Modellelementen gefunden werden.

Diese Eigenschaft gilt auch für das Verfahren von Vanhatalo et al. [182, 181]. Wie bereits in Abschnitt 4.2.2 beschrieben, wird dabei ein GPM zunächst in Teilgraphen zerlegt, die genau eine eingehende und genau eine ausgehende Kante haben. Zur Analyse der durch Zerlegung gebildeten Teilgraphen wird dann ein sehr einfaches heuristisches Verfahren benutzt, das trotz seiner Unvollständigkeit bereits einen großen Teil der Fehler in GPM erkennen kann.

4.2.4 Inhaltliche Prüfungen

Das Werkzeug *Semtalk* [51] verwendet Ontologien, um eine einheitliche Verwendung von Substantiven und Verben über ein oder mehrere Modelle hinweg zu gewährleisten. Bei Verwendung eines solchen ontologiebasierten Konzeptes ist es auch möglich, echte inhaltliche Prüfungen von Geschäftsregeln automatisiert vorzunehmen. Fillies und Weichhardt führen in [50] ein Beispiel an, in dem zwei Geschäftsprozessmodelle „Bestelleingang“ und „Bestellungsbearbeitung“ betrachtet werden. Sie nennen die Geschäftsregel „Nur bestätigte Bestellungen dürfen ausgeführt werden“ als Beispiel einer Regel, die man mit Hilfe ontologiebasierter Modellierung modellübergreifend prüfen kann. Ein ähnliches Beispiel nennen Thomas und Fellmann. Sie beschreiben in [162] einen Ansatz, EPK-Modelle betrieblicher Abläufe mit Ontologien zu verknüpfen.

Eine Übersicht über den Stand der Technik solcher ontologiebasierter Ansätze liefert [44]. Durch die Einbindung von Ontologien in Geschäftsprozessmodellierungsmethoden können mächtige Werkzeuge zur Konsistenzprüfung und Validierung von Geschäftsprozessmodellen geschaffen werden.

Allerdings kann angenommen werden, dass sich ontologiebasierte Verfahren in der betrieblichen Praxis nur in Einzelfällen (etwa bei der Dokumentation medizinischer Prozesse) durchsetzen werden. Dem Ziel (Verbesserung der Modellqualität) steht zumindest in der Einführungsphase ein erhöhter Aufwand im Modellierungsprozess beim Erstellen der Ontologie entgegen.

4.3 Techniken zur Prüfung der pragmatischen Qualität

Mit Verfahren zur Prüfung der pragmatischen Qualität wird untersucht, ob ein Modell für den Leser leicht verständlich modelliert ist.

4.3.1 Metriken zur Bestimmung der Modellkomplexität

Verschiedene Forschungsgruppen untersuchten Komplexitätsmetriken für GPM. Eine solche Metrik ordnet einem GPM eine Zahl zu, die einen oder mehrere Aspekte der Komplexität (worunter wir einfache Verständlichkeit, Überprüfbarkeit und Wartbarkeit verstehen) messen soll. Diesen Komplexitätsmetriken ist ein gesondertes Kapitel dieser Arbeit (Kapitel 6) gewidmet. Ein möglicher Einsatz von GPM-Komplexitätsmetriken besteht darin, den Modellierer darüber zu informieren, wenn die gemessene Komplexität seines Modells einen gewissen Schwellwert überschreitet. Er kann dies zum Anlass nehmen, besser verständliche Modellierungsalternativen in Betracht zu ziehen.

4.3.2 Prüfung von Strukturiertheitsanforderungen

In Abschnitt 7.1 wurde Wohlstrukturiertheit als ein Merkmal von GPM hervorgehoben, das deren Verständnis fördert. Einige Werkzeuge und Sprachen erlauben nur die Erstellung wohlstrukturierter Modelle. So sind Modelle in den Sprachen BPEL4WS [2] oder des Workflow-Managementsystems *ARISTA Flow* [137, 31] grundsätzlich wohlstrukturiert. Im UML-Standard 1.0 gab es eine solche Einschränkung auch für UML-Aktivitätsdiagramme. Dort war gefordert, dass Verzweigungs- und Verschmelzungsknoten paarweise auftreten müssen. In der neueren Version 2.0 des Standards [125] fehlt eine solche Festlegung.

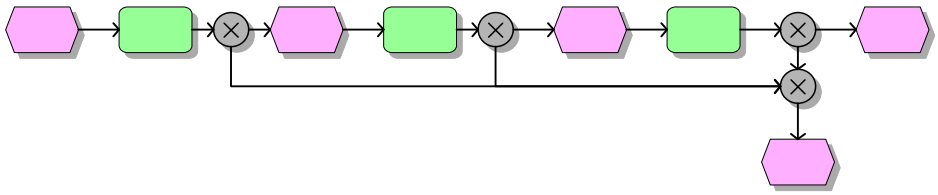


Abbildung 4.1: Prozess mit Ausnahmebehandlung:
nicht wohlstrukturiert, aber leicht verständlich

Eine strenge Einschränkung auf wohlstrukturierte Modelle sieht die Sprache EPK nicht vor. Sie scheint auch nicht sinnvoll, wie das Modellbeispiel in Abb. 4.1 zeigt. Dieses EPK-Modell sollte intuitiv leicht verständlich sein: Nach jedem der nacheinander ablaufenden Prozessschritte kann ein Ausnahmefall eintreten, der dazu führt, dass der Prozess vorzeitig beendet wird (dargestellt durch die Verzweigung zu dem unteren Endereignis). Das Modell ist unstrukturiert, da einem XOR-Join mehrere XOR-Splits gegenüberstehen. Ein Versuch, diesen Prozess durch ein wohlstrukturiertes Modell

zu beschreiben, führt jedoch zu einem komplizierteren Modell als dem in Abb. 4.1 gezeigten.[5]

Die Aussage, dass eine Beschränkung auf wohlstrukturierte GPM nicht sinnvoll ist, wird auch von Gschwind, Koehler und Wong geteilt. Die Autoren schreiben in [69], dass eine solche Beschränkung „für viele GPM-Szenarien sehr einschränkend" sei. Sie stellen dann einen in das kommerzielle GPM-Werkzeug *IBM WebSphere Busines Modeler* integrierten Ansatz vor. Dieser erlaubt ein Abweichen von einer streng blockorientierten Struktur nur dann, wenn das Hinzunehmen einer Kante zum bisherigen Modell nicht dessen Soundness verletzt. Ein solcher Ansatz führt dazu, dass Modelle mit bestimmten Fehlern gar nicht erst konstruiert werden können (*Correctness by Construction*). Ein Import von Modellen, die mit einem anderen Modellierungswerkzeug erstellt wurden, ist dann allerdings nicht ohne weiteres möglich.

4.3.3 Suche nach unnötigen OR-Konnektoren

In Abschnitt 3.3.2 wurde ausgeführt, dass in gewissen Situationen OR-Konnektoren durch XOR- bzw. AND-Konnektoren ersetzt werden sollten, um die Lesbarkeit des Modells zu verbessern. Der Ansatz von Rump [145] sowie das Werkzeug *YAWL Editor* [195] erlauben es, solche OR-Konnektoren zu finden. Beide Ansätze erfordern die Konstruktion des Zustandsraumes. Sie sind somit von dem in Abschnitt 4.2.2 beschriebenen Problem möglicher Zustandsexplosionen betroffen.

[5] Andere Modellierungssprachen wie BPMN unterstützen das Konzept der Exception. Dieses erlaubt in solchen Fällen Modelle, die sowohl strukturiert als auch gut lesbar sind. Die EPK-Sprache unterstützt dieses Konzept nicht.

5 Zielsetzung für ein Werkzeug zur Überprüfung der Qualität von Geschäftsprozessmodellen

Im vorangegangenen Kapitel wurden die gegenwärtig verfügbaren Verfahren und Werkzeuge für die Überprüfung der Qualität von GPM vorgestellt. In diesem Kapitel soll nun herausgearbeitet werden, welche Verbesserungsmöglichkeiten es zu vorhandenen Ansätzen gibt. Dabei konzentrieren wir uns auf Modelle in der Sprache EPK.

5.1 Rückmeldung über Fehlerursachen

Bei einigen der in Abschnitt 4.2 genannten Validierungswerkzeuge ist das Ergebnis einer Validierung lediglich die Information, ob ein bestimmter Fehler vorliegt. Diese Information wird durch Übersetzung der EPK in ein Modell mit wohldefinierter Semantik (meist in ein Petrinetz) und anschließende Analyse des letztgenannten Modells gewonnen. Eine „Zurückübersetzung“ gefundener Fehler in die Original-EPK erfolgt dabei nicht. Der Modellierer erhält somit keine Rückmeldung, welche Modellelemente der EPK für diesen Fehler verantwortlich sind und wie der Fehler beseitigt werden kann.

Ein Beispiel für solch eine fehlende Rückmeldung über die Ursachen von Kontrollflussfehlern findet sich im Werkzeug *EPCTools* [28, 30]. Hier signalisiert eine rote oder grüne „Ampel“, ob das untersuchte Modell sound ist oder nicht, es wird also lediglich eine Ja/Nein-Entscheidung geliefert. Die Ursache des Fehlers (oder ggf. auch mehrerer Fehler) muss der Modellierer selbst ermitteln.

Eine bessere Rückmeldung erhält man etwa beim *EPC Soundness Analysis Plugin* des Werkzeugs *ProM* [177] (hier werden die Konnektoren, die zu Fehlern führen, eingefärbt) oder dem in [145] beschriebenen Werkzeug von Rump (wo eine gut lesbare Beschreibung des Fehlers ausgegeben wird). Unzureichende Rückmeldungen

sind somit keine prinzipielle Schwäche der verwendeten Ansätze, sondern lediglich einiger gegenwärtiger Implementierungen.

Allerdings können selbst in Fällen, in denen das Validierungswerkzeug das in einem formalen Modell (z.B. in einem Petrinetz) erkannte Problem wieder in die Problemdomäne der analysierten EPK „übersetzt“, Informationen zu Fehlern verloren gehen.

Dies soll in Abb. 5.1 illustriert werden. In diesem Modell ergeben sich aus einer unsachgemäßen Kombination eines AND-Splits mit einem XOR-Join offensichtlich zwei Fehler – einer im inneren AND-XOR-Kontrollblock und ein weiterer im äußeren AND-XOR-Kontrollblock.

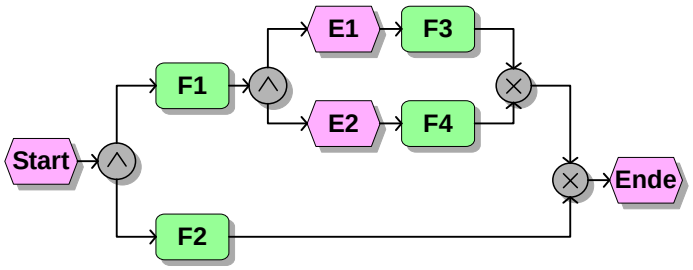


Abbildung 5.1: Zwei verschachtelte Kontrollflussblöcke mit Modellierungsfehler durch die Paarung AND-Split/ XOR-Join

Nehmen wir übereinstimmend mit [123] und [87] an, dass ein XOR-Join blockiert (also keine Markierung an seinen Ausgangskante weiterleitet), wenn mehr als einer seiner Eingangskanten markiert ist. Dann leitet im gezeigten Modell der innere XOR-Join auf Grund dieser Blockade nie eine Markierung an seine Ausgangskante weiter. In der Folge wird am äußeren XOR-Join immer nur genau eine Eingangskante markiert. Ein Analysewerkzeug, das den gesamten Zustandsraum aller möglichen Abläufe im Modell untersucht, wird daher den im äußeren AND-Split/XOR-Join-Block vorhandenen Fehler nicht feststellen.

5.2 Sofortige Rückmeldung über Modellfehler

Es ist wünschenswert, dass eine Überprüfung von EPKs auf mögliche Fehler bereits während des Erstellens der Modelle erfolgen kann. Dadurch wird erreicht, dass der Modellierer zum frühestmöglichen Zeitpunkt eine Rückmeldung über mögliche Probleme im Modell erhält und diese sofort korrigiert werden können.

Im Bereich der Softwareentwicklung ist eine solche frühzeitige Rückmeldung über mögliche Probleme bereits Stand der Technik: In Entwicklungsumgebungen wie *Eclipse*, *Visual Studio* oder *JBuilder* sorgen Compilerdurchläufe oder sogar Software-Testläufe während der Programmeingabe dafür, dass Entwickler bestimmte Klassen möglicher Probleme sofort erkennen [148, 184]. Ebenso existieren Werkzeuge, die Verstöße gegen Prinzipien bekannter Entwurfsmuster erkennen können [120, 17, 43][1].

Im Bereich der UML-Modellierung gibt es mit ARGO/UML [140] ebenfalls ein Werkzeug, bei dem während des Modellierens ständig Tests im Hintergrund ablaufen und der Modellierer, falls nötig, Hinweise erhält. Untersucht werden bei diesen Tests vor allem syntaktische Fehler sowie Konsistenzprobleme zwischen verschiedenen Modellen. Über ähnliche Funktionen verfügen *MetaModelAgent* (eine Erweiterung des *IBM Rational Software Architect*) [8] und *UML/Analyzer* [41].

Modellierungswerkzeuge für GPM bieten eine solche Unterstützung jedoch in der Regel nicht an. Werkzeuge wie *ARISTA Flow* [137, 31] oder *IBM WebSphere Busines Modeler* [69] garantieren zwar, dass keine Kontrollflussfehler vorliegen. Sie erlauben jedoch dem Modellierer kein „freies“ Erstellen von GPM in dem Sinne, dass Modellelemente in beliebiger Reihenfolge gezeichnet werden, um sie später zu einem Gesamtmodell zu verbinden. Statt dessen müssen GPM sukzessive durch wiederholtes Einfügen von (wohlstrukturierten) Blöcken modelliert werden.

Andere gegenwärtig verfügbare Validierungsmethoden gehen von einem mehrstufigen Prozess bestehend aus Modellerstellung, Validierung sowie Korrektur und Weiterentwicklung aus.

Einer der Gründe für die notwendige Trennung von Modellerstellung und Validierung ist, dass einige der bekannten Validierungstechniken nicht direkt auf unvollständig modellierte Modelle anwendbar sind. Diese können u.a. auch aus nicht zusammenhängenden Graphen bestehen, die zu einem späteren Zeitpunkt im Modellierungsprozess zusammengefügt werden sollen. Verfahren, die nicht unmittelbar auf unvollständige Modelle angewendet werden können, finden sich unter den Graph-Reduktionsverfahren ebenso wie unter den auf Petrinetzen basierenden Ansätzen.

5.3 Vermeiden von Zustandsexplosionen

Bei den im Abschnitt 4.2.2 vorgestellten Analyseverfahren, die den gesamten Zustandsraum aus allen denkbaren Abläufen berechnen, kann das bereits in Abschnitt 4.2.2 diskutierte Problem der Zustandsexplosion auftreten.

[1] Eine umfangreiche Übersicht über weitere Quellen ist in [43] zu finden.

Ein Ansatz, der von dem Problem sehr großer Zustandsräume betroffen ist, ist die Berechnung der Fixpunktsemantik nach [87]. Diese wird im Werkzeug *EPC-Tools* genutzt. Obwohl bei der Implementierung sehr effiziente Maßnahmen wie die Benutzung von Binary Decision Diagrams [21] angewendet wurden, bleibt die worst-case-Komplexität exponentiell [30]. [29] gibt ein Praxisbeispiel für eine EPK an (ein Modell einer Zahlungsanweisung), für die mit den in [30] beschriebenen Ansätzen in angemessener Zeit keine Fixpunktsemantik berechnet werden kann. Dies war Ausgangspunkt für eine Verbesserung des Verfahrens: Ein Algorithmus zur Markierung des Zustandsraumes [29] führt zu erheblich verbesserten Werten von Rechenzeit und Speicherverbrauch. Auch hierfür wurde jedoch in [29] ein Beispiel angegeben, für das eine Berechnung der Fixpunktsemantik nicht durchgeführt werden kann.

Kapitel 11 enthält die Ergebnisse einer Analyse von fast 1000 EPK-Modellen mit gängigen Analysewerkzeugen. Diese basieren auf einer Berechnung des Zustandsraumes aller denkbaren Abläufe. Wir werden sehen, dass bei einigen Modellen aus der Praxis eine derartige Untersuchung des Modells auf Fehler mit vertretbarer Rechen- bzw. Speicherkapazität unmöglich ist.

Zielsetzung für die in dieser Arbeit zu entwickelnde Methode soll es dagegen sein, *alle* EPK-Modelle, die eine praxisübliche Größe haben, überprüfen zu können.

5.4 Beachtung verschiedener Interpretationsmöglichkeiten für ein Modell

Es wurde bereits mehrfach erwähnt, dass GPM-Modellierungssprachen wie EPKs oder BPMN eingeführt wurden, ohne deren formale Semantik zu beschreiben. In der betrieblichen Praxis dürften den meisten Personen, die solche Modelle einsetzten, vorhandene wissenschaftliche Ansätze zur formalen Beschreibung der Semantik unbekannt sein.

Selbst zur *Syntax* von EPKs schreiben Langner, Schneider und Wehler in [95]: „Ein Blick in die Literatur ... zeigt schnell, dass die Syntax einer EPK in der Praxis noch nicht standardisiert ist. Vielmehr wird die Syntax pragmatisch dem jeweiligen Zusammenhang angepasst.“

Man kann also davon ausgehen, dass EPKs in der Praxis entwickelt werden, ohne dass sich der Modellierer Gedanken über die in Abschnitt 2.5 aufgeführten formalen Semantiken macht. Andererseits wird aber bei der Analyse solcher Modelle mit Methoden, wie in Abschnitt 4.2.2 beschrieben, eine bestimmte formale Semantik

angenommen. Daraus ergibt sich jedoch eine Gefahr: Die Eigenheiten der jeweilig angenommenen Semantik können zur Folge haben, dass die Validierungsmethode Modellierungsprobleme entdeckt, die eigentlich gar keine sind, oder umgekehrt tatsächliche Probleme nicht erkannt werden. Dies soll an Hand einiger Beispiele illustriert werden.

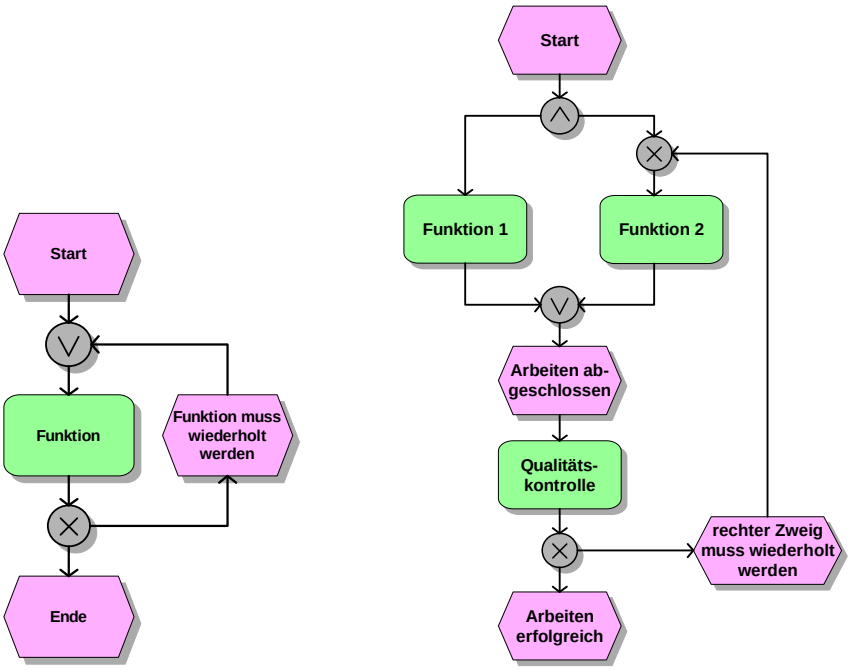


Abbildung 5.2: Beispielmodelle

Abb. 5.2 zeigt zwei Modellfragmente, die in der Praxis völlig unproblematisch sind: Im linken Modell wird die Funktion ein- oder mehrmals durchlaufen, es wird also ein Zyklus modelliert. Der Einstieg in diesen Zyklus ist durch einen OR-Join modelliert. Ein XOR-Join wäre für diesen Zweck zwar besser geeignet, da stets nur an einem der beiden Eingangskanten ein Kontrollfluss den Join erreicht, jedoch ergibt sich auch bei Verwendung eines OR-Joins ein durchaus korrektes Modell. Die Eigenheiten der in [113] vorgeschlagenen Semantik führen aber dazu, dass für solche Modelle ein Deadlock am OR-Join erkannt wird, wenn man für die Analyse ebendiese Semantik zugrundelegt.

Für das rechte Modell kann festgestellt werden, dass keine Fixpunktsemantik laut [87] existiert (siehe Abschnitt 2.5.4). Eine entsprechende Analyse würde also auch hier einen Fehler melden. Trotzdem kann das Modell in der Praxis ohne die Gefahr von

Missverständnissen verwendet werden. Eine ausführlichere Diskussion des Modells findet sich auf Seite 118.

Im dritten Beispielmodell (Abb. 5.3) ist der stark vereinfachte Prozess des Drucks und der Auslieferung eines Buchtitels dargestellt. Zunächst wird eine Auflage gedruckt. Nach Eintreten des Startereignisses „Stichtag (Erscheinungstermin) erreicht“ erfolgt dann die Auslieferung und ggf. der Druck weiterer Auflagen.

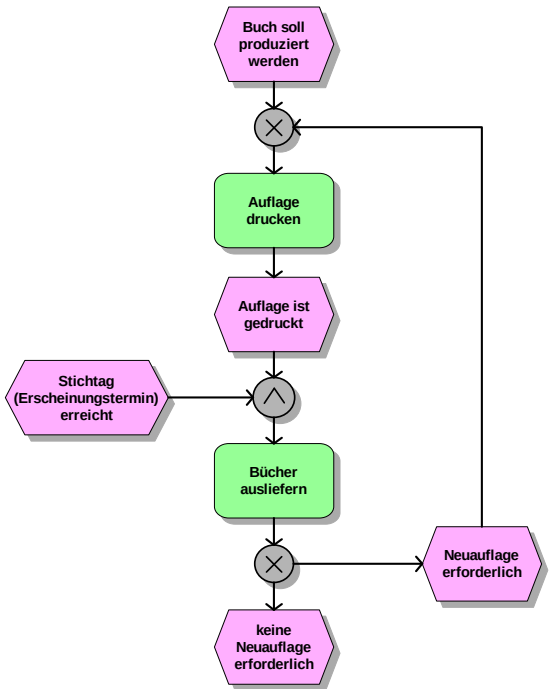


Abbildung 5.3: Startereignis „Erreichen eines Stichtags“

Auch dieses Modell scheint völlig korrekt und auch leicht verständlich zu sein. Es erstaunt daher auf den ersten Blick, dass sämtliche in Abschnitt 4.2.2 erwähnten Validierungstechniken einen Fehler im Modell melden. Der Grund dafür ist, dass Semantikdefinitionen analog Def. 15 annehmen, dass Markierungen an Startereignissen „verbraucht“ werden. Während diese Annahme bei Startereignissen wie „Kundenanfrage trifft ein“ korrekt ist (die Anfrage trifft genau einmal ein), ist sie für das gezeigte Beispiel nicht richtig: Ist der Stichtag einmal erreicht, ist das Modell so zu verstehen, dass das Ereignis dauerhaft markiert bleibt.[2] Das Beispiel

[2] Der Grund für dieses Problem ist, dass die EPK-Modellierungsmethode keine Modellierung von Zuständen vorsieht. Zustände müssen in EPKs als Ereignisse modelliert werden, was zu den dargestellten Schwierigkeiten führt.

zeigt, dass eine von Petrinetzen bekannte Semantik mit „wandernden“ Marken nicht immer tatsächlich die Intention eines EPK-Modells widerspiegelt.

Von einem Werkzeug zur Überprüfung einer EPK soll also erwartet werden können, dass es wenn nötig verschiedene Deutungsmöglichkeiten einer EPK berücksichtigt.

Bisher wurde erörtert, dass bei Methoden, die EPKs in Petrinetze oder Automatenmodelle übersetzen, Annahmen über die Semantik des Ausgangsmodells zugrundegelegt werden, die nicht notwendig der Intention des Modellierers entsprechen müssen.

Ebenso gibt es aber auch Annahmen darüber, was als „korrektes“ Modell zu verstehen ist. Auch diese Annahmen verdienen es, hinterfragt zu werden. Ein Grund für die Beliebtheit des Korrektheitskriterium „Soundness“ dürfte darin liegen, dass es sich - nach Übersetzung einer EPK in ein Petrinetz - durch die Eigenschaften der Lebendigkeit und der Beschränktheit von Petrinetzen ausdrücken lässt [167]. Für die Überprüfung dieser Eigenschaften in Petrinetzen existieren leistungsfähige Werkzeuge.

Während jedoch beispielsweise die Werkzeuge von Kindler [30] oder Mendling [113] Soundness als Maßstab für die Korrektheit von Modellen nehmen, betonen Dehnert [35] und van Dongen [177], dass dieses Kriterium in vielen Fällen nicht geeignet ist. Wünschenswert wäre, dem Modellierer in einem gewissen Rahmen selbst die Wahl zu lassen, was als Fehler anzusehen ist. Dies tut etwa Rump [145] bei der Bewertung von AND-Konnektoren, von denen in einem Ablauf nur eine Eingangskante markiert wird. AND-Konnektoren, bei denen es in jedem denkbaren Ablauf zu einer solchen Deadlock-Situation kommt, werden stets als fehlerhaft ausgegeben. AND-Konnektoren, bei denen es sowohl einen Ablauf mit Deadlock als auch einen Ablauf ohne Deadlock gibt, werden dem Modellierer als Warnung mitgeteilt.

Zielstellung für ein zu entwickelndes Verfahren ist es, Meldungstexte für Fehler- und Warnmeldungen so zu gestalten, dass falls nötig verschiedene Interpretationen der Bedeutung des Modells berücksichtigt werden.

5.5 Betrachtung der Beschriftung von Modellelementen

Die Bedeutung eines Modells lässt sich nach Esswein et al. [48] stets in zwei Teile gliedern: in die Semantik der einzelnen Elemente der Modellierungssprache (für EPKs in Abschnitt 2.5 diskutiert) sowie die „konkrete Semantik“ jedes einzelnen Modellelements. Darunter wird die Bedeutung des Modellelements in der tatsächlichen Anwendungsdomäne verstanden. Bei EPKs wird die „konkrete Semantik“ durch

die Beschriftung von Ereignissen und Funktionen in natürlicher Sprache angegeben. Ähnlich unterteilen Pfeifer und Niehaves [131] die Bedeutung graphischer Modelle in die Anordnung der formalen Modellierungselemente (model element structure) sowie die Bedeutung der Modellelemente in der Sprache der Anwendungsdomäne (terminological structure). Pfeifer und Niehaves unterstreichen, dass die Validierung eines Modells immer beide genannten Aspekte berücksichtigen soll.

Bisher vorgeschlagene Verfahren zur Validierung von Geschäftsprozessmodellen beziehen jedoch (von ganz wenigen Ausnahmen wie beispielsweise [4] abgesehen) kaum die Beschriftung von Modellelementen in die Analyse ein. Die meisten der in Kapitel 4 vorgestellten Validierungsverfahren finden ausschließlich Kontrollflussfehler, die sich aus der Anordnung und den Typen der Konnektoren ergeben. Üblicherweise verlangen Werkzeuge, die die Beschriftung der Funktionen und Ereignisse in die Validierung einbeziehen (vgl. Abschnitt 4.2.4) den Aufbau einer Ontologie, welche die Sachverhalte in der Anwendungsdomäne beschreibt.

In dieser Arbeit werden die Beschriftungen von Modellelementen genutzt, um einige häufige Fehler in EPKs zu identifizieren. Das zur Beschriftung von Modellelementen verwendete Vokabular muss dabei weder eingeschränkt noch durch eine Ontologie beschrieben werden.

5.6 Anpassbarkeit und Erweiterbarkeit der Validierungsregeln

Insbesondere bei der Überprüfung der pragmatischen Qualität sollte es möglich sein, die bei der Überprüfung verwendeten Regeln auszuwählen und wenn nötig anzupassen. Auf diese Weise können eigene organisationsweite Modellierungskonventionen durchgesetzt werden.

Beispielsweise kann eine Organisation fordern, dass die Syntaxforderungen aus Def. 1 streng umzusetzen sind und sich Ereignisse und Funktionen im Kontrollfluss immer abwechseln müssen. In einer anderen Organisation kann die Entscheidung getroffen werden, dass mehrere Funktionen direkt aufeinander folgen dürfen. Es sollte also anpassbar sein, welche Validierungsregeln anzuwenden sind.

Auch die Hinzunahme eigener Modellierungskonventionen kann sinnvoll sein. Ist beispielsweise die Überführung eines EPK-Modells in die Sprache BPEL geplant, so müssen verschiedene Strukturiertheitsanforderungen erfüllt sein [199, 156], die sich in entsprechenden Modellierungskonventionen niederschlagen sollten.

5.7 Beachtung der pragmatischen Modellqualität

Die Mehrzahl der vorhandenen Validierungswerkzeuge für GPM behandelt das zu untersuchende Modell prinzipiell wie maschinell ausführbaren Code. Die Validierung in diesen Werkzeugen beantwortet Fragen wie „Existieren Deadlocks im Modell?“. Die für den wichtigen Einsatzzweck der Unterstützung der Kommunikation zwischen Menschen ebenso bedeutsame Frage „Kann man die dargestellte Situation *leichter verständlich* modellieren?“ findet dagegen wenig Beachtung.

Lediglich die Möglichkeit, einen OR-Join durch einen die Situation besser beschreibenden AND- bzw. XOR-Join zu ersetzen, wird bei einigen Ansätzen (z.B. [197, 145]) überprüft.

Ein Werkzeug, das die Qualität von GPM untersucht, sollte auch Modellteile identifizieren können, die schwer verständlich sind. In bestimmten Situationen sollte dem Modellierer eine besser verständliche Modellierungsalternative vorgeschlagen werden.

In dieser Arbeit werden in Kapitel 9 einige Muster in EPKs vorgestellt, für die besser verständliche Modellierungsalternativen bekannt sind. Weiterhin werden im nächsten Kapitel Komplexitätsmetriken für GPM besprochen, die (unter anderem) zum Ziel haben, Lesbarkeit und Verständlichkeit von Modellen zu messen. Solche Metriken geben Rückschlüsse auf mögliche Modellverbesserungen. Nach der Diskussion von bekannten Komplexitätsmetriken für EPKs wird in Kapitel 7 eine neue Komplexitätsmetrik eingeführt, die den Grad der Strukturiertheit einer EPK misst.

6 Metriken zur Qualitätsbestimmung von Geschäftsprozessmodellen

Piwowarski schreibt in [132]: „Like a good work of art, we know a good program when we see one. Like a good work of art, it is often difficult to quantify what makes a program good.“ Ein Verfahren, das hilft, die Einfachheit, Verständlichkeit und Wartbarkeit von Software zu messen, ist die Bestimmung von Komplexitätsmetriken. Eine solche Komplexitätsmetrik ordnet einer gegebenen Software einen quantitativen Wert (meist eine ganze oder reelle Zahl) zu, der bestimmte Aspekte dieser Software misst.

In der Softwaretechnik haben sich Komplexitätsmetriken seit langem bewährt, um Aussagen über Softwarekomplexität zu treffen. Komplexitätsmetriken wurden unter anderem dafür angewendet, Aussagen über die Verständlichkeit von Quelltexten zu treffen [105], den Aufwand für die Erstellung von Software zu schätzen [1], die Qualität der Modularisierung zu bewerten [92] oder Codeteile zu finden, in denen die Wahrscheinlichkeit von Fehlern besonders hoch ist [5].

Seit einigen Jahren werden Vorschläge publiziert, wie sich die Idee der Komplexitätsmetriken auch für GPM nutzen lässt. Aus den Messungen nach einer solchen Metrik soll beispielsweise abgeleitet werden, wann ein Modell vereinfacht werden sollte, was beispielsweise durch das Zerlegen in mehrere Teilmodelle möglich sein kann.

Im Abschnitt 6.1 werden zunächst die in der Literatur vorgeschlagenen Komplexitätsmetriken für GPM vorgestellt und bewertet. In Abschnitt 6.2 werden Arbeiten zur Validierung der bekannten Metriken vorgestellt.

6.1 In der Literatur vorgestellte Komplexitätsmetriken für GPM

Für Software in höheren Programmiersprachen sind zahlreiche Komplexitätsmetriken bekannt, die seit Jahren erfolgreich eingesetzt werden. In [61] und [62] untersuchten

wir, wie sich die diesen Metriken zugrunde liegenden Ideen auf Modelle von Geschäftsprozessen übertragen lassen. Unabhängig von unseren Forschungen kamen in [25] Cardoso, Mendling, Neumann und Reijers zu sehr ähnlichen Vorschlägen.

In den folgenden Abschnitten werden wichtige aus der Software-Komplexitätsmessung bekannten Metriken betrachtet. Es wird dargestellt, wie sich die diesen Komplexitätsmetriken zugrundeliegenden Ideen auf die Bewertung von GPM übertragen lassen.

6.1.1 Größe des Modells

Die einfachste Komplexitätsmetrik für Software bestimmt die Zahl der Codezeilen (Lines of Code, LOC) bzw. die Zahl der ausführbaren Anweisungen. Die LOC stellen somit die „Programmlänge“ dar, und es ist sofort einsichtig, dass dies eine - wenn auch sehr einfache - Komplexitätsmetrik ist.

Für Software gelten die LOC als Metrik mit nur beschränkter Aussagekraft, da diese Metrik wesentliche Dinge nicht abbilden kann. Es lohnt sich für graphische Modelle aber durchaus, Metriken zu betrachten, die zu den LOC analog sind. Für Geschäftsprozessmodelle ist das etwa die „Zahl der im Modell abgebildeten Knoten“. Es ist naheliegend, dass Modelle mit weniger Aktivitäten leichter zu verstehen sind als Modelle mit vielen Aktivitäten. Allerdings ist es kaum möglich, allgemeingültige Richtlinien aufzustellen, ab welcher Größe ein Modell „zu groß“ ist und in mehrere Teilmodelle geteilt werden sollte. Smith nennt in [153] 40-50 Knoten als maximale Obergrenze für gut verständliche GPM. In [117] wird aus der Tatsache, dass große GPM häufiger Fehler enthalten als kleine, die generelle Modellierungsrichtlinie abgeleitet, so wenig Modellelemente zu nutzen wie möglich. Dieser Ratschlag scheint allerdings zu stark vereinfacht zu sein. Wichtiger als die Größe der Teilmodelle ist nämlich, dass ein Modell den zu beschreibenden Prozess vollständig und mit einheitlichem Abstraktionsgrad wiedergibt. Die Zahl und Art der Modellelemente muss also gerade dem Zweck der Modellierung angemessen sein.

Weiterhin sagt die bloße Zahl der Aktivitäten in einem Geschäftsprozessmodell noch nichts über den Kontrollfluss zwischen den Aktivitäten (also bedingte Verzweigungen und Parallelität) aus. Da aber gerade alternative und parallele Abläufe das Verständnis eines Modells erschweren können, betrachten wir im folgenden Abschnitt eine Metrik zur Messung der Komplexität der Kontrollflussstruktur.

6.1.2 Kontrollflussmetriken nach Cardoso

McCabe [107] schlug die zyklomatische Zahl als Komplexitätsmetrik für Computerprogramme vor. Sie geht vom Kontrollflussgraphen aus und zählt die möglichen Wege, die im Kontrollflussgraphen zurückgelegt werden können. Eine exakte Definition dieser Metrik findet sich in [107]; informell reicht es zu sagen, dass die zyklomatische Zahl eines Kontrollflussgraphen der Zahl der binären Entscheidungen (also der `if`-Anweisungen in einer höheren Programmiersprache) plus 1 entspricht. Nicht-binäre Entscheidungen (also etwa `select` oder `case`-Anweisungen in einer höheren Programmiersprache) mit n möglichen Ausgängen werden wie n-1 binäre Entscheidungen behandelt. Die McCabe-Metrik gibt also Aufschluss über die Zahl der Verzweigungen im Kontrollflussgraphen.

Einfache Strukturen im Kontrollfluss (gemessen durch eine niedrige zyklomatische Zahl) führen zu einer besseren Verständlichkeit eines Programms und (da weniger Testfälle benötigt werden) zu einer einfacheren Testbarkeit. Grady [57] zeigte einen engen Zusammenhang zwischen der zyklomatischen Zahl nach McCabe und der Fehlerrate in Programmen.

Cardoso [26] leitet von der McCabe-Metrik eine analoge Metrik für Geschäftsprozessmodelle ab. Um diese zu bestimmen, sind lediglich für jeden Split im Modell die Zahl der möglichen verschiedenen Kontrollflüsse, die von diesem Split ausgehen können, zu addieren.

Die Zahl der verschiedenen Kontrollflüsse ist:

- für einen AND-Split: 1 (Es müssen immer alle folgenden Pfade parallel bearbeitet werden.)
- für einen XOR-Split mit n Ausgängen: n (Genau einer der n Pfade muss genommen werden, dafür gibt es gerade n Auswahlmöglichkeiten.)
- für einen OR-Split mit n Ausgängen: $2^n - 1$ (Das entspricht den Möglichkeiten, mindestens einen und höchstens n zu durchlaufende Pfade auszuwählen.)

Ein erster (wenn auch wegen seines geringen Umfangs noch wenig aussagekräftiger) Labortest von Cardoso legte nahe, dass diese Metrik geeignet ist, die (subjektiv empfundene) Komplexität eines Geschäftsprozessmodells zu messen [26]. Spätere Untersuchungen [121, 110, 113] ergaben aber keinen statistisch signifikanten Zusammenhang zwischen Cardosos Metrik und tatsächlich im Modell enthaltenen Fehlern. Ebenso verneint [116] einen statistisch relevanten Zusammenhang zwischen Cardosos Kontrollflussmetrik und dem in einem Laborversuch gemessenen tatsächlichen Verständnis von EPK-Modellen.

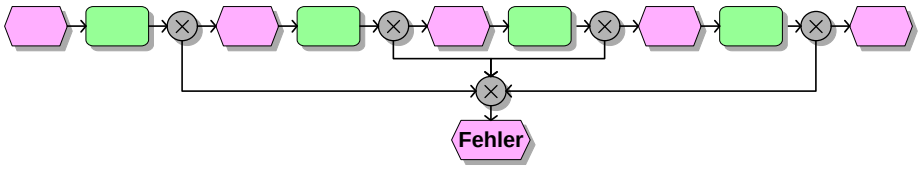


Abbildung 6.1: „fast sequentielles“ Modell (Metrik nach Cardoso = 8)

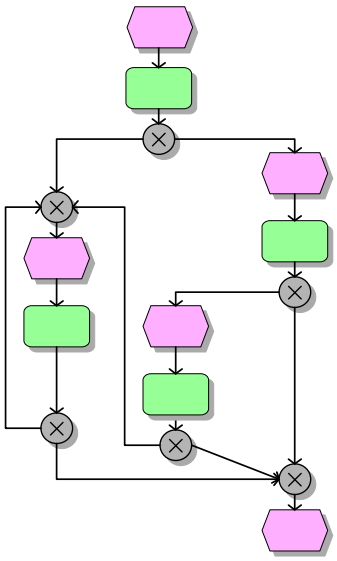


Abbildung 6.2: unstrukturiertes Modell (Metrik nach Cardoso = 8)

Eine Unzulänglichkeit dieser Metrik liegt darin begründet, dass das bloße Zählen der Entscheidungsmöglichkeiten noch zu wenig über die Struktur des Modells aussagt. Von dieser Struktur hängt aber die Verständlichkeit eines Modells entscheidend ab. So zeigen Abb. 6.1 und 6.2 zwei Modelle, deren zyklomatische Zahl gleich ist, da beide dieselbe Zahl binärer Entscheidungen enthalten. Trotzdem ist das eher lineare Modell in Abb. 6.1 deutlich einfacher. Umgekehrt zeigt Abb. 6.3 auf Seite 67 zwei Modelle mit jeweils einem OR-Split/OR-Join-Konstrukt. Für das linke Modell ergibt die Metrik nach Cardoso 3, für das rechte Modell 15. Ein solch deutlicher Unterschied erscheint ungerechtfertigt, wenn wir die Verständlichkeit eines Modells messen wollen. Es kann davon ausgegangen werden, dass die Funktionen zwischen Split und Join vom Modellierer gedanklich zu einer Einheit zusammengefasst werden, deren Verständnis nicht wesentlich von der Zahl der Funktionen abhängt (vgl. [122]).

In den folgenden Abschnitten werden wir daher zusätzliche Komplexitätsmetriken

betrachten, die die Struktur der Modelle in die Berechnung der Komplexität einbeziehen.

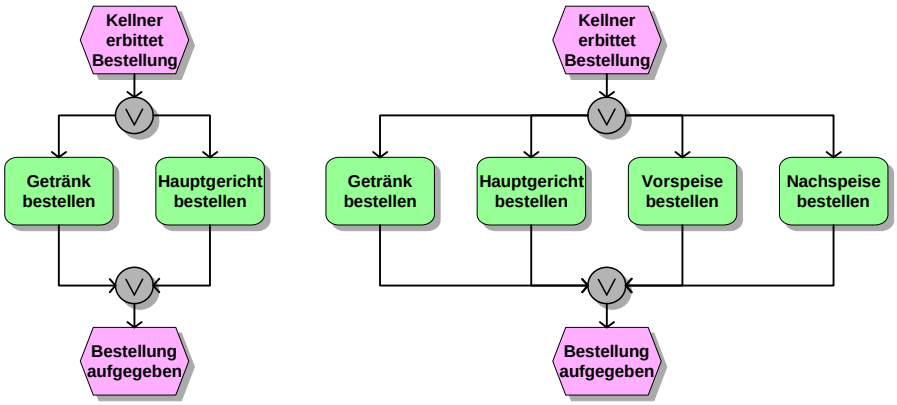


Abbildung 6.3: zwei Modelle mit deutlich verschiedener Metrik nach Cardoso

6.1.3 Innere Struktur des Modells: Verschachtelungstiefe und Strukturiertheit

Für beide Modelle in Abb. 6.1 und 6.2 ergibt sich nach Cardoso eine Metrik von 8. Trotzdem dürfte unstrittig sein, dass Abb. 6.1 ein deutlich einfacheres Modell zeigt als Abb. 6.2. Der Grund hierfür liegt darin, dass Abb. 6.1 einen „nahezu sequentiellen“ Kontrollfluss darstellt, während in Abb. 6.2 mehrere XOR-Split/XOR-Join-Konstrukte ineinander verschachtelt sind. Um den „Grad der Verschachtelung“ zu messen, sind für Kontrollstrukturen von Software die Metriken „maximale Verschachtelungstiefe“ (maximum nesting depth) und „mittlere Verschachtelungstiefe“ (mean nesting depth) bekannt. Die Bedeutung dieser Metrik wurde von Schroeder in [150] diskutiert.

Eine Übertragung dieser Metriken auf Geschäftsprozessmodelle ist leicht möglich: Maximale und mittlere Verschachtelungstiefe beschreiben dann die maximale/mittlere Zahl von Kontrollflussentscheidungen, die getroffen werden müssen, damit ein Knoten des Modells durchlaufen werden kann. Eine formale Definition findet sich in [113].

Während in Abb. 6.1 die maximale Verschachtelungstiefe eins ist (jeder aufgetretener Fehler führt zur Fehlerbehandlung), ist die maximale Verschachtelungstiefe in Abb. 6.2 drei. Die beiden Modelle in Abb. 6.1 und 6.2 unterscheiden sich also in der Verschachtelungstiefe, was Einfluss auf ihre Verständlichkeit hat.

Weiterhin ist zu beachten, dass als EPK oder in anderen GPM-Modellierungssprachen erstellte Modelle in der Regel nicht wohlstrukturiert sein müssen. Verzweigungen und Zusammenführungen des Kontrollflusses müssen also nicht paarweise auftreten, was das Verständnis ebenfalls erschweren kann.

Diese Modellierungssprachen sind vergleichbar mit Programmiersprachen, die Schleifen nicht nur in wohlstrukturierter Art und Weise (`repeat...until` etc.), sondern mit beliebigen `GOTO`-Sprüngen erlauben. Woodward, Hennell und Hedley [194] führten zur Messung der Häufigkeit solcher (nicht wünschenswerter) „Ausprünge“ aus Kontrollstrukturen (bzw. ihrem Gegenstück, den „Einsprüngen“ in eine Kontrollstruktur) in Software-Quelltexten die Knotenzähl- (knot count)-Metrik ein. Informell gesprochen, hat ein Kontrollflussgraph einen Knoten, wenn sich Pfade, entlang derer Kontrollfluss stattfinden kann, überschneiden. Dies ist beispielsweise bei dem durch zwei Sprunganweisungen erzeugbaren Kontrollflussgraphen in Abb. 6.4 der Fall.

Abbildung 6.4: Kontrollfluss mit einem Knoten

Während diese Metrik für Programmiersprachen ausreichend ist, gibt es bei EPK-Modellen noch zu bedenken, dass Verzweigungen und Zusammenführungen im Kontrollfluss durch Konnektoren mit drei verschiedenen möglichen Typen möglich sind. Unstrukturiertheit liegt daher auch vor, wenn der Typ eines Splits nicht mit dem Typ eines zugehörigen[1] Joins übereinstimmt. Eine Metrik zur Messung der Strukturiertheit für EPKs ist somit schwieriger zu realisieren. Diesem Thema ist mit Kapitel 7 ein eigener Abschnitt dieser Arbeit gewidmet.

6.1.4 Stärke des Zusammenhangs zwischen Modellteilen

Wie bereits ausgeführt, kann man annehmen, dass ein graphisches Modell für den Leser leichter verständlich ist, wenn er es gedanklich leicht in Teilmodelle zerlegen kann. Um diese Eigenschaft von EPKs zu messen, wurden verschiedene Metriken definiert.

Mendling schlug in [113] eine Metrik mit der Bezeichnung Separierbarkeit vor. Dazu

[1] Was genau unter „zugehörig“ verstanden werden soll, wird in der Definition der match-Relation auf Seite 18 erklärt.

untersuchte er die Anzahl der Kanten einer EPK, die eine Brücke (entl. cut-vertex) im Graphen darstellen. Diese sind wie folgt definiert:

Definition 19 (Brücke)
Sei (K, A) ein schwach zusammenhängender gerichteter Graph. $a \in A$ heißt Brücke in (K, A), wenn der Graph $(K, A \setminus \{a\})$ nicht mehr schwach zusammenhängend ist.

Eine Kante ist also Brücke, wenn durch ihr Entfernen aus dem Graphen die Eigenschaft des Zusammenhangs zerstört wird. Somit ermöglicht jede Brücke in einer EPK auch das gedankliche Aufteilen der EPK in zwei Teilmodelle. Als Separierbarkeit definiert Mendling dann den Quotienten zwischen der Zahl der Brücken in einer EPK und der Gesamtzahl der Kanten in dieser EPK.

Ein ähnliches Ziel verfolgt die in [112] vorgestellte Dichte-Metrik. Sie ist definiert als das Verhältnis zwischen der Zahl der Kanten im Modell und der Zahl der theoretisch möglichen Kanten in einer EPK mit einer Knotenzahl, die der Knotenzahl der zu untersuchenden EPK entspricht.

Vanderfeesten et al. schlagen in [180] eine sog. Cross-Connectivity-Metrik vor. Dafür definieren sie ein Maß für jeden Pfad im Modell. Dieses Maß soll die Schwierigkeit, diesen Pfad gedanklich nachzuvollziehen, quantifizieren. Ähnlich zur Metrik von Cardoso (siehe Abschnitt 6.1.2) wird dabei angenommen, dass AND-Konnektoren leichter verständlich sind als XOR- und OR-Konnektoren. Die Konnektoren erhalten daher verschiedene Gewichtungen. Es scheint derzeit keine fundierte Begründung dafür zu existieren, warum die Gewichtungen gerade so wie in der Veröffentlichung angegeben sinnvoll sind.

Grund zur Kritik an der Cross-Connectivity-Metrik gibt beispielsweise die Tatsache, dass sie etwa die in Abb. 6.5 gezeigten Modelle sehr deutlich verschieden bewertet.

6.1.5 Erfassbarkeit des Modells: kognitive Komplexitätsmetriken

Im Abschnitt 6.1.2 wurde an zwei Beispielen die Unzulänglichkeit der in [26] vorgeschlagenen Metrik kritisiert. Die in Abschnitt 6.1.3 genannten zusätzlichen Metriken erlauben es, zwischen den im ersten Beispiel (Abb. 6.1 und 6.2) dargestellten Modellen zu differenzieren.

Sehen wir uns nun das in Abb. 6.3 auf Seite 67 gezeigte zweite Beispiel an: Beide Modelle zeigen eine Kontrollstruktur, bei der eine oder mehrere Kontrollflusszweige parallel ausgeführt werden. Der zum Verständnis eines solchen Konstrukts nötige Aufwand hängt nur unwesentlich davon ab, ob 2, 4 oder 10 Kontrollflusszweige zwischen OR-Split und OR-Join liegen, da das gesamte Konstrukt als eine Einheit

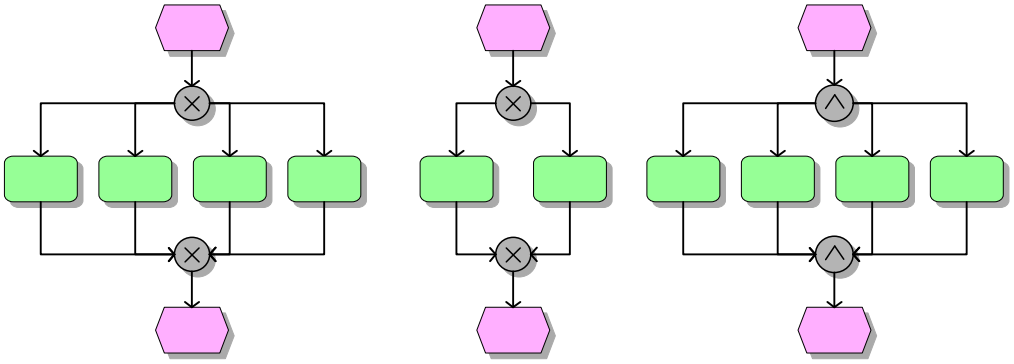


Abbildung 6.5: Beispiele für die Cross-Connectivity-Metrik

betrachtet wird. Shao und Wang [152] schlugen das kognitive Gewicht (cognitive weight) als eine Komplexitätsmetrik für Software vor. Diese Komplexitätsmetrik orientiert sich an den kognitiven Fähigkeiten des Menschen. Shao und Wang ordneten verschiedenen Kontrollstrukturen (z.B. „if-Anweisung“, „Unterprogrammaufruf“ oder „rekursiver Aufruf“) ein kognitives Gewicht zu, das den geistigen Aufwand, ein entsprechendes Konstrukt zu erfassen, wiedergibt. So wird etwa eine Sequenz von aufeinanderfolgenden Anweisungen ohne dazwischenliegende Verzweigungen mit dem kognitiven Gewicht 1 bewertet, `if-then`-Strukturen erhalten das Gewicht 2 und ein rekursiver Aufruf das Gewicht 3. Das kognitive Gewicht einer Softwarekomponente wird dann definiert als Summe der kognitiven Gewichte ihrer Bestandteile.

In [60] schlugen wir eine Übertragung des Ansatzes der kognitiven Gewichte auf GPM vor. Bevor eine solche Metrik angewendet werden kann, sind jedoch umfangreiche Experimente zur Bestimmung sinnvoller kognitiver Gewichte nötig, was außerhalb des Schwerpunkts dieser Arbeit lag. Solche Experimente existieren derzeit auch für den ursprünglichen Anwendungsfall der Komplexitätsmessung von Software nicht in zufriedenstellender Qualität, wie wir in [66] gezeigt haben. Somit bilden kognitive Gewichte zur Komplexitätsmessung einen zwar interessanten, aber derzeit praktisch noch nicht einsetzbaren Ansatz.

6.1.6 Muster und Anti-Muster

Die den im vorherigen Abschnitt besprochenen kognitiven Gewichten zugrundeliegende Idee ist es, häufig auftretende Muster als eine Einheit zu betrachten,

da sie dem Leser eines Modells als ein zusammenhängendes Gebilde verständlich sind. Einen verwandten Ansatz verfolgt auch Gustafsson [70]. Er untersucht, in welcher Zahl bekannte Designmuster (etwa aus der Sammlung [52]) in einem UML-Modell von Software vorkommen. Dahinter steht die These, dass die Verwendung von erprobten und dokumentierten Designmustern die Verständlichkeit, Wartbarkeit und Zuverlässigkeit einer Software erhöht. Wird die Verwendung solcher Muster entdeckt, die als zuverlässig bekannt sind, so wird eine positive Auswirkung auf die Komplexität angenommen. Diese Überlegung ist sicher nicht in allen Fällen zutreffend (viele verwendete Design-Muster garantieren noch kein gutes Programm), hat jedoch eine gewisse Berechtigung. In jedem Falle erfordern Metriken, die die Zahl auftretender Muster zählen, eine umfangreiche Erfahrung mit diesen Mustern, um sinnvoll gedeutet werden zu können.

Neben den soeben diskutierten „guten“ Designmustern untersucht [70] auch das Gegenteil: als „schlecht“ bekannte Designmuster (sog. Anti-Patterns), die auf mangelhafte Programmierung hindeuten.

Eine Übertragung der Gedanken aus [70] auf Geschäftsprozessmodelle scheint sehr sinnvoll. Insbesondere ist zu erwarten, dass das Entdecken der „schlechten“ Anti-Patterns hilft, Verbesserungsmöglichkeiten in einem Modell aufzudecken. In Kapitel 9 wird dieser Gedanke weiter verfolgt und ein umfangreicher Musterkatalog vorgestellt.

6.1.7 Zusammenfassende Übersicht

Tabelle 6.1 auf Steite 73 fasst die Aussagen der vorherigen Abschnitte zusammen: Bekannten Metriken zur Messung der Komplexität von *Software* werden analoge Metriken zur Messung der Komplexität von *Geschäftsprozessmodellen* gegenübergestellt und kurz bewertet. Da sich die Ideen der genannten Software-Komplexitätsmetriken in der Praxis bewährt haben, ist zu erwarten, dass die Anwendung analoger Ideen im Bereich der Geschäftsprozessmodellierung ebenfalls zu guten Ergebnissen führt.

Wie auch bei der Messung von Softwarekomplexität, gibt es auch für Geschäftsprozessmodelle nicht eine einzige universelle Metrik, die alle wesentlichen Einflussgrößen berücksichtigt. Es ist daher sinnvoll, die Metriken kombiniert anzuwenden.

Sämtliche betrachtete Metriken sind relativ leicht maschinell berechenbar und können z.B. in einen Modell-Editor integriert werden. In [68] zeigten wir die Integration einer Auswahl mehrerer Komplexitätsmetriken in das EPK-Modellierungswerkzeug *EPCTools* (siehe Abb. 6.6). Bei Verwendung dieses Werkzeugs wird der Modellierer beim Überschreiten von zuvor festgelegten Schwellwerten für die berechneten Metriken gewarnt. Einzelheiten zur Implementierung sind in [121] beschrieben.

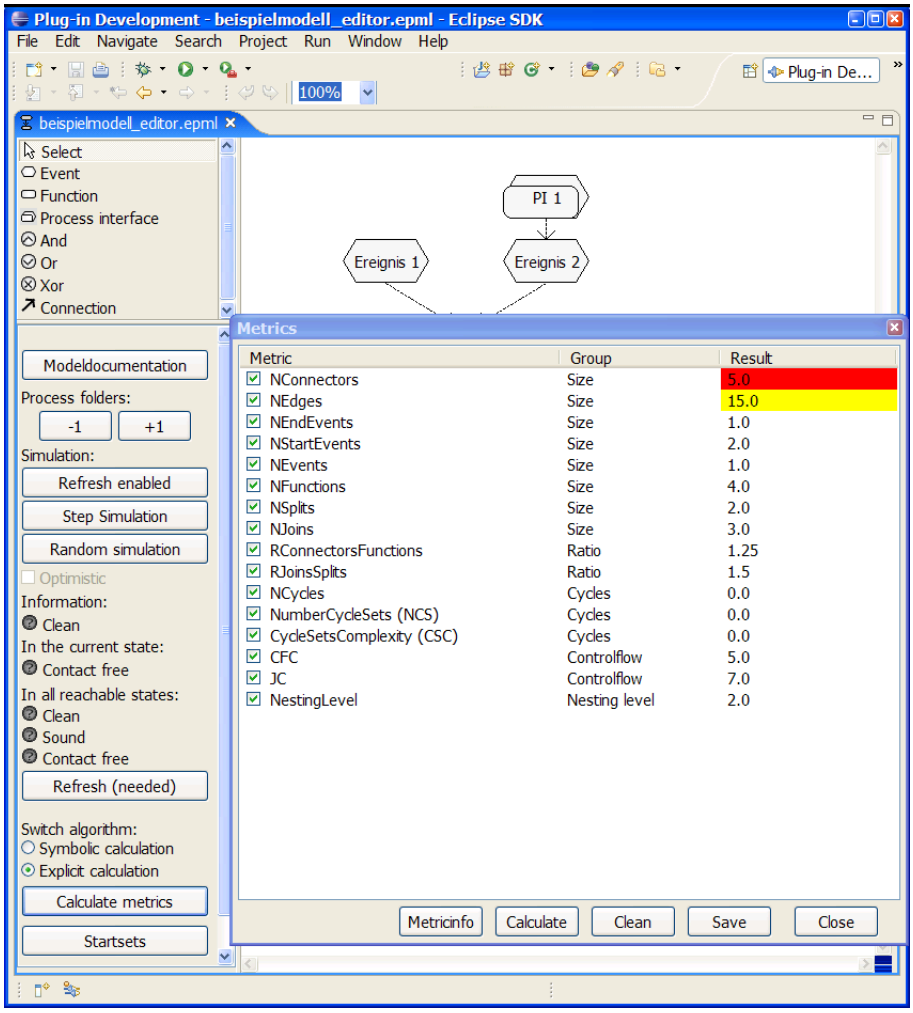


Abbildung 6.6: Integration von Metriken in das Modellierungswerkzeug *EPCTools*

Auch das Werkzeug *ProM* [9], das u.a. der Analyse von GPM dient, enthält ein Modul zur Berechnung von Komplexitätsmetriken.

Die vorgestellten Metriken sind (ggf. mit kleinen Variationen) unabhängig von der verwendeten Modellierungssprache einsetzbar.

Software-Komplexitäts-metrik	**entsprechende Metrik für GPM**	**Anwendung, Bewertung**
Lines of Code	Zahl der Aktivitäten	sehr einfache Metrik, Kontrollfluss nur ungenügend berücksichtigt, Vorgabe einer Höchstzahl von Aktivitäten in einem Modell kann aber sinnvoll sein
zyklomatische Zahl [107]	Metrik nach Cardoso [26]	berücksichtigt Komplexität an Entscheidungsknoten, daher gut geeignet, um die Zahl der zum Testen nötigen Testfälle zu bestimmen Weitere das Verständnis beeinflussende Informationen zur Struktur bleiben unberücksichtigt.
maximale bzw. mittlere Verschachtelungstiefe	maximale bzw. mittlere Verschachtelungstiefe	gut geeignet
Knotenzähl-Metrik [194]	siehe Kapitel 7	misst „Strukturiertheit“ (Ein- und Aussprünge in bzw. aus Kontrollstrukturen), ist bei wohlstrukturierten Modellen stets 0
kognitives Gewicht [152]	siehe [60]	misst den vom Leser eines Modells zu erfassenden „Informationsgehalt“ des Modells, ist daher ein guter Anhaltspunkt dafür, ob ein Modell modularisiert werden sollte, Derzeit fehlt noch eine experimentelle Grundlage.
(Anti)-Patterns [130]	siehe Kapitel 9	Zahl der im Modell vorkommenden Anti-Patterns gut geeignet, um mangelhafte Modellierung aufzudecken

Tabelle 6.1: Komplexitätsmetriken für Software und Geschäftsprozessmodelle

6.2 Metriken und Modellqualität - Bekannte Ergebnisse

Mendling [113] untersuchte, inwiefern ein statistisch relevanter Zusammenhang zwischen GPM-Komplexitätsmetriken und Fehlern in EPK-Modellen besteht. Als Fehler wurde dabei die Verletzung der Soundness-Eigenschaft nach Def. 17 unter Zugrundelegung der in [113] definierten Übergangsrelation betrachtet. Hierzu berechnete Mendling die Werte für 30 Metriken für einen Satz von 2003 Modellen, von denen 215 nicht sound waren. Anschließend untersuchte er die Korrelation zwischen jeder dieser Metriken und und der binären (zweiwertigen) Variablen, die den Wert 1 annimmt, falls das Ereignis „Die EPK ist sound“ eintritt, andernfalls den Wert 0.

Mendling berechnete den Rangkorrelationskoeffizienten nach Spearman für den Zusammenhang zwischen jeder der 30 betrachteten Metriken und der binären Variablen, welche die Soundness des Modells beschreibt.

Für zwei der betrachteten Metriken lag der Betrag des berechneten Rangkorrelationskoeffizienten unter 0,2, was laut [22] als „sehr geringe Korrelation“ interpretiert wird. Für alle anderen Metriken lag der Betrag unter 0,5, was [22] als „geringe Korrelation“ bezeichnet. Der Höchstwert von 0,48 wurde für die Zählmetrik „Zahl der AND-Joins“ erreicht.

In [180] wurde eine analoge Analyse auch für die Cross-Connectivity-Metrik durchgeführt, wobei sich ein Rangkorrelationskoeffizient von -0,434 ergab. Damit schnitt diese Metrik fast ebenso ab wie die deutlich einfachere Zählmetrik „Zahl der Konnektoren im Modell“, für die ein Rangkorrelationskoeffizient von 0,43 berechnet wurde.

Zur Untersuchung der Abhängigkeit der Soundness-Eigenschaft von mehreren Metriken (also mehreren unabhängigen Variablen) nutzte Mendling das Verfahren der logistischen Regression. Er bezog 30 Komplexitätsmetriken in seine Untersuchungen ein. Mit diesen erstellte er ein Regressionsmodell, das aus den Werten von sieben der Komplexitätsmetriken die Soundness einer EPK mit einer Korrektheit von 95,2% voraussagen kann.

Während die vorstehenden Untersuchungen den Zusammenhang zwischen den Werten von Komplexitätsmetriken und der Soundness-Eigenschaft untersuchten, befassten sich [116] und [118] mit dem Zusammenhang von Metriken und dem tatsächlichen Verständnis von GPM. Hierzu wurden einer Gruppe von Modellierern GPM vorgelegt, zu denen Fragen zu beantworten waren. Daraufhin wurde geprüft, ob ein statistisch relevanter Zusammenhang zwischen den Werten der Komplexitätsmetriken und der Zahl der richtigen und falschen Antworten existiert. Nur für zwei Metriken,

nämlich für die Dichte (siehe Abschnitt 6.1.4) und die Metrik „durchschnittliche Zahl der Ein- bzw. Ausgangskanten eines Konnektors“, zeigte sich in [116] ein statistisch relevanter Zusammenhang mit dem auf diese Weise gemessenen Modellverständnis. In [118] ergab sich nur für die in Abschnitt 6.1.4 beschriebene Separabilitäts-Metrik ein statistisch relevanter Zusammenhang. Beide Studien wurden dafür kritisiert, dass die zu beantwortenden Fragen möglicherweise nicht alle wichtigen Aspekte des Modellverständisses abbildeten [109, 108].

Im nächsten Kapitel wird eine neue Metrik eingeführt, die den Grad der (Un)strukturiertheit einer EPK misst. Auch für diese Metrik werden wir den Zusammenhang zwischen dem Wert der Metrik und der Soundness-Eigenschaft analysieren.

7 Unstrukturiertheit - Definition und Messung

In der Literatur wurde bislang mit Mendlings *Degree of Structuredness* eine einzige Metrik vorgeschlagen, die den Grad der Strukturiertheit von Modellen misst. Im folgenden Abschnitt wird diese Metrik vorgestellt und diskutiert. Insbesondere werden einige Unzulänglichkeiten dieser Metrik benannt. Anschließend wird eine eigene Metrik zur Messung von Unstrukturiertheit entwickelt, die diese Nachteile nicht aufweist. Im Anschluss daran wird eine Untersuchung der Eignung der vorgeschlagenen Metrik zur Vorhersage von Fehlern in GPM vorgestellt.

7.1 Wohlstrukturiertheit

In dieser Arbeit wird angestrebt, verschiedene Arten von Unstrukturiertheit in EPKs zu unterscheiden ihre Auswirkungen zu untersuchen. Zunächst soll jedoch der Begriff der Wohlstrukturiertheit für EPKs definiert werden. Anschaulich gesprochen, verstehen wir unter Wohlstrukturiertheit, dass nur Konstrukte aus paarweise auftretenden Split- und Joinkonnektoren gleichen Typs sowie Iterationen im Modell vorkommen. Die „erlaubten“ Konstrukte sind in Abb. 7.1 gezeigt. Hierbei wurden Ereignisse und Funktionen weggelassen, da es bei der Definition des Begriffs „Wohlstrukturiertheit“ lediglich auf die Anordnung der Konnektoren ankommt.

Eine wohlstrukturierte EPK entsteht durch das wiederholte Zusammenfügen von Konstrukten, wie sie in Abb. 7.1 abgebildet sind (und dem Hinzufügen von Funktionen und Ereignissen entsprechend der EPK-Semantikdefinition Def. 1).

Formal lässt sich die Bildung einer wohlstrukturierten EPK durch Angabe einer Graphgrammatik (siehe etwa [42]) definieren. Etwas einfacher gelingt jedoch eine auf Reduktionsregeln basierende Definition von wohlstrukturierten EPKs. Die Grundidee ist es, dass eine EPK wohlstrukturiert ist, wenn nach fortgesetztem Entfernen der in Abb. 7.1 abgebildeten Konstrukte aus dem Graphen ein „trivialer Graph“ übrig bleibt.

Die folgende Definition fasst die in Abb. 7.1 gezeigten Fälle zusammen und berücksichtigt auch die in dieser Abbildung nicht betrachteten Funktionen und

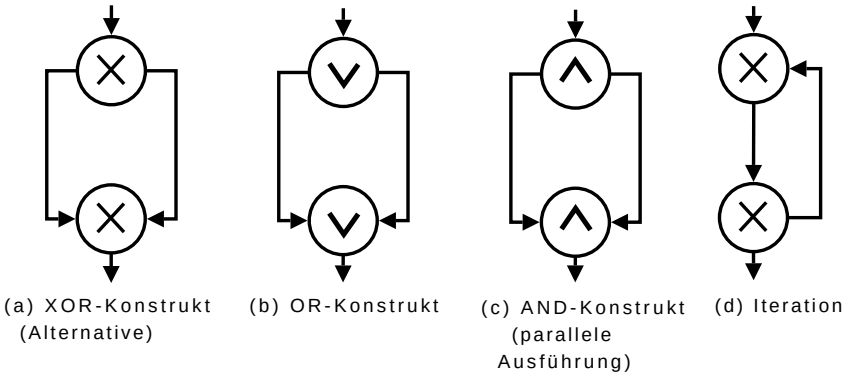


Abbildung 7.1: Bestandteile wohlstrukturierter EPKs

Ereignisse. Durch die Reduzierung können aus einer EPK Gebilde entstehen, die keine EPK sind (da z.B. nach Entfernen von Ereignissen und Funktionen der Graph nicht mehr einfach ist; zwischen Konnektoren gibt es dann mehr als eine Kante). Aus diesem Grunde arbeiten die folgenden Definitionen auf Multigraphen, bei denen es mehr als eine Kante zwischen zwei Knoten geben darf.

Definition 20 (Menge reduzierbarer Knoten)
Sei $\Lambda = (K, A)$ ein gerichteter Multigraph, dessen Knotenmenge wie in Def. 1 in paarweise disjunkte Mengen von Funktionen, Ereignisse und Konnektoren aufgeteilt sei. Weiterhin sei für Konnektoren die Funktion type wie in Def. 1 erklärt.

Eine Teilmenge $\overline{K}$ der Knotenmenge K nennen wir eine Menge reduzierbarer Knoten, wenn einer der folgenden Fälle zutrifft:

1. *(Funktion)*
   $\overline{K} = \{f\}$, wobei f eine Funktion ist.
2. *(Ereignis, das weder Start- noch Endereignis ist)*
   $\overline{K} = \{e\}$, wobei e ein Ereignis mit $card \triangleright e = card\ e\triangleleft = 1$ ist.
3. *(Kontrollflussblock wie in Abb. 7.1 (a)-(c))*
   $\overline{K} = \{k_{in}, k_{out}\}$, wobei k_{in} ein Split und k_{out} ein Join ist und gilt: $type(k_{in}) = type(k_{out})$. Ferner gibt es kein $(k_{in}, x) \in A$ mit $x \neq out$ sowie kein $(x, k_{out}) \in A$ mit $x \neq k_{in}$.
4. *(Iteration wie in Abb. 7.1 (d))*
   $\overline{K} = \{k_{in}, k_{out}\}$, wobei k_{in} ein Join und k_{out} ein Split ist und gilt:
   $type(k_{out}) = type(k_{in}) = XOR$ sowie $card\ k_{in}\triangleleft = card \triangleright k_{out} = 1$ und
   $card \triangleright k_{in} = card\ k_{out}\triangleleft = 2$. Ferner sei $(k_{in}, k_{out}) \in K$ sowie $(k_{out}, k_{in}) \in K$.

Die einzelnen Fälle sind in Abb. 7.2 dargestellt.

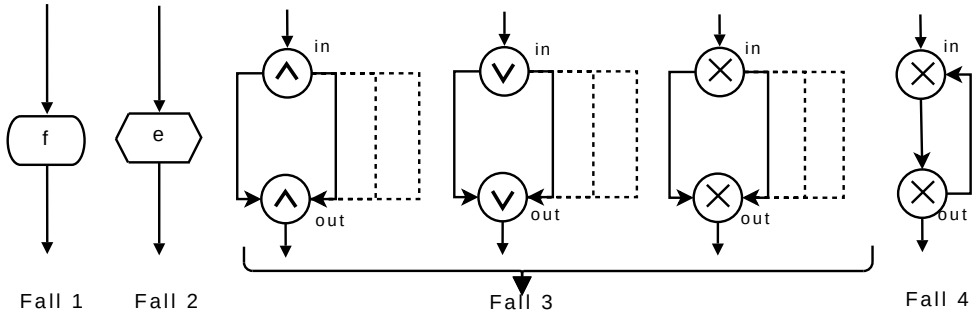


Abbildung 7.2: Mengen reduzierbarer Knoten lt. Def. 20

In allen genannten Fällen gibt es in der Menge $\overline{K}$ einen Knoten, der genau einen Vorgängerknoten in $K \setminus \overline{K}$ hat. Für die einelementigen Mengen in Fall 1 und 2 hat der einzige in $\overline{K}$ enthaltene Knoten (in der Definition als f bzw. e bezeichnet) genau einen Vorgängerknoten in $K \setminus \overline{K}$. In den Fällen 3 und 4 hat der jeweils mit k_{in} bezeichnete Knoten genau einen Vorgängerknoten in $K \setminus \overline{K}$. Wir bezeichnen diesen eindeutig bestimmten Knoten als Vorgängerknoten der Menge $\overline{K}$, dargestellt mit dem Symbol $pred(\overline{K})$.

Analog gibt es in allen Fällen in $\overline{K}$ einen Knoten, der genau einen Nachfolgeknoten in $K \setminus \overline{K}$ hat. In Fall 1 und 2 betrifft das wieder die Knoten f bzw. e, in den Fällen 3 und 4 die jeweils mit k_{out} bezeichneten Knoten. Wir bezeichnen diesen eindeutig bestimmten Knoten als Nachfolgeknoten der Menge $\overline{K}$, dargestellt mit dem Symbol $succ(\overline{K})$.

Eine Reduktion formt nun einen Graphen in einen Graphen mit weniger Knoten um, indem eine Menge reduzierbarer Knoten entfernt und durch eine Kante ersetzt wird:

Definition 21 (Reduktion)
Sei $\Lambda = (K, A)$ wie in Def. 20 definiert. Eine Reduktion bildet den Multigraphen (K, A) auf einen Multigraphen (K', A') ab, wobei gilt: $\overline{K} \subset K$ ist eine Menge reduzierbarer Knoten und

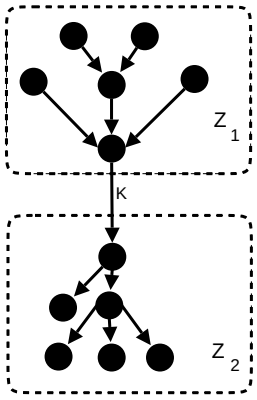
- *$K' = K \setminus \overline{K}$ (d.h. alle Knoten in $\overline{K}$ werden aus der Knotenmenge entfernt)*
- *$A' = \left(A \setminus \left\{(x, y) \in A | x \in \overline{K} \vee y \in \overline{K}\right\}\right) \cup \left\{(pred(\overline{K}, succ(\overline{K})\right\}$ (d.h. alle Kanten von oder zu einem Knoten aus $\overline{K}$ werden aus der Kantenmenge entfernt, und es wird eine neue Kante vom Vorgängerknoten von $\overline{K}$ zum Nachfolgeknoten von $\overline{K}$ zum Graphen hinzugefügt.)*

Hat eine EPK genau ein Start- und genau ein Endereignis, so bedeutet Wohlstrukturiertheit, dass sie sich durch wiederholte Reduktion in einen Graphen abbilden lässt, dessen Knotenmenge aus genau zwei Ereignissen besteht. Von einer wohlstrukturierten EPK können nämlich außer dem Start- und dem Endereignis alle Knoten durch Reduktionen entfernt werden.

Da jedoch EPKs mehr als ein Start- und mehr als ein Endereignis haben können, muss sich auch die Definition der Wohlstrukturiertheit auf EPKs mit mehreren Start- bzw. Endereignissen erstrecken.

Definition 22 (wohlstrukturierte EPK)
*Sei $\Lambda = (K, A)$ wie in Def. 20 definiert. Λ heißt **vollständig reduzierte EPK**, falls die Kantenmenge A eine Brücke (vgl. Def. 19, Seite 69) κ enthält, deren Entfernen aus dem Graphen dazu führt, dass seine Knotenmenge in zwei Teilmengen Z_1 und Z_2 zerfällt, für die gilt, dass alle in Z_1 enthaltenen Konnektoren Joins und alle in Z_2 enthaltenen Konnektoren Splits sind.*

*Eine EPK heißt **wohlstrukturiert**, wenn sie sich durch mehrfache Anwendung der in Def. 21 beschriebenen Reduktionsabbildung in eine vollständig reduzierte EPK überführen lässt.*



Anmerkung: In der Sprechweise der Graphentheorie ist Z_1 ein In-Tree, also ein gerichteter Baum, in dem es von jedem Knoten einen Pfad zur Wurzel gibt, und Z_2 ein Out-Tree, also ein gerichteter Baum, in dem es von der Wurzel aus einen Pfad zu jedem Knoten gibt.

Die Brücke κ ist eine Kante, die die Wurzeln der beiden Bäume verbindet.

Beispiel:
Abb. 7.3 zeigt, wie sich durch mehrfache Anwendung einer Reduktion eine wohlstrukturierte EPK in eine vollständig reduzierte EPK überführen lässt. Die Pfeile in der Abbildung zeigen jeweils auf das Konstrukt, das im jeweiligen Reduktionsschritt entfernt wird. Falls es mehrere mögliche Mengen reduzierbarer Knoten gibt, können die möglichen Reduktionen in beliebiger Reihenfolge angewendet werden.

Nachdem nun der Begriff der Wohlstrukturiertheit von EPKs formal eingeführt wurde, sollen in den folgenden Abschnitten Metriken betrachtet werden, die den „Grad der Wohlstrukturiertheit“ von GPM messen. In Abschnitt 7.2 wird die von Mendling für diesen Zweck vorgeschlagene Metrik „Degree of Structuredness“ vorgestellt und bewertet. Anschließend wird in Abschnitt 7.3 eine neue Metrik zur

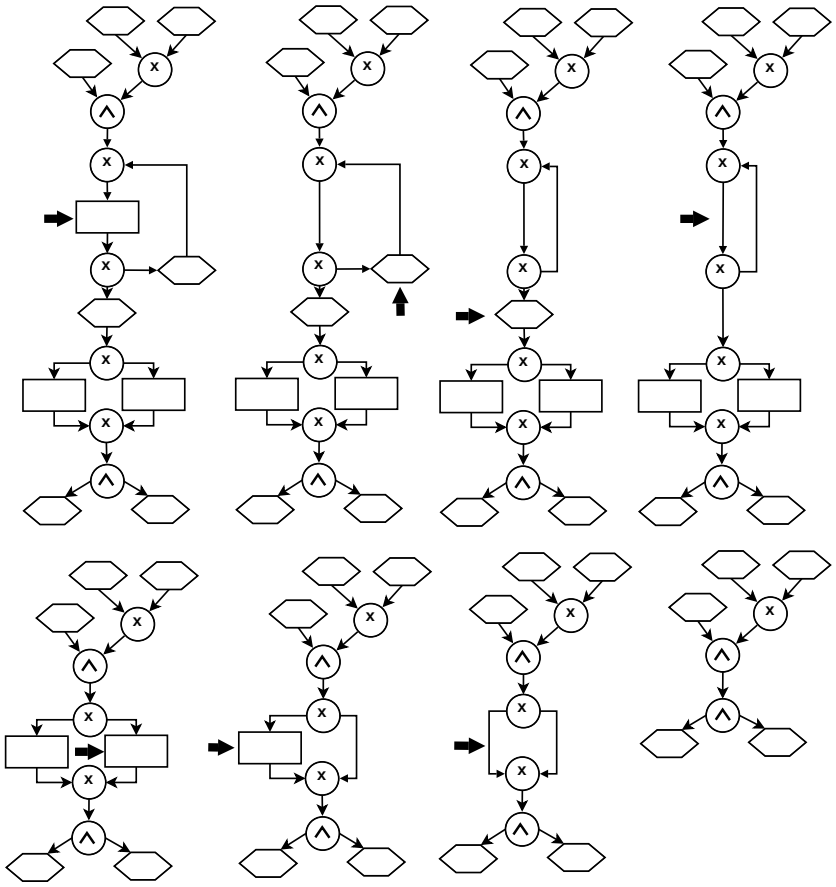


Abbildung 7.3: Reduktion einer wohlstrukturierten EPK

Messung der Unstrukturiertheit von GPM vorgeschlagen. Nach der Herleitung und Definition dieser Metrik erfolgt eine Validierung, in der ein statistisch signifikanter Zusammenhang zwischen der eingeführten Metrik und in GPM erkannten Fehlern nachgewiesen wird.

7.2 Mendlings Degree of Structuredness

Mendling schlug den *Degree of Structuredness* (DoS) als eine Metrik vor, die Unstrukturiertheit in EPKs messen soll [113, 98].

Diese Metrik beruht auf Reduktionsregeln ähnlich den im vorangegangenen Abschnitt vorgestellten. Mendling reduziert neben den in Def. 21 genannten Fällen auch mehrere Startereignisse, die alle Vorgänger desselben Joins sind sowie mehrere Endereignisse, die alle Nachfolger desselben Splits sind (am Beispiel von Startereignissen

gezeigt an Abb. 7.4). Weiterhin reduziert er einen Graphen, der nur aus Startereignis und Endereignis (verbunden durch eine Kante) besteht, zum leeren Graphen. Damit gelingt es, auch die bei der Reduktion einer wohlstrukturierten EPK entstehenden vollständig reduzierten EPKs vollständig „verschwinden zu lassen“[1].

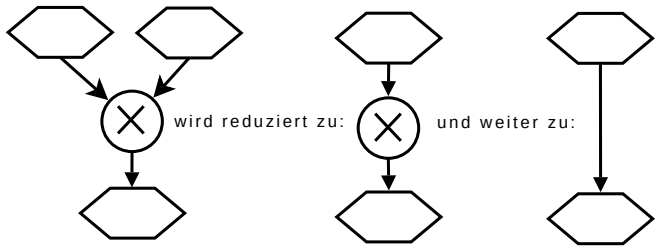


Abbildung 7.4: Reduktion von Startereignissen

Auf diese Weise wird eine wohlstrukturierte EPK immer zu einem leeren Graphen reduziert. Mendling definiert nun den *Degree of Structuredness*, abgekürzt DoS, durch $1 - \frac{card\ K_{reduziert}}{card\ K_{original}}$. Hierbei ist $K_{original}$ die Knotenmenge der ursprünglichen EPK und $K_{reduziert}$ die Knotenmenge in dem daraus entstandenen maximal reduzierten Graphen (im Sinne von: Es können für diesen Graphen keine weiteren Reduktionen mehr ausgeführt werden).

Es ist leicht zu sehen, dass für eine wohlstrukturierte EPK $K_{reduziert} = 0$ gilt und folglich der Degree of Structuredness 1 wird.

Die Idee dieser Metrik ist, dass sie umso näher bei 1 liegt, je mehr Knoten Teil eines Konstrukts aus Abb. 7.2 sind. Eine EPK wird als „besser strukturiert“ angesehen, je höher der Degree of Structuredness ist.

Dieser Ansatz hat zwei Nachteile: Zunächst sieht man am Beispiel von Abb. 7.5, dass bei verschachtelten Kontrollflussblöcken Unstrukturiertheit in einem äußeren Block anders behandelt wird als Unstrukturiertheit im inneren Block.

Das linke Modell von Abb. 7.5 zeigt eine wohlstrukturierte EPK, deren DoS 1 ist. Im mittleren Modell wurde eine zusätzliche Verbindung zwischen den beiden Kontrollflüssen des inneren Kontrollflussblocks hinzugenommen, im rechten Modell analog eine zusätzliche Verbindung zwischen den beiden Kontrollflüssen des äußeren Kontrollflussblocks. Im mittleren Modell lassen sich per Reduktion lediglich drei Funktionen entfernen, somit ist dessen DoS $1 - \frac{8}{11}$. Im rechten Modell jedoch lässt

[1] Die Reduktion eines Graphen, der nur aus Startereignis und Endereignis (verbunden durch eine Kante) besteht, zu einem leeren Graphen, ist in den in [113] angegebenen Reduktionsregeln nicht ausdrücklich erwähnt. Implizit wird eine solche Reduktion jedoch in [113] verwendet.

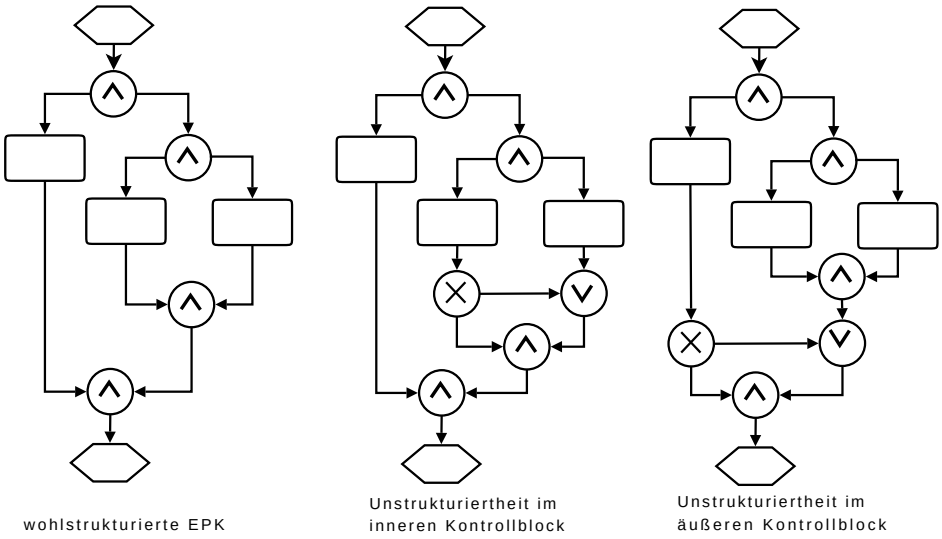


Abbildung 7.5: Modelle mit unterschiedlicher DoS-Metrik

sich darüber hinaus der gesamte innere Kontrollflussblock entfernen, was zu einem „besseren“ Wert des DoS von $1 - \frac{6}{11}$ führt. Ein solcher Unterschied scheint nicht gerechtfertigt.

Als weiterer Nachteil erweist sich, dass bei Verwendung der DoS-Metrik Modellvergrößerungen zu einer Senkung der Komplexität führen können: Fügt man einer nicht wohlstrukturierten EPK weitere Funktionen und Ereignisse (jedoch keine Konnektoren) hinzu, *verringert* sich deren DoS, da man diese Funktionen und Ereignisse per Reduktion entfernen kann. Im Bruch $\frac{card\ K_{reduziert}}{card\ K_{original}}$ erhöht sich zwar der Nenner, aber nicht der Zähler. Eine Komplexitätsmetrik sollte jedoch nie sinken, wenn dem Modell zusätzliche Elemente hinzugefügt werden [190, 6].

In [113] ermittelte Mendling den Rangkorrelationskoeffizienten nach Spearman für den Zusammenhang zwischen DoS und der Soundness einer EPK mit -0,36. Dies ist ein eher geringer Wert, der noch unter dem Rangkorrelationskoeffizienten für die einfache Metrik „Zahl der Elemente der EPK“ liegt.

Im kommenden Abschnitt wird eine Metrik zur Messung von Unstrukturiertheit vorgestellt, die die bei der DoS-Metrik auftretenden Nachteile vermeidet.

7.3 Zahl der unstrukturierten Konnektoren

In diesem Abschnitt wird die Zahl der unstrukturierten Konnektoren (unmatched connector count, UCC) als eine neue Komplexitätsmetrik für EPK-Modelle eingeführt.

Die dieser Metrik zugrundeliegende Idee ist es, eine Heuristik zu schaffen, die zu messen versucht, an wie vielen Stellen im Modell die Eigenschaft „Wohlstrukturiertheit“ verletzt ist. Die in Abschnitt 4.2.1 vorgestellten Graph-Reduktionsalgorithmen (auf denen auch der in Abschnitt 7.2 diskutierte Degree of Structuredness aufbaut) können lediglich eine globale ja/nein-Entscheidung treffen (Modell ist wohlstrukturiert / Modell ist nicht wohlstrukturiert). Demgegenüber soll unsere Metrik Unstrukturiertheit im Modell eher als lokale Eigenschaft betrachten. Sie soll messen, an wie vielen Stellen im Modell Konnektoren so angeordnet sind, dass sie nicht Bestandteil eines wohlstrukturierten Kontrollblocks sein können.

In einer wohlstrukturierten EPK nach Def. 22, in der nur Split und Joins gleichen Typs paarweise sauber verschachtelt vorkommen, gibt es für einen Split s genau drei Möglichkeiten:

1. s hat überhaupt keinen Pfad zu einem Join (dann verzweigt der Split s nur noch zu Endereignissen)
2. s startet einen strukturierten Kontrollblock zwischen einem Split und einem Join gleichen Typs (vgl. Abschnitt 7.1)
3. s ist ein Zyklenausstiegsknoten (vgl. Def. 7, Seite 17), der eine Iteration beendet.

Weiter gilt:

- Ist s Zyklenausstiegsknoten, so gibt es in diesem Zyklus keine weiteren Zyklenausstiegsknoten, ebenso gibt es genau einen Zykleneinstiegsknoten j. Weiterhin gilt $type(s) = type(j) = XOR$.
- Ist s kein Zyklenausstiegsknoten, so gibt es einen und nur einen Join j, für den $match(s, j)$ nach Def. 9 gilt. Für dieses Paar (s, j) stimmen die Typen der Konnektoren überein, es gilt also $type(s) = type(j)$.

Wird eine der soeben genannten Forderungen verletzt, soll der betroffene Split *unstrukturiert* heißen. Formal erklären wir den Begriff des unstrukturierten Split-Konnektors in der folgenden Definition. Die Nummerierung durch die Buchstaben a) bis i) in dieser Definition stimmt mit den entsprechenden Modellfragmenten in Abb. 7.6 überein, in der Beispiele unstrukturierter Split-Konnektoren zusammengestellt sind.

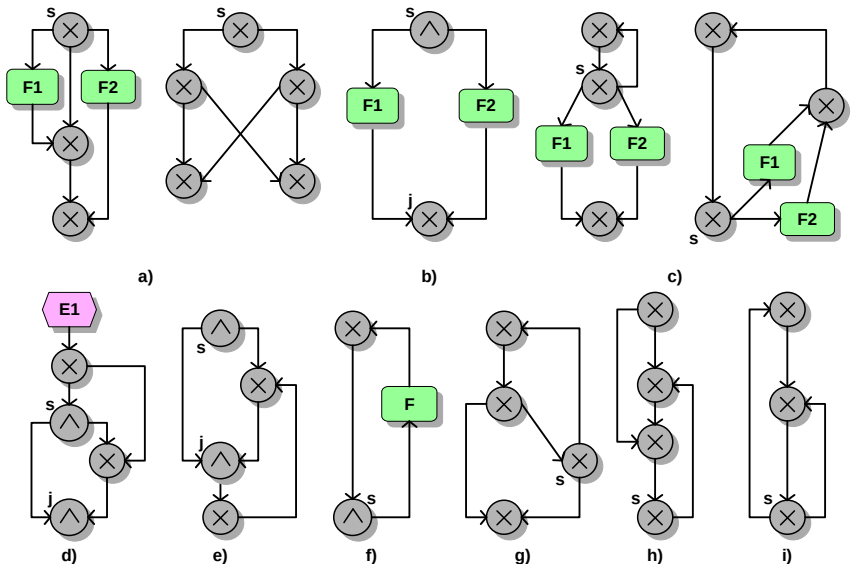


Abbildung 7.6: Unstrukturierte Modellfragmente lt. Def. 23

Definition 23 (unstrukturierter Split)
Sei $\Lambda = (K, A)$ ein EPK-Modell. Ein Split $s \in K$ heißt unstrukturiert, wenn mindestens eine der folgenden Eigenschaften erfüllt ist:

1. *Es gibt einen Join j, für den $match(s, j)$ gilt, und eine der folgenden Bedingungen gilt:*
   a) *Neben j gibt es einen weiteren Join $j' \neq j$, für den $match(s, j')$ gilt.*
   b) *j hat einen anderen Typ als s, d.h. $type(s) \neq type(j)$.*
   c) *s ist Zyklenausstiegsknoten.*
   d) *Es gibt einen Pfad von einem Startereignis zu j, der s nicht enthält, oder einen Pfad von s zu einem Endereignis, der j nicht enthält.*
   e) *Es gibt einen Zyklus von j nach j, der s nicht enthält, oder einen Zyklus von s nach s, der j nicht enthält.*
2. *s ist ein Zyklenausstiegsknoten, und eine der folgenden Bedingungen gilt:*
   f) *s ist nicht vom Typ XOR ($type(s) \neq XOR$).*
   g) *Ein Zyklus von s nach s besitzt neben s einen weiteren Zyklenausstiegsknoten $s' \neq s$.*
   h) *Ein Zyklus von s nach s besitzt mehr als einen Zykleneinstiegsknoten.*

i) s ist Zyklenausstiegsknoten aus zwei verschiedenen Zyklen Z_1 und Z_2. Ferner ist ein Join e Zykleneinstiegsknoten in den Zyklus Z_1 sowie ein Join f Zykleneinstiegsknoten in den Zyklus Z_2, und es gilt $e \neq f$.

Beispiele für die in der Definition angesprochenen Fälle werden in Abb. 7.6 gezeigt: Abb. 7.6 a) zeigt zwei Beispiele dafür, dass für ein gegebenen Split s mehr als ein Join j mit $match(s, j)$ existieren kann.

Abb. 7.6 b) zeigt den in der Praxis häufig zu findenden Fall einer falschen Kombination aus Split und Join. In der Abbildung ergibt sich aus der Kombination eines XOR-Splits mit einem AND-Join ein Deadlock am Join. Eine Kombination von AND- bzw. XOR-Split mit einem OR-Join ergibt zwar keine Kontrollflussfehler, wird jedoch trotzdem als unstrukturiert betrachtet.

Abb. 7.6 c) stellt zwei Fälle dar, in denen der Split sowohl einen strukturierten Kontrollblock (hier eine Alternative zwischen dem Ausführen von F1 und F2) einleitet, als auch Ausgangspunkt für einen Zyklus ist.

Die Abb. 7.6 d) und e) zeigen „Einsprünge“ in Kontrollblöcke. In beiden Abbildungen bilden AND-Split und AND-Join einen Kontrollblock, in dem zwei Kontrollpfade parallel zu durchlaufen sind. Diese Struktur wird dadurch gestört, dass die Ausführung eines Teils des jeweils rechten Kontrollpfades auch dadurch veranlasst werden kann, dass am XOR-Join eine Markierung von außerhalb des aus AND-Split und AND-Join gebildeten Blocks eintreffen kann. Analog sind „Aussprünge“ aus dem Kontrollblock möglich, hier kehrt sich in Abb. 7.6 d) und e) die Richtung des Kontrollflusspfeils zwischen den XOR-Konnektoren um.

Die Abb. 7.6 f) stellt einen Fall vor, in dem der Split s, sobald er einmal aktiviert ist, wieder und wieder die Iteration durchlaufen muss, so dass der modellierte Geschäftsprozess niemals enden kann.

In 7.6) g) wird ein Zyklus durch einen „Aussprung“ verlassen.

In den Abb. 7.6 h)-i) werden schließlich Fälle gezeigt, in denen sich Zyklen überschneiden.

Es ist wichtig, an dieser Stelle darauf hinzuweisen, dass die aufgeführten Fälle in ihrer Auswirkung nicht als gleichermaßen problematisch einzustufen sind. So enthält das Modellfragment b) einen offensichtlichen Kontrollflussfehler. Demgegenüber stellen etwa die in a), c), h) und i) dargestellten Fälle Modellteile dar, die nicht zur Verletzung der Soundness-Eigenschaft führen, in diesem Sinne also völlig korrekt sind. Wir haben jedoch beim Entwurf unserer Unstrukturiertheitsmetrik den menschlichen Modellierer oder Leser eines Modells im Blick, für den alle dargestellten Fälle unstruk-

turierten Modellierens mögliche Quellen für Modellierungs- bzw. Verständnisfehler sein können.

Wir wollen in einer EPK die Konnektoren zählen, die „problematische“ Modellkonstrukte einleiten. Anders als bei der im vorangegangenen Abschnitt vorgestellten Metrik von Mendling wird bei diesem Ansatz Unstrukturiertheit eher als lokale Eigenschaft innerhalb eines Modells betrachtet.

In Def. 23 wurden verschiedene Symptome berücksichtigt, die entstehen können, wenn zu einem gegebenen Split kein „passender“ Join (im Sinne „sauber verschachtelter“ Kontrollflussblöcke) gefunden werden kann. Auf diese Art ist es also möglich, die Zahl der unstrukturierten Splits zu bestimmen. Da diese Splits ein unstrukturiertes Modellfragment einleiten, stellt die Metrik eine Heuristik zur Schätzung der Anzahl unstrukturierter Modellfragmente in einer EPK dar.

Dass es jedoch zur Beurteilung der Komplexität einer EPK nicht ausreicht, lediglich diese Splits zu zählen, verdeutlicht Abb. 7.7. Beide Modelle haben genau einen unstrukturierten Split. Während in Abb. 7.7(a) von diesem Split genau eine Kante ausgeht, die zu Unstrukturiertheit im Modell führt, sind es in Abb. 7.7(b) drei solcher Kanten. Von einer Metrik, die Unstrukturiertheit messen soll, darf man erwarten, dass das rechte Modell als unstrukturierter als das linke bewertet wird.

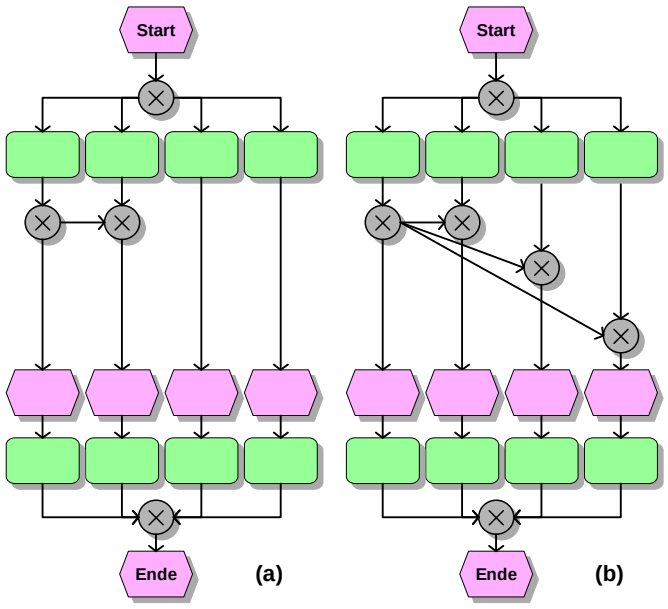


Abbildung 7.7: Zwei EPKs mit genau einem unstrukturierten Split

Um dies zu gewährleisten, bedienen wir uns der folgenden Definition der Inversen einer EPK:

Definition 24 (Inverse einer EPK)
Sei $\Lambda = (K, A)$ eine EPK. Dann heißt $\Lambda^{-1} = (K, \widetilde{A})$ die Inverse von Λ, wenn gilt: $(a, b) \in A$ genau dann, wenn $(b, a) \in \widetilde{A}$.

Es ist leicht zu sehen, dass Λ^{-1} selbst wieder die in Def. 1 aufgelisteten Syntaxanforderungen für EPKs mit Ausnahme von Forderung 10, die XOR- und OR-Splits nach einem Ereignis verbietet, erfüllt.

Ein Split in Λ wird zum Join in Λ^{-1} und umgekehrt. Zählen wir nun die unstrukturierten Splits in Λ und Λ^{-1} zusammen, erhalten wir eine realistische Einschätzung der Zahl der problematischen Konnektoren.

Somit definieren wir:

Definition 25 (Zahl der unstrukturierten Konnektoren)
Sei Λ eine EPK und Λ^{-1} ihre Inverse. Die Zahl der unstrukturierten Konnektoren in Λ ist die Summe der unstrukturierten Splits in Λ und in Λ^{-1}. Wir bezeichnen diese Zahl kurz mit UCC (für „Unmatched Connector Count“).

Im linken Modell von Abb. 7.7 gibt es zwei unstrukturierte Konnektoren - einen unstrukturierten Split im Modell selbst und einer in der inversen EPK. Im rechten Modell zählt man vier unstrukturierte Konnektoren, nämlich einen unstrukturierten Split im Modell selbst und drei in der inversen EPK. Somit liefert die Zahl der unstrukturierten Konnektoren, dass das rechte Modell als weniger gut strukturiert bewertet wird – so wie es der Erwartung entspricht.

Validierung der Tauglichkeit der UCC-Metrik

In Abschnitt 6.2 wurde bereits der von Mendling [113] und anderen Autoren genutzte Ansatz vorgestellt, die Tauglichkeit einer Komplexitätsmetrik mit statistischen Mitteln zu validieren. Wir wollen bei der Prüfung der Tauglichkeit der UCC-Metrik nach demselben Verfahren vorgehen. Das hat den positiven Effekt, dass sich die Ergebnisse direkt mit denen aus Mendlings Arbeit vergleichen lassen.

Die Grundidee der Validierung ist es, zu prüfen, inwiefern sich eine statistisch relevante Beziehung zwischen der UCC-Metrik und der Soundness-Eigenschaft (siehe Kapitel 2) als Korrektheitskriterium einer EPK nachweisen lässt.

Wie auch bei Mendling, wurden für diese Untersuchung die EPK-Modelle des SAP R/3-Referenzmodells verwendet. Dieses Referenzmodell beschreibt formal auf

betriebswirtschaftlicher Ebene die durch das SAP R/3-System unterstützten Geschäftsprozesse. Das Referenzmodell war bis zum Auslieferungsstand 4.6 Bestandteil der SAP R/3-Installation und fand daher eine große Anwenderbasis. Die Wahl dieses Modells für unsere statistischen Untersuchungen hat neben der direkten Vergleichbarkeit mit den Arbeiten von Mendling und anderen Autoren [113, 177] den Vorteil, dass es eine große Zahl von praxisrelevanten EPK-Modellen umfasst. Somit sind statistisch relevante Ergebnisse zu erwarten. Das SAP R/3-Referenzmodell wurde über mehrere Jahre hinweg (1992-2000) von einer namhaften Softwarefirma entwickelt. Damit kann angenommen werden, dass die Modelle eine Qualität aufweisen, wie sie auch in anderen Industrieprojekten erwartet werden kann.

Das SAP R/3-Referenzmodell enthält 604 nichttriviale EPK-Modelle. Für unsere Analysen konnten vier dieser Modelle nicht berücksichtigt werden, da sie Funktionen oder Ereignisse mit mehr als einer eingehenden oder mehr als einer ausgehenden Kontrollflusskante besitzen und somit nicht der formalen Definition einer EPK (siehe Kapitel 2) entsprechen. Weitere 60 Modelle bilden keinen zusammenhängenden Graphen und entsprechen somit ebenfalls nicht der Definition einer EPK.

Somit verbleiben 540 EPK-Modelle des SAP-Referenzmodells, die in die statistischen Auswertungen einbezogen werden konnten.

Für diese 540 Modelle wurden die Ergebnisse der Soundness-Überprüfung von Mendling, publiziert in [113], genutzt.

In unserem statistischen Modell wird nun die Abhängigkeit des Eintretens des Ereignisses „die EPK ist sound“ bzw. des Komplementärereignisses „die EPK ist nicht sound“ vom Wert der UCC-Metrik als unabhängige Variable untersucht. Die Variable, deren Abhängigkeit untersucht werden soll, kann die Werte 1 (EPK ist fehlerhaft) oder 0 (EPK ist sound) annehmen.

In derselben Weise wie bei anderen Autoren [113, 180] wurde zunächst der Rangkorrelationskoeffizient nach Spearman für den Zusammenhang zwischen Soundness und UCC-Metrik bestimmt[2]. Dieser lag bei +0,777 und damit höher als bei jeder anderen untersuchten Komplexitätsmetrik für EPKs. Als einzige publizierte Komplexitätsmetrik für EPKs erreicht die Zahl der unstrukturierten Konnektoren einen Wert des Rangkorrelationskoeffizienten, der nach der Klassifikation von [22] als „hohe Korrelation“ einzustufen ist.

[2]Ein herzlicher Dank für das Durchführen der statistischen Analysen geht an Jan Mendling. Durch die Verwendung der von ihm zur Verfügung gestellten Modelle konnte sichergestellt werden, dass die Ergebnisse direkt mit denen von [113] vergleichbar sind. Weitere Details zur statistischen Auswertung finden sich in [98].

Mendling präsentierte als ein zentrales Ergebnis seiner Untersuchungen von Komplexitätsmetriken für EPKs in [113] ein Regressionsmodell, in dem die Wahrscheinlichkeit des Ereignisses „Die EPK ist sound“ aus den Werten verschiedener Metriken bestimmt wurde.

Zur Untersuchung der Abhängigkeit der binären Variablen, die für das Ereignis „eine EPK ist sound“ steht, von den Werten der Komplexitätsmetriken als unabhängige Variablen wurde in [113] das Verfahren der logistischen Regression genutzt. Bei diesem Verfahren wird die Wahrscheinlichkeit p des Eintretens eines Ereignisses (in unserem Fall „EPK ist fehlerhaft“) in Abhängigkeit von den Werten der unabhängigen Variablen $x_1, x_2, \ldots x_n$ (in unserem Fall den Werten der Metriken) bestimmt. Dies geschieht nach dem Ansatz:

$$p = \frac{1}{(1 + e^{-(\beta_0 + \sum_{i=1}^{k} \beta_i x_i)})} \tag{7.1}$$

Beim Verfahren der logistischen Regression werden die Koeffizienten $\beta_0, \ldots, \beta_n$ in Gleichung 7.1 nach dem Maximum-Likelihood-Prinzip bestimmt. Eine genauere Beschreibung des Verfahrens der logistischen Regression findet man u.a. in [7] und [73].

Mendling stellte ein Regressionsmodell aus sieben verschiedenen Komplexitätsmetriken vor, das die Wahrscheinlichkeit des Ereignisses „Soundness“ voraussagt.

Ein Indikator für die Qualität eines solchen logistischen Regressionsmodells ist die Zahl der durch dieses Modell richtig getroffenen Vorhersagen. Hierzu werden für jedes Modell der betrachteten Stichprobe die Werte der Metriken und daraus die aus Gleichung 7.1 ermittelte Wahrscheinlichkeit p berechnet. Ergibt sich $p > 0,5$, wird das Ergebnis „EPK ist fehlerhaft“ angenommen, andernfalls lautet die Annahme, dass die EPK sound ist. Anschließend kann man die tatsächliche Zugehörigkeit einer EPK zur Gruppe „fehlerhaft“ oder „sound“ und die auf Grund der Berechnung des Regressionsmodells vorhergesagte Gruppenzugehörigkeit gegenüberstellen.

Bei einer Verwendung von sieben Komplexitätsmetriken als unabhängige Variable erreicht Mendling in 95,2% aller Fälle eine korrekte Klassifikation. Wird nach dem gleichen Modell lediglich die in dieser Arbeit eingeführte UCC-Metrik als einzige unabhängige Variable in Gleichung 7.1 genutzt, wird immer noch in 90,6% der Fälle die richtige Gruppenzugehörigkeit bestimmt.

Der Prozentsatz der richtig klassifizierten Fälle gibt zwar einen ersten Eindruck über die Qualität eines logistischen Regressionsmodells, ist jedoch mit Vorsicht zu betrachten. So sagt dieser Prozentsatz nichts darüber aus, wie deutlich das Eintreten des Ereignisses vorhergesagt wurde (oder wie stark die Vorhersage danebenlag); ein

Wert von p=0,51 wird bei der Bestimmung dieses Prozentsatzes ebenso behandelt wie ein Wert von p=0,999.

Als aussagekräftiger gilt der Nagelkerke R^2-Test [73]. Werte der Nagelkerke R^2-Testgröße von 0,4 und größer lassen auf einen statistisch signifikanten Zusammenhang zwischen den unabhängigen Variablen und dem Eintreten bzw. Nichteintreten des beobachteten Ereignisses schließen [73].

Für unser Modell mit der UCC-Metrik als einzige unabhängige Variable wurde ein Nagelkerke R^2-Wert von 0,707 berechnet, was als hervorragender Wert gilt. Damit reiht sich die CFC-Metrik in die aussagekräftigsten Metriken zur Vorhersage der Fehlerwahrscheinlichkeit einer EPK ein: Unter den 15 in [113] untersuchten Metriken kamen nur die vier besten auf Nagelkerke R^2-Werte zwischen 0,7 und 0,8.

Dass die UCC-Metrik sehr gut dazu geeignet ist, die Soundness von EPKs vorauszusagen, ist wenig überraschend. Zum einen ist die Metrik so angelegt, dass typische Fehler, die unweigerlich zur Verletzung der Soundness-Eigenschaft führen (wie z.B. die in Abb. 7.6 b) gezeigte Deadlock-Situation), lokalisiert und gezählt werden. Zum anderen wird allen wohlstrukturierten Modellen, die stets auch sound sind, stets die UCC-Metrik 0 zugeordnet.

Die UCC-Metrik soll jedoch mehr leisten, als Rückschlüsse auf Kontrollflussfehler im Modell zu erlauben: Sie soll auch die Verständlichkeit von Modellen für menschliche Leser messen. Daher werden auch diejenigen Konstellationen unstrukturierter Splits, die die Soundness-Eigenschaft nicht beeinträchtigen (vgl. etwa Abb. 7.6 a), c), h) und i)) in die Betrachtungen einbezogen.

Schließlich ist anzumerken, dass das statistische Modell mit einer binären Variablen die Intention der UCC-Metrik nur unzureichend abbildet. Die UCC-Metrik versucht festzustellen, *an wie vielen Stellen* das Modell Probleme haben kann, während die abhängige Variable im statistischen Modell nur zwei Ausprägungen (die EPK ist entweder sound oder nicht) hat.

Ein zentrales (wenn auch nicht überraschendes) Ergebnis dieses Kapitels ist, dass es einen hohen Zusammenhang zwischen Nichtstrukturiertheit und Kontrollflussfehlern gibt. Die der UCC-Metrik zugrundeliegende Idee, Symptome von Nichtstrukturiertheit zu ermitteln, wird nun in den kommenden Kapiteln weiter ausgebaut. Während es Gegenstand dieses Kapitels war, Unstrukturiertheit zu messen, beschreiben die folgenden Kapitel die automatisierte Erkennung (unstrukturierter) Muster, die auf fehlerhafte Modellierung hindeuten.

8 Musterkataloge für Fehler und Unstrukturiertheit in Modellen

In diesem Kapitel sollen die Probleme, die in unstrukturierten GPM häufig anzutreffen sind, systematisch kategorisiert werden.

In Abschnitt 8.1 werden zunächst aus der Literatur bekannte Klassifizierungen solcher Probleme vorgestellt. Anschließend wird in Abschnitt 8.2 die Systematik dargestellt, nach der ein eigener Musterkatalog erstellt wird. Der Musterkatalog selbst findet sich dann im Folgekapitel 9.

8.1 Existierende Systematisierungen von Fehlermustern

Während es, wie in Kapitel 4 beschrieben, umfangreiche Literatur zum Testen von GPM auf Korrektheit gibt, befassen sich nur wenige Arbeiten mit der Frage, welche typischen Fehlermuster zu unkorrekten Modellen führen.

Rump analysiert in [145] ausführlich, welche Fehler bei der Modellierung mit EPKs auftreten können und nennt zehn Fälle, die einen Fehler darstellen oder zu einer Warnung Anlass geben. Um diese zu finden, wird in [145] der Zustandsraum der untersuchten EPK berechnet. Uns interessieren jedoch an dieser Stelle Muster für Fehler, die direkt durch statische Analyse der EPK gefunden werden können.

8.1.1 Deadlock-Muster von Onoda et al.

Der nach Wissen des Autors erste Versuch, Fehlermuster in GPM zu kategorisieren, wurde von einer Forschungsgruppe der Universität Osaka unternommen. In [126] werden fünf sog. „Deadlock-Muster“ beschrieben.

Mit den im Abschnitt 2.2 eingeführten Bezeichnungen lassen sich die Muster 1 bis 3 von [126] wie folgt zusammenfassen:

In der EPK $\Lambda = (K, A)$ existiert ein XOR-Split $x \in K$ sowie ein AND-Join $a \in K$. Seien weiter out_1 und out_2 zwei voneinander verschiedene Ausgangskanten von x sowie in_1 und in_2 zwei voneinander verschiedene Eingangskanten von a.

Dann gilt für:

- **Muster 1:** Es gibt einen Pfad von x nach a, der out_1 und in_1 enthält, jedoch keinen Pfad von x nach a, der out_2 und in_2 enthält.
- **Muster 2:** Es gibt einen Pfad von x nach a, der out_1 und in_1 enthält sowie einen weiteren Pfad von x nach a, der out_2 und in_2 enthält.
- **Muster 3:** Es gibt einen Pfad von x nach a, der out_1 und in_1 enthält, jedoch keinen Pfad von x nach a, der out_1 und in_2 enthält.

Muster 4 behandelt einen Zyklus im GPM, in dem ein AND-Join Zykleneinstiegsknoten und ein AND-Split Zyklensausstiegsknoten ist.

Beispiele für das Auftreten der Fehlermuster 1 bis 4 sind in Abb. 8.1 dargestellt.

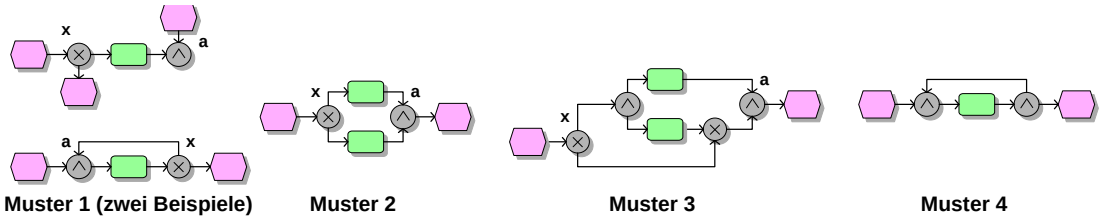


Abbildung 8.1: Deadlockmuster nach [126]

Das Muster 5 aus [126] ist für EPKs irrelevant, da es von einer Semantik ausgeht, die nicht der von EPKs entspricht [32]. Nach diesem Muster würden zwei Startereignisse, die mit einem AND-Join zusammengeführt werden, bereits zu einem Deadlock führen.

Um die angegebenen Muster in Modellen zu analysieren, führt [126] die Begriffe „Erreichbarkeit“ und „Überführbarkeit“ ein. Erreichbarkeit entspricht der Existenz eines Pfades im Graphen. Überführbarkeit einer Kante a_1 zu einer Kante a_2 bedeutet, dass in jedem möglichen endlichen[1] Ablauf des Modells auf einen Zustand, in dem Kante a_1 markiert ist, ein Zustand folgt, in dem Kante a_2 markiert ist (möglicherweise über mehrere Zwischenzustände).

Muster 2 führt laut [126] zu einem Deadlock, wenn zwischen einem XOR-Split x und einem AND-Join a zwar Erreichbarkeit, jedoch keine Überführbarkeit gegeben ist. Im Modell von Abb. 8.2 wird kein Fehler gemeldet, da Überführbarkeit von x zu

[1] Die Forderung „endlich“ ist in [126] nicht ausdrücklich erwähnt, aber notwendig. Sie entspricht der (auch als Fairness bezeichneten) Annahme, dass keine Schleife im Modell unendlich oft durchlaufen wird.

a gegeben ist: Nachdem beide Ausgangskanten von x markiert sind, wird in jedem denkbaren Ablauf ein Zustand erreicht, in dem die beiden Eingangskanten von a markiert sind.

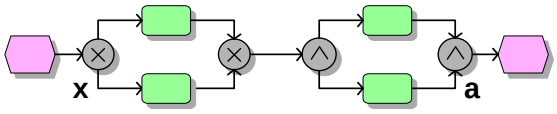


Abbildung 8.2: Kein Deadlock zwischen XOR-Split x und AND-Join a

Bei der Untersuchung von Deadlocks zwischen einem XOR-Split x und einem AND-Join a wird die Überführbarkeit zwischen den Ausgangskanten von x und den Eingangskanten von a aber ohne Berücksichtigung anderer markierter Kanten im Modell untersucht. Für das in Abb. 8.3 gezeigte Modell heißt das, dass nach der in [126] gegebenen Definition keine Überführbarkeit zwischen x und a existiert. Es kann nämlich mit dem in [126] vorgestellten Verfahren nicht erkannt werden, dass am AND-Join a_1 stets beide Eingangskanten erreicht werden und a_1 somit stets Markierungen von den Eingangskanten an die Ausgangskante weitergibt. Somit wird im gezeigten Modell fälschlicherweise ein Deadlock diagnostiziert. Fälle wie in Abb. 8.3 gezeigt sind jedoch in der Praxis nicht selten, was den in [126] vorgestellten Ansatz für den Praxiseinsatz untauglich macht. Dieses Problem betrifft auch den Ansatz von Awad und Puhlmann [?], der - aufbauend auf [126] - mittels einer graphischen Abfragesprache BPMN-Q mögliche Fehler in BPMN-Modellen lokalisieren soll.

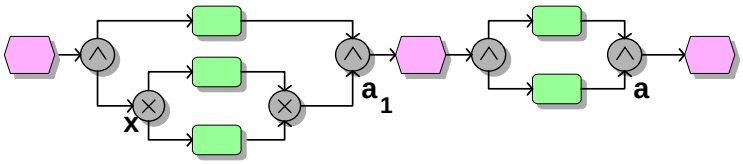


Abbildung 8.3: Von [126] nicht erfasster Fall

Die tiefere Ursache dafür, dass das Mustersystem aus [126] und [?] bei Fällen wie dem in Abb. 8.3 gezeigten versagt, liegt darin, dass nicht zuverlässig erkannt wird, dass die mit dem XOR-Split eröffneten alternativen Pfade bereits wieder zusammengeführt wurden, bevor sie den AND-Join a erreichen.

Wie diese Fälle korrekt behandelt werden, zeigt van der Aalst in [168, 171]. In diesen Artikeln werden als Workflow-Netze dargestellte GPM betrachtet. Diese spezielle Klasse von Petrinetzen zeichnet sich dadurch aus, dass es genau eine Stelle ohne Vorgänger und genau eine Stelle ohne Nachfolger gibt. Diese beiden

Stellen stehen für den (eindeutig bestimmten) Start- und Endzustand eines Ablaufes des Geschäftsprozesses. In [168] wurde gezeigt, dass sich in einem solchen Modell Deadlocks und mehrfache Beendigung dadurch äußern, dass das Petrinetz Handles [47] aufweist. Dies ist der Fall genau dann, wenn es zwischen einer Stelle und einer Transition zwei verschiedene Pfade gibt, die außer dem Anfangs- und dem Endknoten keine weiteren Knoten gemeinsam haben.

8.1.2 GPM-Fehlerliste des IBM-Forschungslabors Zürich

Eine Untersuchung von tatsächlichen Fehlern in GPM, die in industriellem Kontext entwickelt wurden, wurde im IBM Forschungslabor Zürich vorgenommen [89]. Die Untersuchung bezog sich auf „hunderte Prozessmodelle, die im *IBM WebSphere Business Modeler* ... und anderen Modellierungswerkzeugen wie *ARIS*, *Adonis* oder dem *MID Innovator* zwischen 2004 und 2006 erstellt wurden. Die Modelle resultierten aus realen Projekten quer durch verschiedene Industriezweige wie dem Bankwesen, der Versicherungsbranche, dem Einzelhandel, der pharmazeutischen Industrie und der Telekommunikationsbranche“ (wörtlich zitiert aus [89]). Als Ergebnis der Untersuchung wurden sog. Anti-Muster aufgestellt, also Muster für typische Fehlersituationen.

Jeweils ein Beispiel für die in [89] behandelten Fehlermuster sind in Abb. 8.4 dargestellt. Funktionen und Ereignisse wurden in dieser Darstellung weggelassen, da es bei der Identifikation von Kontrollflussfehlern nur auf die Kombination von Konnektoren ankommt. Innerhalb der Pfeile zwischen den Konnektoren kann prinzipiell jedes Konstrukt mit genau einer Eingangs- und genau einer Ausgangskante stehen. Die dargestellten Fehlermuster beziehen sich auf die bekannten Fehlertypen „XOR-Split gepaart mit AND-Join“, „AND-Split gepaart mit XOR-Join“ sowie „Zyklus im Modell, der als Zykleneinstiegs- oder Zyklenausstiegsknoten einen AND- statt eines XOR-Konnektors hat“.

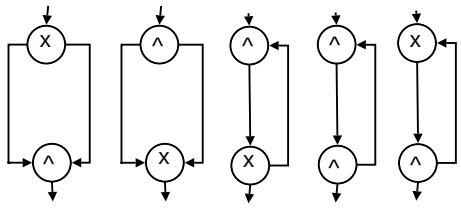


Abbildung 8.4: Fehlermuster aus [89]

Ebenso im IBM-Forschungslabor Zürich entstand der in [182] beschriebene Ansatz, GPM in Teilgraphen mit genau einer Eingangs- und genau einer Ausgangskante zu

unterteilen. Betrachtet werden Workflow-Graphen. Diese zeichnen sich im Vergleich zu EPKs dadurch aus, dass es genau einen Anfangs- und genau einen Endknoten gibt und dass die Typen der Konnektoren nur XOR und AND sein können. Zur Untersuchung der Soundness des Gesamtmodells reicht es, die Soundness aller Teilgraphen, in die es zerlegt wurde, zu prüfen [165]. Hierfür wird in [182] ein heuristisches Verfahren angegeben, dass die folgenden Fehlermuster enthält:

1. Der Graph enthält einen XOR-Split, jedoch keinen XOR-Join.
2. Der Graph enthält einen XOR-Join, jedoch keinen XOR-Split.
3. Der Graph enthält einen AND-Split, jedoch keinen AND-Join.
4. Der Graph enthält einen AND-Join, jedoch keinen AND-Split.
5. Der Graph enthält einen Zyklus, jedoch keinen XOR-Split.
6. Der Graph enthält einen Zyklus, jedoch keinen XOR-Join.

Die Fehlermuster 1-4 besagen, dass zu jedem Split ein passender Join gehören muss. Dies ist nicht unmittelbar auf EPKs zu übertragen, da hier z.B. ein Split auch einfach zu einem Endereignis verzweigen kann. Die beiden letzten Muster verbieten AND-Konnektoren als Zykleneinstiegs- bzw. Zyklenausstiegsknoten.

[182] beschreibt die Anwendung von Mustern für bekanntermaßen korrekte Modellteile sowie von Fehlermustern zur Prüfung der Soundness-Eigenschaft von 200 Modellen. Für 68,5% dieser Modelle kann mit den betrachteten Mustern entweder Soundness festgestellt oder ein Fehler nachgewiesen werden. Ein einfaches Beispiel eines fehlerhaften Modells, das mit der Heuristik von [182] nicht gefunden wird, zeigt Abb. 8.5.

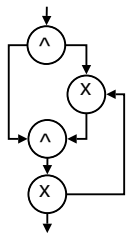


Abbildung 8.5: ein Modellfehler, der von der Heuristik in [182] nicht erkannt wird

8.1.3 Kategorisierung von Unstrukturiertheit nach Liu und Kumar

Liu und Kumar untersuchten in [103], in welchen Fällen ein unstrukturiertes GPM mit AND- und XOR-Konnektoren in ein verhaltensgleiches wohlstrukturiertes GPM umgewandelt werden kann und wie Deadlocks und Synchronisationsfehler entdeckt

werden können. Zu diesem Zweck werden Muster unstrukturierter Modellfragmente aus einem Kontrollblock mit Split S und Join J wie in Abb. 8.6 betrachtet. Offenbar gibt es für jedes der beiden Modellfragmente aus Abb. 8.6 $2^4 = 16$ Möglichkeiten, als Typ eines Konnektors AND oder XOR zu wählen. In einer Tabelle zeigen Liu und Kumar, dass jeweils 9 dieser 16 Möglichkeiten zu einem Deadlock oder Synchronisationsfehler führen.

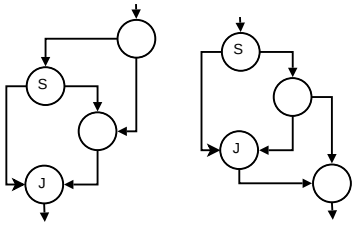


Abbildung 8.6: In [103] betrachtete Unstrukturiertheits-Muster

Liu und Kumar zeigten weiterhin, wie durch Verdoppeln oder Verschieben von Knoten jedes Modell ohne Zyklen in ein verhaltensgleiches Modell überführt werden kann, für das die Korrektheit anhand der oben erwähnten Tabelle bestimmt werden kann.

Die Erweiterung des Ansatzes auf Modellen mit Zyklen bleibt in [103] unvollständig. Es wird lediglich der Fall von Zyklen mit genau einem Zykleneinstiegs- und genau einem Zyklenausstiegsknoten umfassend behandelt.

Modelle mit mehr als einem Start- oder Endereignis sowie Modelle mit OR-Konnektoren, beides wichtig bei der Untersuchung von EPKs, waren auf Grund der von Liu und Kumar betrachteten einfachen Modellierungssprache nicht Gegenstand der Untersuchungen in [103].

8.1.4 Fehlermuster in den Reduktionsregeln von Mendling

Fehlermuster werden auch im Reduktionsansatz von Mendling [113] genutzt. Da dieser mit EPK-Modellen arbeitet, werden (anders als in allen bisher genannten Aufstellungen von Fehlermustern) auch Fälle betrachtet, in denen neben AND- und XOR-Konnektoren auch OR-Konnektoren vorkommen.

Mendlings Reduktionsregeln umfassen fünf Fälle, in denen im Modell ein fehlerhaftes Konstrukt reduziert und eine entsprechende Fehlermeldung ausgegeben wird. Diese sind in Abb. 8.7 dargestellt. Die mit den Buchstaben S, J, A, B und E bezeichneten Knoten stellen jeweils Konnektoren dar. Je nachdem, ob es sich dabei um AND-, OR- oder XOR-Konnektoren handelt, wird entschieden, ob

das entsprechende Konstrukt reduziert werden kann und ob eine Fehlermeldung auszugeben ist.

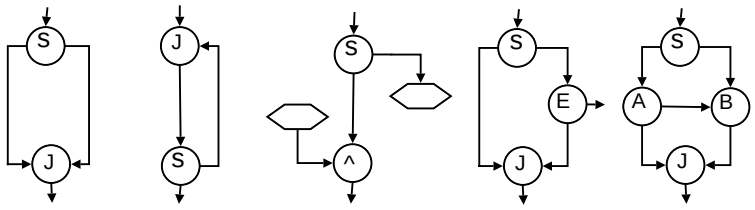


Abbildung 8.7: Muster, die von den Reduktionsregeln in [113] behandelt werden

Die beiden ersten in Abb. 8.7 dargestellten Situationen beinhalten die schon im Katalog von IBM vorhandenen Fehlermuster (vgl. Abb. 8.4). Das dritte Muster führt zu einem Fehler, wenn S ein OR- oder XOR-Split ist. Dann ist es nämlich möglich, dass nach S die Ausgangskante markiert wird, die nicht auf dem Pfad zu J liegt, womit sich ein Deadlock am AND-Join ergibt. Das vierte Muster behandelt den Fall, dass es innerhalb eines Kontrollblocks aus Split und Join einen Konnektor E gibt, der den Kontrollfluss in eine andere Richtung leitet. Durch diesen Konnektor E ergeben sich im Vergleich zum ersten Muster neue Fehlermöglichkeiten: Ist $type(S) = AND$ und $type(J) = AND$, so kann es trotz der richtigen Paarung eines AND-Splits mit einem AND-Join zu einem Deadlock am Join kommen, wenn E ein OR- oder XOR-Konnektor ist. Im fünften Muster gibt es zwischen zwei Zweigen eines mit einem Split beginnenden und mit einem Join endenden Kontrollblock eine „Querverbindung“. Diese Konstruktion führt zu verschiedenen Fehlermustern. Die Reduktionsregeln werden in [113] auf 604 Modelle des SAP R/3-Referenzmodells angewendet, wobei 178 Instanzen der dargestellten Fehlermuster in 90 Modellen gefunden werden.

Die dargestellten Muster werden in Reduktionsregeln beschrieben. Bei der Ausführung der Reduktionsregeln werden Fehler erkannt, sobald (ggf. nach wiederholter Reduktion des Modells) eines der Fehlermuster im Modell auftritt. Bei diesem Ansatz kann allerdings das Vorhandensein nichtreduzierbarer Modellteile dazu führen, dass die Anwendung der Reduktionsregeln stoppt und kein Fehlermuster erkannt wird. So wird beispielsweise der Deadlock am unteren AND-Join im Modell von Abb. 8.8 auf Seite 100 nicht festgestellt, da das Modell wegen der Unstrukturiertheit im rechten Teil des Modells nach den Regeln von [113] nicht weiter reduziert werden kann.

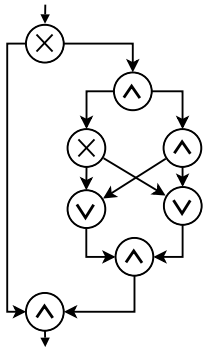


Abbildung 8.8: Deadlock, der durch die Reduktionsregeln in [113] nicht erkannt werden kann

8.2 Erarbeiten des Katalogs von Problemmustern in EPKs

Aufbauend auf den im vorangegangenen Abschnitt dargestellten aus der Literatur bekannten Fehlermustern wurde ein Katalog von Problemmustern erarbeitet. In diesem Abschnitt werden zunächst einige Definitionen, die zur Beschreibung von Problemmustern hilfreich sind, eingeführt. Anschließend wird der generelle Aufbau des Problemmuster-Katalogs dargestellt und erläutert, welche Systematik bei seiner Zusammenstellung angewendet wurde. Der Katalog selber findet sich dann in Kapitel 9.

8.2.1 Definitionen

Aufbauend auf den Definitionen aus Abschnitt 2.3 führen wir im Folgenden einige Begriffe ein, die zur Kategorisierung von Problemfällen in EPKs nützlich sind:

Definition 26 (SESE-Region)
Sei $\Lambda = (K, A)$ eine EPK und $S \subseteq K$ eine Teilmenge der Knoten von Λ. S heißt Single-Entry-Single-Exit-Region (kurz: SESE-Region), falls es zwei Knoten $k_{entry}, k_{exit} \in S$ mit den folgenden Eigenschaften gibt:

1. *$card\ {\triangleright} k_{entry} = card\ k_{exit}{\triangleleft} = 1$*
2. *Für jedes Startereignis $e_{start} \in K$ und jeden Knoten $s \in S$ gilt: Jeder Pfad von e_{start} nach s enthält k_{entry}.*
3. *Für jedes Endereignis $e_{end} \in K$ und jeden Knoten $s \in S$ gilt: Jeder Pfad von s nach e_{end} enthält k_{exit}.*

Wir betrachten nun einen Split s und einen Join j einer EPK $\Lambda = (K, A)$, für die die in Abschnitt 7.3 eingeführte match-Relation gilt (d.h. es ist: $match(s, j)$) und untersuchen bestimmte Spezialfälle.

Für unseren Katalog von Fehlermuster interessieren uns Fehler in einem Teilmodell einer EPK, das durch die Konnektoren s und j begrenzt ist. Was wir darunter verstehen wollen, legen die folgenden Definitionen fest:

Definition 27 (begrenzte Nachfolgermenge)
Ein Knoten $k \in K$ liegt genau dann in der durch j begrenzten Nachfolgermenge von s, wenn es einen Pfad von s nach k gibt, der j nicht enthält. Wir bezeichnen diese Menge mit dem Symbol $\boldsymbol{N}_j(s)$.

Definition 28 (begrenzte Vorgängermenge)
Ein Knoten $k \in K$ liegt genau dann in der durch s begrenzten Vorgängermenge von j, wenn es einen Pfad von k nach j gibt, der s nicht enthält. Wir bezeichnen diese Menge mit dem Symbol $\boldsymbol{V}_s(j)$.

Definition 29 (Split-Join-Block (s-j-Block))
Für gegebene s und j mit $match(s, j)$ bezeichnen wir den Durchschnitt der Mengen $\boldsymbol{N}_j(s)$ und $\boldsymbol{V}_s(j)$ als Split-Join-Block (kurz: s-j-Block, Symbol: $\boldsymbol{B}(s, j)$):
$\boldsymbol{B}(s, j) = \boldsymbol{N}_j(s) \cap \boldsymbol{V}_s(j)$.

Beispiel:
Für die in Abb. 8.9 auf Seite 102 gezeigte EPK gilt mit $j = J$ und $s = S$ die Relation $match(S, J)$, da beispielsweise mit $(S, F1, C1, C3, J)$ und $(S, F2, C2, E3, C4, F4, J)$ zwei Pfade von S nach J existieren, deren einzige gemeinsame Knoten S und J sind. Weiter gilt:
$\mathbf{N}_J(S) = \{F1, F2, C1, C2, E3, C3, C4, F3, F4, E4, F5, E5\}$,
$\mathbf{V}_S(J) = \{F1, F2, C1, C2, E2, E3, C3, C4, F3, F4\}$ und
$\mathbf{B}(S, J) = \mathbf{N}_J(S) \cap \mathbf{V}_S(J) = \{F1, F2, C1, C2, E3, C3, C4, F3, F4\}$.

Der s-j-Block $\mathbf{B}(S, J)$ enthält anschaulich gesprochen die Knoten, die „zwischen“ dem Split S und dem Join J liegen. Für die in Abb. 8.9 gezeigte EPK ist $\mathbf{B}(S, J)$ eine echte Teilmenge sowohl von $\mathbf{N}_J(S)$ als auch von $\mathbf{V}_S(J)$.

Im gezeigten Beispiel stimmt $\mathbf{B}(S, J)$ nicht mit $\mathbf{N}_J(S)$ überein, da mit E4, F5 und E5 Knoten existieren, von denen aus kein Pfad zu J führt. Weiterhin stimmt $\mathbf{B}(S, J)$ nicht mit $\mathbf{V}_S(J)$ überein, da letztere Menge das Element E2 enthält, zu dem kein Pfad von S führt.

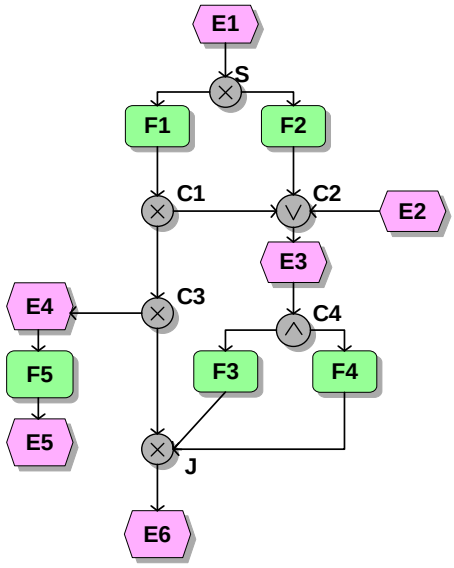


Abbildung 8.9: s-j-Block

Anschaulich gesprochen, gibt es mit dem Join $C2$ einen Einsprung in den s-j-Block $\mathbf{B}(S, J)$ und mit dem Split $C3$ einen Aussprung aus dem s-j-Block $\mathbf{B}(S, J)$.

Die Begriffe Ein- und Aussprung definieren wir formal wie folgt:

Definition 30 (Einsprung)
Sei $\Lambda = (K, A)$ eine EPK und $\boldsymbol{B}(s, j)$ ein s-j-Block in Λ.
Ein Knoten $e \in \boldsymbol{B}(s, j)$ heißt Einsprung in $\boldsymbol{B}(s, j)$, falls es eine Kante $(x, e) \in A$ gibt, so dass $x \notin \boldsymbol{B}(s, j)$.

Aus der Definition von $\mathbf{B}(s, j)$ folgt leicht, dass ein Einsprung $e \in \mathbf{B}(s, j)$ immer ein Join ist, von dem ein Vorgängerknoten in $\mathbf{B}(s, j)$ und ein weiterer Vorgängerknoten außerhalb von $\mathbf{B}(s, j)$ liegt.

Definition 31 ((echter) Aussprung)
Sei $\Lambda = (K, A)$ eine EPK und $\boldsymbol{B}(s, j)$ ein s-j-Block in Λ.
Ein Knoten $e \in \boldsymbol{B}(s, j)$ heißt Aussprung aus $\boldsymbol{B}(s, j)$, falls es eine Kante $(e, x) \in A$ gibt, so dass $x \notin \boldsymbol{B}(s, j)$. Falls zusätzlich gilt $type(e) \neq AND$, heißt e ein echter Aussprung aus $\boldsymbol{B}(s, j)$.

Weiterhin unterscheiden wir die Einsprünge in einen s-j-Block danach, ob der Einsprung erst erreicht werden kann, nachdem der s-j-Block bereits einmal durchlaufen wurde:

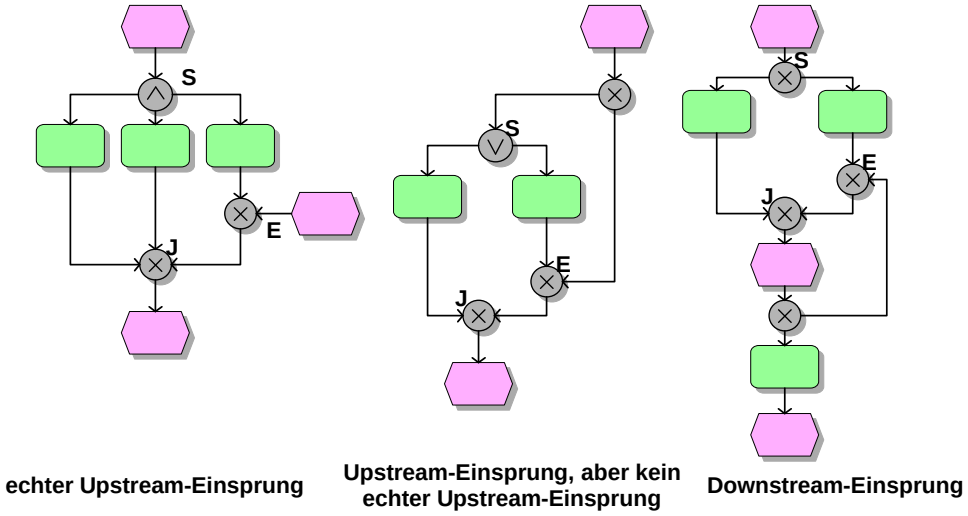


Abbildung 8.10: Unterscheidung zwischen verschiedenen Klassen von Einsprüngen

Definition 32 (Upstream-Einsprung / Downstream-Einsprung)
Sei e ein Einsprung in den s-j-Block $\boldsymbol{B}(s,j)$. e heißt Upstream-Einsprung, wenn es einen Pfad von einem Startereignis zu e gibt, der nicht durch s geht. Andernfalls heißt e Downstream-Einsprung.

Definition 33 (echter Upstream-Einsprung)
In einer EPK $\Lambda = (K, A)$ sei e ein Upstream-Einsprung in den s-j-Block $\boldsymbol{B}(s,j)$. e heißt echter Upstream-Einsprung, falls kein XOR-Split $x \in K$ existiert, für den die folgenden Eigenschaften gelten:

1. *Es gibt einen Pfad P_1 von x nach s und einen Pfad $P_2 \neq (x,s,e)$ von x nach e.*
2. *P_1 und P_2 sind schnittfrei.*
3. *Jeder Pfad von einem Startereignis zu s sowie jeder Pfad von einem Startereignis zu e geht durch x.*

Abb. 8.10 zeigt Beispiele für die in den Definitionen 32 und 33 erklärten Begriffe.

Für das Vorliegen einer match-Relation zwischen einem Split s und einem Join j wurde bisher nur die Existenz zweier Pfade P_1 und P_2 von s nach j gefordert, die (außer s und j selber) keine gemeinsamen Knoten haben. Nach dieser Definition kann es noch Pfade zwischen den Knotenmengen der beiden Pfade - genauer: zwischen $\mathbf{M}(P_1) \setminus \{s,j\}$ und $\mathbf{M}(P_2) \setminus \{s,j\}$ - geben; einige Beispiele hierfür sind schematisch in Abb. 8.11 auf Seite 104 gezeigt.

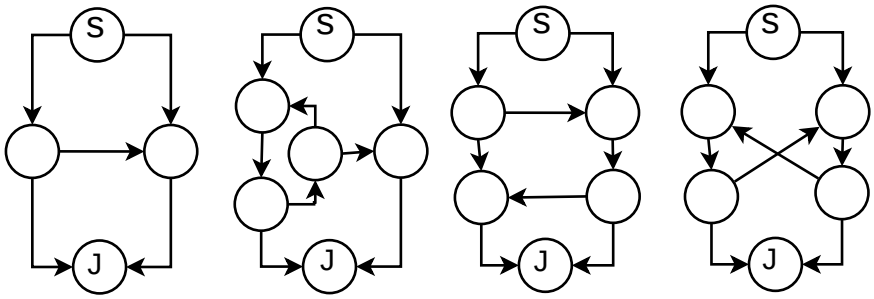


Abbildung 8.11: Transfers zwischen den beiden Pfaden vom Split zum Join

Wir definieren:

Definition 34 (Transfer in einem s-j-Block)
Sei $\boldsymbol{B}(s,j)$ ein s-j-Block und P_1 sowie P_2 zwei schnittfreie Pfade von s nach j. Ein Pfad T von $\boldsymbol{M}(P_1) \setminus \{s,j\}$ nach $\boldsymbol{M}(P_2) \setminus \{s,j\}$, heißt Transfer von P_1 nach P_2.

Unter Nutzung der oben gegebenen Definitionen können mögliche Muster von Modellierungsproblemen in nicht wohlstrukturierten EPKs effizient beschrieben werden.

8.2.2 Vorgehensweise bei der Erarbeitung des Problemmusterkatalogs

In diesem Abschnitt wird beschrieben, welche Systematik angewandt wurde, um den im kommenden Kapitel 9 vorgestellte Problemmusterkatalog zu erstellen.

Die Grundidee war, zu untersuchen, auf welche Weise die in Def. 22 gegebene Definition der Wohlstrukturiertheit von EPKs verletzt werden kann. Für die identifizierten Verletzungen wurde analysiert, inwiefern sich aus der Verletzung der Wohlstrukturiertheits-Eigenschaft ein tatsächliches Modellierungsproblem ergibt.

Dabei wurde insbesondere sichergestellt, dass die in der Literatur bereits vorgestellten und in Abschnitt 8.1 beschriebenen Muster erfasst sind.

Besondere Beachtung fanden die Fälle, die in [89] als die am häufigsten in realen GPM vorkommende Fehlerklassen identifiziert wurden: Kontrollblöcke zwischen einem Split und einem im Typ abweichenden Join (also Probleme in s-j-Blöcken) sowie fehlerhafte Zyklen.

8.2.3 Probleme in s-j-Blöcken

Es wurde bei der Erfassung von Typkonflikten zwischen einem Split und einem zugehörigen Join angestrebt, solche Probleme im möglichst allgemeinen Fall identifizieren zu können. Ein Mittel hierzu war es, die in der Literatur vorgestellten

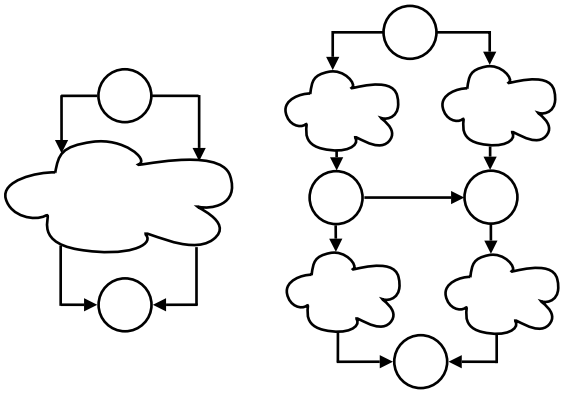


Abbildung 8.12: s-j-Block ohne Transfer (links) und mit einer Transfer-Kante

Fehlermuster durch die Beachtung möglicher Einsprünge in bzw. Aussprünge aus s-j-Kontrollblöcken zu verallgemeinern.

Von entscheidender Bedeutung dafür, dass Typkonflikte nicht nur in „fast wohlstrukturierten“ Spezialfällen erkannt werden (wie das etwa beim Musterkatalog von IBM [89] der Fall ist), ist die Verwendung der in Def. 9 auf Seite 18 eingeführten Match-Relation. Diese beschreibt die „Zusammengehörigkeit“ eines Splits mit einem Join auf eine sehr allgemeine Weise. Bei einer Überführung von EPKs in Petrinetze wie in [169] beschrieben, resultiert ein $match(s, j)$ zwischen AND-Join s und XOR-Split j (=Synchronisationsfehler) oder zwischen XOR-Split s und AND-Join j (=Deadlock) stets darin, dass das Petrinetz ein Handle [47] aufweist. Dies ist genau dann der Fall wenn es in einem Petrinetz zwei Pfade zwischen einer Stelle und einer Transition gibt, die außer dem Anfangs- und Endpunkt keine weiteren gemeinsamen Knoten haben. Wie von van der Aalst [168, 166] ausgeführt, können Kontrollflussfehler durch die Analyse solcher Handles gefunden werden.

In einem s-j-Block kann sowohl der Split s als auch der Join j jeden der Typen AND,OR und XOR annehmen. Somit ergeben sich $3^2 = 9$ mögliche Kombinationen der Typen. Alle denkbaren Kombinationen von Typen der Konnektoren und alle denkbaren Ein- und Aussprünge wurden auf mögliche Probleme hin untersucht. Das Ergebnis zeigt Tabelle 8.1 auf Seite 107: In dieser Tabelle ist aufgeführt, zu welchen Mustern sich mögliche Probleme in s-j-Blöcken zusammenfassen lassen.

s-j-Blöcke, bei denen die Typen von Split und Join nicht zueinander passen, sind erfahrungsgemäß für die Mehrzahl von Kontrollflussfehlern in einem Modell verantwortlich. Der allgemeine untersuchte Fall ist in der linken Darstellung von

Abb. 8.12 gezeigt. Das Wolkensymbol steht hier für beliebige Modellelemente, auch Ein- und Aussprünge sind möglich.

Weiterhin wurde der wichtige Sonderfall untersucht, dass eine Kontrollflusskante zu einem Transfer im s-j-Block (vgl. Def. 34 auf Seite 104) führt. Dieser Fall ist in Abb. 8.12 rechts dargestellt.

Dieser Fall wurde auch in [69] als häufiger Anwendungsfall bei der Konstruktion von Modellen beschrieben. Es zeigte sich, dass alle Probleme, die durch einen zusätzlichen Transfer im s-j-Block entstehen können, bereits durch Anwendung der in Tabelle 8.1 erfassten Muster erkannt werden. Durch die Untersuchung aller Arten von s-j-Blöcken mit einem durch eine Kante hervorgerufenen Transfer wurde allerdings mit Muster 18 (siehe Seite 146) ein Muster identifiziert, bei dem eine Ersetzung eines OR-Splits durch einen XOR-Split die Lesbarkeit des Modells verbessern kann.

type(s)	∧	∧	∧	∨	∨	∨	×	×	×
type(j)	∧	∨	×	∧	∨	×	∧	∨	×
kein Einsprung								Muster 7, Seite 122	
Upstream-Einsprung vom Typ ∧								Muster 13, Seite134	
Upstream-Einsprung vom Typ ∨		Muster 7, Seite 122	Muster 4, Seite 117	Muster 1, Seite 112		Muster 4, Seite 117	Muster 1, Seite 112		Muster 19, Seite 147
Upstream-Einsprung vom Typ ×	Muster 2, Seite 114								
Downstream-Einsprung	Muster 3, Seite 115	Muster 5, Seite 118						Muster 7, Seite 122	

Tabelle 8.1: Klassifizierung von Problemmustern in s-j-Blöcken

Ist ein Feld in dieser Tabelle grau unterlegt, so führt die entsprechende Kombination nicht zu Fehlern.

Anmerkung: Aussprünge aus s-j-Blöcken sind in der Tabelle nicht separat erfasst. Für diese gibt es nur einen bemerkenswerten Fall: type(s)=type(j)=AND mit einem echten Aussprung. Dieser Fall ist ein Spezialfall des auf Seite 134 vorgestellten Musters 13.

8.2.4 Probleme in Zyklen

In einer wohlstrukturierten EPK laut Def. 22 sind Zykleneinstiegs- und -ausstiegsknoten stets XOR-Konnektoren. Die Verwendung von OR- oder AND-Splits als Zyklenausstiegsknoten führt ebenso wie die Verwendung von AND-Joins als Zykleneinstiegsknoten zu Kontrollflussfehlern im Modell. Diese sind in den Mustern 11 (Seite 131) und 12 (Seite 133) erfasst.

Wird ein OR-Join als Zykleneinstiegsknoten verwendet, ist es in der Regel empfehlenswert, diesen der besseren Lesbarkeit wegen durch einen XOR-Join zu ersetzen. Dieser Fall wird auf Seite Seite 125 unter der Musternummer 8 diskutiert.

8.2.5 Fehler außerhalb von s-j-Blöcken und Zyklen

Die in den vorigen Abschnitten betrachteten Probleme ergeben sich daraus, dass in einem s-j-Block oder einem Zyklus nicht zueinander passende Split- und Join-Konnektoren verwendet werden.

Wir betrachten nun Situationen, in denen sich ein Kontrollflussfehler aufgrund der Tatsache ergeben kann, dass zu einem Split s im Modell überhaupt kein Join j mit $match(s, j)$ existiert.

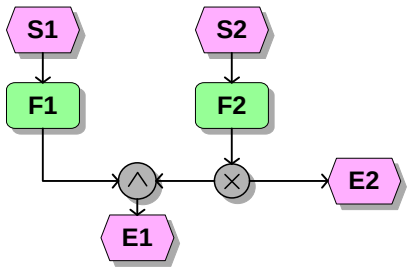


Abbildung 8.13: möglicher Deadlock am AND-Join

Der wichtigste Fall ist in Abb. 8.13 dargestellt. In diesem Modell ist ein Deadlock am AND-Join möglich, falls die von S1 aus erreichbare Eingangskante markiert ist, jedoch am XOR-Split die Entscheidung getroffen wurde, den Pfad in Richtung E2 zu durchlaufen.

Bei der Analyse der EPKs des SAP R/3-Referenzmodells durch Mendling [113] erwies sich dieses Fehlermuster als das am häufigsten gefundene unter allen in [113] betrachteten Mustern. In unserem Musterkatalog ist es unter der Musternummer 13 (Seite 134) zu finden.

Ein selteneres Problemmuster, das aus der Erreichbarkeit von Joins von mehreren Startereignissen aus resultiert, wird auf Seite 142 unter der Musternummer 16 diskutiert.

8.2.6 Unnötige OR-Konnektoren im Modell

Neben dem schon in Abschnitt 8.2.4 beschriebenen Fall, dass ein Zykleneinstiegsknoten als OR statt XOR modelliert ist, nennt Abschnitt 3.3.2 verschiedene weitere Fälle, in denen es zu empfehlen ist, OR-Konnektoren durch XOR-Konnektoren zu ersetzen. Muster 7 befasst sich mit s-j-Blöcken, in denen ein OR-Join durch einen besser verständlichen AND- oder XOR-Join ersetzt werden kann. Muster 9 beschreibt den Fall, dass die alternative Abarbeitung von Knoten durch einen OR-Join begonnen wurde, statt den zu diesem Zwecke besser geeigneten XOR-Join zu nutzen. Muster 18 beschreibt schließlich einen Fall, in dem ein OR-Split durch einen XOR-Split ersetzt werden kann.

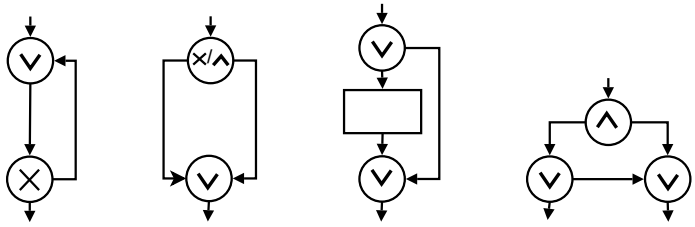


Abbildung 8.14: OR-Konnektoren können ersetzt werden,
von links nach rechts: Muster 8, 7, 9 und 18

8.2.7 Verwendung von Startereignissen

In einer wohlstrukturierten EPK nach Def. 22 ist sichergestellt, dass Startereignisse tatsächlich nur am „Anfang“ des Modells zu finden sind und im Falle mehrerer Startereignisse vor Beginn der eigentlichen Abarbeitungslogik im Modell durch Joins zusammengeführt werden. In einem solchen Modell ist auf einen Blick erkennbar, welches die erlaubten Startzustände sind, d.h. welche Kombinationen von Startereignissen eintreten dürfen. Werden dagegen im unstrukturierten Falle die von den Startereignissen ausgehenden Pfade erst zusammengeführt, nachdem bereits Splits, Funktionen oder Ereignisse durchlaufen wurden, ist dies weniger gut zu erkennen.

Abb. 8.15 zeigt die Struktur des Modells „Meldung“ aus dem Zweig „Projektbezogene Instandhaltungsabwicklung“ aus dem SAP R/3-Referenzmodell. (Beschriftungen

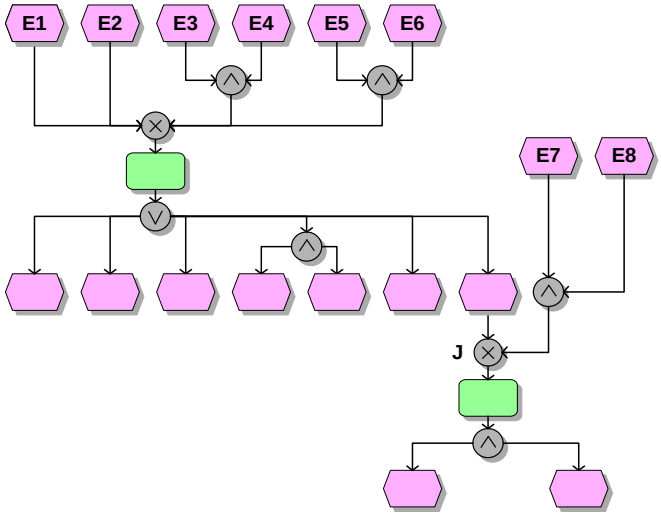


Abbildung 8.15: Die zulässigen Startzustände sind nicht unmittelbar zu erkennen

von Ereignissen und Funktionen wurden weggelassen). Während sofort erkennbar ist, welche Kombinationen der Ereignisse E1 bis E6 in einem Startzustand vorkommen dürfen, ist der Zusammenhang zu den Startereignissen E7 und E8 erst bei etwas genauerem Hinsehen zu identifizieren. Er ergibt sich aus der Tatsache, dass an dem mit J bezeichneten XOR-Join nicht beide Eingangskanten markiert werden dürfen.

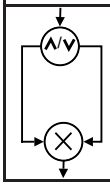
Auf eine derartige Verletzung der Wohlstrukturiertheit durch Startereignisse, die nicht am „Anfang“ des Modells stehen, weist das Muster 19 (Seite 147) hin.

Weitere stilistische Probleme, die durch eine Verwendung von Startereignissen unter Verletzung der Wohlstrukturiertheits-Eigenschaft auftreten können, werden durch die Muster 14 (Seite 138), 15 (Seite 140) und 17 (Seite 144) betrachtet.

9 Katalog für Problemmuster in EPK-Modellen

In diesem Kapitel werden die identifizierten Problemmuster vorgestellt und in Form eines Musterkatalogs zusammengefasst.

Die Darstellung jedes Musters beginnt mit einem Schema der folgenden Art, das die wichtigsten Informationen zu diesem Muster enthält:

Muster 1 (Fehler)	s-j-Block, $type(s) \in \{AND, OR\}$, $type(j) = XOR$
	Beschreibung

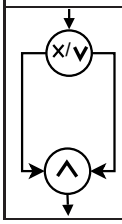
Der Identifikation des Musters durch eine Nummer folgt eine Einstufung in eine der Klassen „Fehler“ bzw. „Warnung“ sowie eine genaue Definition des Musters. Im darunterstehenden zweigeteilten Feld ist neben einer das Muster beschreibenden Skizze eine kurz gefasste Beschreibung des Modellierungsproblems zu finden.

In einer solchen Skizze stehen Kreise ohne eines der Symbole $\vee$, $\wedge$ oder $\times$ für beliebige Konnektoren. Zu beachten ist, dass die Skizze nicht immer alle Fälle darstellt, die das Muster umfasst. Sie soll lediglich dazu dienen, mit einem Blick eine mögliche Situation, für die das Muster zutrifft, zu erfassen. Ausführlichere Erläuterungen sowie Beispiele folgen unter den Punkten „Beispiele“ und „Diskussion“.

Die Beispiele zeigen Modelle oder Modellfragmente (also nicht notwendig syntaktisch vollständige EPKs), in denen das entsprechende Muster zu finden ist.

Wenn möglich, wurden die Beispiele aus realen Modellen der in Kapitel 11 beschriebenen EPK-Modellsammlung entnommen.

9.1 s-j-Block mit (X)OR-Split und AND-Join

Muster 1 (Fehler oder Warnung)	s-j-Block, $type(s) \in \{XOR, OR\}$, $type(j) = AND$
	Deadlock am AND-Join ist möglich.

Beispiele:

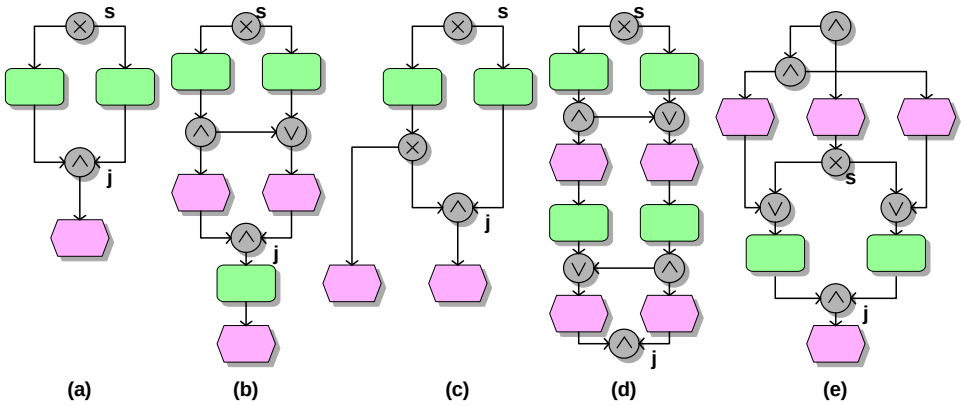


Abbildung 9.1: Deadlock in den Fällen (a) bis (c)

Diskussion: Ein XOR- oder OR-Split kombiniert mit einem AND-Join stellt den typischen Fall eines Deadlocks dar. In den in Abb. 9.1 gezeigten Modellfragmenten (a) bis (c) ist leicht zu sehen, dass ein Deadlock am AND-Join j auftritt.

Die Beispiele (d) und (e) in Abb. 9.1 zeigen allerdings, dass unter besonderen Umständen trotz Vorliegen des Musters die Soundness-Eigenschaft erfüllt sein kann.

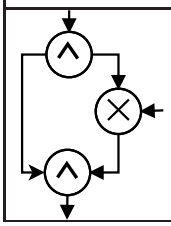
Der entscheidende Unterschied zwischen den Beispielen (a) bis (c) einerseits und (d) und (e) andererseits, ist folgender: In allen Fällen gibt es (lt. Definition der match-Relation) zwei schnittfreie Pfade P_1 und P_2 von s nach j. In den Beispielen (a) bis (c) gilt weiterhin: Sobald am Split die Entscheidung getroffen wurde, einen dieser Pfade zu durchlaufen (wir nehmen o.B.d.A. an, dass dies P_2 sei), kann kein Knoten des anderen Pfades P_1 markiert werden, bevor der Join j erreicht wurde.

Eine hinreichende Bedingung für das Vorliegen eines Deadlocks ist also:

- dass es zwischen dem (X)OR-Split s und dem AND-Join j zwei schnittfreie Pfade P_1 und P_2 gibt und
- dass P_1 keine Einsprünge in den s-j-Block enthält und
- dass es keinen Transfer von P_2 nach P_1 gibt.

Sind all diese Bedingungen erfüllt, soll unser Algorithmus die Meldung „Fehler“ ausgeben. Andernfalls wird der Hinweis auf einen möglichen Deadlock als „Warnung“ formuliert. Dabei soll nicht weiter geprüft werden, ob ein Deadlock tatsächlich vorliegt oder dies (so wie in den Beispielen (d) und (e)) nicht der Fall ist. Dieser Entscheidung liegt die Annahme zugrunde, dass Fälle wie in Abb. 9.1 (d) und (e) dargestellt in der Praxis selten vorkommen.

9.2 s-j-Block mit AND-Split, AND-Join und Upstream-Einsprung vom Typ XOR

Muster 2 (Fehler oder Warnung)	s-j-Block, $type(s) = type(j) = AND$, Upstream-Einsprung e mit $type(e) = XOR$
	Synchronisationsfehler an e oder Deadlock an j möglich

Beispiele:

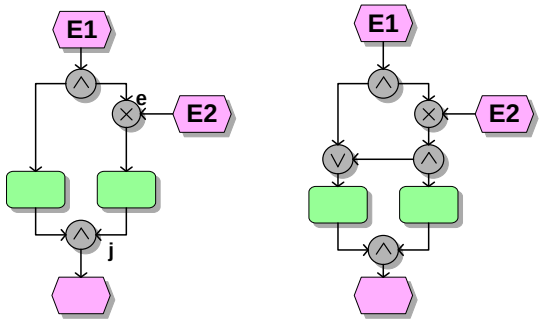


Abbildung 9.2: Synchronisationsfehler im linken Modell

Diskussion:

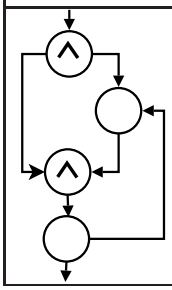
Das Problemmuster soll anhand des linken Modells von Abb. 9.2 dargestellt werden. Damit es weder zu einem Deadlock an j noch zu einem Synchronisationsfehler an e kommt, darf E2 nicht im Startzustand enthalten sein. Somit wäre das Startereignis überflüssig, da es in keinem Startzustand, der zu einem fehlerfreien Ablauf des Modells führt, enthalten sein kann. Laut Soundness-Definition Def. 17 ist wegen dieses Startereignisses das Modell nicht sound.

Ähnlich wie bei Muster 1 sind auch bei Muster 2 Fälle denkbar, in denen das Modell trotz Vorliegen des Musters korrekt ist. Ein solcher Fall ist im rechten Modell von Abb. 9.2 gezeigt.

Eine Entscheidung, ob die Meldung „Fehler“ oder die Meldung „Warnung“ ausgegeben wird, erfolgt daher auf die gleiche Weise wie für Muster 1 (Seite 112) beschrieben.

Verwandte Muster: Dieses Muster ist ein wichtiger Spezialfall von Muster 19.

9.3 s-j-Block mit AND-Split, AND-Join und Downstream-Einsprung

Muster 3 (Fehler oder Warnung)	s-j-Block, $type(s) = type(j) = AND$, Downstream-Einsprung e mit $type(e) \in \{OR, XOR\}$
	Deadlock am AND-Join möglich, wenn dieser über den Einsprung e zum zweiten Male erreicht wird

Beispiel:

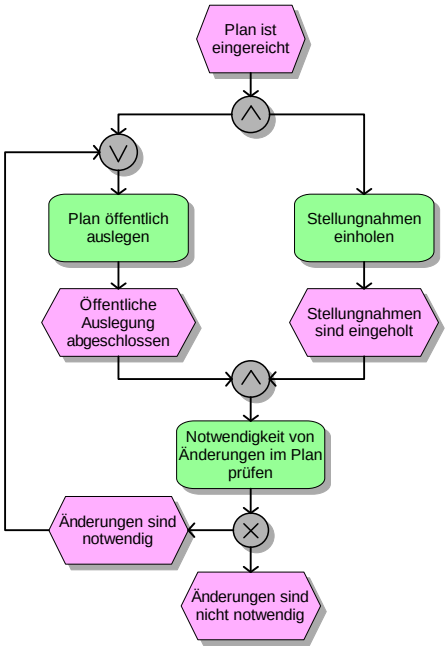


Abbildung 9.3: Deadlock am AND-Join nach Ereignis „Änderungen sind notwendig“

Das Beispiel zeigt ein abgewandeltes und vereinfachtes Modell aus [163]. Falls das Ereignis „Änderungen sind notwendig“ eintritt, kommt es zu einem Deadlock am AND-Join, da nur eine seiner Eingangskanten markiert wird.

Diskussion: Wird der s-j-Block vom AND-Split s aus durchlaufen, so gibt es

zunächst keine Probleme. Erreicht allerdings dann der Kontrollfluss durch den Einsprung e erneut den s-j-Block, so kommt es zu einem Deadlock am AND-Join.

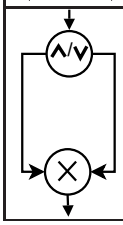
Analog zu Muster 1 (Seite 112) sind auch hier Fälle denkbar, in denen das Modell trotz Vorliegen des Musters korrekt ist. Die Einstufung eines gefundenen Musters in die Klassen „Fehler“ bzw. „Warnung“ erfolgt wie bei Muster 1 beschrieben.

Verwandte Muster: Der AND-Join ist bei Vorliegen dieses Musters stets ein Zykleneinstiegsknoten. Der daraus resultierende Deadlock wird als Muster 11 erkannt. Für das Finden aller Kontrollflussfehler wäre es ausreichend, lediglich nach dem Vorkommen von Muster 11 zu suchen. Mit dem spezielleren Muster ist es aber möglich, dem Modellierer eine besser verständliche Rückmeldung zu liefern.

Ist e ein OR-Join, unterbleibt die zusätzliche Warnung, dass er als XOR-Join modelliert werden sollte (vgl. Muster 8, Seite 125), da dies nichts an dem prinzipiellen Fehler ändern würde.

Wird der AND-Join j durch einen OR-Join ersetzt, so wird der Deadlock vermieden. Allerdings hat ein solches Modell möglicherweise andere Schwächen, wie in Muster 5 (Seite 118) diskutiert wird.

9.4 s-j-Block mit AND- oder OR-Split und XOR-Join

Muster 4 (Fehler)	s-j-Block, $type(s) \in \{XOR, OR\}$, $type(j) = AND$
	Synchronisationsfehler am XOR-Join

Beispiel:

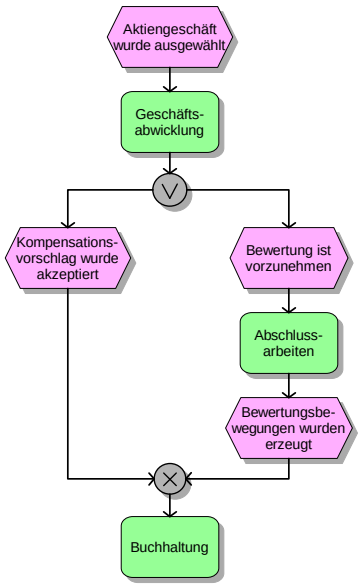
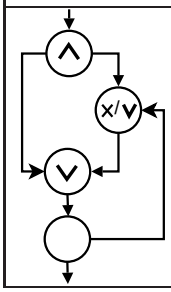


Abbildung 9.4: Synchronisationsfehler am XOR-Join möglich

Das Beispiel zeigt einen Ausschnitt des Modells „Treasury / Aktien TR/SE“ aus dem SAP R/3-Referenzmodell. Um einen Synchronisationsfehler zu vermeiden, muss entweder der OR-Split durch ein XOR-Split oder der XOR-Join durch einen OR-Join ersetzt werden.

Diskussion: Da am Split mehr als eine ausgehende Kante markiert werden kann, kann in der Folge ein Zustand erreicht werden, in dem mehr als eine Eingangskante am XOR-Join markiert ist. Es kann also zu einem Synchronisationsfehler am Join kommen, und zwar unabhängig von eventuellen Ein- und Aussprüngen.

9.5 s-j-Block mit AND-Split, OR-Join und Downstream-Einsprung

Muster 5 (Warnung)	s-j-Block, $type(s) = AND$, $type(j) = OR$, Downstream-Einsprung vom Typ XOR oder OR Im s-j-Block gibt es außer dem genannten Downstream-Einsprung keine weiteren Ein- und Aussprünge. Im s-j-Block gibt es keine Transfers.
	Partial Redo-Muster [65], alternative Modellierung sollte in Betracht gezogen werden

Beispiel:

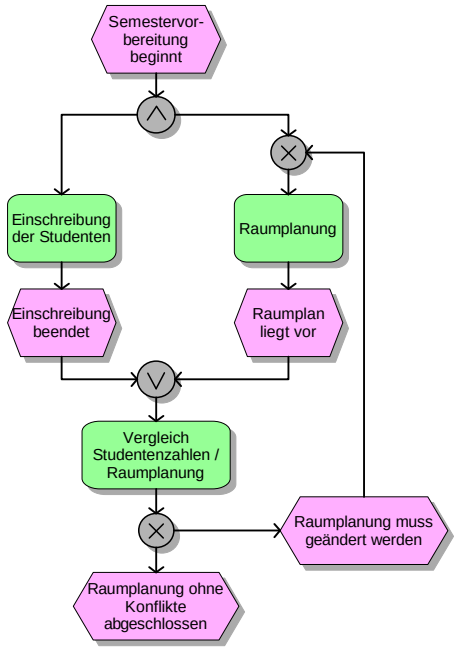


Abbildung 9.5: „Partial-Redo“-Muster [65]

Ein Beispielprozess, der mit dem vorliegenden Muster modelliert sein könnte,

beschreibt die Abläufe an einer Hochschule zu Beginn eines neuen Semesters: Die Vorgänge „Einschreiben von Studenten“ und „Raumplanung“ laufen parallel ab. Nach Beendigung dieser parallelen Aktivitäten erfolgt eine Überprüfung, ob die geplante Raumbelegung mit der Zahl der eingeschriebenen Studenten vereinbar ist. Ist dies nicht der Fall, so ist die Aktivität „Raumplanung“ (aber natürlich nicht das Einschreiben der Studenten) erneut durchzuführen.

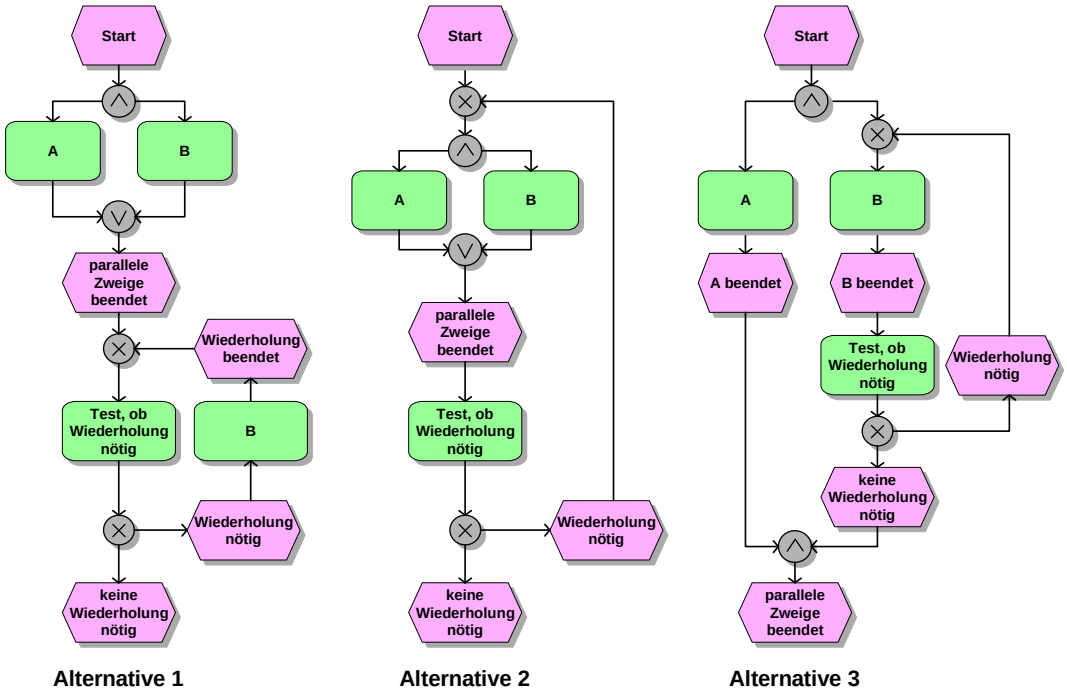


Abbildung 9.6: Modellierungsalternativen

Diskussion: Die beiden Joins bilden einen „Teufelskreis“ wie in Abschnitt 2.5.3 beschrieben. Man kann zeigen, dass ein Modell, das dieses Muster enthält, keine Fixpunktsemantik nach Kindler [87] besitzt [39]. Legt man die strengen Maßstäbe von [87] an, so ist ein solches Modell also in jedem Falle fehlerhaft, da keine zufriedenstellende Semantikdefinition möglich ist. Mit einer Modellierung wie in Alternative 1 von Abb. 9.6 gezeigt (bei der das Ereignis B verdoppelt wurde) werden die Probleme vermieden.

Interessant im Hinblick auf mögliche Modellverbesserungen ist aber auch eine inhaltliche Betrachtung des dargestellten Prozesses: Zunächst werden beginnend mit dem AND-Join zwei parallele Vorgänge gestartet. Nachdem *beide* abgeschlossen sind, erfolgt ein Rücksprung, zum Beispiel nach dem negativen Ausgang einer

Qualitätsprüfung. Ziel dieses Rücksprungs ist jedoch nur einer der ursprünglich parallel bearbeiteten Zweige, d.h. nur einer der Vorgänge ist zu wiederholen.

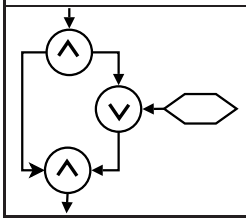
Es ist zu vermuten, dass eine Situation, in der genau einer der Pfade vor dem OR-Join zu wiederholen ist, in der Praxis eher selten ist. Beim Vorliegen dieses Musters sollte der Modellierer die folgenden Fragen beantworten:

1. Soll im Falle eines erforderlichen Rücksprungs tatsächlich nur einer der beiden Pfade wiederholt durchlaufen werden? Andernfalls ergibt sich ein wohlstrukturiertes Modell wie in Alternative 2 von Abb. 9.6.

2. Muss die Entscheidung, ob ein Rücksprung erfolgt, tatsächlich erst nach Abschluss der beiden parallel bearbeiteten Pfade getroffen werden? Falls nein, ist ein wohlstrukturiertes Modell wie in Alternative 3 von Abb. 9.6 vorzuziehen.

3. Ist nach einem Rücksprung tatsächlich derselbe Prozess wiederholt auszuführen oder sind beim erneuten Durchlauf andere Aktivitäten notwendig? Beim Raumplanungs-Beispiel von Abb. 9.5 wäre es beispielsweise ein berechtigter Einwand, dass die im Wiederholungsfall notwendige Aktivität „Anpassung der Raumplanung an die gemeldeten Studentenzahlen“ sich so deutlich von der ursprünglichen Aktivität „Raumplanung“ unterscheidet, dass sie auch als eine andere Aktivität modelliert werden sollte. Damit ergäbe sich ein Modell ähnlich Alternative 1 von Abb. 9.6, wobei lediglich die untere Funktion B durch eine andere Funktion B’ zu ersetzen ist.

9.6 s-j-Block mit AND-Split, AND-Join und Startereignis vor Upstream-Einsprung vom Typ OR

Muster 6 (Warnung)	s-j-Block, $type(s) = type(j) = AND$, Upstream-Einsprung e vom Typ OR Es gibt eine Kante von einem Startereignis zu e.
	Das Startereignis hat keinen Einfluss auf die Abarbeitung des Geschäftsprozesses.

Beispiel:

Abb. 9.7 zeigt einen vereinfachten Ausschnitt des Modells „Operative Unternehmensplanung“ aus dem Zweig „Unternehmenscontrolling“ des SAP R/3-Referenzmodells. Ob das Startereignis „Geschäftsjahr/Periode ist zu kalkulieren“ eintritt oder nicht, hat hier keinerlei Einfluss auf die Abarbeitung. Offenbar war beabsichtigt, dass es von einem AND-Join gefolgt werden soll (vgl. Muster 15, Seite 140).

Diskussion: Das Vorliegen dieses Musters führt nicht zu einer Verletzung der Soundness-Eigenschaft. Dennoch erfolgt eine Warnung, da die Bedeutung des Startereignisses x, dessen Ausgangskante zum Einsprung e führt, fraglich ist.

Bei einem korrekten Ablauf des Prozesses muss nämlich, damit es am Join j keinen Deadlock gibt, stets von beiden Seiten ein Kontrollfluss den Join erreichen. Ob nun das Startereignis x im Startzustand enthalten ist oder nicht, ist dann für den Ablauf irrelevant. Das Startereignis x ist in diesem Falle überflüssig.

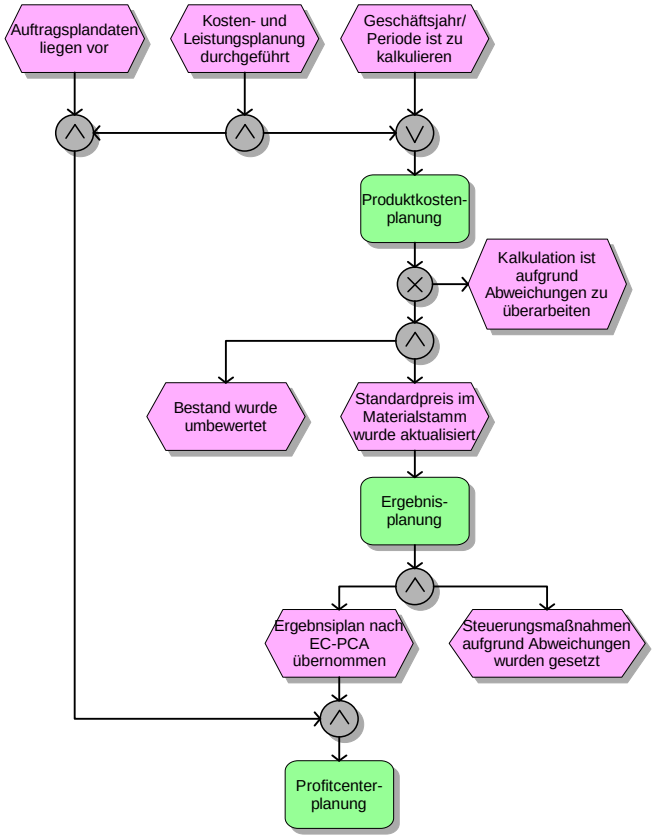
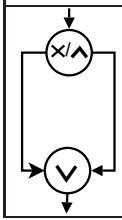


Abbildung 9.7: Startereignis „Geschäftsjahr/Periode ist zu kalkulieren“ ist ohne Einfluss

9.7 s-j-Block mit AND- oder XOR-Split und OR-Join

Muster 7 (Warnung)	s-j-Block, $type(s) \in \{AND, XOR\}$, $type(j) = AND$ Weitere Bedingungen sind in der Erklärung unten angegeben.
	Der Join j sollte nicht als OR-Join modelliert sein, sondern den gleichen Typ wie s (AND bzw. XOR) haben.

Beispiel:
Im gezeigten Modellfragment, entnommen der EPK „Rahmenverträge“ aus dem

Zweig „Beschaffung von Material / externe Dienstleistungen“ des SAP R/3-Referenzmodells, kann (und sollte!) der OR-Join durch einen AND-Join ersetzt werden.

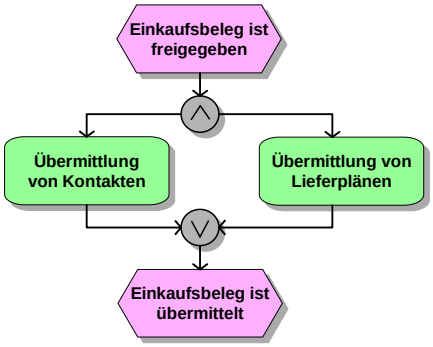


Abbildung 9.8: Der OR-Split kann hier durch AND ersetzt werden.

Diskussion: Die (informell beschriebene) Semantik eines OR-Joins heißt: „Warte auf die Beendigung aller Pfade, von denen aus Kontrollfluss eintreffen kann.“ Somit verhält sich ein OR-Join nach einem AND-Split (wie in Abb. 9.8) wie ein AND-Join – er wartet, bis an *allen* Eingangskanten eine Markierung anliegt. Analog verhält sich ein OR-Join nach einem XOR-Split wie ein XOR-Join. Modelle wie das in Abb. 9.8 gezeigte sind somit durchaus formal korrekt. Die Verwendung des für die Situation tatsächlich „passenden“ Joins (also etwa in Abb. 9.8 eines AND-Joins) vereinfacht jedoch die Lesbarkeit des Modells. So wäre in Abb. 9.8 sofort erkennbar, dass die Vorbedingung für das Ereignis „Einkaufsbeleg ist übermittelt“ ist, dass *beide* Funktionen ausgeführt wurden. In einem so einfachen Modell wie dem in Abb. 9.8 gezeigten mag dies auch ohne Modelländerung noch sofort zu erkennen sein; bei komplexeren Modellen (in denen Split und Join räumlich getrennt sein können) ist dies jedoch nicht ohne weiteres der Fall.

Unter Umständen muss eine Warnung für dieses Muster unterbleiben, und zwar wenn einer der folgenden Fälle vorliegt:

- Es existieren zwei Startereignisse E_1 und E_2, für die es schnittfreie Pfade von E_1 zu j und von E_2 zu j gibt (schematisch dargestellt in Abb. 9.9(a)).
- j ist Zykleneinstiegsknoten (schematisch dargestellt in Abb. 9.9(b)).
- Rs gibt zwei Splits s_1 und s_2 mit $type(s_1) \neq type(s_2)$, für die sowohl $match(s_1, j)$ als auch $match(s_2, j)$ gilt (schematisch dargestellt in Abb. 9.9(c)).
- Es gibt einen Split s vom Typ AND, für den $match(s, j)$ gilt und es einen echten Aussprung aus dem s-j-Block gibt (schematisch dargestellt in Abb. 9.9(d)).

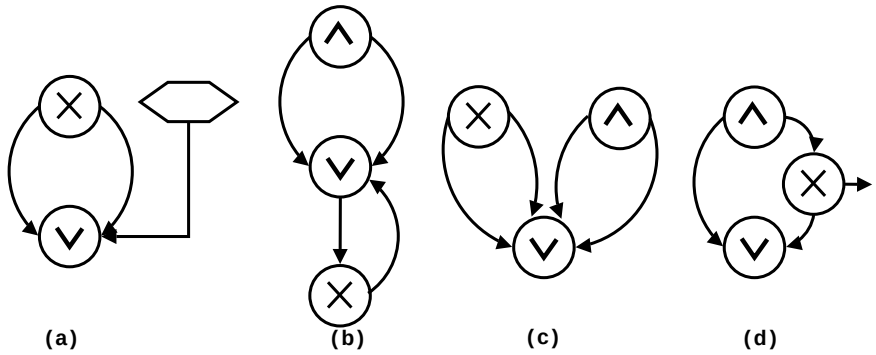
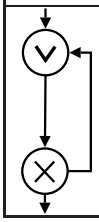


Abbildung 9.9: Der OR-Split kann in diesen Fällen nicht ersetzt werden.

Verwandte Muster: Muster 8 zeigt eine ähnliche Situation, in der ein OR-Join durch einen XOR-Join ersetzt werden kann. Während im vorliegenden Muster zwischen Split und Join eine match-Beziehung vorliegt, bilden Split und Join in Muster 8 die Ein- und Ausstiegsknoten eines Zyklus.

9.8 OR-Join als Zykleneinstiegsknoten

Muster 8 (Warnung)	Zykleneinstiegsknoten hat den Typ OR Weitere Bedingungen sind in der Erklärung unten angegeben.
	Der Zykleneinstiegsknoten kann unter Umständen besser als XOR modelliert werden.

Beispiel:

Im folgenden Modellfragment (Quelle: [23]) kann der OR-Join durch einen XOR-Join ersetzt werden.

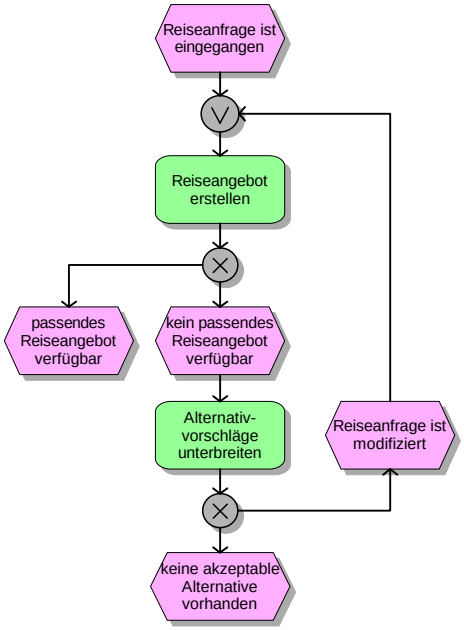


Abbildung 9.10: Der OR-Split kann hier durch XOR ersetzt werden.

Diskussion: Eine Warnung für dieses Muster muss unterbleiben, wenn einer der folgenden Fälle vorliegt:

- Es existieren zwei Startereignisse E_1 und E_2, für die es schnittfreie Pfade von E_1 zu j und von E_2 zu j gibt (schematisch dargestellt in Abb. 9.11(a)).

- Es gibt einen Split s vom Typ AND oder OR, für den $match(s, j)$ gilt (schematisch dargestellt in Abb. 9.11(b)).

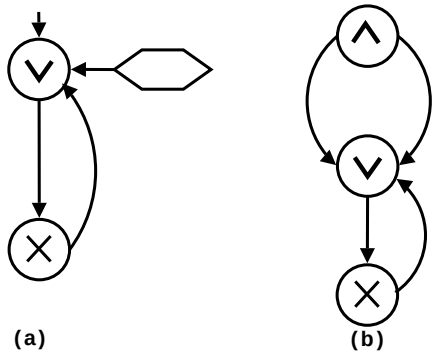


Abbildung 9.11: Der OR-Split kann in diesen Fällen nicht ersetzt werden.

Generell gilt, dass nicht pauschal entschieden werden kann, ob beim Auftreten des Musters der Zykleneinstiegsknoten tatsächlich durch einen XOR-Join ersetzt werden sollte.

Ein Argument für eine Ersetzung ist, dass mit den in Abschnitt 2.5 besprochenen Semantiken in jedem denkbaren Zustand der EPK genau eine Kante markiert ist und somit nie zwei Eingangskanten am OR-Join markiert werden können. Diese Tatsache ist für den Leser eines Modells aber leichter erkennbar, wenn statt des OR-Joins ein Join vom Typ XOR verwendet wird. Erwähnenswert ist, dass dieses Muster nach der Semantik von Mendling [113] sogar zu einer Verletzung der Soundness-Eigenschaft führt[1].

Ein Argument gegen eine Ersetzung ist, dass durch das EPK-Modellierungselement „Ereignis“ in der Praxis neben Ereignissen oft auch Zustände modelliert werden. Während aber ein Ereignis zu einem bestimmten Zeitpunkt auftritt, kann ein Zustand über einen längeren Zeitraum andauern. Im Beispiel von Abb. 9.10 könnte es etwa statt „Reiseanfrage ist eingegangen“ (was ein Ereignis beschreibt) durchaus auch heißen „Bedarf nach Reisebuchung besteht“. Dies beschriebt einen Zustand, der auch dann noch andauert, wenn bereits die Funktion „Reiseangebot erstellen“ abgearbeitet wird. Insbesondere dauert er auch dann noch an, wenn nach dem Ereignis „Reiseanfrage ist modifiziert“ die Schleife im Modell erneut durchlaufen wird. Ein OR-Join kann diese Tatsache möglicherweise besser ausdrücken, da er nicht wie

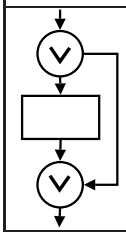
[1] Nach Meinung des Autors ist das ein Nachteil dieser Semantikdefinition.

ein XOR-Join den Eindruck vermittelt, dass sich „Reiseanfrage ist modifiziert“ und „Bedarf nach Reisebuchung besteht“ ausschließen.

Beide Modellierungsvarianten haben also Vor- und Nachteile. Um einen Einheitlichkeit zu erreichen, ist eine organisationsweite Modellierungsrichtlinie sinnvoll.

Verwandte Muster: Muster 11 befasst sich mit dem Fall, dass der Zykleneinstiegsknoten ein AND-Join ist (was stets zu einem Deadlock führt).

9.9 OR-Split startet optionale Ausführung

Muster 9 (Warnung)	s-j-Block, $type(s) = type(j) = OR$, $card\ s\triangleleft = card\ \triangleright j = 2$ Es gibt eine Kante (s, j).
	Split und Join sollten als XOR modelliert werden.

Beispiel:

Im Modell „Produktionsplanung“ im Zweig „Unternehmenscontrolling“ des SAP R/3-Referenzmodells findet sich der folgende Modellabschnitt[2]:

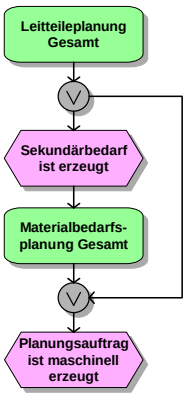


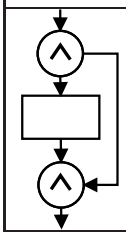
Abbildung 9.12: Statt OR-Konnektoren sollten hier besser XOR verwendet werden.

Diskussion: Im vorliegenden Falle soll die Situation modelliert werden, dass ein bestimmter Zweig entweder durchlaufen oder übersprungen wird. Es gibt also genau zwei Möglichkeiten, was durch einen XOR-Operator ausgedrückt werden sollte. Eine Benutzung von OR-Konnektoren wie in Abb. 9.12 würde erlauben, beide Pfade parallel zu durchlaufen. Das ist aber sinnlos, da einer der Pfade keine Ereignisse und Funktionen enthält.

Verwandte Muster: Muster 10 behandelt den Fall, dass die „leere“ Kontrollflusskante von einem AND-Split zu einem AND-Join führt.

[2]Das gezeigte Modellfragment gibt es nur in der englischsprachigen Version des Referenzmodells; in der deutschsprachigen wurde die fragliche Kante entfernt.

9.10 AND-Split startet optionale Ausführung

Muster 10 (Warnung)	s-j-Block, $type(s) = AND$, $type(j) \in \{AND, OR\}$, $card\ s\triangleleft = card\ \triangleright j = 2$ Es gibt eine Kante (s, j).
	Die „leere“ Kontrollflusskante zwischen dem AND-Split und dem Join ist nutzlos und sollte (ggf. mitsamt des Splits und des Joins) entfernt werden.

Beispiel:

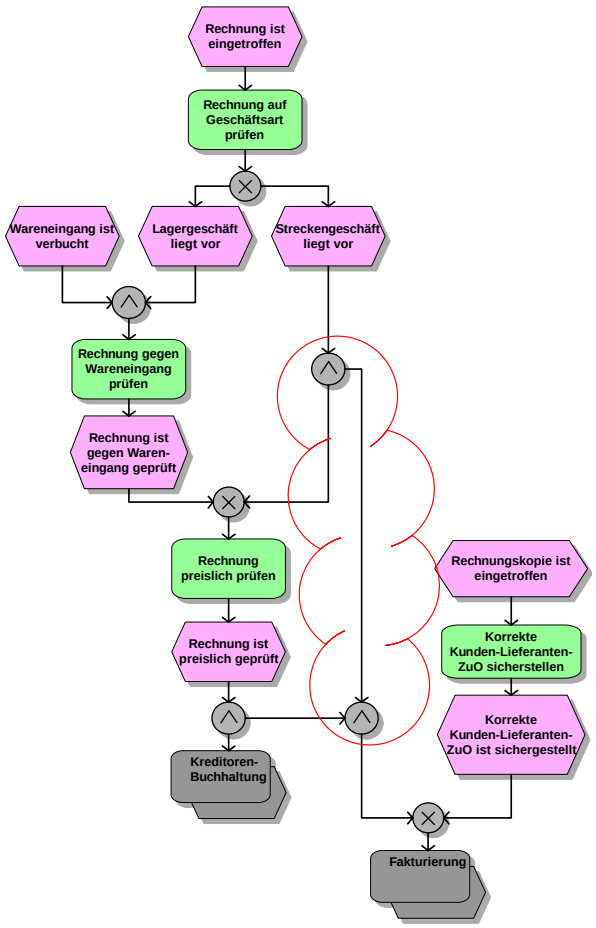


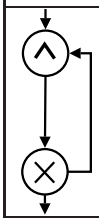
Abbildung 9.13: Kontrollflusskante und AND-Konnektoren sind nutzlos

Das in Abb. 9.13 gezeigte Modell wurde [10] entnommen. Die mit dem Wolkensymbol gekennzeichneten Modellelemente (AND-Split, Kontrollflusskante und AND-Join) können ohne jeden Informationsverlust weggelassen werden, woraus sich ein einfacher lesbares Modell ergibt.

Diskussion: Das Modellieren einer „leeren“ Kontrollflusskante zwischen einem AND-Split und einem dazugehörigen Join ist ohne jeden inhaltlichen Sinn. Der Leser kann durch unnötige Notationselemente verwirrt werden. Die leere Kontrollflusskante sollte entfernt werden. Falls der AND-Split genau zwei Ausgangskanten und der dazugehörige Join genau zwei Eingangskanten hat, können auch Split und Join entfallen.

Verwandte Muster: Muster 9 behandelt den Fall, dass die „leere“ Kontrollflusskante von einem OR-Split zu einem OR-Join führt.

9.11 AND-Join als Zykleneinstiegsknoten

Muster 11 (Fehler)	Zykleneinstiegsknoten hat den Typ AND
	Wird ein AND-Join als Zykleneinstiegsknoten verwendet, heißt dies, dass fast immer am Zykleneinstiegsknoten ein Deadlock möglich ist (siehe aber „Diskussion“ unten!).

Beispiel:

Abb. 9.14 zeigt das Modell „Anwerbungsauftragsüberwachung“ aus dem Zweig „Arbeitgeberleistungsadministration“ des SAP R/3-Referenzmodells.

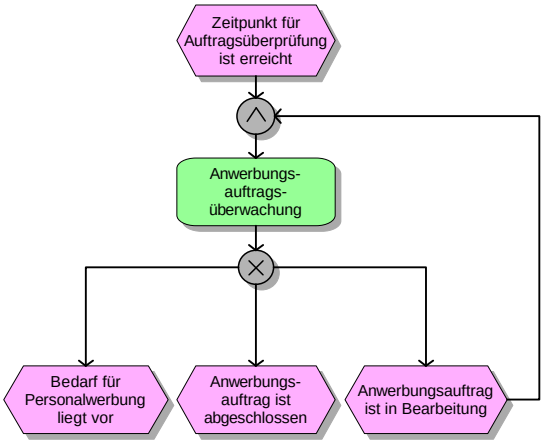


Abbildung 9.14: AND-Join als Zykleneinstiegsknoten

Diskussion: Auch wenn sich Modelle wie das in Abb. 9.15 gezeigte konstruieren lassen, die trotz des AND-Joins j als Zykleneinstiegsknoten sound sind[3], liegt die Annahme nahe, dass es sich bei den allermeisten in der Praxis vorkommenden Fällen um eine tatsächliche Verletzung der Soundness-Eigenschaft handelt. Daher wird beim Auftreten dieses Musters stets eine Meldung der Klasse „Fehler“ ausgegeben.

Wie schon in Abschnitt 5.4 erörtert wurde, entsprechen die in der Literatur eingeführten Semantiken unter Umständen nicht der intuitiven Vorstellung der Anwender in der betrieblichen Praxis. Inspiriert von Petrinetzen, gehen die Semantiken (wie in Abschnitt 2.5 beschrieben) von einem *Verschieben* von Markierungen

[3] unter Annahme der Fixpunktsemantik nach Kindler [87]

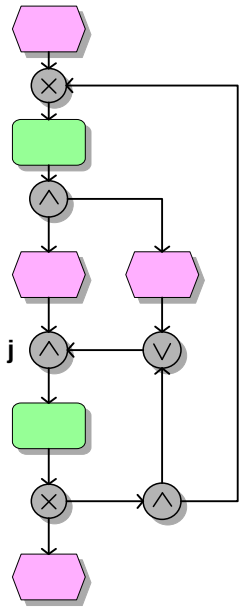


Abbildung 9.15: AND-Join als Zykleneinstiegsknoten, Modell ist sound

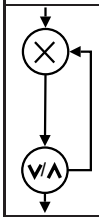
im EPK-Graphen aus. Abb. 9.14 ist ein gutes Beispiel dafür, dass dieser Ansatz Schwächen hat, wenn das Modellierungselement „Ereignis“ genutzt wird, um einen Zustand zu modellieren. Ein solcher Zustand kann nämlich - im Gegensatz zu einem einmal eintretenden Ereignis - fortbestehen. Man kann für das Modell in Abb. 9.14 argumentieren, dass der Zustand „Zeitpunkt der Auftragsüberprüfung ist erreicht“ dauerhaft bestehen bleibt, wenn er erst einmal eingetreten ist. Eine Markierung an der Ausgangskante dieses Zustandes dürfte somit bei einem Zustandsübergang nicht einfach weiterverschoben werden. Vielmehr müsste die obere Eingangskante des AND-Joins dauerhaft markiert bleiben. Entsprechende Semantiken wurden bisher nicht vorgeschlagen, und eine diesbezügliche Diskussion ist auch nicht Hauptgegenstand dieser Arbeit.

Für unseren musterbasierten Ansatz ist es lediglich wichtig, dem Modellierer beim Vorliegen dieses Musters eine aussagekräftige Information zu liefern, da ein bloßer Verweis auf einen Deadlock am AND-Join aus den genannten Gründen eventuell nicht nachvollzogen werden kann.

Verwandte Muster: Ein Spezialfall dieses Musters ist Muster 3.

Muster 8 befasst sich mit dem weniger kritischen Fall, dass der Zykleneinstiegsknoten ein OR-Join ist.

9.12 AND- oder OR-Split als Zyklenausstiegsknoten

Muster 12 (Fehler)	Zyklenausstiegsknoten hat nicht den Typ XOR
	Wird ein OR- oder AND-Split als Zyklenausstiegsknoten verwendet, kann das dazu führen, dass die Ausgangskante, die aus dem Zyklus herausführt (und somit ein ihr folgendes Endereignis), mehrfach „erreicht“ wird.

Beispiel:

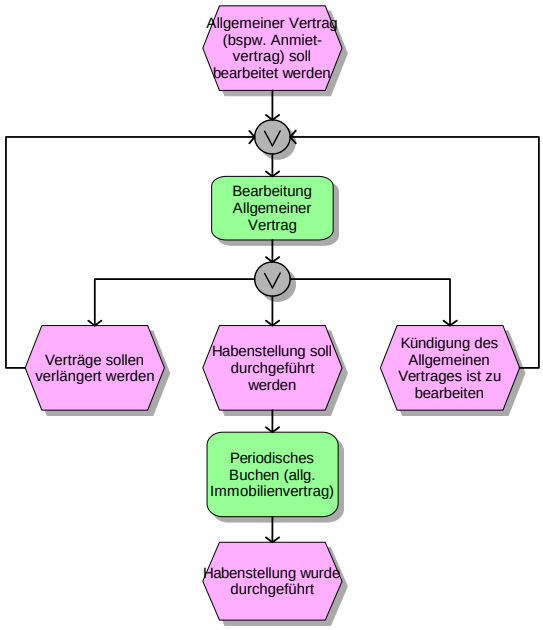
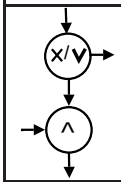


Abbildung 9.16: AND-Join als Zykleneinstiegsknoten

Das Beispielmodell „Allgemeiner Vertrag“ aus dem Zweig „Immobilienmanagement“ des SAP R/3-Referenzmodells lässt es zu, dass am OR-Split, der den Zyklus beendet, sowohl der Kontrollflusspfad in Richtung Endereignis „Habenstellung wurde durchgeführt“ eingeschlagen wird als auch der Zyklus ein weiteres Mal durchlaufen wird. Das periodische Buchen wird dann mehrfach durchgeführt, und das Endereignis kann mehrfach erreicht werden.

Diskussion: Bei einem Modell mit diesem Muster kann von einem Modellierungsfehler (mehrfache Beendigung) ausgegangen werden.

9.13 Kontrollfluss erreicht nach einem (X)OR-Split nicht den AND-Join

Muster 13 (Warnung)	Der von einem (X)OR-Join ausgehende Kontrollfluss erreicht möglicherweise nur eine Eingangskante am AND-Join (formale Definition siehe unten)
	Nach Durchlaufen des Splits ist es möglich, dass der Kontrollfluss nur eine Eingangskante des AND-Joins erreicht und es somit zum Deadlock kommt.

Beispiele:

In Abb. 9.17 (ein vereinfachter Abschnitt aus dem Modell „Serviceauftrag“ aus dem Zweig „Qualitätsmanagement“ des SAP R/3-Referenzmodells) tritt dieses Muster in Form eines XOR-Aussprungs aus einem AND-Kontrollblock auf. Als Folge kann am AND-Join ein Deadlock entstehen.

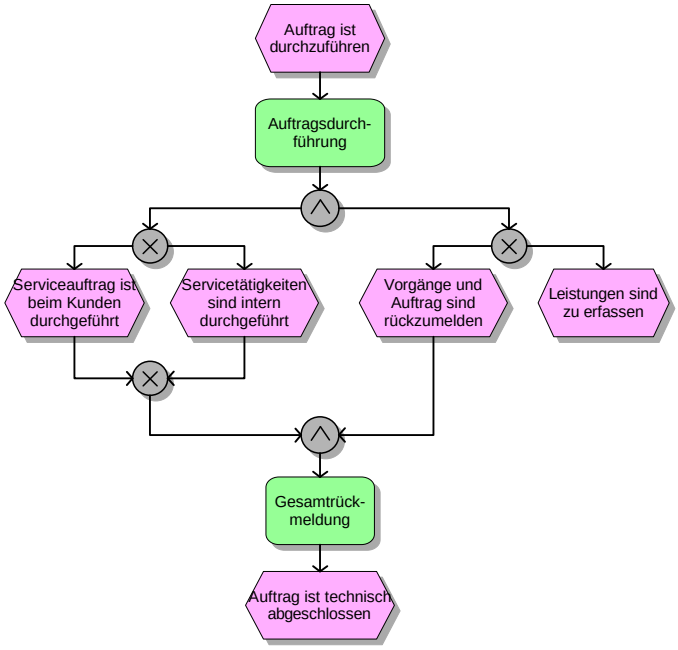


Abbildung 9.17: XOR-Aussprung aus AND-Kontrollblock

Abb. 9.18 hingegen (entnommen dem Modell „Veranstaltungsplanung und -durchführung“ aus dem Zweig „Veranstaltungsmanagement“ des SAP R/3-Referenzmodells) zeigt eine Situation, die als fehlerfrei interpretiert werden kann.

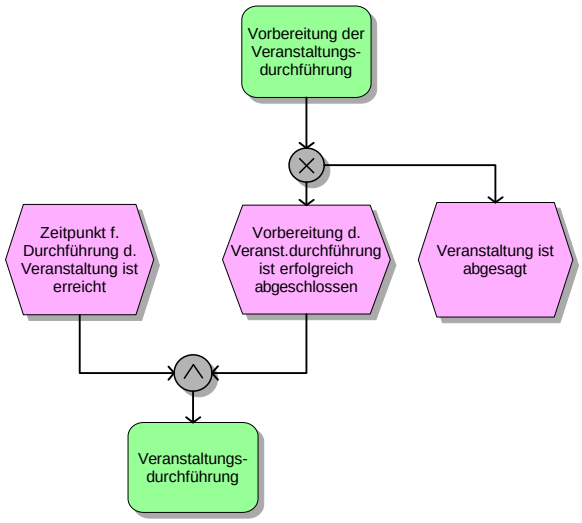


Abbildung 9.18: AND-Join kann bei abgesagter Veranstaltung nicht synchronisieren

Diskussion: Soll ein Vorkommen dieses Musters lokalisiert werden, sind (X)OR-Splits zu finden, von denen aus der Kontrollfluss zu nur einer Eingangskante eines AND-Joins führt. Formal kann das Muster durch die folgenden Forderungen definiert werden:

- Es gibt einen Pfad von einem XOR- oder OR-Split s zu einem AND-Join a.
- Es existiert eine Kante (s, s^*) mit $s^* \neq j$ sowie eine Kante (j^*, j) mit $j^* \neq s$.
- Es gibt keinen Pfad von s nach j^* sowie keinen Pfad von s^* nach j.
- Es gibt keinen AND-Split a und keinen Join $e \neq j$, so dass es schnittfreie Pfade von a nach s sowie von a nach e gibt und e in einem nichtzyklischen Pfad von s nach j enthalten ist.

Die oben eingeführten Bezeichnungen sind in Abb. 9.19(a) auf Seite 136 dargestellt. Abb. 9.19(b) zeigt den Fall, der durch die letzte der obigen Forderungen ausgeschlossen wird.

Formal gesehen, ist bei allen Modellen mit diesem Muster ein Deadlock am AND-Join möglich. Daher werden etwa von Mendling [113] solche EPKs grundsätzlich als fehlerhaft eingestuft. In vielen Fällen ist allerdings der Deadlock vom Modellierer durchaus in Kauf genommen oder sogar beabsichtigt. Dies ist z.B. der Fall, wenn Startereignisse als Bedingungen interpretiert werden, die erst während der

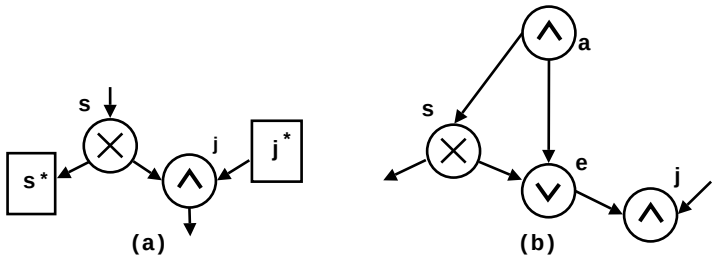


Abbildung 9.19: Bezeichnungen in der oben eingeführten formalen Beschreibung des Musters (s^* und j^* können beliebige Knoten sein.)

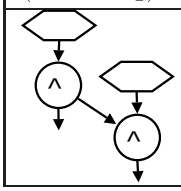
tatsächlichen Abarbeitung des Geschäftsprozesses eintreten bzw. geprüft werden. Beispielsweise kann in der in Abb. 9.18 dargestellten Situation aus inhaltlichen Gründen das Startereignis „Zeitpunkt für die Durchführung der Veranstaltung ist erreicht“ gar nicht mehr eintreten, wenn der Weg zum Endereignis „Veranstaltung ist abgesagt“ durchlaufen wurde. Decker und Mendling schlugen in [32] für solche Fälle eine andere Semantik für Startereignisse vor, die von früheren Definitionen, wie sie etwa von Rump [145] verwendet wurde, abweicht: Sobald garantiert ist, dass ein Prozess terminiert, ohne dass die Markierung von der Ausgangskante eines Startereignisses entfernt werden muss, wird diese Markierung aus dem Startzustand gestrichen. Eine solche Semantik stimmt gut mit der intuitiven Vorstellung überein. Bisher wurde dieser semantische Ansatz jedoch eher informell beschrieben. Eine formale Definition wurde noch nicht veröffentlicht.

Einen „echten“ Fehler dürfte dieses Muster dagegen in Abb. 9.17 bewirken: Wird der Weg zum Endereignis „Leistungen sind zu erfassen“ genommen, kommte es zu einem Deadlock am AND-Join. Hier liegt die Vermutung nahe, dass der Modellierer vergessen hat, den XOR-Kontrollblock (so wie im linken Teil des Modells geschehen) zu schließen.

In manchen Fällen soll durch einen Aussprung aus dem s-j-Block beim XOR-Split e modelliert werden, dass der gesamte dargestellte Prozess beendet werden kann (dass es also gar nicht zu einer Synchronisation am AND-Join kommen soll). In diesem Falle wäre zu überlegen, ob die beiden mit dem AND-Split s eingeleiteten Ausführungspfade tatsächlich parallel abgearbeitet werden sollten. Unter Umständen werden die Arbeiten in einem der Pfade nämlich umsonst ausgeführt, wenn im anderen parallel laufenden Pfad bereits entschieden wurde, dass der Aussprung zum Endereignis erfolgt. Eventuell ist hier sogar eine Verbesserung des tatsächlichen Geschäftsprozesses - und nicht nur des Modells - möglich.

Verwandte Muster: Ein Spezialfall dieses Musters ist Muster 15, bei dem vor dem blockierenden AND-Join lediglich Startereignisse liegen. Es ist ein Beispiel dafür, dass Deadlocks unter Umständen vom Modellierer durchaus bewusst in das Modell eingebaut werden.

9.14 AND-Join nach Startereignis könnte blockieren

Muster 14 (Warnung)	Es existiert ein AND-Split a_1, ein AND-Join a_2, ein Startereignis E sowie die Kanten (a_1, a_2) und (E, a_2).
	deutet auf einen beabsichtigten Deadlock am AND-Join a_2 hin, ggf. alternative Modellierung prüfen

Beispiel:

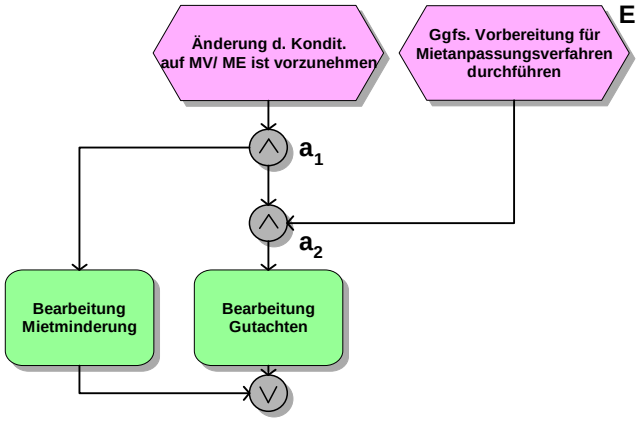


Abbildung 9.20: AND-Join nach „Kann“-Startereignis

Im Modell „Mietänderung“ aus dem Zweig „Immobilienmanagement“ des SAP R/3-Referenzmodells findet sich die Konstellation, die vereinfacht in Abb. 9.20 dargestellt ist. Hier wird schon durch die Beschriftung des rechten Startereignisses („gegebenenfalls“) deutlich, wie das Modell verstanden werden soll: Das linke Startereignis tritt immer ein, und die Funktion „Bearbeitung Mietminderung“ ist immer auszuführen. Unter besonderen Umständen (nämlich wenn außerdem das rechte Startereignis ebenfalls eintritt) ist außerdem die Funktion „Bearbeitung Gutachten“ durchzuführen. Ein möglicher Deadlock am AND-Join wird vom Modellierer ausdrücklich beabsichtigt.

Diskussion: Ist eine wohlstrukturierte Modellierung angestrebt (etwa, weil das Modell später in ausführbaren BPEL-Code zu übertragen ist), sollte die in Abb. 9.21 gezeigte Modellierungsalternative in Betracht gezogen werden. Das links gezeigte Modellfragment kann in die rechts dargestellte strukturierte Form überführt werden.

Welche Form als besser verständlich betrachtet wird, hängt von den Erfahrungen und Gewohnheiten der Leser des Modells ab.

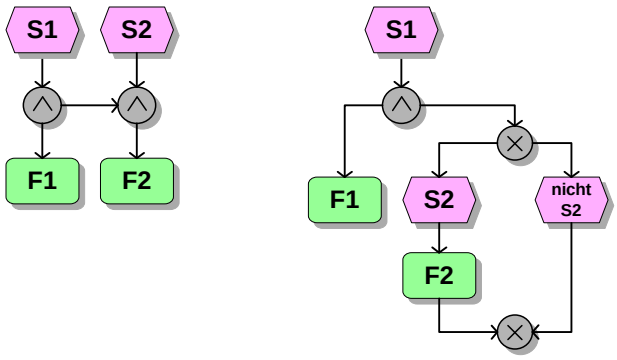
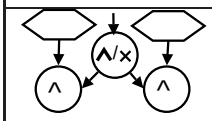


Abbildung 9.21: AND-Join nach „Kann“-Startereignis

Verwandte Muster: Ein wichtiger Spezialfall wird in Muster 15 (Seite 140) diskutiert.

9.15 AND-Join nach Startereignis soll blockieren

Muster 15 (Warnung)	Es gibt einen Split s mit $type(s) \in \{XOR, AND\}$, zwei AND-Joins a_1 und a_2, zwei Startereignisse E_1 und E_2 sowie die Kanten (s, a_1), (s, a_2), (E_1, a_1) und (E_2, a_2)
	deutet auf einen beabsichtigten Deadlock hin, ggf. alternative Modellierung prüfen

Beispiele:

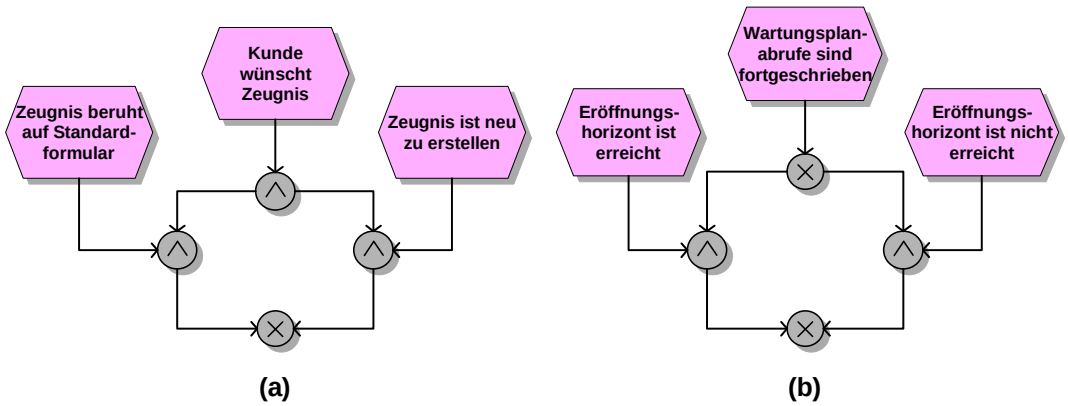


Abbildung 9.22: Von den beiden Startereignissen rechts und links im Modell tritt stets genau eines ein.

Das Modell in Abb. 9.22 (a) zeigt einen Ausschnitt aus dem bereits früher erwähnten Modell „Qualitätsmanagement im Vertrieb - Zeugniserstellung“ des SAP R/3-Referenzmodells.

In Abb. 9.22 (b) ist ein Ausschnitt aus dem Modell „Qualitätsmanagement - Prüfmittelverwaltung - Wartungsplanung“ gezeigt.

Diskussion: Formal gesehen, kommt es bei diesem Muster stets zu einem Synchronisationsfehler am XOR-Join (im Falle von Abb. 9.23 (a)) bzw. zu einem Deadlock an einem der beiden AND-Joins. Trotzdem wird dieses Muster häufig in Modellen gefunden und offenbar von Praktikern durchaus auch verstanden: Synchronisationsfehler bzw. Deadlock sind ausdrücklich beabsichtigt. Es soll damit ausgedrückt werden, dass sich zwei der Startereignisse ausschließen (in Abb. 9.23(a) und (b) sind das die Ereignisse E1 und E3).

Die Verwendung der inhaltsgleichen Modellierungsalternative von Abb. 9.23 (c) macht diesen Zusammenhang besser deutlich, ist leichter zu überblicken und vermeidet einen Synchronisationsfehler bzw. Deadlock.

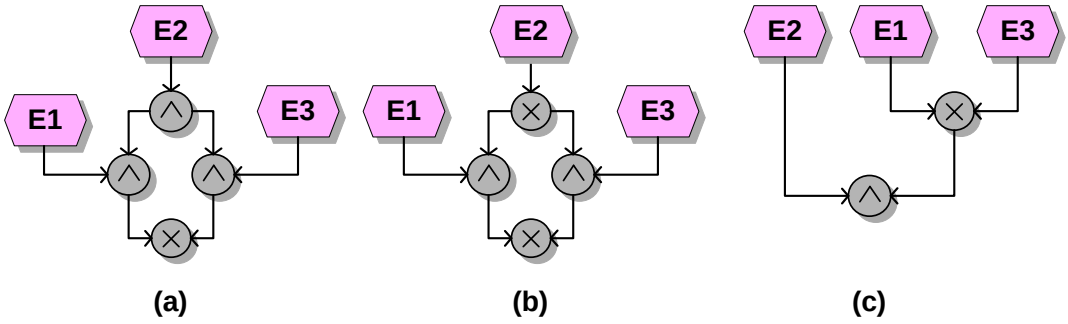
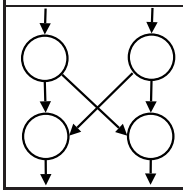


Abbildung 9.23: Modellierungsalternative (c) ist zu bevorzugen.

Verwandte Muster: Das Muster ist ein Spezialfall von Muster 13 (falls der Split vom Typ XOR ist) bzw. von Muster 14 (falls der Split vom Typ AND ist).

9.16 Paar von Kanten liegt im Einflussbereich von Splits verschiedenen Typs

Muster 16 (Warnung)	Für zwei Joins j_1 und j_2 und zwei Splits s_1 und s_2 gilt: Es gibt schnittfreie Pfade von s_1 nach j_1 und j_2 sowie schnittfreie Pfade von s_2 nach j_1 und j_2.
	deutet in jedem Falle auf ein Verständnisproblem hin, unter Umständen auch auf Modellierungsfehler

Beispiele:

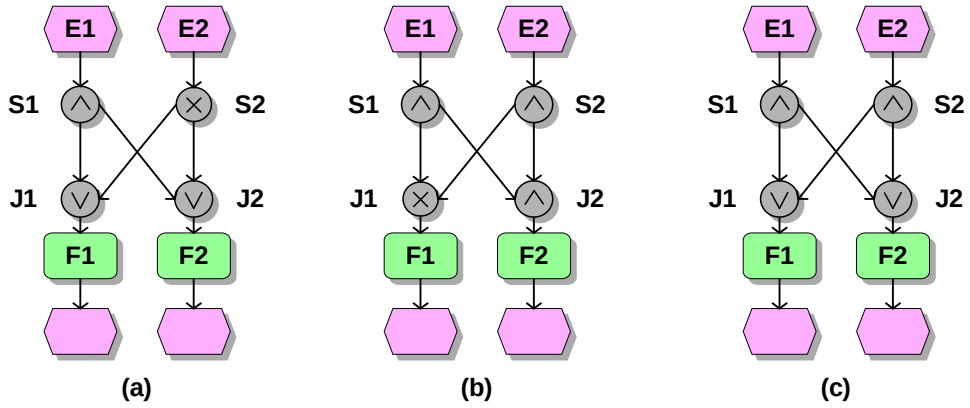


Abbildung 9.24: Verschiedene Ausprägungen von Muster 16

Diskussion: In einer gut lesbaren EPK sollte für zwei Knoten (z.B. Funktionen) k_1 und k_2 leicht erkennbar sein, ob sie in einer der folgenden Beziehungen zueinander stehen:

1. k_1 und k_2 können zeitgleich ausgeführt werden.
2. k_1 und k_2 können nicht zeitgleich ausgeführt werden.
3. k_1 und k_2 schließen einander aus, d.h. höchstens einer der Knoten k_1 und k_2 wird zu einem gegebenen Zeitpunkt ausgeführt.

In der Regel genügt ein Blick auf die EPK-Struktur, um zu entscheiden, welcher dieser Fälle gegeben ist. Der erste Fall tritt u.a. dann ein, wenn k_1 und k_2 auf einen

AND- oder OR-Split folgen; der letzte Fall tritt ein, wenn k_1 und k_2 auf einen XOR-Split folgen.

Schwierig wird es dagegen, wenn k_1 und k_2 im „Einflussbereich“ mehrerer Konnektoren stehen.

Das kann zu verschiedenen Problemen führen, wie in Abb. 9.24 gezeigt wird:

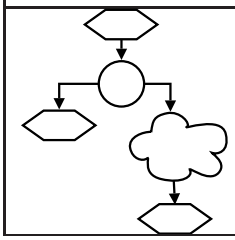
In Abb. 9.24(a) gibt es korrekte Abläufe, wenn der Startzustand entweder $\{E1\}$ oder $\{E2\}$ ist. Die Konstruktion in diesem Modell kann jedoch zu falschen Schlussfolgerungen darüber verleiten, ob die Funktionen F1 und F2 zeitgleich ausgeführt werden dürfen.

Einen echten Fehler gibt es dagegen in Abb. 9.24(b). In diesem Modell gibt es keinen Startzustand, der zu einem Ablauf ohne Kontrollflussfehler führt. Damit nämlich an J1 kein Synchronisationsfehler auftritt, dürfen nicht E1 und E2 zugleich in einem Startzustand enthalten sein. Dann aber kommt es immer zu einem Deadlock an J2.

Abb. 9.24(c) schließlich stellt einen weniger schwerwiegenden Fall dar, da jeder der möglichen Startzustände $\{E1\}$, $\{E2\}$ und $\{E1, E2\}$ zu einem fehlerfreien Ablauf führt. Jedoch empfiehlt sich hier eine deutlich einfache Modellierung: E1 und E2 gefolgt von einem OR-Join und einem AND-Split, der zu F1 und F2 führt.

Selbst wenn das Modell nicht wie im Falle von Abb. 9.24(b) fehlerhaft ist, ist das gezeigte Muster zu vermeiden, da Modelle mit diesem Muster schwer verständlich werden.

9.17 Ablauf ohne Funktionen

Muster 17 (Warnung)	Es gibt einen Pfad von einem Startereignis zu einem Endereignis, der keine Funktionen enthält. (In der Abbildung steht die „Wolke“ für ein beliebiges Teilmodell, das auch Funktionen enthalten kann. In dem Pfad ohne Wolkensymbol gibt es keine Funktionen.)
	sollte auf mögliche inhaltliche Fehler kontrolliert werden

Beispiel:

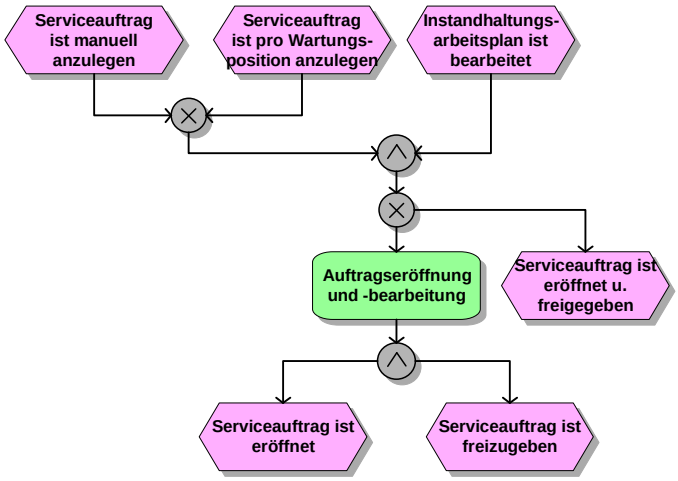


Abbildung 9.25: Endereignis kann ohne Ausführen einer Funktion erreicht werden

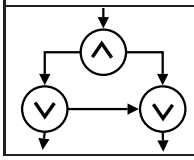
Im Modell „Qualitätsmanagement - Serviceauftrag“ des SAP R/3-Referenzmodells, das in Abb. 9.25 ausschnittsweise dargestellt ist, kann das rechte Endereignis erreicht werden, ohne dass jemals eine Funktion ausgeführt wurde. Vermutlich entspricht das nicht der betrieblichen Realität.

Diskussion: Gibt es einen Pfad von einem Startereignis zu einem Endereignis, der keine Funktionen enthält, so ist dies bei strenger Anwendung der Syntaxanforderungen von Def. 1 ein syntaktischer Fehler im Modell: Ereignisse und Funktionen wechseln sich dann nämlich nicht ab.

Wie bereits in Abschnitt 2.2 diskutiert wurde, wird jedoch auf diese Forderung in der Praxis oft verzichtet, ohne dass das zu Problemen führen muss.

Verzichtet man auf die Forderung, dass sich Ereignisse und Funktion entlang eines Pfades abwechseln müssen, so ist dennoch ein Pfad von einem Startereignis zu einem Endereignis ohne dazwischenliegende Funktionen ein Indiz für einen möglichen inhaltlichen Modellierungsfehler. Das entsprechende Modell sollte manuell überprüft werden.

9.18 Transfer, der mit XOR statt OR eingeleitet werden sollte

Muster 18 (Warnung)	Sei a ein AND-Split, o ein OR-Split sowie j ein Join. Es existieren Pfade P_1 von a nach o und P_2 von a nach j sowie eine Kante (o, j). Die durch $P_1 \backslash \{a, o\}$ und $P_2 \backslash \{a, j\}$ induzierten Untergraphen der EPK sind SESE-Regionen.
	Der OR-Split sollte durch einen XOR-Split ersetzt werden.

Beispiel:

Beide Modelle in Abb. 9.26 stellen die gleiche Situation dar: Die Funktionen A und B werden parallel ausgeführt. Endereignis E1 kann, Endereignis E2 muss stets erreicht werden.

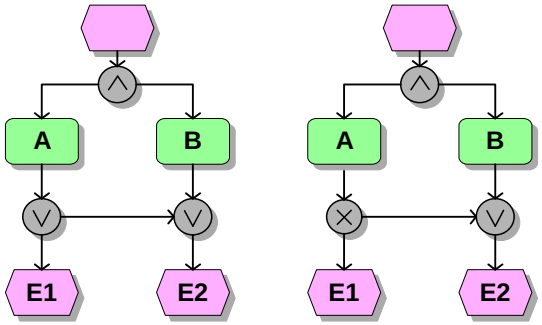
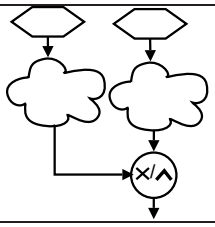


Abbildung 9.26: Der OR-Split kann durch einen XOR-Split ersetzt werden.

Diskussion: Die Modellierung mit einem XOR-Split ist zu bevorzugen, da hier unmittelbar deutlich wird, dass es genau zwei Möglichkeiten der Fortsetzung gibt. Wird der OR-Split verwendet, so muss der Leser als dritte Möglichkeit in Betracht ziehen, dass beide Ausgänge des OR-Splits durchlaufen werden. Daraus ergibt sich jedoch kein Ablauf, der nicht schon durch den XOR-Split modelliert wird.

9.19 Erlaubte Startzustände schwer erkennbar

Muster 19 (Warnung)	Zu einem XOR- oder AND-Join führen Pfade von zwei verschiedenen Startereignissen. Mindestens einer dieser Pfade enthält Ereignisse oder Splits (im Bild durch das Wolkensymbol dargstellt).
	Da möglicherweise schwer zu erkennen ist, ob es an dem Join zu einem Deadlock bzw. Synchronisationsfehler kommen kann, sollte eine alternative Modellierung in Betracht gezogen werden.

Beispiel:

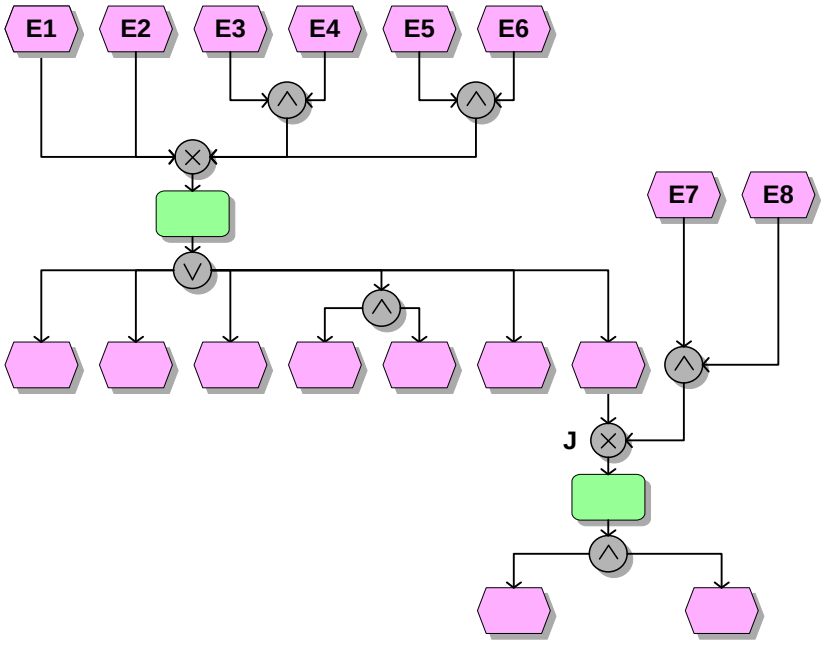


Abbildung 9.27: Erlaubte Startzustände sind hier nur mühsam zu erkennen

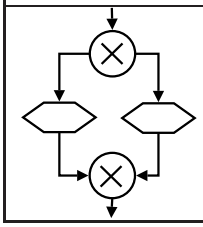
In Abb. 9.27 sind die Ereignisse und Funktionen des Modells „Instandhaltung - geplante Instandhaltungsabwicklung - Meldung“ aus dem SAP R/3-Referenzmodell (ohne Beschriftung der Ereignisse und Funktionen) gezeigt. Das Modell ist sound, jedoch ist es nur mühsam zu erkennen, welche Startzustände erlaubt sind (d.h. nicht zu einem Synchronisationsfehler an dem mit J beschrifteten XOR-Join führen).

Diskussion: Eine EPK mit diesem Muster ist nicht wohlstrukturiert (vgl. Def. 22 auf Seite 80). Wenn möglich, sollte eine wohlstrukturierte Modellierungsalternative gesucht werden, bei der alle Startereignisse am Anfang des Modells stehen. Das wird jedoch nicht in allen Fällen möglich sein. Schwierig kann eine Ersetzung z.B. dann werden, wenn ein Ereignis als Startereignis dargestellt ist, das erst im Verlaufe eines Geschäftsprozessablaufs eintritt.

In der Praxis wird beispielsweise durch ein solches Ereignis oft das Erreichen eines bestimmten Zeitpunktes beschrieben (siehe etwa Abb. 5.3 auf Seite 58) oder das Eintreffen einer Antwort von einem externen Partner dargestellt. Beim Versuch, das Modell als wohlstrukturierte EPK darzustellen, kann die Information verlorengehen, dass der besagte Zeitpunkt unter Umständen gar nicht erreicht wird bzw. die erwartete Antwort nicht immer eintreffen muss.

Verwandte Muster: Muster 2 auf Seite 114 ist ein Spezialfall dieses Musters. Bei diesem Spezialfall kommt es stets zu einem Kontrollflussfehler.

9.20 nur Ereignisse im XOR-Kontrollblock

Muster 20 (Warnung)	Zwischen einem XOR-Split und einem XOR-Join stehen ausschließlich Ereignisse.
	Eine Rückfrage an den Modellierer, ob der weitere Prozessablauf tatsächlich davon unbeeinflusst bleibt, welches der Ereignisse eintritt, erscheint sinnvoll.

Beispiel:

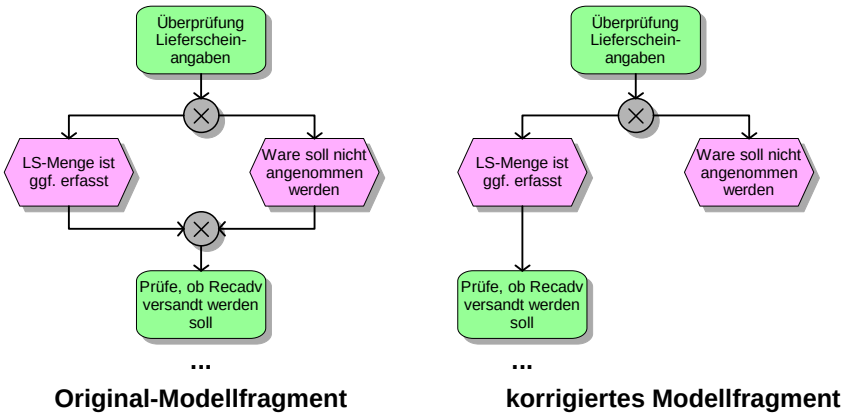


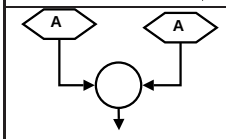
Abbildung 9.28: Welches Ereignis eintritt, hat keinen Einfluss auf den weiteren Verlauf.

Abb. 9.28 zeigt einen Ausschnitt aus dem Modell „Wareneingang Lager“ aus [12, Seite 345]. In der linken Abbildung wurde so modelliert, dass der Prozess stets auf die gleiche Art und Weise weitergeführt wird, auch wenn das Ereignis „Ware soll nicht angenommen werden“ eintritt. In der weiteren Folge der Prozesschritte findet sich dann tatsächlich stets die Funktion „Einlagerung der Ware“. Korrekt wäre vermutlich eine Modellierung wie im rechten Modell gewesen, bei dem „Ware soll nicht angenommen werden“ ein Endereignis ist.

Diskussion: Dieses Muster weist keinesfalls immer auf einen Fehler hin. Oft ist es zu Dokumentationszwecken tatsächlich angebracht, die möglichen Ereignisse zu benennen. Sinnvoll scheint jedoch in jedem Falle eine Rückfrage an den Modellierer,

ob der weitere Prozessablauf tatsächlich davon unbeeinflusst bleibt, welches der Ereignisse eintritt.

9.21 Logisch identische Ereignisse vor/nach einem Konnektor

Muster 21 (Fehler oder Warnung)	Ein Konnektor hat zwei logisch identische Ereignisse als Vorgänger oder Nachfolger.
	inhaltlicher Fehler (wenn zwei logisch identische Ereignisse auf einen XOR-Split folgen) oder Hinweis auf die Möglichkeit, das Modell durch Weglassen redundanter Elemente zu verkleinern

Beispiele:

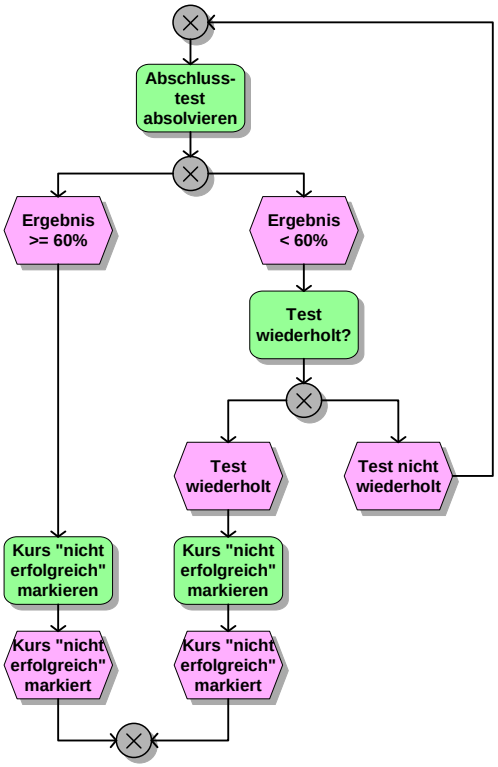


Abbildung 9.29: inhaltlich fehlerhaftes Modell

Diskussion: Stehen die logisch identischen Ereignisse nach einem XOR-Split, ist dies offenbar stets ein inhaltlicher Fehler, da der XOR-Split ja gerade aussagt, dass nur eines der Ereignisse eintreten kann (und das andere nicht).

In allen anderen Fällen sind verschiedene Situationen möglich: Abb. 9.29 ist ein

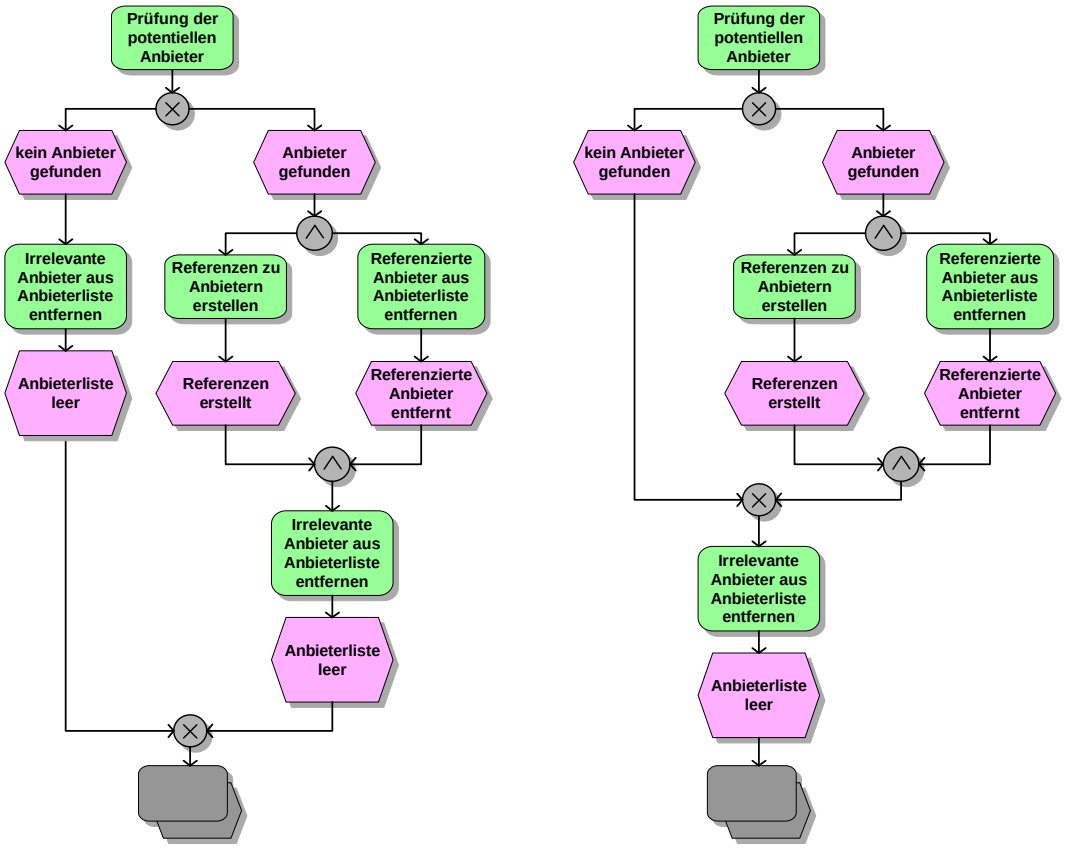


Abbildung 9.30: links: Originalmodell, rechts: vereinfachtes Modell

Modellausschnitt aus [58]. Hier weist das doppelte Vorkommen des Ereignisses auf einen Modellfehler hin: Vermutlich hätte im linken Zweig die Situation „Kurs erfolgreich“ modelliert werden sollen.

In den meisten Fällen jedoch dürfte das doppelte Vorkommen eines Ereignisses lediglich ein Indiz dafür sein, dass die EPK vereinfacht werden kann. Ein Beispiel zeigt Abb. 9.30, entnommen einer Diplomarbeit [187].

Zu „Fehlalarmen“ durch dieses Muster kommt es, wenn Standard-Texte wie „erfolgreich durchgeführt“ oder „OK“ für Ereignisbeschriftungen genutzt werden. Da hier der Bezug darauf fehlt, *was* erfolgreich durchgeführt wurde, sind die Ereignisse trotz gleichlautender Beschriftung erkennbar verschieden.

Ein Beispiel für einen solchen Fall zeigt das Modellfragment in Abb. 9.31 (entnommen aus [90]):

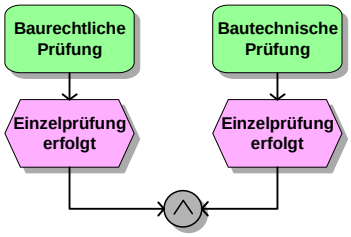
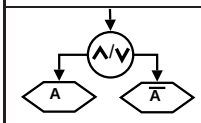


Abbildung 9.31: Fehlalarm bei Muster 21

Ebenso sind zwei Ereignisse trotz gleichlautender Beschriftung erkennbar verschieden, wenn ihnen in einer erweiterten EPK verschiedene Organisationseinheiten zugeordnet sind. Ein Ereignis „Prüfung erfolgreich“, ausgeführt von der Buchhaltung, ist ein anderes Ereignis als „Prüfung erfolgreich“, ausgeführt von der Rechtsabteilung.

9.22 Zwei sich ausschließende Ereignisse durch AND-bzw. OR-Konnektor vereint

Muster 22 (Warnung)	Ein Konnektor vom Typ OR oder AND hat zwei Ereignisse, die sich logisch widersprechen, als Vorgänger oder Nachfolger.
	inhaltlicher Fehler Möglicherweise sollte der Konnektor durch einen XOR-Konnektor ersetzt werden.

Beispiel:

In dem Modellfragment aus Abb. 9.32 (das einer Diplomarbeit entnommen wurde), wurde statt des die Situation korrekt beschreibenden XOR-Splits ein OR-Split verwendet. Bei unerfahrenen Modellierern ist eine solche Verwechslung zwischen OR und XOR ein häufiger Fehler.

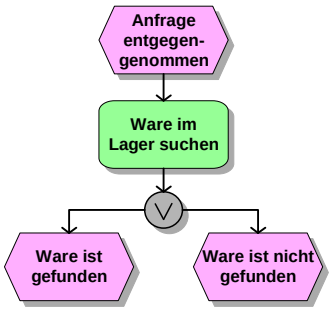


Abbildung 9.32:

Diskussion:

Leitet ein OR- oder AND-Split mehrere Kontrollflusszweige ein, so können diese im Modell parallel durchlaufen werden. Insbesondere heißt das, dass mehrere Ereignisse, die auf einen AND- oder OR-Join folgen, gemeinsam eintreten dürfen.

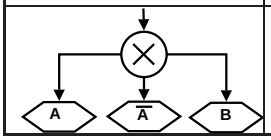
Schließen sich nun zwei auf einen OR- oder AND-Split folgende Ereignisse logisch aus (z.B. „Genehmigung erteilt“ / „Genehmigung verwehrt“), so ist davon auszugehen, dass ein Fehler im Modell vorliegt. Dieser besteht meist darin, dass statt eines XOR-Splits fälschlicherweise ein OR- oder AND-Split verwendet wurde.

Eine analoge Aussage gilt für den Fall, dass einander widersprechende Ereignisse direkt vor einem OR- bzw. AND-Join stehen.

Es ist anzumerken, dass sich die beiden Ereignisse auch logisch ausschließen können, wenn nicht eines die Negation des anderen ist. Ein Vorliegen des Musters soll auch erkannt werden, wenn z.B. eine Kombination von Ereignissen wie „Der Wert x hat zugenommen“ / „Der Wert x hat abgenommen“ gefunden wird, auch wenn außerdem noch der dritte Fall „Der Wert x ist konstant geblieben“ möglich wäre.

Verwandte Muster: Einander widersprechende Ereignisse vor bzw. nach einem XOR-Konnektor werden von Muster 23 behandelt.

9.23 Ereignis, dessen Negation und weiteres Ereignis am XOR-Konnektor vereint

Muster 23 (Warnung)	Ein Konnektor vom Typ XOR hat die Ereignisse A, $\neg A$ sowie ein weiteres Ereignis B als Vorgänger oder Nachfolger.
	Da stets entweder A oder $\neg A$ eintritt, ist die Berechtigung eines weiteren Ereignisses B fraglich (*Tertium non datur*).

Beispiel:

Das Modellfragment in Abb. 9.33 ist dem Zeitschriftenartikel [56] entnommen. In einem solchen Modell leidet zumindest die Verständlichkeit[4]: Da offenbar stets genau eines der beiden linken Ereignisse eintritt, ist die Berechtigung der beiden anderen Ereignisse unklar.

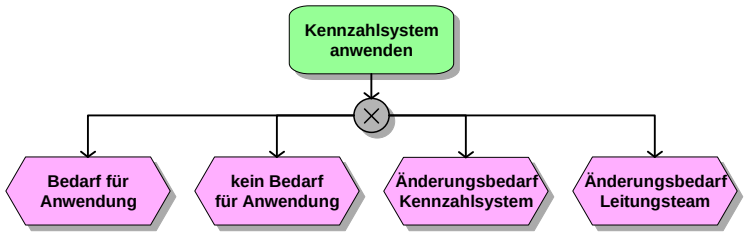


Abbildung 9.33: Es tritt stets eines der beiden linken Ereignisse ein.

Einen möglichen durch dieses Muster erzeugten „Fehlalarm“ zeigt Abb. 9.34. Hier entsteht der Eindruck eines Fehlers dadurch, dass die Aussage „CR ist vorhanden und (un)vollständig“ verkürzt wurden auf „CR ist (un)vollständig“. Für den Leser des Modells dürfte diese verkürzte Schreibweise sogar leichter verständlich sein.

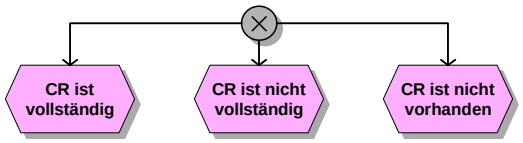


Abbildung 9.34: Fehlalarm bei Muster 23

[4] Den fachlichen Inhalt des Modellausschnitts vermag der Autor nicht zu beurteilen.

Diskussion: In diesem Muster folgen auf einen XOR-Split ein Ereignis A, dessen Negation $\neg A$ und mindestens ein weiteres Ereignis B. Aus den gezeigten Beispielen wird deutlich, dass das Vorkommen dieses Musters ein Hinweis auf mögliche Verständnisprobleme oder Modellfehler sein kann. In manchen Fällen (z.B. in Abb. 9.34) liegt jedoch kein wirkliches Problem vor.

Ein besonderer Fall, in dem kein Problem vorliegt, ist eine Situation, in der A, $\neg A$ und B Entscheidungen der Art „ja“ / „nein“ / „unter Umständen“ beschreiben. Um auch solche Situationen richtig zu behandeln, soll eine Mustersuche in den Fällen, in denen B einschränkende Begriffe wie „zum Teil“, „eventuell“ oder „unter Vorbehalt“ enthält, kein Auftreten des Musters melden.

In den im Rahmen dieser Arbeit untersuchten Praxis-Modellen (vgl. Kapitel 11) wurde beispielsweise für die folgenden Kombinationen von Ereignissen nach einem XOR-Split *kein* Auftreten von Muster 23 gemeldet:

- „Berichte sind in Ordnung“ / „Berichte sind nicht in Ordnung“ / Berichte sind nur teilweise in Ordnung“
- „Bestellstatus zurückgewiesen“ / „Bestellstatus zugestimmt“ / „Bestellstatus teilweise zugestimmt“

Verwandte Muster: Einander widersprechende Ereignisse vor bzw. nach einem AND- oder OR-Konnektor werden von Muster 22 behandelt.

10 Untersuchung von GPM mit Hilfe logischer Programmierung

10.1 Grundlagen

Die Grundidee unseres Ansatzes zur Überprüfung der Qualität von GPM besteht darin, im Modell Muster (wie im vorangegangenen Kapitel beschrieben) zu finden, die auf Verbesserungsmöglichkeiten hindeuten.

Die Erkennung von Modellteilen einer EPK, die möglicherweise verbessert werden sollten, erfordert das Identifizieren von Mustern im Graphen. Zur Erkennung solcher Muster haben sich Sprachen bewährt, die dem Prinzip der logischen Programmierung folgen. Der Vorteil solcher Sprachen besteht darin, dass es ausreicht, die gesuchten Muster zu beschreiben. Das Finden von Musterinstanzen erfolgt dann durch den Interpreter der logischen Programmiersprache. Dieser Interpreter hat zu diesem Zweck insbesondere Backtracking-Algorithmen bereits effizient implementiert. Die Implementierung der Erkennung von Mustern wurde daher in der Sprache Prolog vorgenommen, womit eine verbreitete und bewährte logische Programmiersprache gewählt wurde.

10.2 Übersetzung des Modells in Prolog-Fakten

Liegt eine EPK im Austauschformat EPML vor, so enthält diese EPML-Datei alle Informationen zu den Knoten und Kanten des Graphen, der die EPK darstellt. Der Abschnitt

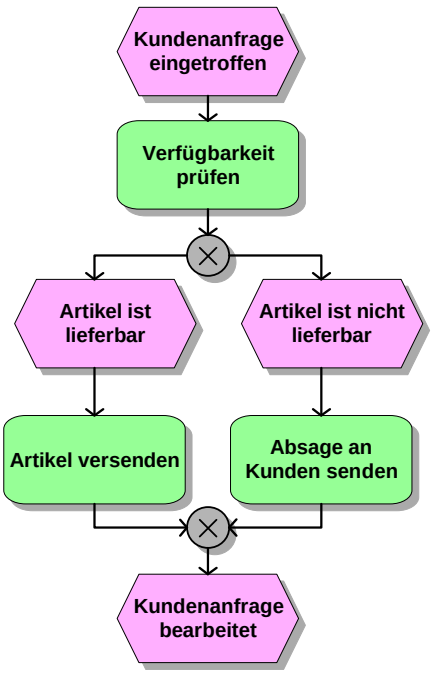
```
<event id="10">
  <name>Kundenanfrage bearbeitet</name>
</event>
<arc id="11">
  <flow source="1" target="2"/>
</arc>
```

beinhaltet drei logische Aussagen:

1. Es gibt im Modell ein Ereignis mit id=10.
2. Das Ereignis mit id=10 hat den Namen „Kundenanfrage bearbeiten“.
3. Es gibt im Modell eine Kante vom Element mit id=1 zum Element mit id=2.

Im Folgenden werden einem EPK-Modell die in diesem Modell enthaltenen logischen Aussagen als Prolog-Fakten gegenübergestellt.

```
event(i_1).
elementname(i_1,'Kundenanfrage eingetroffen').
event(i_5).
elementname(i_5,'Artikel ist lieferbar').
event(i_6).
elementname(i_6,'Artikel nicht lieferbar').
event(i_10).
elementname(i_10,'Kundenanfrage bearbeitet').
function(i_2).
elementname(i_2,'Verfügbarkeit prüfen').
function(i_7).
elementname(i_7,'Artikel versenden').
function(i_8).
elementname(i_8,'Absage an Kunden senden').
arc(i_1,i_2).
arc(i_2,i_4).
arc(i_4,i_5).
arc(i_4,i_6).
arc(i_5,i_7).
arc(i_6,i_8).
arc(i_7,i_9).
arc(i_8,i_9).
arc(i_9,i_10).
xor(i_4).
xor(i_9).
```



Diese Fakten enthalten dieselben Informationen wie das Modell bzw. die zugehörige EPML-Datei. Um die Fakten aus der EPML-Datei zu generieren, kann eine einfache in der Sprache XSLT geschriebene Transformation benutzt werden, deren Quelltext in [63] zu finden ist.

10.3 Grundlegende Prädikate

Ein EPK-Modell, das in eine Prolog-Wissensbasis überführt wurde, beinhaltet lediglich die einstelligen Prädikate `function` (Existenz einer Funktion), `event` (Existenz eines Ereignisses), `xor`, `and` und `or` (Existenz eines Konnektors) sowie das zweistellige

Prädikat `arc` (Existenz einer Kante zwischen zwei Knoten). Weiterhin existiert das zweistellige Prädikat `elementname`, das jeder Funktion und jedem Ereignis ihre Beschriftung im Modell als Zeichenkette zuordnet.

Um diese Wissensbasis effektiv nutzen zu können, werden ihr nun zunächst Regeln hinzugefügt, die das in Abschnitt 2.2 eingeführte Vokabular beschreiben. Das Prolog-System wird also mit einem „Grundwissen“ über ereignisgesteuerte Prozessketten in Form logischer Schlussregeln ausgestattet.

Dies soll an einigen Beispielregeln gezeigt werden:

```
connector(X) :- clause(and(X),true) ; clause(or(X),true);
clause(xor(X),true).
```

besagt, dass wir unter einem Konnektor einen AND-, XOR- oder OR-Konnektor verstehen. Das Prädikat „X ist Konnektor“ gilt also, wenn im betrachteten Modell für `X` eines der Prädikate `and(X)`, `xor(X)` oder `or(X)` gilt.

```
no_outgoing_arcs(X) :- not(arc(X,_)).
```

besagt, dass für ein Element das Prädikat „hat keine ausgehenden Kanten“ erfüllt ist, wenn es keine Kante (d.h. kein arc-Element in der EPML-Datei) gibt, die von diesem Element ausgeht. Mit Hilfe dieses Prädikats kann dann ein Endereignis definiert werden als ein Ereignis, von dem keine Kanten ausgehen:

```
endevent(X) :- event(X),no_outgoing_arcs(X).
```

Auf diese Weise werden die folgenden Prädikate definiert, die die grundlegenden in Abschnitt 2.2 eingeführten Begriffe beschreiben.

Definition 35 (Grundlegende Prädikate)
Sei $\Lambda = (K, A)$ ein EPK-Modell wie in Def. 1 definiert. Es sei wieder F die Menge der Funktionen, E die Menge der Ereignisse, C_{xor}, C_{or} und C_{and} die Mengen der XOR-, OR- bzw. AND-Konnektoren sowie $C = C_{xor} \cup C_{or} \cup C_{and}$. Für einen Knoten $k \in C$ sind die einstelligen Prädikate definiert:
`orconnector(k)` genau dann, wenn $k \in C_{or}$
`andconnector(k)` genau dann, wenn $k \in C_{and}$
`xorconnector(k)` genau dann, wenn $k \in C_{xor}$
`connector(k)` genau dann, wenn $k \in C$
`split(k)` genau dann, wenn $k \in C$ und card $\triangleright k = 1$
`join(k)` genau dann, wenn $k \in C$ und card $k\triangleleft = 1$.

Daraus lassen sich leicht sechs weitere Prädikate ableiten:
`orsplit(k)` $\equiv_{def}$ `orconnector(k)` $\wedge$ `split(k)`
`andsplit(k)` $\equiv_{def}$ `andconnector(k)` $\wedge$ `split(k)`
`xorsplit(k)` $\equiv_{def}$ `xorconnector(k)` $\wedge$ `split(k)`
`orjoin(k)` $\equiv_{def}$ `orconnector(k)` $\wedge$ `join(k)`
`andjoin(k)` $\equiv_{def}$ `andconnector(k)` $\wedge$ `join(k)`
`xorjoin(k)` $\equiv_{def}$ `xorconnector(k)` $\wedge$ `join(k)`

Für Ereignisse, also Knoten $e \in E$, wird definiert:
`startevent(e)` genau dann, wenn event(e) gilt und es kein $x \in K$ gibt, für das $(x, e) \in A$ sowie
`endevent(e)` genau dann, wenn event(e) gilt und es kein $x \in K$ gibt, für das $(e, x) \in A$.

Beim überwiegende Teil der Muster ist es notwendig, Pfade zwischen Knoten im Modell zu finden. Auch die Existenz zweier schnittfreier Pfade zwischen zwei Knoten wird oft gefordert, etwa immer dann, wenn für zwei Knoten x_1 und x_2 die Relation $match(x_1, x_2)$ gelten soll.

Dafür wurden zwei Prolog-Regeln definiert:
`path(A,B,Path)` gilt gilt genau dann, wenn ein Pfad vom Knoten A zum Knoten B durch die Liste in der Variablen $Path$ beschrieben ist.
`secondpath(A,B,Path,Exclude)` gilt genau dann, wenn ein Pfad vom Knoten A zum Knoten B durch die Liste in der Variablen $Path$ beschrieben ist und in $Path$ kein Element von $Exclude$ vorkommt.

Die match-Relation $match(A, B)$, welche die Existenz zweier schnittfreier Pfade fordert, lässt sich dann leicht dadurch ausdrücken, dass `path(A,B,Path1)` und `secondpath(A,B,Path2,Path1)` gilt.

Mit Hilfe der bisher genannten Prädikate sowie den in bestehenden Prolog-Bibliotheken bereits zur Verfügung stehenden Prädikaten (etwa zum Testen, ob ein Element in einer Menge enthalten ist) ist es leicht, Sachverhalte wie „Jeder Pfad von einem Startereignis zum Knoten k muss den Knoten p enthalten.“ zu formulieren.

10.4 Einfache syntaktische Prüfungen

Eine Überprüfung der syntaktischen Korrektheit eines Modells ist leicht möglich, indem die EPK-Syntaxanforderungen nach Def. 1 in Prolog-Regeln übersetzt werden. Bei der Überprüfung der Korrektheit wird dann an die Wissensbasis die Anfrage

gestellt, ob im Modell Elemente gefunden werden, durch die die entsprechende Syntaxanforderung verletzt wird.

Um etwa zu überprüfen, ob der Graph antisymmetrisch ist (Forderung 2 in Def. 1), wird zunächst nach zwei Kanten der Form $(a, b)/(b, a)$ gesucht. Wird kein solches Gegenbeispiel gefunden, ist die betreffende Eigenschaft für die untersuchte EPK erfüllt. Wird ein Gegenbeispiel gefunden, so erhält der Modellierer dadurch die Information, an welcher Stelle sein Modell syntaktisch falsch ist. Die Prolog-Regeln, mit denen sämtliche Forderungen an die Syntax von EPKs nach Def. 1 getestet werden können, sind in [63] veröffentlicht.

10.5 Suche nach Mustern semantischer und pragmatischer Probleme

Wie in Abschnitt 10.3 beschrieben, wird im Prolog-Programm zunächst ein „Grundwortschatz", der im Wesentlichen die Definitionen aus Abschnitt 8.2.1 abbildet, durch Prädikate definiert.

Ist dieser „Grundwortschatz" durch Prolog-Regeln ausgedrückt, ist es leicht, die Muster unseres Musterkatalogs zu beschreiben.

So lässt sich Muster 4 von Seite 117 (Match zwischen AND- bzw. OR-Split und XOR-Join) beschreiben durch das Prädikat:

```
muster4(S,J) :- (andsplit(S);orsplit(S)),xorjoin(J),match(S,J).
```

Für jedes Muster unseres Musterkatalogs wurde ein solches Prädikat formuliert. Die Suche nach möglichen Problemen in einem Modell entspricht dann der Anfrage an den Prolog-Interpreter, Instanzen zu finden, für die das jeweilige Prädikat erfüllt ist. Als Ergebnis dieser Anfrage liefert das Prolog-System dann die Information zurück, *wo* im Modell das Muster auftritt. Bei der obigen Anfrage nach Vorkommen von Muster 4 gibt das Prolog-System alle Kombinationen von Split S und Join J aus, für die `muster4(S,J)` gilt.

10.6 Behandlung der Modellbeschriftung

Zum Auffinden der Muster 21 bis 23 (siehe Seite 151) ist eine Untersuchung der Beschriftungen von Ereignissen nötig. Diese soll erkennen, ob zwei Ereignisse logisch identisch sind bzw. sich logisch widersprechen. Um dies zu erreichen, werden im ersten Schritt die Beschriftungen der Modellelemente in eine Normalform gebracht. In dieser sind beispielsweise Stoppwörter wie *der*, *die*, *das*, *ein*, *eine* u.ä. entfernt. Dadurch wird

erreicht, dass etwa die Ereignisbeschriftungen „Der Antrag wurde geprüft“ und „Es wurde der Antrag geprüft“ beide in dieselbe Normalform „Antrag geprüft“ überführt werden.

Anschließend werden Synonyme und Antonyme behandelt. In dem für diese Arbeit entwickelten Prototypen für die musterbasierte Problemsuche mit Prolog wurden 183 Synonyme und 73 Antonyme zu Begriffen aufgenommen, die sich sehr häufig in Modellen betrieblicher Abläufe finden. Diese Begriffe beziehen sich vor allem auf den positiven bzw. negativen Ausgang von Maßnahmen, Entscheidungen und Prüfungen. Dies sind gerade die Ereignisse, die in EPKs häufig zu finden sind, um die Nachbedingung einer Funktion zu modellieren.

Durch die Analyse kann z.B. festgestellt werden, dass die Ereignisbeschriftungen „Der Antrag wurde genehmigt“ und „Der Antrag ist bewilligt“ den gleichen Sachverhalt beschreiben oder dass „Anlage ist in Ordnung“ die Negation von „Anlage defekt“ darstellt. Im Prolog-Programm ist die Behandlung von Synonymen und Antonymen dadurch realisiert, dass Wörter, für die Synonyme bzw. Antonyme behandelt werden, durch eine spezielle Zeichenkette ersetzt werden. So werden die beiden genannten Zeichenketten „Der Antrag wurde genehmigt“ und „Der Antrag ist bewilligt“ beide in die Normalform „`Antrag nf_erlaubnis_erteilt`“ überführt. Im Falle von Antonymen wird zusätzlich ein Flag gesetzt, das anzeigt, dass ein Ereignis die Negation des anderen ist.

Schließlich werden Beschriftungen gefunden, die sich lediglich durch negierende Zeichenketten wie „nicht“, „kein(e)“ / oder „Un-“ unterscheiden. Auch diese drücken offenbar das Gegenteil voneinander aus („Genehmigung wurde erteilt“ / „Genehmigung wurde nicht erteilt“ bzw. „Daten vollständig“ / „Daten unvollständig“).

Auf diese Weise werden Ereignisse identifiziert, von denen eines die Negation des anderen ist. Für das Muster 22 war jedoch schon das Vorliegen von zwei Ereignissen, die einander ausschließen, ausreichend. Neben Beschriftungen, die sich durch negierende Ausdrücke unterscheiden, werden daher auch Gleichheiten, Ungleichheiten und Informationen zu Größenveränderungen untersucht. Diese tauchen in GPM recht häufig auf.

Damit können auch Paare von Ereignissen E_1 und E_2 gefunden werden, für die zwar nicht $E_1 = \neg E_2$, jedoch $E_1 \wedge \neg E_2 = falsch$ gilt. Ein Beispiel hierfür ist die Kombination E_1=„Der Wert hat sich erhöht“, E_2=„Der Wert ist konstant geblieben“ (auch wenn als dritte Möglichkeit der Fall „Der Wert ist gesunken“ eintreten kann).

10.7 Maßnahmen zur Laufzeitoptimierung

Die Laufzeit der Mustersuche mit Prolog steigt mit der Zahl der im Modell vorhandenen Knoten bzw. Pfade. Somit kann durch die Anwendung von Reduktionsregeln die Laufzeit entscheidend verbessert werden. Wie in Abschnitt 4.2.2 beschrieben, kommen hierfür Reduktionsregeln in Betracht, die eigenschaftserhaltend sind. Das reduzierte Modell muss also dieselben Probleme aufweisen wie das Ausgangsmodell.

Im Prolog-Programm wurden alle in Abschnitt 7.1 dargestellten Reduktionsregeln implementiert. Außerdem werden, wie bei Mendling [113], mehrere Startereignisse, die denselben Join als gemeinsamen Nachfolgeknoten haben, zu einem einzelnen Startereignis zusammengefasst. Analog werden mehrere Endereignisse, die alle Nachfolgeknoten desselben Splits sind, zu einem einzelnen Endereignis zusammengefasst. Entfernt werden weiterhin AND-Splits, von denen aus eine Kante zu einem Endereignis führt.

Schließlich werden SESE-Regionen komplett entfernt, in denen sämtliche Konnektoren vom Typ XOR sind. Solche Modellteile können keine Kontrollflussfehler enthalten. Nachdem nämlich die Eingangskante einer solchen SESE-Region markiert wurde, gibt es stets genau eine markierte Kante in der SESE-Region, bis schließlich die Ausgangskante markiert wird. Dies folgt unmittelbar aus der Definition der in Abschnitt 2.5.1 beschriebenen Übergangsrelationen.

Es muss angemerkt werden, dass die aufgeführten Reduktionsregeln nur in Bezug auf Kontrollflussfehler eigenschaftserhaltend sind. Die Muster 17 und 20, bei deren Vorliegen die Soundness-Eigenschaft nicht verletzt ist, müssen *vor* der Anwendung der Reduktionsregeln gesucht werden. Gleiches gilt für die Muster 21 bis 23, bei denen mögliche Modellierungsprobleme auf Grund der Beschriftung von Modellelementen gefunden werden.

Betrachtet man die Definitionen der Muster in dem Musterkatalog von Kapitel 9, stellt man fest, dass ein großer Teil der Muster die Berechnung der match-Relation erfordert. Eine Maßnahme zur Verbesserung der Laufzeit ist es daher, sämtliche match-Beziehungen für ein Modell bei Programmbeginn (nach Anwendung der Reduktionsregeln) einmalig zu berechnen. Jedes erkannte Auftreten eines Paares (s, j) mit $match(s, j)$ wird dann dauerhaft als Prädikat zur Prolog-Regelmenge hinzugefügt. Durch dieses Cachen der Ergebnisse entfällt das wiederholte Suchen nach Split s und Join j mit $match(s, j)$, was jeweils ein laufzeitintensives Suchen nach Pfaden im EPK-Graphen erfordern würde.

Verzichtet wurde auf eine Aufteilung des Modells in einzelne SESE-Regionen, wie

bei [182, 181] vorgeschlagen. Der Hauptgrund dafür ist, dass auch ohne eine solche Aufteilung bereits eine zufriedenstellende Geschwindigkeit erreicht wurde. Zudem ist zu erwarten, dass gerade bei nicht wohlstrukturierten EPKs mit mehreren Start- und Endereignissen eine Aufteilung in SESE-Regionen weniger effektiv ist als bei den in [182, 181] betrachteten Workflow-Netzen mit genau einem Start- und genau einem Endknoten. Alle Startereignisse müssten nämlich in einer gemeinsamen SESE-Region liegen, was das Bilden kleiner SESE-Regionen behindert. Gleiches gilt für alle Endereignisse.

Die Aufteilung des zu untersuchenden Modells in SESE-Regionen bietet ebenso wie die Anwendung zusätzlicher Reduktionsregeln noch Potential für weitere Verbesserungen der Laufzeit des Prolog-Programms.

Soll Prolog während des Modellierens im Hintergrund laufen und Fehler sofort melden, besteht ein weiterer Ansatz zur Verbesserung der Laufzeit darin, die am Modell vorgenommenen Änderungen zu überwachen. Es ist dann möglich, nur diejenigen Prolog-Regeln zu prüfen, die von der gerade vorgenommenen Änderung betroffen sein können [16, 15]. Auch diese Möglichkeit ist im gegenwärtig fertiggestellten Prototypen noch nicht implementiert.

11 Validierung

11.1 Beschreibung des Validierungsansatzes

Um die Tauglichkeit des in dieser Arbeit vorgestellten Ansatzes zu bewerten, wurde eine große Zahl von EPK-Modellen mit den im vorangegangenen Kapitel vorgestellten Methoden untersucht. Dieses Kapitel beschreibt die Ergebnisse dieser Untersuchung und vergleicht sie mit den Ergebnissen, die andere Validierungswerkzeuge für denselben Satz von Modellen liefern.

Die Untersuchung erfolgte an einem Satz von EPK-Modellen. Diese wurden aus den folgenden Quellen zusammengestellt:

- 531 Modelle aus dem SAP R/3-Referenzmodell: Von dem 604 Modelle umfassenden SAP-Referenzmodell wurden 4 syntaktisch unkorrekte Modelle ausgeschlossen, bei denen eine Funktion oder ein Ereignis sowohl mehr als eine eingehende als auch mehr als eine ausgehende Kante hat. Von den verbleibenden 600 Modellen wurden weitere 69 Modelle ausgeschlossen, die (ggf. mit geringfügigen Umbenennungen der Beschriftungen von Ereignissen und Funktionen) mehrfach im Referenzmodell vorkommen.
- 112 Modelle aus 2 Bachelor-, 22 Diplom- und 6 Seminararbeiten von deutschen Hoch- und Fachschulen sowie einem Proposal zu einer Diplomarbeit. Von den Bachelor-, Diplom- und Seminararbeiten wurden 18 im Bereich Informatik oder Wirtschaftsinformatik sowie 12 im Bereich Wirtschaftswissenschaften erstellt.
- 25 Modelle aus 7 Dissertationsschriften: Von den 7 Dissertationen wurden 3 im Bereich der Ingenieurwissenschaften, 3 in den Wirtschaftswissenschaften sowie eine im Bereich der Forst- und Umweltwissenschaften verfasst.
- 13 Modelle aus 2 Handbüchern: 11 Modelle wurden der Prozessbeschreibung des Projekt-Controlling Systems *PCS* der Firma Schulz & Löw Wiesbaden entnommen. Zwei weitere Modelle entstammen dem firmeninternen Konventionenhandbuch zum Einsatz des Werkzeugs *ARIS Toolset* in mySAP.com-Projekten der Wikima AG Zug.
- 82 Modelle aus 48 wissenschaftlichen Veröffentlichungen, die in Konferenzbänden, in wissenschaftlichen Zeitschriften oder als Forschungsbericht publiziert

wurden. Da üblicherweise die Seitenzahl in Konferenzbänden und Zeitschriften limitiert ist, enthalten diese Publikationen eher kleine Beispielmodelle. Für die Untersuchung wurden nur solche Modelle berücksichtigt, die trotz ihres Beispielcharakters einen tatsächlichen betriebswirtschaftlichen Sachverhalt abbilden.

- 12 Modelle aus 6 Vorlesungsskripten deutscher Hoch- und Fachschulen. Dabei hatten drei Vorlesungen die Geschäftsprozessmodellierung zum Inhalt, zwei befassten sich mit der Verwendung von SAP R/3, eine mit Medizininformatik. Wieder wurde darauf geachtet, dass trotz des Beispielcharakters von in einer Vorlesung angegebenen Modellen tatsächliche betriebswirtschaftliche Sachverhalte abgebildet sind.
- 88 Modelle aus 7 Büchern:

Autor und Titel	EPKs
Becker/Schütte: Handelsinformationssysteme [12]	66
Keller: SAP R/3 prozessorientiert anwenden [85]	4
Seidlmeier: Prozessmodellierung mit ARIS [151]	3
Stahlknecht/Hasenkamp: Einführung in die Wirtschaftsinformatik [155]	1
vom Brocke: Referenzmodellierung [185]	1
Becker/Vossen: Geschäftsprozeßmodellierung und Workflows [13]	12
Wölfle/Schubert: Integrierte Geschäftsprozesse mit Business Software [192]	1

- 4 Modelle aus Musterlösungen zu Prüfungsaufgaben, erstellt für Prüfungen zum Thema „Systementwicklung“ an der Katholischen Universität Eichstätt-Ingolstadt sowie zum Thema „Requirements Engineering“ an der Universität Bern
- 88 Modelle aus 11 Praxisprojekten: Dieser Teil der Modellsammlung zeichnet sich dadurch aus, dass er besonders große und komplexe Modelle beinhaltet. Das Themenspektrum der Projekte ist weit gestreut: Es reicht von der Beschreibung der Geschäftsprozesse eines Energieversorgers über die Modellierung juristischer

Prozesse in der Europäischen Union bis hin zur Modellierung der Funktionalität der elektronischen Gesundheitskarte in Deutschland.

- 29 Modelle aus 14 anderen im Internet verfügbaren Quellen: In diesen Modellen wurde die EPK-Notation benutzt, um Geschäftsprozesse zu dokumentieren und per Internet zu publizieren. Beispiele für solche Prozesse sind eine Abbildung der Prüfungsordnung an der TU München oder die Modellierung des Prozesses „Schreiben eines Wikipedia-Artikels“ in der tschechischen Version der Online-Enzyklopädie Wikipedia (`cs.wikipedia.org`).

Insgesamt umfasste die Modellauswahl 984 Modelle aus 130 Quellen.

Bei der Auswahl der Modelle wurde darauf geachtet, keine Doppelungen zu erhalten. Beispielsweise wurden zahlreiche Modelle aus dem Buch „SAP R/3 prozessorientiert anwenden“ [85] nicht berücksichtigt, da sie bereits Bestandteil des SAP R/3-Referenzmodells sind.

Gelegentlich war es notwendig, leicht erkenn- und behebbare syntaktische Fehler in den untersuchten Modellen zu korrigieren. Endete etwa ein EPK-Modell fälschlicherweise mit einer Funktion, so wurde ein Endereignis hinzugenommen, um ein syntaktisch korrektes Modell zu erhalten. Ließ sich jedoch die Bedeutung eines syntaktisch falschen Modells nicht unmittelbar erschließen (etwa bei Funktionen oder Ereignissen, die mehr als eine Eingangs- oder Ausgangskante besitzen), so wurde das entsprechende Modell verworfen.

Durch Anwendung der in Abschnitt 10.7 aufgeführten Reduktionsregeln konnte für 492 der 984 Modelle (also genau die Hälfte) gezeigt werden, dass das Modell wohlstrukturiert ist. Diese Modelle wurden vollständig reduziert. Weitere 58 Modelle waren nicht wohlstrukturiert, konnten jedoch trotzdem unter Nutzung der Reduktionsregel für SESE-Regionen, in denen ausschließlich XOR-Konnektoren vorkommen, vollständig reduziert werden. Die verbleibenden 434 Modelle wurden auf Muster möglicher Probleme untersucht.

11.2 Ergebnis der Modellanalysen mit Prolog

Die Analyse der Modelle mit Prolog wurde auf einem PC mit Intel Core2Duo-Prozessor mit einer Geschwindigkeit von 3 GHz durchgeführt. Verwendet wurde die 32-bit-Version von SWI-Prolog 5.6.51 mit den Default-Einstellungen für die verwendeten Stack-Bereiche.

Die Analyse der Modelle erfolgte durch Abarbeitung eines Batch-Auftrags. Bei diesem wurde für jedes der Modelle erneut der Prolog-Interpreter gestartet, das

Prolog-Programm geladen und abgearbeitet. Die Analyse aller 984 EPKs benötigte insgesamt 65 Sekunden. In dieser Zeit ist die XSLT-Transformation, die die im EPML-Format vorliegenden EPKs in Prolog-Fakten übersetzt, nicht enthalten. Der Zeitaufwand für diese Transformation ist allerdings praktisch vernachlässigbar.

Tabelle 11.1 auf Seite 171 zeigt, wie viele Instanzen der Muster 1 bis 20 in den 984 untersuchten Modellen gefunden wurden.

Wie bereits bei Mendling [113] liegt Muster 13 deutlich an der Spitze der Muster, die zu einem Kontrollflussfehler führen und daher in die Kategorie „Fehler“ eingeordnet sind. Übertroffen wird dieses Muster lediglich von Muster 19, das für die Warnung steht, dass Ablauflogik und Startereignisse vermischt sind. Häufig anzutreffen sind zudem die diversen falschen Kombinationen von Split und Join. Der häufigste Fall war hier die Kombination eines AND- bzw. OR-Splits mit einem XOR-Join (Muster 4), die 99mal auftrat und und zu einem Ablauf mit Synchronisationsfehler führen kann.

Ebenso gab es einige Muster, die nicht oder sehr selten identifiziert wurden. Hier lohnt es sich, zu untersuchen, auf welche Regeln verzichtet werden kann, um die Ausführungszeit der Prolog-Tests nochmals zu verkürzen. Um dies festzustellen, wurde die Prolog-Auswertung aller Modelle nochmals durchgeführt, wobei auf die Regeln für die seltener als 5mal gefundenen Muster (Muster 5, 9, 16 und 18), verzichtet wurde.

Muster 5 wurde in den 984 untersuchten EPKs überhaupt nicht gefunden. Allerdings führt ein Verzicht auf die Prolog-Regeln für dieses Muster zu keiner nennenswerten Zeitersparnis. Da aber durchaus Modelle außerhalb der für diese Arbeit verwendeten Modellsammlung in der Praxis vorkommen (etwa in [138], modelliert als UML Aktivitätsdiagramm) scheint es sinnvoll, die Suche nach Muster 5 in den Prolog-Regeln zu belassen. Gerade beim Vorfinden dieses Musters können dem Modellierer hilfreiche Vorschläge für eine mögliche inhaltliche Verbesserung des Modells gegeben werden.

Die Regeln für Muster 9 (dreimal gefunden) und Muster 18 (einmal gefunden) belasten ebensowenig die Ausführungsgeschwindigkeit des Prolog-Programms und sollten (da sie nützliche Modellverbesserungen anzeigen) im Programm verbleiben.

Bei dem 4mal gefundenen Muster 16 allerdings war festzustellen, dass die Suche nach diesem Muster die Ausführung des gesamten Prolog-Programms deutlich verlangsamt: Ohne eine Suche nach diesem Muster dauerte die Analyse aller 984 Modelle statt 65 Sekunden nur noch 52 Sekunden. Da zudem das Muster nur an Stellen gefunden wurde, an denen auch durch die Anwendung anderer Muster bereits

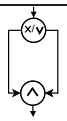
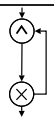
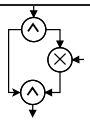
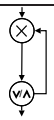
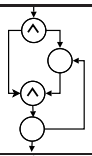
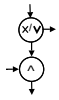
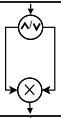
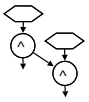
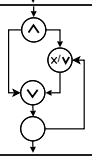
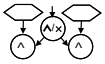
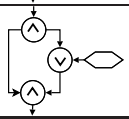
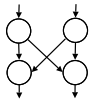
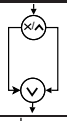
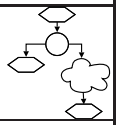
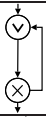
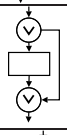
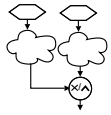
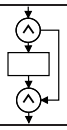
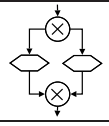
Muster 1 	38 in 26 EPKs (F) 21 in 16 EPKs (W)	Muster 11 	33 in 27 EPKs (F)
Muster 2 	13 in 8 EPKs (F) 12 in 5 EPKs (W)	Muster 12 	48 in 35 EPKs (F)
Muster 3 	8 in 5 EPKs (F), 0 Warnungen	Muster 13 	302 in 183 EPKs (F)
Muster 4 	99 in 52 EPKs (F)	Muster 14 	63 in 34 EPKs (W)
Muster 5 	0	Muster 15 	67 in 34 EPKs (W)
Muster 6 	12 in 4 EPKs (W)	Muster 16 	4 in 4 EPKs (W)
Muster 7 	72 in 40 EPKs (W)	Muster 17 	21 in 5 EPKs (W)
Muster 8 	30 in 25 EPKs (W)	Muster 18 	1 in 1 EPK (W)
Muster 9 	3 in 3 EPKs (W)	Muster 19 	490 in 201 EPKs (W)
Muster 10 	30 in 21 EPKs (W)	Muster 20 	23 in 23 EPKs (W)

Tabelle 11.1: Gefundene Musterinstanzen
(F=Kategorie „Fehler“, W=Kategorie „Warnung“)

ein Fehler lokalisiert wurde, erscheint es sinnvoll, im Interesse einer schnelleren Ausführungszeit auf eine Suche nach Muster 16 zu verzichten.

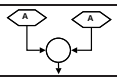
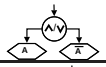
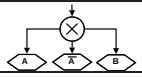
Problemklasse	korrekt erkannte Fehler	unberechtigte Fehlermeldungen und Zweifelsfälle
Muster 21 	12 in 12 EPKs	4 in 4 EPKs
Muster 22 	21 in 12 EPKs	keine
Muster 23 	8 in 6 EPKs	6 in 5 EPKs

Tabelle 11.2: Gefundene Muster 21 bis 23 in den deutschsprachigen Modellen

Die Muster 21 bis 23, die eine Untersuchung der Modellbeschriftung erfordern, wurden nur an denjenigen 357 Modellen der EPK-Sammlung getestet, deren Ereignisse und Funktionen in deutscher Sprache beschriftet sind. Das Prolog-System benötigte für die Suche nach den Mustern 21 bis 23 in diesen 357 EPKs insgesamt 26 Sekunden.

Es kann festgestellt werden, dass in 24 der 357 Modelle (was einem Anteil von knapp 7% entspricht) mit Hilfe der Muster 21 und 22 ein tatsächlicher Modellierungsfehler gefunden wurde. In der Regel handelte es sich dabei um einen falsch verwendeten Konnektortypen; besonders häufig trat hier die Verwendung von OR statt des in dieser Situation korrekten XOR auf.

Für Muster 23 stehen 8 eindeutig korrekt erkannten Problemen 6 unberechtigte Fehlermeldungen oder Zweifelsfälle gegenüber. Ob tatsächlich ein Fehler vorliegt, war fachlich nicht in jedem Falle nachvollziehbar. In Zweifelsfällen wurde der Fall als „unberechtigte Fehlermeldung“ eingestuft.

Beispiele für Kombinationen von Aussagen, bei es auch ohne tiefere Kenntnisse der Anwendungsdomäne nachvollziehbar ist, dass die Fehlermeldung unberechtigt ist, waren:

- „Kundenauftrag angenommen“ / „Kundenauftrag abgelehnt“ / „Kundenauftrag gesperrt“
- „CR ist vollständig“ / „CR ist nicht vollständig“ / „CR ist nicht vorhanden“ (siehe Abb. 9.34)

Obwohl rein formallogisch die jeweils erste Aussage die Negation der jeweils zweiten Aussage ist, ist in den vorgestellten Fällen die Modellierung der drei Ereignisse mit XOR-Verknüpfung sachlich nachvollziehbar.

11.3 Ergebnis der Modellanalysen mit vorhandenen Werkzeugen

Um die Ergebnisse der musterbasierten Analyse mit den Resultaten anderer Werkzeuge zu vergleichen, wurden die untersuchten Modelle ebenfalls mit anderen aus der wissenschaftlichen Literatur bekannten Werkzeugen analysiert.

Hierzu wurden Werkzeuge gewählt, die verschiedene Ansätze - insbesondere bezüglich der Definition der Übergangsrelation für nichtlokale Konnektoren - repräsentieren: Das Werkzeug *EPCTools* [28, 30, 29] berechnet eine Fixpunktsemantik. Das *ProM EPC Soundness Analysis Plugin* [9] nutzt Mendlings Semantik aus [113]. Der *YAWL Editor* [197] schließlich benutzt die von Wynn eingeführte YAWL-Semantik [195].

Wie auch die musterbasierten Analysen mit Prolog, wurden die Auswertungen mit den genannten Werkzeugen auf einem PC mit Intel Core2Duo-Prozessor mit einer Geschwindigkeit von 3 GHz durchgeführt. Als Betriebssystem kam Microsoft Windows Vista Business in der 64-bit-Version zu Einsatz. Die Java Virtual Machine (SUN Java SE, 1.6.0_07-b06), die die jeweiligen Programme startete, wurde mit einer Heap-Größe von 1,5 GB ausgeführt. Eine solche Konfiguration ist zum gegenwärtigen Zeitpunkt für Arbeitsplatzrechner eher überdurchschnittlich.

11.3.1 Analyse mit EPCTools

Das Werkzeug *EPCTools* [28, 30, 29] von Cuntz und Kindler zeichnet sich dadurch aus, dass es für das zu untersuchende Modell eine Fixpunktsemantik zu berechnen versucht. Für die Modelle, für die eine solche berechnet werden kann, überprüft *EPCTools* Soundness sowie schwache Soundness der EPK. Hierzu wird der symbolische Model-Checker des Projekts Model Checking in Education (MCiE) der Universität Paderborn genutzt.

Für alle 984 Modelle wurde versucht, eine Fixpunktsemantik zu finden. Sofern eine solche berechnet werden konnte, wurde das Modell auf die Eigenschaften „Soundness“ und „schwache Soundness“ getestet. Hierfür wurde das Werkzeug *EPCTools* so erweitert, dass es diese Modelleigenschaften gemäß Def. 17 prüft, also alle denkbaren Kombinationen von Startereignissen im Startzustand überprüft. Motiviert von der in Def. 17 angegebenen Soundness-Definition erfolgte die Ausgabe „sound“ genau dann, wenn es einen Startzustand gibt, für den *EPCTools* zu dem Ergebnis kommt, dass die EPK mit diesem Startzustand sound ist.

Zunächst wurde die Untersuchung an den unveränderten (nicht reduzierten) Modellen durchgeführt. 14 Berechnungen konnten nicht innerhalb einer Stunde abgeschlossen werden und wurden nach Ablauf dieser Zeit abgebrochen. Für weitere 25 Modelle beendete sich das Werkzeug *EPCTools* mit der Fehlermeldung, dass der zur Verfügung stehende Arbeitsspeicher für die Berechnung nicht ausreiche. Somit konnte das Werkzeug *EPCTools* für etwa 4% der untersuchten Modelle die Frage nach der Korrektheit nicht beantworten. Nachdem die in Abschnitt 10.7 beschriebenen Reduktionsregeln auf alle Modelle angewendet wurden, ergab sich bei der Analyse der reduzierten Modelle mit *EPCTools* noch für 7 Modelle einen Programmabbruch wegen fehlenden Arbeitsspeichers. Für eine EPK benötigte die Analyse des reduzierten Modells mehr als eine Stunde (genau: 63 Minuten).

Für genau drei der 984 untersuchten Modelle stellte *EPCTools* fest, dass keine Fixpunktsemantik existiert. Dies entspricht der Erwartung, dass solche Modelle in der Praxis sehr selten sind. Gleichwohl zeigt die Tatsache, dass drei Modelle ohne Fixpunktsemantik gefunden wurden, dass EPKs ohne Fixpunktsemantik nicht nur in konstruierten wissenschaftlichen Beispielen wie dem in [172] gezeigten Teufelskreis vorkommen.

11.3.2 Analyse mit ProM

Das *EPC Soundness Analysis Plugin* des Werkzeugs *ProM* [9] nutzt die von Mendling in [113] eingeführte EPK-Semantik. Es berechnet nach dieser Semantik zu einer EPK das Transitionssystem (also den Zustandsraum, der alle möglichen Abläufe einer EPK beschreibt). An Hand des berechneten Transitionssystems wird dann die Soundness-Eigenschaft überprüft.

Auch mit dem *ProM*-Plugin wurden alle 984 EPKs auf Soundness getestet. Von den unreduzierten Modellen konnten 831 innerhalb von einer Stunde überprüft werden.

Nach Anwendung der Reduzierungsregeln erhöhte sich diese Zahl auf 953. Für 31 Modelle konnte das *ProM*-Plugin auch nach der Reduzierung die Soundness-Eigenschaft nicht innerhalb einer Stunde überprüfen.

11.3.3 Analyse mit dem YAWL Editor

Die Soundness-Analyse des Werkzeugs *YAWL Editor* wurde für die Überprüfung von in der Sprache YAWL geschriebenen GPM entwickelt. Daher mussten vor einer Analyse die als EPK vorliegenden Modelle in die Sprache YAWL übertragen werden. Das ist ohne Problem möglich, da die Sprache YAWL sämtliche in der Sprache EPK vorhandenen Notationselemente unterstützt.

Allerdings haben YAWL-Modelle die Eigenschaft, dass es genau einen Anfangs- und genau einen Endknoten des Prozessmodells geben muss. Eine Analyse von EPKs mit mehr als einem Startereignis wird somit vom *YAWL Editor* nicht unterstützt, da unklar ist, welche Startzustände erlaubt sind. Um trotz dieser Einschränkung möglichst viele Ergebnisse der YAWL-Soundnessanalyse zu erhalten, wurden sämtliche Modelle unter Anwendung der in Abschnitt 10.7 beschriebenen Reduktionsregeln reduziert. Von den ursprünglich 984 Modellen verblieben 737, die in der reduzierten Form genau ein Startereignis haben.

Diese 737 Modelle wurden mit dem Werkzeug *YAWL Editor* untersucht. Der *YAWL Editor* bietet neben einem Test auf Soundness auch die Suche nach OR-Joins an, die durch AND oder XOR ersetzt werden können. Dieser Test wurde gemeinsam mit der Soundness-Analyse ausgeführt. Der *YAWL Editor* erlaubt es, bei der Analyse zwei Arten von Reduktionsregeln zu verwenden, um die Geschwindigkeit der Analyse zu erhöhen [195]. Zunächst wurden die Analysen unter Anwendung beider Typen von Reduktionsregeln durchgeführt; die in diesem Abschnitt genannten Ausführungszeiten sind die bei Anwendung sämtlicher Reduktionsregeln. Da jedoch in der Implementierung der YAWL-Reduktionsregeln ein Fehler entdeckt wurde, mussten die Analysen, um korrekte Ergebnisse zu garantieren, noch einmal ohne Anwendung der YAWL-Reduktionsregeln wiederholt werden.

Die YAWL-Semantik erlaubt beliebig viele Marken auf einer Kante des Modells. Dies stellt einen Unterschied zu der in dieser Arbeit verwendeten Def. 11 dar. Eine direkte Folge dieser Tatsache ist, dass die Menge aller möglichen Zustände eines YAWL-Modells unendlich werden kann. Im Analyseprogramm des *YAWL Editors* musste also eine Endlos-Schleife verhindert werden. Dies gelang durch Einführung einer Beschränkung: Sobald die Zahl der berechneten Zustände 10000 überschreitet, beendet sich die YAWL-Analyse ohne Ergebnis. Dies war für 13 Modelle im Test der Fall. Bis zur Feststellung, dass es mehr als 10000 Zustände gibt, vergingen im Durchschnitt 36:05 Minuten. Die längste Berechnung, die mit der Meldung „10000 Zustände überschritten“ endete, dauerte 5:12 Stunden.

Für 14 EPKs endete die Berechnung mit einem Programmabbruch, der offenbar auf einen Programmfehler zurückzuführen war, der nicht im Zusammenhang mit einer Zustandsexplosion stand.

Für die verbleibenden 710 Modelle wurde die Soundnessberechnung erfolgreich durchgeführt. Diese erfolgte dann auch sehr schnell, für 622 Modelle in weniger als einer Sekunde. Drei Modelle benötigten über 10 Sekunden, die längste Berechnung dauerte 256 Sekunden.

11.4 Vergleich der Ergebnisse von EPCTools, ProM und YAWL Editor

EPCTools und *ProM* sind die beiden Werkzeuge, mit denen alle 984 EPKs unserer Modellsammlung analysiert wurden. Wir beginnen den Vergleich der Ergebnisse mit einer Gegenüberstellung der Analysen dieser beiden Werkzeuge. Ein Vergleich war für die 947 Modelle möglich, für die sowohl *EPCTools* als auch *ProM* ein Ergebnis lieferte.

EPCTools fand genau drei Modelle, die keine ideale Semantik (wie auf Seite 26 beschrieben) besitzen. *ProM* stellte für alle diese Modelle fest, dass sie nach der von Mendling eingeführten Semantik nicht sound sind.

Es gab zwei Klassen von Modellen, für die die Ergebnisse der beiden Werkzeuge nicht übereinstimmen.

Ein Unterschied betraf Modelle mit mehreren Startereignissen, die von *EPCTools* als „sound“, von *ProM* dagegen als „nicht sound“ eingestuft wurden. Der Grund hierfür liegt darin, dass die beiden Werkzeuge verschiedene Definitionen des Begriffs Soundness nutzen. Das *ProM EPC Soundness Analysis Plugin* nutzt die in dieser Arbeit auf Seite 33 angegebene Def. 17. Dagegen beruht der in *EPCTools* integrierte Soundness-Test auf einer Soundness-Definition, die von der in Abschnitt 3.2.1 beschriebenen abweicht. Die dritte Forderung in der Soundness-Definition von van der Aalst [167] („Es gibt kein Modellelement, das nicht zu einem korrekten Ablauf beiträgt.“) wird nicht beachtet. Das führt dazu, dass etwa das Modell in Abb. 11.1 mit dem Startzustand $\{E1\}$ als sound eingestuft wird, obwohl offenbar das Startereignis E2 in keinem Ablauf enthalten ist, der nicht zu einem Deadlock führt.

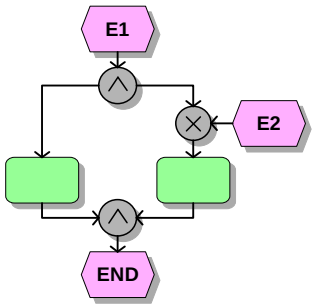


Abbildung 11.1: Modell wird von *EPCTools* als sound betrachtet

Die zweite Klasse von Modellen, für die das Ergebnis von *EPCTools* und *ProM* unterschiedlich ausfiel, betraf Modelle mit einem OR-Join innerhalb eines Zyklus.

Alle in Abb. 11.2 gezeigten Modelle haben laut Mendlings Semantik einen Deadlock am OR-Join. Nach Überzeugung des Autors entspricht dies jedoch nicht der unter Anwendern der EPK-Notation vorherrschenden Auffassung von der Bedeutung dieser Modelle.

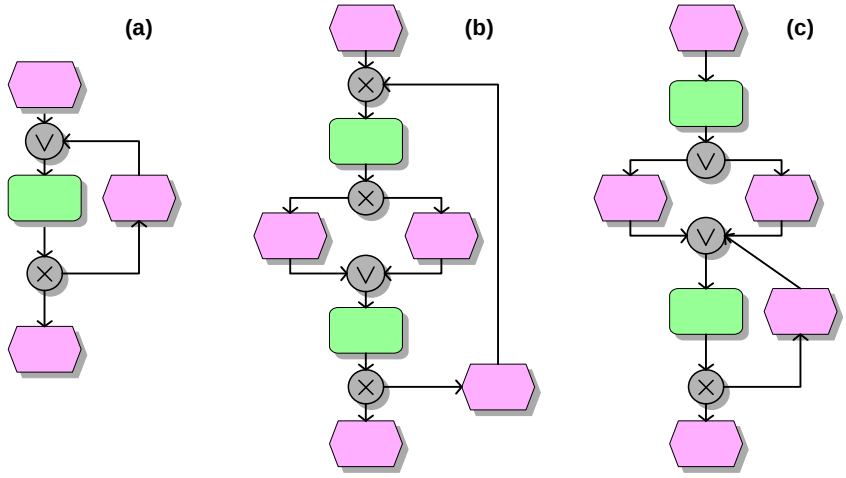


Abbildung 11.2: Modelle haben lt. Mendlings Semantik einen Deadlock am OR-Join

Von den beiden genannten Unterschieden abgesehen, stimmten die Ergebnisse von *EPCTools* und dem *ProM*-Plugin überein.

Noch größer war die Übereinstimmung zwischen den Ergebnissen von *EPCTools* und *YAWL Editor*. Wie erwähnt, wurden die *YAWL Editor*-Auswertungen für diejenigen 737 EPKs durchgeführt, die sich per Reduktionsregeln zu einem Modell mit genau einem Startereignis reduzieren lassen.

Für zwei der drei Modelle ohne Fixpunktsemantik konnte der *YAWL Editor* ein Ergebnis berechnen. Beide Modelle sind nicht sound nach YAWL-Semantik.

Es konnte genau ein Fall festgestellt werden, in dem das Ergebnis der *YAWL Editor*-Analyse nicht mit dem von *EPCTools* übereinstimmte. Dies ließ sich darauf zurückführen, dass die YAWL-Semantik der beiden OR-Joins für das Modell in Abb. 11.3 auf Seite 178 nicht mit der von *EPCTools* berechneten Fixpunktsemantik übereinstimmt. Während die Fixpunktsemantik von *EPCTools* annimmt, dass beide OR-Joins eine Markierung weiterleiten können, blockieren laut YAWL-Semantik (und auch nach Mendlings Semantik) beide OR-Joins.

Interessant ist auch die Betrachtung der Fälle, in denen der *YAWL Editor* kein Ergebnis liefert, da die Zahl von 10000 berechneten Zuständen überschritten wird. Dies trat nur für Modelle auf, die lt. *EPCTools* nicht sound sind und deren Zu-

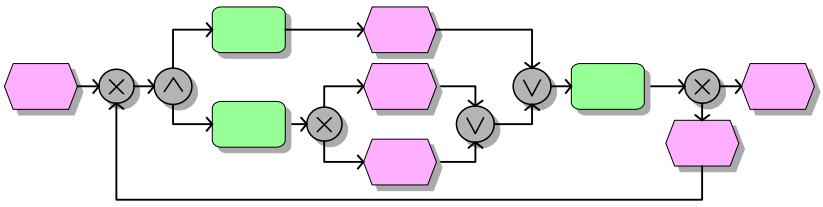


Abbildung 11.3: YAWL-Semantik weicht von Fixpunktsemantik ab

standsmenge laut YAWL-Semantik unendlich wird. Zwei charakteristische Beispiele für solche Modelle sind in Abb. 11.4 zu sehen. Für beide ist leicht zu sehen, dass sie fehlerhaft sind. Dass solche Fehler vom *YAWL Editor* nicht erkannt werden können, ist ein Nachteil der YAWL-Semantik. Es wäre hilfreich, die *YAWL Editor*-Implementierung um zusätzliche Maßnahmen zur Erkennung der gezeigten Fehler zu erweitern. In [183] wurde zu diesem Zweck die Betrachtung von Invarianten in Petrinetzen vorgeschlagen.

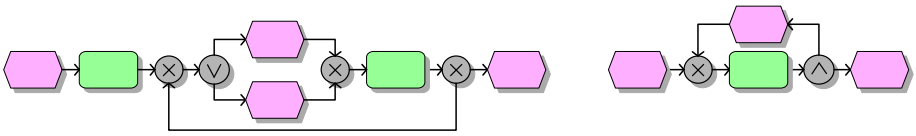


Abbildung 11.4: Modelle, deren Zustandsmenge in YAWL unendlich wird

Zusammenfassend zeigt Tab. 11.3 auf Seite 179 eine Gegenüberstellung der Ergebnisse der drei untersuchten Werkzeuge für diejenigen Modelle, für die eine Soundnessanalyse in allen drei Werkzeugen durchgeführt werden konnte. Die vier Modelle, für die lediglich *ProM* Soundness feststellt, lassen sich leicht auf einen noch nicht behobenen Fehler im *ProM*-Plugin zurückführen. Bei korrekter Anwendung von Mendlings Semantikdefinition sind die betroffenen Modelle nicht sound, so wie es auch die anderen Werkzeuge feststellen. Die anderen Fälle, in denen es Differenzen zwischen den Ergebnissen der einzelnen Werkzeuge gibt (in der Tabelle grau unterlegt), entsprechen den in den Abbildungen Abb. 11.2 und Abb. 11.3 gezeigten Fällen.

Es kann festgestellt werden, dass die drei Werkzeuge in der überwiegenden Zahl der Fälle zu einem einheitlichen Ergebnis kommen. Trotzdem existieren gelegentliche Differenzen.

Ein Vergleich der durch Mustersuche mit Prolog erzielten Ergebnisse mit den Ergebnissen der betrachteten Werkzeuge erfolgt in Abschnitt 11.5.2. Zuvor sollen

ProM-Plugin	EPCTools	YAWL Editor	Zahl der Modelle
unsound	unsound	unsound	24
unsound	unsound	sound	0
unsound	sound	unsound	1
unsound	sound	sound	9
sound	unsound	unsound	(4)
sound	unsound	sound	0
sound	sound	unsound	0
sound	sound	sound	119

Tabelle 11.3: Ergebnis der Soundness-Analysen

jedoch in Abschnitt 11.5.1 die Ergebnisse der Syntaxüberprüfung mit Prolog mit anderen bekannten Verfahren verglichen werden.

11.5 Vergleich der Ergebnisse des musterbasierten Ansatzes mit bekannten Methoden

11.5.1 Überprüfung der syntaktischen Qualität

Eine Überprüfung der syntaktischen Korrektheit eines Modells ist leicht möglich, indem die EPK-Syntaxanforderungen nach Def. 1 in Prolog-Regeln übersetzt werden. Bei der Überprüfung der Korrektheit wird dann an die Wissensbasis die Anfrage gestellt, ob im Modell Elemente gefunden werden, durch die die entsprechende Syntaxanforderung verletzt wird.

Um etwa zu überprüfen, ob der Graph antisymmetrisch ist, wird zunächst nach zwei Kanten der Form $(a, b)/(b, a)$ gesucht. Wird kein solches Gegenbeispiel gefunden, ist die betreffende Eigenschaft für die untersuchte EPK erfüllt. Wird ein Gegenbeispiel gefunden, so erhält der Modellierer dadurch die Information, an welcher Stelle das Modell syntaktisch falsch ist.

Sämtliche Regeln aus Def. 1 lassen sich leicht als Prolog-Regeln formulieren. Dabei ist die Kürze der Regeln erwähnenswert: Lediglich die Regeln, die Aussagen zu Themen wie „Erreichbarkeit“ oder „Zusammenhang“ treffen, benötigen mehr als eine einzelne Programmzeile, und auch diese Regeln zur Erreichbarkeit umfassen zusammen lediglich 13 Prolog-Zeilen. Wie Störrle in [158] an ähnlichen Beispielen zeigte, sind Prolog-Regeln deutlich kürzer und einfacher lesbarer als vergleichbare

Regeln, die in der Object Constraint Language (OCL) notiert wurden. Zudem zeigten sich in [158] deutliche Vorteile in der Ausführungsgeschwindigkeit von Prolog- gegenüber OCL-Abfragen.

Durch die beschriebenen Tests auf syntaktische Korrektheit können alle in Def. 1 aufgeführten Syntaxregeln für EPKs überprüft werden. Dies leisten bisher weder Metamodell-basierte Ansätze noch die in [115] vorgestellte Lösung unter Nutzung der Sprache Schematron.

Ein Vorteil des vorgestellten Ansatzes gegenüber dem in [115] beschriebenen besteht darin, dass (wie in Abschnitt 4.1 diskutiert) im Austauschformat EPML auf überflüssige Attribute, die lediglich redundante Informationen darstellen, verzichtet werden kann. Dies führt dazu, dass das Austauschformat einfacher wird.

Die vorgestellten Tests auf syntaktische Korrektheit können sinnvoll gemeinsam mit Metamodell-basierten Verfahren eingesetzt werden, die manche Fehler im Modell gar nicht erst zulassen. Das wird dadurch erreicht, dass das Metamodell bereits Einschränkungen an die zu modellierenden Modellinstanzen trifft. Eine Regel, für die ein solches Vorgehen sinnvoll ist, ist z.B. das Verbot reflexiver Kanten im Graphen (vgl. Forderung 2 in Def. 1). Vorhandene Werkzeuge wie das *ARIS Toolset* gestatten dem Modellierer gar nicht, solche Kanten zu zeichnen. Gleichwohl müssen bei einer Validierung einer Eingabedatei auch solche Regeln überprüft werden, insbesondere wenn ein Modell importiert werden soll, das mit einem anderen Werkzeug erstellt wurde.

11.5.2 Überprüfung der semantischen Qualität

Wir wollen nun die Ergebnisse der Prolog-Suche nach Mustern für Kontrollflussfehler mit den Ergebnissen der Soundness-Überprüfungen vergleichen, die durch die exakten dynamischen Analysewerkzeuge gewonnen wurden.

Für die Fälle, in denen die Werkzeuge zu verschiedenen Ergebnissen kommen, wurde zum Vergleich das Resultat herangezogen, das nach Auffassung des Autors dem intuitiven Modellverständnis am nächsten kommt. Konkret heißt das, dass die in Abb. 11.2 auf Seite 177 gezeigten Fälle ebenso als korrekt gewertet wurden wie das Modell von Abb. 11.3 auf Seite 178. Modelle wie in Abb. 11.1 auf Seite 176, die *EPCTools* als sound einstuft, wurden in Übereinstimmung mit Def. 17 als fehlerhaft betrachtet.

In diesem Abschnitt betrachten wir nur die Ergebnisse derjenigen Prolog-Regeln, die Muster für einen Kontrollflussfehler finden sollen. Dies sind die Regeln, für die das Prolog-Programm laut Musterkatalog in Kapitel 9 die Meldung „Fehler“ ausgeben

kann. In 191 Modellen aus unserer Modellsammlung von 984 EPKs wurde ein Muster festgestellt, für das das Prolog-Programm die Meldung „Fehler“ ausgibt. Drei dieser Modelle konnten mit keinem der drei dynamischen Analyseverfahren analysiert werden. Die verbleibenden 188 EPKs wurden durch dynamische Analyse tatsächlich als „nicht sound“ erkannt.

Für 4 Modelle gab Prolog die Meldung „Warnung“, jedoch keine Meldung „Fehler“ aus. Drei dieser Modelle waren, wie das Ergebnis der dynamischen Analyseverfahren zeigt, tatsächlich fehlerhaft. Das vierte Modell war dagegen sound. Dieses Modell war der einzige Fall, in dem Prolog (bei Muster 1) eine Warnung für eine EPK ohne Kontrollflussfehler erzeugte. Bemerkenswert ist die Herkunft des Modells: Es wurde in [117] als ein Beispiel für ein GPM publiziert, für das diverse Verbesserungen ratsam sind. Tatsächlich führte gerade der in [117] kritisierte stark unstrukturierte Modellierungsstil dazu, dass Prolog die Warnung über einen vermutlichen Fehler anzeigt.

Für die verbleibenden 789 EPKs meldete das Prolog-Programm weder Fehler noch Warnungen. Tatsächlich wurden all diese Modelle bei Anwendung der dynamischen Analyseverfahren als „sound“ eingestuft.

Zusammenfassend lässt sich feststellen, dass mit dem musterbasierten Verfahren in sämtlichen Modellen, die laut dynamischer Analyse nicht sound waren, tatsächlich mindestens einen Kontrollflussfehler gefunden wurde. Umgekehrt gab es nur in einem einzigen Falle eine unberechtigte Warnung.

11.5.3 Überprüfung der pragmatischen Qualität

Der in dieser Arbeit vorgestellte Ansatz kann verschiedene Probleme identifizieren, die mit der Einfachheit und Lesbarkeit der Modelle zusammenhängen. Einige davon (Muster 6, 9, 10, 14 sowie 16 bis 20) werden nach Wissen des Autors von keinen anderen Ansätzen bzw. Modellierungswerkzeugen untersucht.

Eine Überprüfung, ob ein OR-Join durch einen XOR- bzw. AND-Join ersetzt werden sollte, um den modellierten Sachverhalt verständlicher zu beschreiben, gibt es auch bei Rump [145] und Wynn [195].

Der Ansatz von Wynn ist als eine der Analysen im *YAWL Editor* integriert. Somit können wir für die 710 Modelle aus unserer Modellsammlung, die vom *YAWL Editor* analysiert werden konnten, die Ergebnisse des *YAWL Editors* mit denen der Mustersuche mit Prolog vergleichen. Die YAWL-Analyse muss hierzu ohne Verwendung der YAWL-Reduktionsregeln durchgeführt werden, da diese nicht eigenschaftserhaltend bezüglich der Ersetzbarkeit von OR-Joins sind.

In 39 EPKs lieferte die Anwendung der Muster 7 und 8 in Prolog-Analyse das Ergebnis, dass mindestens ein OR-Join im Modell ersetzt werden kann. *YAWL Editor* kam für all diese EPKs zum selben Ergebnis.

In den verbleibenden 671 EPKs fand die Prolog-Analyse keine Möglichkeit, einen OR-Join zu ersetzen. Für genau eine dieser EPKs konnte der *YAWL Editor* dennoch feststellen, dass eine Ersetzung möglich ist. In allen anderen Fällen stimmen die Ergebnisse von Prolog und *YAWL Editor* überein.

Zusammenfassend kann also festgestellt werden, dass die Prolog-Analyse im Vergleich zum exakten vom *YAWL Editor* berechneten Ergebnis genau einen ersetzbaren OR-Join nicht finden konnte.

Andererseits wurden durch die Prolog-Analyse vier Instanzen der Muster 18 und 9 identifiziert, wodurch vier weitere OR-Splits und drei OR-Joins durch XOR-Konnektoren ersetzt werden können. Diese wurden durch den *YAWL Editor* nicht gefunden.

Bemerkenswert ist die Feststellung, dass von den insgesamt 443 in den Modellen vorhandenen OR-Joins nicht weniger als 105 durch einen AND- bzw. XOR-Join ersetzt werden können (Muster 7, 8 und 9). Muster 18 und 9 fanden außerdem 4 weitere ersetzbare OR-Splits. Weiterhin wurden in den deutschsprachigen Modellen durch das Muster 22 zwölf weitere OR-Splits identifiziert, die ersetzt werden *müssen*. Diese Ergebnisse machen deutlich, dass OR-Konnektoren zu häufig eingesetzt werden. Die Modelle in einigen Quellen zeigten, dass den Modellierern der Unterschied zwischen XOR- und OR-Konnektor nicht hinreichend bewusst war.

11.6 Zusammenfassung der Ergebnisse der Validierung

Der in dieser Arbeit vorgestellte musterbasierte Ansatz liefert für in der Praxis vorkommende EPKs sehr schnell Informationen über mögliche Fehler und Verbesserungsmöglichkeiten. Bei formalen Ansätze, die den gesamten Zustandsraum eines Modells untersuchen müssen, kann es dagegen zu der als Zustandsexplosion bezeichneten Situation kommen, in der Rechenkapazität oder verfügbarer Speicher nicht ausreichen, um die Berechnung in einer angemessenen Zeit abzuschließen.

Tabelle 11.4 auf Seite 183 zeigt für die vier verglichenen Werkzeuge, bei wie vielen Modellen die Analysezeit über 10 Minuten bzw. über 1 Stunde lag, welches die längste beobachtete Auswertungszeit[1] war und wie viele Modelle auf Grund

[1] Beim *YAWL Editor* wurden nur die Berechnungsdauern der erfolgreich analysierten Modelle betrachtet. Die Maximalzeit bis zur Feststellung, dass wegen Überschreitens der Zahl von 10000 Zuständen keine Aussage getroffen werden kann, lag bei 5:12 Stunden.

eines Speicherüberlaufs (bzw. der im *YAWL Editor* eingebauten Beschränkung auf maximal 10000 untersuchte Zustände) nicht analysiert werden konnten.

In dieser Gegenüberstellung zeigen sich deutlich die Vorteile des musterbasierten Ansatzes.

	Auswertungsdauer			
Werkzeug	> 10 Minuten	> 1 Stunde	max.	keine Auswertung möglich
musterbasiert mit Prolog	0 EPKs	0 EPKs	16 Sekunden	0 EPKs
EPCTools	1 EPK	1 EPK	63 Minuten	7 EPKs
ProM	4 EPKs	0 EPKs	22 Minuten	31 EPKs
YAWL Editor	0 EPKs	0 EPKs	256 Sekunden	13 EPKs

Tabelle 11.4: Laufzeiten der Werkzeuge (bei reduzierten Modellen) im Vergleich

Der in dieser Arbeit vorgestellte musterbasierte Ansatz ist heuristischer Natur. Es ist typisch für heuristische Verfahren, dass sie schnell eine „gute“ Lösung für ein Problem finden, diese Lösung jedoch nicht als „optimal“ oder „vollständig“ angesehen werden kann. Die Motivation zur Nutzung heuristischer Verfahren liegt darin, Ressourcen (Zeit bzw. Rechenkapazität) im Vergleich zum Einsatz exakter Verfahren zu sparen.

Dies geschieht für solche Fälle, in denen ein exaktes Verfahren die vollständige Lösung nur unter unvertretbar hohem Ressourcenaufwand finden kann oder kein exaktes Verfahren bekannt ist.

Die Güte eines heuristischen Verfahrens zeigt sich dann darin, wie nahe eine gefundene Problemlösung an der vollständigen Lösung liegt. Es wurde gezeigt, dass die in dieser Arbeit vorgestellte Lösung im Vergleich zu den Ergebnissen exakter Methoden eine sehr hohe Fehlererkennungsrate aufweist: In *allen* EPKs, die laut dynamischer Analyse nicht sound sind, meldete Prolog einen Fehler. Es gab lediglich eine Meldung (der Klasse „Warnung“) für ein Modell ohne Kontrollflussfehler.

Es kann jedoch trotz der guten Fehlererkennungsrate nicht mit Sicherheit davon ausgegangen werden, dass die durch Mustersuche erkannten Analyseergebnisse 100%ig korrekt und vollständig sind.

Eine musterbasierte Modellüberprüfung zur Modellierungszeit kann und soll also eine formale Validierung nach der Fertigstellung des Modells nicht ersetzen, sondern kann gemeinsam mit dieser angewendet werden. Der entscheidende Gewinn beim

Einsatz des heuristischen Verfahrens zur Modellierungszeit ist, dass mögliche Probleme sofort erkannt werden, sobald sie ins Modell eingefügt wurden. Sie können somit sofort durch den Modellierer behoben werden. Der folgende Abschnitt beschreibt ein Experiment, das belegt, wie auf diese Art die Qualität eines GPM verbessert werden kann.

11.7 Experiment zum praktischen Nutzen von Hintergrundvalidierung

Um einen Eindruck davon zu bekommen, wie sich eine im Hintergrund laufende Modellüberprüfung auf die Qualität der Modelle auswirkt, wurde ein Experiment mit einer Gruppe von Studenten der Wirtschaftswissenschaften an der FH Bonn-Rhein-Sieg durchgeführt. Dieses fand im Rahmen der Lehrveranstaltung Geschäftsprozessmodellierung statt.

Die Studenten wurden in zwei Gruppen eingeteilt. Beiden Gruppen wurde dieselbe Modellierungsaufgabe gestellt. Eine der beiden Gruppen konnte einen EPK-Editor mit im Hintergrund ablaufender Validierung nutzen. Hierfür wurden verschiedene Korrektheitsüberprüfungen in das auf dem Eclipse-Framework basierende Open-Source-Modellierungswerkzeug *bflow Toolbox* integriert[2]. Dazu gehörten neben den Überprüfungen auf korrekte Syntax die wichtigsten Muster für Kontrollflussfehler aus Kapitel 9. Einzelheiten zu dieser Implementierung sind in [94] zu finden.

Den 15 teilnehmenden Studenten wurden zunächst Fragebögen vorgelegt. Darin waren Fragen nach Alter, Geschlecht, Studienrichtung, Semester und Muttersprache zu beantworten. Weiterhin wurden die Teilnehmer gebeten, ihre Vertrautheit mit EPKs sowie ihre Erfahrungen mit GPM-Editoren und mit dem Eclipse-Framework einzuschätzen. Ausgehend von den erhaltenen Antworten wurden die Studenten in zwei Gruppen mit annähernd gleichen Merkmalen bezüglich der im Fragebogen erfragten Eigenschaften eingeteilt. Insbesondere wurden die beiden Teilnehmerinnen, deren Muttersprache nicht deutsch war, in verschiedene Gruppen eingeteilt.

Die Teilnehmer des Experimentes hatten in einem früheren Semester einen Kurs zur GPM-Modellierung mit EPKs belegt. Bei dieser Gelegenheit hatten sie auch EPKs zu erstellen, allerdings lediglich auf Papier ohne Nutzung eines EPK-Editors. Den

[2] Ein herzlicher Dank für die Durchführung der *bflow Toolbox*-Integration gebührt den Kollegen Stefan Kühne und Heiko Kern von der Abteilung Betriebliche Informationssysteme der Universität Leipzig.

Selbsteinschätzungen der Studenten konnte entnommen werden, dass der überwiegende Teil der Teilnehmer mit EPK-Modellierung wenig vertraut war.

Nachdem kurz die grundlegenden Eigenschaften von EPK-Modellen (insbesondere die Syntaxanforderungen) wiederholt wurden, erhielten die Studenten eine Fallstudie in deutscher Sprache zu einem vereinfachten Geschäftsprozess aus dem Maschinenbau. Zu dieser Fallstudie war von jedem Studenten innerhalb von 45 Minuten eine EPK mit dem Werkzeug *bflow Toolbox* zu modellieren. Gruppenarbeit war dabei nicht gestattet.

Einer der Gruppen stand *bflow Toolbox* mit im Hintergrund ablaufender Validierung zur Verfügung. Gefundene Probleme wurden durch eine Markierung am Modell kenntlich gemacht. Außerdem gab es in einem speziellen Bildschirmfenster eine textuelle Information zum Hintergrund des Fehlers. Die andere Gruppe modellierte ohne Validierung im Hintergrund, aber ansonsten mit demselben Modellierungswerkzeug.

Alle Teilnehmer nutzten die für die Modellierung zur Verfügung stehenden 45 Minuten aus. Nach Ablauf dieser Zeit wurden die Modelle zur Auswertung gespeichert. Auf Grund eines technischen Problems konnte dabei für jeweils einen Teilnehmer aus jeder Gruppe kein Ergebnis gespeichert werden. Somit standen nur für 13 Studenten auswertbare Ergebnisse zur Verfügung. Die Vergleichbarkeit der Gruppen in Bezug auf die im Fragebogen erhaltenen Antworten blieb dabei jedoch erhalten.

Auf Grund der Einfachheit der Fallstudie gab es keine Modelle, in denen eines der in Kapitel 9 katalogisierten Probleme gefunden wurde. Die von den Studenten erstellten Modelle erhielten jedoch eine nennenswerte Zahl syntaktischer Fehler: In der Gruppe ohne Hintergrundvalidierung gab es 24 Syntaxfehler in 6 Modellen, in der Gruppe mit Hintergrundvalidierung gab es 6 Syntaxfehler in 7 Modellen. Eine genaue Auflistung der in den Modellen gefundenen Fehler zeigt Tab. 11.5 auf Seite 186.

Zwei weitere Arten syntaktischer Fehler wurden nicht gezählt:

Folgten zwei Funktionen im Kontrollfluss direkt aufeinander, so wurde dies nicht als Fehler gewertet. Im vorausgehenden Kurs zur EPK-Modellierung war den Studenten erklärt worden, dass eine solches Weglassen von Ereignissen gestattet ist, wenn die Ereignisse keinen Informationsgehalt haben.

Ebenso nicht als Fehler gewertet wurde, wenn ein Student versehentlich statt des korrekten Pfeils für die Modellierung von Kontrollflüssen den von *bflow Toolbox* ebenfalls angebotenen (visuell leicht anders dargestellten) Pfeil für die Modellierung von Datenflüssen benutzte. Wären die beiden vorstehenden Klassen von Fehlern berücksichtigt worden, so hätten in der Gruppe ohne Validierung 53 Fehler in 6

	ohne Validierung						mit Validierung						
Modell	1	2	3	4	5	6	7	8	9	10	11	12	13
Ereignis oder Funktion hat ≥ 2 Eingangs- oder ≥ 2 Ausgangskanten	0	0	3	1	1	1	0	0	0	0	0	0	2
EPK startet mit Funktion statt mit einem Ereignis	0	0	1	0	0	1	0	0	0	0	0	0	0
EPK endet mit Funktion statt mit einem Ereignis	1	0	2	1	5	2	0	0	0	0	0	0	1
Konnektor hat sowohl ≥ 2 Eingangs- als auch ≥ 2 Ausgangskanten	0	0	0	0	0	0	0	1	0	0	0	0	0
isoliertes Element	0	0	2	0	0	0	1	0	0	0	0	0	0
Ereignis wird von einem (X)OR-Split gefolgt	2	0	0	0	0	1	0	0	0	0	0	0	1
Summe	3	0	8	2	6	5	1	1	0	0	0	0	4

Tabelle 11.5: In den EPKs gefundene syntaktische Fehler

Modellen, in der Gruppe mit Validierung 12 Fehler in 7 Modellen gezählt werden müssen. Während es in den Modellen der Gruppe ohne Validierung im Durchschnitt 4,0 Fehler pro Modell gab, waren es bei den Modellen der Gruppe mit Validierung im Durchschnitt nur 0,86 Fehler. Dies ist ein statistisch signifikantes Ergebnis (überprüft mit dem einseitigen Mann-Whitney U-Test zum Signifikanzniveau 0,025). Dennoch muss das Ergebnis auf Grund der kleinen Gruppengröße mit Vorsicht betrachtet werden.

Trotz der Einschränkung, dass das Experiment mit einer eher kleinen Gruppe von Teilnehmern durchgeführt wurde, zeigt das Resultat ein ermutigendes Ergebnis: Die Nutzung einer im Modellierungswerkzeug laufenden Hintergrundvalidierung kann die Qualität der modellierten GPM erhöhen. Es ist zu vermuten, dass der Qualitätsgewinn bei erfahrenen Modellierern weniger gravierend ausfällt, da die im Experiment beobachteten eher simplen Syntaxfehler dann deutlich seltener vorkommen dürften. Steigen jedoch Größe und Komplexität der Modelle, so dürften die Modellierer auch mehr von einer im Hintergrund ablaufenden Überprüfung auf Kontrollflussfehler im Modell profitieren. Weitere Experimente hierzu sind als weiterführende Arbeiten geplant.

12 Zusammenfassung und Ausblick

12.1 Zusammenfassung

Ausgangspunkt für diese Arbeit war die Überlegung, dass unstrukturierte Modellierung von GPM Fehler verursachen und zur Erstellung schwer verständlicher Modelle führen kann. Nachdem zunächst der Begriff der Unstrukturiertheit für EPKs formal definiert wurde, wurde eine Metrik vorgestellt, die den Grad der Unstrukturiertheit misst. Durch Untersuchung der Rangkorrelation zwischen dieser Metrik und der Soundness-Eigenschaft einer EPK wurde gezeigt, dass Unstrukturiertheit einen entscheidenden (negativen) Einfluss auf die Korrektheit von EPKs (genauer: auf das Vorhandensein von Kontrollflussfehlern in EPKs) hat.

Diese Beobachtung wurde zum Anlass genommen, verschiedene Formen von Unstrukturiertheit systematisch zu untersuchen. Daraus entstand ein Katalog von Mustern, die Probleme in EPKs beschreiben. Weiterhin wurde ein auf logischer Programmierung basierendes Verfahren vorgestellt, mit dem Instanzen der beschriebenen Muster in EPKs gefunden werden können.

Um die Tauglichkeit dieses Verfahrens zu überprüfen, wurden die per Mustersuche erkannten Probleme mit den Ergebnissen der Soundnessanalyse dreier bekannter Werkzeuge verglichen. Es zeigte sich, dass das musterbasierte Verfahren Kontrollflussfehler fast ähnlich zuverlässig erkennt wie diese Werkzeuge, die dynamische Analyseverfahren benutzen. Das vom Autor vorgeschlagene musterbasierte Verfahren weist jedoch deutliche Vorteile in Bezug auf Ausführungsgeschwindigkeit und Speicherverbrauch auf. Für alle Modelle war mit vertretbarem Aufwand an Rechenzeit und Speicher eine Analyse möglich. Dies war bei den zum Vergleich benutzten Werkzeugen nicht der Fall.

12.2 Wesentliche Beiträge der Arbeit

Ein aus theoretischer Sicht interessantes Ergebnis dieser Arbeit ist die in Kapitel 7 vorgestellte UCC-Metrik. Für die untersuchte Auswahl von Modellen war der Rangkorrelationskoeffizient zwischen den Werten dieser Metrik und der Variablen,

welche die Soundness einer EPK beschreibt, deutlich größer als für jede andere in der Literatur vorgestellte GPM-Komplexitätsmetrik. Dieses Ergebnis unterstreicht die Bedeutung guter Strukturierung für die Qualität eines Modells.

Das zentrale Resultat dieser Arbeit ist jedoch der in Kapitel 9 vorgestellte Katalog von Fehlermustern in EPKs sowie das in Kapitel 10 beschriebene Verfahren dafür, Vorkommen solcher Muster zu finden.

Im Vergleich zu bekannten Aufstellungen von Fehlermustern wurde die Menge der Problemmuster deutlich erweitert und die einzelnen Muster möglichst allgemein formuliert. Anders als in den in Abschnitt 8.1 zitierten Arbeiten wurden auch Muster vorgestellt, die zwar nicht zu Kontrollflussfehlern führen, aber auf Modelle hindeuten, deren Modellierung fragwürdig ist (vgl. Muster 6) oder die leichter verständlich modelliert werden können (vgl. Muster 9). Ebenfalls neu ist die Einbeziehung der Beschriftung von Ereignissen in die Korrektheitsuntersuchungen. Dadurch konnten Fehler gefunden werden, die von bisherigen Ansätzen nicht oder nur nach aufwendiger Erstellung einer formalen Domänenontologie erkannt werden.

Mit der Realisierung der Mustersuche als Prolog-Programm wurde ein Verfahren vorgestellt, das Modellierungsprobleme sehr schnell erkennen kann. Dieses Verfahren hat darüber hinaus die positive Eigenschaft, dass nicht nur das Vorhandensein eines Fehlers, sondern dessen *Ursachen* identifiziert werden können. Dadurch wird es möglich, dem Modellierer bereits während der Modellierung eine Rückmeldung über mögliche Verbesserungen im Modell zu geben. Da eigene Regeln ohne größere Mühen zur Regelbasis hinzugefügt werden können, erlaubt der vorgestellte Ansatz weiterhin, die Einhaltung eigener organisationsweiter Modellierungskonventionen zu überprüfen.

Mit dem Prolog-basierten Ansatz lassen sich alle Anforderungen an die Syntax eines EPK-Modells testen. Das ist eine Verbesserung gegenüber den bisher bekannten Verfahren (vgl. [115]). Dank der einfacheren Syntaxtests wird es möglich, auf redundante Informationen im Austauschformat EPML zu verzichten und so das Austauschformat EPML zu verschlanken. Tests zu syntaktischen, semantischen und pragmatischen Fehlern können mit demselben Ansatz (Mustersuche in Prolog) erfolgen, während bisher üblicherweise Syntax und Semantik mit verschiedenen Werkzeugen behandelt werden.

Die an 984 EPKs durchgeführte Suche nach den betrachteten Fehlermustern lieferte als Nebenprodukt Informationen über die Häufigkeit, mit der verschiedene Modellierungsprobleme in der Praxis vorkommen. Ebenso lieferte die mit den Werkzeugen *EPCTools*, *ProM* und *YAWL Editor* durchgeführte Analyse dieser EPKs interessante

Einsichten in die Unterschiede innerhalb der Semantikdefinitionen, auf denen diese Werkzeuge aufbauen [67].

Der in dieser Arbeit vorgestellte Ansatz zum Finden von Modellfehlern beruht auf dem Finden von Fehlermustern im Modell. Das vorgestellte Verfahren zum Finden von Problemen in EPKs ist heuristischer Natur. Anspruch auf Vollständigkeit der gefundenen Fehler wird nicht erhoben.

Damit unterscheidet sich das Verfahren deutlich von bisher veröffentlichten Werkzeugen (z. B. [29, 113, 197, 135]). Diese nehmen (zu einem gewissen Grade willkürlich) eine bestimmte Semantik für ein Modell an. Unter der Annahme dieser Semantik werden dann Verletzungen der Soundness-Eigenschaft garantiert aufgespürt.

Der in dieser Arbeit vorgestellte Ansatz verfolgt dagegen andere Ziele: Es werden mögliche Interpretationen eines Modells (also Vorstellungen des Erstellers bzw. Lesers des Modells über dessen Semantik) in Betracht gezogen. Davon ausgehend wird versucht, dem Modellierer die Teile des Modells zu benennen, die einen Fehler beinhalten oder aber unter Umständen anders modelliert werden sollten, um die Verständlichkeit zu erhöhen.

Ausgehend vom jeweiligen Muster erhält der Modellierer Hinweise auf mögliche Modellverbesserungen.

Wie häufig bei heuristischen Verfahren, werden die Geschwindigkeitsvorteile, die der musterbasierte Ansatz verspricht, durch Verzicht auf 100%ige Genauigkeit erkauft. Es kann weder garantiert werden, dass *jeder* Kontrollflussfehler gefunden wird, noch, dass jeder gemeldete Fehler auch tatsächlich einem Kontrollflussfehler entspricht. Wie in Kapitel 11 gezeigt wurde, ist jedoch die Zahl der nicht bzw. falsch erkannten Probleme sehr gering. Für die 984 untersuchten Modelle lag die Abweichung zwischen den mit Prolog ermittelten Ergebnissen und den Resultaten der exakten Werkzeuge sogar noch unter der Abweichung, die diese Werkzeuge (auf Grund verschiedener Annahmen über die Semantik von EPKs) untereinander aufwiesen.

Mit dem Werkzeug *bflow Toolbox* (`www.bflow.org`) existiert mittlerweile ein EPK-Editor, in dem die wichtigsten der in der Arbeit vorgestellten Regeln implementiert sind.

12.3 Ziele für weiterführende Forschungen

Der vorgestellte Katalog von Mustern beschreibt Modellierungsprobleme für ereignisgesteuerte Prozessketten. Damit ist er prinzipiell auch für andere GPM-Sprachen

wie BPMN oder YAWL anwendbar, da diese Sprachen den Sprachumfang von EPKs umfassen.

Tatsächlich wurden diejenigen Muster aus Kapitel 9, die zur Meldung „Fehler“ führen, bereits erfolgreich als Abfragemuster für die BPMN-Abfragesprache BPMN-Q implementiert und angewendet[1]. Bei einer Analyse von 109 Modellen aus dem öffentlichen Modellspeicher des Modellierungswerkzeugs *Oryx* [33] zeigte sich, dass sich auf diese Weise Kontrollflussfehler in BPMN-Modellen gut lokalisieren lassen. In BPMN und anderen GPM-Modellierungssprachen kommen jedoch weitere Modellierungselemente hinzu, beispielsweise zur Darstellung der Behandlung von Ausnahmen. Daraus resultieren weitere Problemmuster [65, 144], deren Analyse Gegenstand weiterführender Arbeiten sein wird.

Ein weites Feld für zukünftige Forschungen ist auch die Untersuchung der Beschriftungen von Ereignissen und Funktionen. Im Rahmen dieser Arbeit wurde die Erkennung von logisch identischen sowie von sich logisch ausschließenden Ereignissen prototypisch mit einem eher einfachen Algorithmus implementiert. Es ist eine Verbesserung der Fehlererkennung zu erhoffen, wenn stattdessen mächtigere Verfahren (wie in [4], [93] oder [54] beschrieben) verwendet werden.

Awad et al. verwenden in [4] Verfahren aus dem Bereich des Information Retrieval, um ein Ähnlichkeitsmaß zwischen Namen von Aktivitäten innerhalb eines BPMN-Diagramms zu definieren. Dieser Ansatz, der auf die englische Wortschatz-Datenbank *WordNet* [49] zurückgreift, kommt ohne Beschränkung des verwendbaren Wortschatzes aus. Er erzielt gute Ergebnisse beim Erkennen gleicher oder ähnlicher Aktivitäten, auch wenn diese nicht identisch bezeichnet sind.

Eine Kombination des in [4] beschriebenen Verfahrens mit dem von der gleichen Forschungsgruppe beschriebenen Validierungsansatz [3] erlaubt auch eine Überprüfung inhaltlicher Aussagen wie „Es darf kein Konto eröffnet werden, bevor die Identität des Inhabers überprüft wurde“.

Eine Ausweitung des in dieser Arbeit entwickelten musterbasierten Ansatzes auf das Erkennen solcher inhaltlicher Modellierungsprobleme bietet ein reiches Potential für zukünftige Forschungen. Gegenwärtig wird beispielsweise das mustergesteuerte Erkennen von Situationen untersucht, in denen im Modell zwar eine Prüfung modelliert ist, der Ausgang dieser Prüfung jedoch keinen Einfluss auf den weiteren Ablauf hat (wie im Modell von Abb. 12.1, entnommen aus [186]).

Schließlich bietet der vorgestellte Algorithmus zur Mustersuche noch weitere

[1] Herzlicher Dank für die Kooperation bei der Formulierung und Anwendung der BPMN-Q-Abfragen gebührt Ahmed Awad vom Hasso-Plattner-Institut Potsdam.

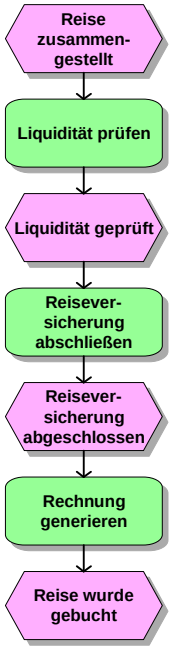


Abbildung 12.1: Es könnte sinnvoll sein, auch den Fall zu modellieren, dass die Liquiditätsprüfung negativ verläuft.

Möglichkeiten, die Ausführungsgeschwindigkeit zu verbessern. Wie in [182, 181] dargestellt, kann es sinnvoll sein, das zu untersuchende Modell zur Analyse in Teilmodelle zu zerlegen. Erfolgt die Validierung während des Erstellens eines Modells im Hintergrund, können Laufzeitverbesserungen auch auf andere Weise erreicht werden: Es spart Rechenzeit, wenn stets nur diejenigen Problemmuster gesucht werden, die durch die im GPM-Editor aktuell ausgeführten Änderungen tatsächlich eintreten können [16, 15].

Literaturverzeichnis

[1] Albrecht, A. ; Gaffney, J. : Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. In: *IEEE Transactions on Software Engineering* 9 (1983), Nr. 6, S. 639–648

[2] Andrews, T. : Business Process Execution Language for Web Services. (2003). `http://www.ibm.com/developerworks/library/specification/ws-bpel/`. – Online; zuletzt abgerufen am 3. Juli 2009

[3] Awad, A. ; Decker, G. ; Weske, M. : Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: *BPM '08: Proceedings of the 6th International Conference on Business Process Management*. Berlin, Heidelberg : Springer, 2008, S. 326–341

[4] Awad, A. ; Polyvyanyy, A. ; Weske, M. : Semantic Querying of Business Process Models. In: *12th International Conference on Enterprise Distributed Object Computing EDOC*, 2008

[5] Bache, R. ; Tinker, R. : A rigorous approach to metrication: a field trial using Kindra. In: *Software Engineering, 1988 Software Engineering 88., Second IEE/BCS Conference*, 1988, S. 28–32

[6] Bache, R. : *Graph Theory Models of Software*, South Bank University, London, Diss., 1990

[7] Backhaus, K. ; Erichson, B. ; Plinke, W. ; Weiber, R. : *Multivariate Analysemethoden: Eine anwendungsorientierte Einführung*. Berlin, Heidelberg, New York : Springer, 2003

[8] Bäckström, F. ; Ivarsson, A. : Verification and Correction of UML Models. (2006). `http://www.objektfabriken.se/mma/mma_rose_productsheet.pdf`. – Online; zuletzt abgerufen am 3. Juli 2009

[9] Barborka, P. ; Helm, L. ; Köldorfer, G. ; Mendling, J. ; Neumann, G. ; Dongen, B. F. ; Verbeek, E. ; van der Aalst, W. M. P.: Integration

of EPC-related Tools with ProM. In: *EPK 2006, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten* Bd. 224 (CEUR Workshop Proceedings), 105-120

[10] Becker, J. ; Delfmann, P. ; Knackstedt, R. : Eine Modellierungstechnik für die konfigurative Referenzmodellierung. In: *Referenzmodellierung 2002 Methoden - Modelle - Erfahrungen*, 2002, S. 35–79

[11] Becker, J. ; Rosemann, M. ; Schütte, R. : Grundsätze ordnungsgemäßer Modellierung. In: *Wirtschaftsinformatik* 37 (1995), Nr. 5, S. 435–445

[12] Becker, J. ; Schütte, R. : *Handelsinformationssysteme*. 2. Auflage. Verlag Moderne Industrie, 2004

[13] Becker, J. ; Vossen, G. : *Geschäftsprozeßmodellierung und Workflows, Modelle, Methoden Werkzeuge*. Bonn : Thomson Verlag, 1995

[14] Berthelot, G. : Checking properties of nets using transformation. In: *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory in Petri Nets-selected papers*. London : Springer, 1986, S. 19–40

[15] Blanc, X. ; Mougenot, A. ; Mounier, I. ; Mens, T. : Incremental Detection of Model Inconsistencies Based on Model Operations. In: *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings* Bd. 5565, Springer, 2009 (Lecture Notes in Computer Science), S. 32–46

[16] Blanc, X. ; Mounier, I. ; Mougenot, A. ; Mens, T. : Detecting model inconsistency through operation-based model construction. In: *ICSE '08: Proceedings of the 30th International Conference on Software engineering*. New York, USA : ACM, 2008, S. 511–520

[17] Blewitt, A. ; Bundy, A. ; Stark, I. : Automatic verification of design patterns in Java. In: *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. New York, USA : ACM, 2005, S. 224–232

[18] Bögl, A. ; Schrefl, M. ; Pomberger, G. ; Weber, N. : Semantic Annotation of EPC Models in Engineering Domains by Employing Semantic Patterns. In: *ICEIS 2008 - Proceedings of the Tenth International Conference*

on Enterprise Information Systems, Volume AIDSS, Barcelona, Spain, 2008, S. 106–115

[19] Börger, E. ; Thalheim, B. : A Method for Verifiable and Validatable Business Process Modeling. In: *Advances in Software Engineering: Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*. Berlin, Heidelberg : Springer, 2008, S. 59–115

[20] Broy, M. ; Crane, M. ; Dingel, J. ; Hartman, A. ; Rumpe, B. ; Selic, B. : 2nd UML 2 Semantics Symposium: Formal Semantics for UML. In: *Models in Software Engineering* (2007), S. 318–323

[21] Bryant, R. E.: Binary decision diagrams and beyond: enabling technologies for formal verification. In: *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*. Washington, USA : IEEE Computer Society, 1995, S. 236–243

[22] Bühl, A. ; Zöfel, P. : *SPSS für Windows Version 6 : Praxisorientierte Einführung in die moderne Datenanalyse*. 1. Auflage. Addison-Wesley, 1994

[23] Burlefinger, S. ; Mayer, I. ; Petersen, L. ; Schweitzer, M. : Maßnahmen und Modelle zur Analyse von Dienstleistungsprozessen / Veröffentlichung Nr. 1 des Arbeitskreises Dienstleistungsmanagement, Lehrstuhl für Industriebetriebslehre und Controlling,Universität des Saarlandes. 2006. – Forschungsbericht

[24] Business Process Management Initiative: Business Process Modeling Notation. (2004). `http://www.bpmi.org`. – Online; zuletzt abgerufen am 3. Juli 2009

[25] Cardoso, J. ; Mendling, J. ; Neumann, G. ; Reijers, H. : A Discourse on Complexity of Process Models. In: *Proceedings of the BPM 2006 Workshops, Workshop on Business Process Design BPI 2006, Vienna, Austria*, 2006

[26] Cardoso, J. : How to Measure the Control-flow Complexity of Web Processes and Workflows. In: *The Workflow Handbook*, 2005, S. 199–212

[27] Chen, R. ; Scheer, A. : Modellierung von Prozessketten mittels Petri-Netz-Theorie. In: *Veröffentlichungen des Instituts für Wirtschaftsinformatik, Universität des Saarlandes* (1994), Nr. 107

[28] Cuntz, N. : *Über die effiziente Simulation von Ereignisgesteuerten Prozessketten*, Universität Paderborn, Diplomarbeit, 2004

[29] Cuntz, N. ; Freiheit, J. ; Kindler, E. : On the Semantics of EPCs: Faster calculation for EPCs with small state spaces. In: *EPK 2005, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2005, S. 7–23

[30] Cuntz, N. ; Kindler, E. : On the semantics of EPCs: Efficient calculation and simulation. In: *EPK 2004: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings*, 2004, S. 7–26

[31] Dadam, P. ; Reichert, M. ; Rinderle, S. ; Jurisch, M. ; Acker, H. ; Göser, K. ; Kreher, U. ; Lauer, M. : Towards Truly Flexible and Adaptive Process-Aware Information Systems. In: *UNISCON* Bd. 5, Springer, 2008 (Lecture Notes in Business Information Processing), S. 72–83

[32] Decker, G. ; Mendling, J. : Instantiation Semantics for Process Models. In: *BPM '08: Proceedings of the 6th International Conference on Business Process Management* Bd. 5240, Springer, 2008 (Lecture Notes in Computer Science), S. 164–179

[33] Decker, G. ; Overdick, H. ; Weske, M. : Oryx — An Open Modeling Platform for the BPM Community. In: *BPM '08: Proceedings of the 6th International Conference on Business Process Management* Bd. 5240, Springer, 2008 (Lecture Notes in Computer Science), S. 382–385

[34] Dehnert, J. ; Rittgen, P. : Relaxed Soundness of Business Processes. In: *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*. London : Springer, 2001, S. 157–170

[35] Dehnert, J. ; Zimmermann, A. : On the Suitability of Correctness Criteria for Business Process Models. In: *Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings* Bd. 3649, Springer, 2005 (Lecture Notes in Computer Science), S. 386–391

[36] Dijkman, R. M. ; Dumas, M. ; Ouyang, C. : Semantics and analysis of business process models in BPMN. In: *Inf. Softw. Technol.* 50 (2008), Nr. 12, S. 1281–1294

[37] Dongen, B. van ; Mendling, J. ; van der Aalst, W. M. P.: Structural Patterns for Soundness of Business Process Models. In: *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*. Los Alamitos, USA : IEEE Computer Society, 2006, S. 116–128

[38] Dufourd, C. ; Finkel, A. ; Schnoebelen, P. : Reset Nets Between Decidability and Undecidability. In: *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*. London : Springer, 1998, S. 103–115

[39] Dumas, M. ; Grosskopf, A. ; Hettel, T. ; Wynn, M. T.: Semantics of Standard Process Models with OR-Joins. In: *OTM Conferences (1)* Bd. 4803, Springer, 2007 (Lecture Notes in Computer Science), S. 41–58

[40] Egan, D. E. ; Schwartz, B. J.: Chunking in recall of symbolic drawings. In: *Memory & Cognition* 7 (1979), Nr. 2, S. 149–158

[41] Egyed, A. : Instant consistency checking for the UML. In: *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, USA : ACM, 2006, S. 381–390

[42] Ehrig, H. : Introduction to the Algebraic Theory of Graph Grammars (A Survey). In: *Proceedings of the International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*. London : Springer, 1979, S. 1–69

[43] Eichberg, M. ; Kloppenburg, S. ; Klose, K. ; Mezini, M. : Defining and Continuous Checking of Structural Program Dependencies. In: *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*. New York, USA : ACM, 2008, S. 391–400

[44] El Kharbili, M. ; Medeiros, A. K. d. ; Stein, S. ; Aalst, W. M. P. d.: Business Process Compliance Checking: Current State and Future Challenges. In: *Modellierung betrieblicher Informationssysteme (MobIS)* Bd. 141 (Lecture Notes in Informatics), S. 107–113

[45] Eshuis, R. : *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*, University of Twente, Enschede, Diss., 2002

[46] Esparza, J. : Reduction and synthesis of live and bounded free choice Petri nets. In: *Inf. Comput.* 114 (1994), Nr. 1, S. 50–87

[47] Esparza, J. ; Silva, M. : Circuits, handles, bridges and nets. In: *Applications and Theory of Petri Nets*, 1989, S. 210–242

[48] Esswein, W. ; Gehlert, A. ; Seiffert, G. : Towards a Framework for Model Migration. In: *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings* Bd. 3084, Springer, 2004 (Lecture Notes in Computer Science), S. 463–476

[49] Fellbaum, C. (Hrsg.): *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, 1998

[50] Fillies, C. ; Weichhardt, F. : Towards the Corporate Semantic Process Web. In: *Berliner XML Tage*, 2003, S. 78–90

[51] Fillies, C. ; Weichhardt, F. : On Ontology-based Event-driven Process Chains. In: *EPK 2005, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2005

[52] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J. : Design Patterns: Abstraction and Reuse of Object-Oriented Design. In: *ECOOP '93: Proceedings of the 7th European Conference on Object-Oriented Programming* Bd. 707, Springer, 1993 (Lecture Notes in Computer Science), S. 406–431

[53] Gausmeier, J. ; Fahrwinkel, U. : Strategiekonforme Geschäftsprozesse und CIM-Maßnahmen. In: *CIM-Management* 10 (1994), Nr. 6, S. 56–61

[54] Gervasi, V. ; Zowghi, D. : Reasoning about inconsistencies in natural language requirements. In: *ACM Trans. Softw. Eng. Methodol.* 14 (2005), Nr. 3, S. 277–330

[55] Gibson, V. R. ; Senn, J. A.: System structure and software maintenance performance. In: *Commun. ACM* 32 (1989), Nr. 3, S. 347–358

[56] Giesa, F. ; Kopfer, H. : Management logistischer Dienstleistungen der Kontraktlogistik. In: *Logistik Management* 2 (2000), Nr. 1, S. 43–53

[57] Grady, R. B.: Successfully applying software metrics. In: *Computer* 27 (1994), Nr. 9, S. 18–25

[58] Grohmann, G. ; Kraemer, W. ; Milius, F. ; Zimmermann, V. : Modellbasiertes Curriculum-Design für Learning Management Systeme: Ein Integra-

tionsansatz auf Basis von ARIS und IMS Learning Design. In: *Wirtschaftsinformatik*, Universitätsverlag Karlsruhe, S. 795–812

[59] Gruhn, V. ; Laue, R. : Using Timed Model Checking for Verifying Workflows. In: *Computer Supported Activity Coordination*, INSTICC Press, 2005, S. 75–88

[60] Gruhn, V. ; Laue, R. : Adopting the Cognitive Complexity Measure for Business Process Models. In: *Fifth IEEE International Conference on Cognitive Informatics, Beijing, China*, IEEE Computer Society, 2006, S. 236–241

[61] Gruhn, V. ; Laue, R. : Complexity Metrics for Business Process Models. In: *9th International Conference on Business Information Systems (BIS 2006), Klagenfurt, Austria*, Springer, 2006, S. 1–12

[62] Gruhn, V. ; Laue, R. : Komplexitätsmetriken für Geschäftsprozessmodelle. In: *Modellierung 2006, 22.-24. März 2006, Innsbruck, Tirol, Austria, Proceedings* Bd. 82, 2006 (Lecture Notes in Informatics), S. 289–292

[63] Gruhn, V. ; Laue, R. : Validierung syntaktischer und anderer EPK-Eigenschaften mit PROLOG. In: *EPK 2006, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 5. Workshop der Gesellschaft für Informatik e. V.*, 2006, S. 69–84

[64] Gruhn, V. ; Laue, R. : Forderungen an hierarchische EPK-Schemata. In: *EPK 2007, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2007, S. 59–76

[65] Gruhn, V. ; Laue, R. : Good and Bad Excuses for Unstructured Business Process Models. In: *Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007)*, 2007

[66] Gruhn, V. ; Laue, R. : On Experiments for Measuring Cognitive Weights for Software Control Structures. In: *IEEE International Conference on Cognitive Informatics*. Los Alamitos, USA : IEEE Computer Society, 2007, S. 116–119

[67] Gruhn, V. ; Laue, R. : A Comparison of Soundness Results Obtained by Different Approaches. In: *1st International Workshop on Empirical Research in Business Process Management*, 2009

[68] Gruhn, V. ; Laue, R. ; Meyer, F. : Berechnung von Komplexitätsmetriken für ereignisgesteuerte Prozessketten. In: *EPK 2006, Geschäftsprozessmanage-*

ment mit Ereignisgesteuerten Prozessketten Bd. 224, 2006 (CEUR Workshop Proceedings), S. 189–202

[69] Gschwind, T. ; Koehler, J. ; Wong, J. : Applying Patterns during Business Process Modeling. In: *6th Int. Conference on Business Process Management (BPM)*, Springer, 2008 (Lecture Notes in Computer Science 5240), S. 4–19

[70] Gustafsson, J. : Metrics calculation in MAISA. 2000. – Forschungsbericht

[71] Hammer, M. ; Champy, J. : *Business Reengineering.* Campus, 1995

[72] Holl, A. ; Valentin, G. : Structured Business Process Modeling (SBPM). In: *Information Systems Research in Scandinavia (IRIS 27) (CD-ROM)*, 2004

[73] Hosmer, D. ; Lemeshow, S. : *Applied Logistic Regression.* Wiley-Interscience, 1989

[74] Hübscher, H. : *IT- Handbuch IT-Systemelektroniker/-in sowie Fachinformatiker/-in.* Westermann Verlag, 2007

[75] International Organization for Standardization: *ISO 9001.2000, Quality Management Systems - Requirements for quality assurance, ISO/TC 176/SC 2/WG 18/N 52.* 2000

[76] Janssen, W. ; Mateescu, R. ; Mauw, S. ; Springintveld, J. : Verifying business processes using SPIN. In: *Proceedings of the 4th International SPIN Workshop, Paris, France, Nov. 1998*, 1998, S. 21–36

[77] Janssen, W. ; Mateescu, R. ; Mauw, S. ; Fennema, P. ; van der Stappen, P. : Model Checking for Managers. In: *5th and 6th International SPIN Workshops*, 1999, S. 92–107

[78] Jelliffe, R. : The Schematron Assertion Language 1.5. (2002), October. `http://www.ascc.net/xml/resource/schematron/Schematron2000.html`. – Online; zuletzt abgerufen am 3. Juli 2009

[79] Johnson, R. ; Pearson, D. ; Pingali, K. : The program structure tree: computing control regions in linear time. In: *PLDI '94: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation.* New York, USA : ACM, 1994, S. 171–185

[80] Jones, C. B.: *Systematic software development using VDM.* Hertfordshire : Prentice Hall International Ltd., 1986

[81] Juan, E. Y. T. ; Tsai, J. J. P. ; Murata, T. : Compositional verification of concurrent systems using Petri-net-based condensation rules. In: *ACM Trans. Program. Lang. Syst.* 20 (1998), Nr. 5, S. 917–979

[82] Jungo, D. ; Buchmann, D. ; Nitsche, U. U.: Testing of semantic properties in XML documents. In: *Proceedings of the 4th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, Paphos, Cyprus*, 2006

[83] Kawalek, P. ; Kueng, P. : The Usefulness of Process Models: A Lifecycle Description of how Process Models are used in Modern Organisations. In: *Proceedings IFIP WG8.1 Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD)*, IEEE Computer Society Press, 1997

[84] Keller, G. ; Nüttgens, M. ; Scheer, A. : Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). In: *Veröffentlichungen des Instituts für Wirtschaftsinformatik, Universität des Saarlandes* (1992), Nr. 89

[85] Keller, G. : *SAP R/3 prozessorientiert anwenden.* Addison-Wesley, München, 1999

[86] Kern, H. ; Kühne, S. : Verarbeitung von ARIS-EPK-Modellen im Eclipse Modeling Framework. In: *EPK 2007, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2007, S. 97–109

[87] Kindler, E. : On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: *Business Process Management: Second International Conference, BPM 2004, Potsdam, Germany, June 17-18, 2004. Proceedings* Bd. 3080, Springer, 2004 (Lecture Notes in Computer Science), S. 82–97

[88] Koehler, J. ; Tirenni, G. ; Kumaran, S. : From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In: *EDOC '02: Proceedings of the Sixth International Enterprise Distributed Object Computing Conference*, 2002, S. 96

[89] Koehler, J. ; Vanhatalo, J. : Process anti-patterns: How to avoid the common traps of business process modeling, Part 1 - Modelling control flow. In: *IBM WebSphere Developer Technical Journal* (2007), April, Nr. 10.4.

[90] König, M. : Workflow-Management in der Baupraxis. In: *4. Tag des Baubetriebs 2004 - Tagungsbeiträge Nachtragsmanagement in Praxis und Forschung, Schriften der Professur Baubetrieb und Bauverfahren*, Bauhaus-Universität Weimar, 2006

[91] Korherr, B. ; List, B. : A UML 2 Profile for Event Driven Process Chains. In: *Proceedings of the 1st IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006)*, Springer, 2006

[92] Korson, T. D. ; Vaishnavi, V. K.: An empirical study of the effects of modularity on program modifiability. In: *Papers presented at the first workshop on Empirical Studies of Programmers*. Norwood, USA : Ablex Publishing Corp., 1986, S. 168–186

[93] Koschmider, A. ; Oberweis, A. : How to detect semantic business process model variants? In: *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. New York, USA : ACM, 2007, S. 1263–1264

[94] Kühne, S. ; Kern, H. ; Gruhn, V. ; Laue, R. : Business Process Modelling with Continuous Validation. In: *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 2008, Revised Papers* Bd. 17, Springer, 2008 (Lecture Notes in Business Information Processing), S. 212–223

[95] Langner, P. ; Schneider, C. ; Wehler, J. : Ereignisgesteuerte Prozessketten und Petri-Netze. In: *Berichte des Fachbereichs Informatik der Universität Hamburg* (1997), Nr. 106

[96] Langner, P. ; Schneider, C. ; Wehler, J. : Prozessmodellierung mit ereignisgesteuerten Prozessketten (EPKs) und Petri-Netzen. In: *Wirtschaftsinformatik* 39 (1997), Nr. 5, S. 479–489

[97] Langner, P. ; Schneider, C. ; Wehler, J. : Relating Event-driven Process Chains to Boolean Petri Nets. München : Institut für Informatik, Ludwig-Maximilians-Universität, Dez. 1997 (9707). – Forschungsbericht

[98] Laue, R. ; Mendling, J. : The Impact of Structuredness on Error Probability of Process Models. In: *Information Systems and e-Business Technologies, 2nd International United Information Systems Conference, UNISCON 2008, Klagenfurt, Austria, April 22-25, 2008, Proceedings* Bd. 5, Springer, 2008 (Lecture Notes in Business Information Processing), S. 585–590

[99] Leelaprute, P. ; Kikuno, T. ; Nakamura, M. ; Matsumoto, K. : Definition and Detection of Semantic Warnings for VoiceXML. In: *IASTED Conf. on Software Engineering*, IASTED/ACTA Press, 267-275

[100] Lin, H. ; Zhao, Z. ; Li, H. ; Chen, Z. : A Novel Graph Reduction Algorithm to Identify Structural Conflicts. In: *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9.* Washington, USA : IEEE Computer Society, 2002, S. 289

[101] Lindland, O. I. ; Sindre, G. ; Sølvberg, A. : Understanding Quality in Conceptual Modeling. In: *IEEE Softw.* 11 (1994), Nr. 2, S. 42–49

[102] Lindsay, A. ; Downs, D. ; Lunn, K. : Business processes–attempts to find a definition. In: *Information & Software Technology* 45 (2003), Nr. 15, S. 1015–1019

[103] Liu, R. ; Kumar, A. : An Analysis and Taxonomy of Unstructured Workflows. In: *Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings* Bd. 3649, Springer, 2005 (Lecture Notes in Computer Science), S. 268–284

[104] Martens, A. : On Compatibility of Web Services. In: *Petri Net Newsletter* 65 (2003), S. 12–20

[105] Mathias, K. S. ; Cross, J. H. II ; Hendrix, T. D. ; Barowski, L. A.: The role of software measures and metrics in studies of program comprehension. In: *ACM-SE 37: Proceedings of the 37th annual Southeast Regional Conference (CD-ROM).* New York, USA : ACM, 1999

[106] Matousek, P. : *Verification of Business Process Models*, Technical University of Ostrava, Diss., 2003

[107] McCabe, T. J.: A Complexity Measure. In: *IEEE Trans. Software Eng.* 2 (1976), Nr. 4, S. 308–320

[108] Melcher, J. ; Seese, D. : Process Measurement: Insights from Software Measurement on Measuring Process Complexity, Quality and Performance / Universität Karlsruhe (TH), Institut für Angewandte Informatik und Formale Beschreibungsverfahren. 2008. – Forschungsbericht

[109] Melcher, J. ; Seese, D. : Towards Validating Prediction Systems for Process Understandability: Measuring Process Understandability. In: *SYNASC 2008, 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 26-29 September 2008*, IEEE Computer Society, 2008. – ISBN 978–0–7695–3523–4, S. 564–571

[110] Mendling, J. ; Moser, M. ; Neumann, G. ; Verbeek, E. ; van Dongen, B. F. ; van der Aalst, W. M. P.: A Quantitative Analysis of Faulty EPCs in the SAP Reference Model / BPM Center Report, BPMcenter.org. 2006 (BPM-06-08). – Forschungsbericht

[111] Mendling, J. ; Nüttgens, M. : Exchanging EPC Business Process Models with EPML. In: *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004*, 2004, S. 61–80

[112] Mendling, J. : Testing Density as a Complexity Metric for EPCs / Vienna University of Economics and Business Administration (JM-2006-11-15). – Forschungsbericht

[113] Mendling, J. : *Detection and Prediction of Errors in EPC Business Process Models*, Wirtschaftsuniversität Wien, Diss., 2007

[114] Mendling, J. ; Nüttgens, M. : EPC Modelling based on Implicit Arc Types. In: *Information Systems Technology and its Applications, International Conference ISTA'2003, June 19-21, 2003, Kharkiv, Ukraine, Proceedings* Bd. 30, 2003 (Lecture Notes in Informatics), S. 131–142

[115] Mendling, J. ; Nüttgens, M. : EPC Syntax Validation with XML Schema Languages. In: *EPK 2003, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2003, S. 19–30

[116] Mendling, J. ; Reijers, H. A. ; Cardoso, J. : What Makes Process Models Understandable? In: *Business Process Management, 5th International Confe-*

rence, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings Bd. 4714, Springer, 2007 (Lecture Notes in Computer Science), S. 48–63

[117] Mendling, J. ; Reijers, H. A. ; van der Aalst, W. M. P.: Seven Process Modeling Guidelines (7PMG) / Queensland University of Technology. 2008 (QUT ePrints, Report 12340). – Forschungsbericht

[118] Mendling, J. ; Strembeck, M. : Influence Factors of Understanding Business Process Models. In: *Business Information Systems, 11th International Conference, BIS 2008, Innsbruck, Austria, May 2008*, Springer, 2008, S. 142–153

[119] Mendling, J. ; van der Aalst, W. M. P.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In: *Proc. of the the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, 2007

[120] Mens, K. ; Mens, T. ; Wermelinger, M. : Supporting Software Evolution with Intentional Software Views. In: *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution*. New York, USA : ACM, 2002, S. 138–142

[121] Meyer, F. : *Untersuchung von Komplexitätsmetriken graphischer Geschäftsprozessmodelle*, Universität Leipzig, Diplomarbeit, 2006

[122] Moody, D. L.: Cognitive Load Effects on End User Understanding of Conceptual Models: An Experimental Analysis. In: *Proceedings of 8th East European Conference on Advances in Databases and Information Systems, ADBIS 2004*, Springer, 2004 (Lecture Notes in Computer Science 3255), S. 129–143

[123] Nüttgens, M. ; Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, 2002, S. 64–77

[124] Oanea, O. : *Verification of soundness and other properties of business processes*. Eindhoven, The Netherlands, Department of Mathematics and Computer Science, Eindhoven University of Technology, Diss., Dez. 2007

[125] Object Management Group, I. : Unified Modeling Language (UML) 2.0 Superstructure Specification. (2003). http://www.omg.org/docs/ptc/03-08-02.pdf. – Online; zuletzt abgerufen am 3. Juli 2009

[126] Onoda, S. ; Ikkai, Y. ; Kobayashi, T. ; Komoda, N. : Definition of Deadlock Patterns for Business Processes Workflow Models. In: *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences-Volume 5*, IEEE Computer Society, 1999, S. 5065

[127] Opočenská, K. ; Kopecký, M. : Incox - A language for XML Integrity Constraints Description. In: *DATESO 2008* Bd. 330, 2008 (CEUR Workshop Proceedings), S. 1–12

[128] Ouyang, C. ; Dumas, M. ; Breutel, S. ; Hofstede, A. H. M.: Translating Standard Process Models to BPEL. In: *Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006, Proceedings* Bd. 4001, Springer, 2006 (Lecture Notes in Computer Science), S. 417–432

[129] Ouyang, C. ; Dumas, M. ; van der Aalst, W. M. P. ; ter Hofstede, A. H.: From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way / BPM Center Report, BPMcenter.org. 2006 (BPM-06-27). – Forschungsbericht

[130] Paakki, J. ; Karhinen, A. ; Gustafsson, J. ; Nenonen, L. ; Verkamo, A. I.: Software Metrics by Architectural Pattern Mining. In: *Proc. International Conference on Software: Theory and Practice*, 2000, S. 325–332

[131] Pfeiffer, D. ; Niehaves, B. : Evaluation of Conceptual Models - A Structuralist Approach. In: *Proceedings of the 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy, ECIS 2005, Regensburg, Germany, May 26-28, 2005*, 2005

[132] Piwowarski, P. : A nesting level complexity measure. In: *SIGPLAN Not.* 17 (1982), Nr. 9, S. 44–50

[133] Puhlmann, F. : Soundness Verification of Business Processes Specified in the Pi-Calculus. In: *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November*

25-30, 2007, Proceedings, Part I Bd. 4803, Springer, 2007 (Lecture Notes in Computer Science), S. 6–23

[134] Puhlmann, F. ; Weske, M. : Investigations on Soundness Regarding Lazy Activities. In: *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings* Bd. 4102, Springer, 2006 (Lecture Notes in Computer Science), S. 145–160

[135] Raedts, I. ; Petkovic, M. ; Usenko, Y. S. ; van der Werf, J. M. E. M. ; Groote, J. F. ; Somers, L. J.: Transformation of BPMN Models for Behaviour Analysis. In: *Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS) 2007*, INSTICC PRESS, 126-137

[136] Recker, J. ; Mendling, J. : On the Translation between BPMN and BPEL. In: *Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'06), held in conjunctiun with the 18th Conference on Advanced Information Systems (CAiSE'06), Luxembourg, Luxembourg, EU*, Namur University Press, Namur, Belgium, EU, 2006, S. 521–532

[137] Reichert, M. ; Dadam, P. : ADEPTflex -Supporting Dynamic Changes of Workflows Without Losing Control. In: *Journal of Intelligent Information Systems* 10 (1998), Nr. 2, S. 93–129

[138] Reis, S. ; Metzger, A. ; Pohl, K. : Integration Testing in Software Product Line Engineering: A Model-Based Technique. In: *Fundamental Approaches to Software Engineering, 10th International Conference, FASE 2007, Held as Part of the Joint European Conferences, on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings* Bd. 4422, Springer, 2007 (Lecture Notes in Computer Science), S. 321–335

[139] Rittgen, P. : Quo vadis EPK in ARIS? In: *Wirtschaftsinformatik* 42 (2000), Nr. 1, S. 27–35

[140] Robbins, J. E. ; Redmiles, D. F.: Cognitive support, UML adherence, and XMI interchange in Argo/UML. In: *Information & Software Technology* 42 (2000), Nr. 2, S. 79–89

[141] Roberts, E. S.: Loop exits and structured programming: reopening the debate. In: *SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education.* New York, USA : ACM Press, 1995, S. 268–272

[142] Rombach, H. D.: A Controlled Experiment on the Impact of Software Structure on Maintainability. In: *IEEE Trans. Softw. Eng.* 13 (1987), Nr. 3, S. 344–354

[143] Rosemann, M. : *Komplexitätsmanagement in Prozeßmodellen*, Universität Münster, Diss., 1996

[144] Rozman, T. ; Polancic, G. ; Horvat, R. V.: Analysis of Most Common Process Modeling Mistakes in BPMN Process Models. In: *2008 BPM and Workflow Handbook*, 2008

[145] Rump, F. J.: *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten.* B. G. Teubner Verlag Stuttgart Leipzig, 1999

[146] Rumpe, B. : A Note on Semantics (with an Emphasis on UML). In: *Second ECOOP Workshop on Precise Behavioral Semantics*, Technische Universität München, TUM-I9813, 1998

[147] Sadiq, W. ; Orlowska, M. E.: Analyzing process models using graph reduction techniques. In: *Information Systems* 25(2) (2000), S. 117–134

[148] Saff, D. ; Ernst, M. D.: Continuous testing in Eclipse. In: *ICSE '05: Proceedings of the 27th international Conference on Software engineering.* New York, USA : ACM, 2005, S. 668–669

[149] Sarshar, K. ; Loos, P. : Comparing the Control-Flow of EPC and Petri Net from the End-User Perspective. In: *Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings* Bd. 3649, Springer, 2005 (Lecture Notes in Computer Science), S. 434–439

[150] Schroeder, A. : Integrated program measurement and documentation tools. In: *ICSE '84: Proceedings of the 7th International Conference on Software engineering.* Piscataway, USA : IEEE Press, 1984, S. 304–313

[151] Seidlmeier, H. : *Prozessmodellierung mit ARIS : eine beispielorientierte Einführung für Studium und Praxis.* 2. Auflage. Vieweg-Verlag, 2006

[152] Shao, J. ; Wang, Y. : A New Measure of Software Complexity based on Cognitive Weights. In: *IEEE Canadian Journal of Electrical and Computer Engineering* 28 (2003), Nr. 2, S. 69–74

[153] Smith, G. : Improving Process Model Quality to Drive BPM Project Success. (2008). http://www.bpm.com/improving-process-model-quality-to-drive-bpmproject-success.html. – Online; zuletzt abgerufen am 3. Juli 2009

[154] Spivey, J. M.: *The Z notation: a reference manual.* Upper Saddle River, USA : Prentice-Hall, Inc., 1989

[155] Stahlknecht, P. ; Hasenkamp, U. : *Einführung in die Wirtschaftsinformatik.* 10. Auflage. Springer, 2002

[156] Stein, S. ; Ivanov, K. : EPK nach BPEL Transformation als Voraussetzung für praktische Umsetzung einer SOA. In: *Software Engineering* Bd. 105 (Lecture Notes in Informatics), 75-82

[157] Störrle, H. : Semantics of UML 2.0 Activities. In: *Symposium on Visual Languages - Human Centric Computing (VL/HCC'04, Proceedings)*, IEEE, 2004, S. 235–242

[158] Störrle, H. : A PROLOG-based Approach to Representing and Querying Software Engineering Models. In: *VLL* Bd. 274 (CEUR Workshop Proceedings), 71-83

[159] Sweller, J. : Cognitive load theory, learning difficulty, and instructional design. In: *Learning and Instruction* 4 (1994), Nr. 4, S. 295 – 312

[160] Takata, Y. ; Nakamura, T. ; Seki, H. : Accessibility Verification of WWW Documents by an Automatic Guideline Verification Tool. In: *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4.* Washington, USA : IEEE Computer Society, 2004, S. 40098.1

[161] Taubenberger, S. ; Jürjens, J. : IT Security Risk Analysis based on Business Process Models enhanced with Security Requirements. In: *Modeling Security Workshop (ModSec @ MoDELS 2008)*, 2008

[162] Thomas, O. ; Fellmann, M. : Semantic EPC: Enhancing Process Modeling Using Ontology Languages. In: *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, June 7, 2007* Bd. 251, 2007 (CEUR Workshop Proceedings)

[163] Thomas, O. ; Seel, C. ; Seel, C. ; Hermes, B. ; Martin, G. ; Nüttgens, M. ; Rump, F. : EPK-Referenzmodelle für Verwaltungsverfahren. In: *Proceedings des 3. GI-Workshop EPK 2004 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2004

[164] Trcka, N. ; Aalst, W. M. P. d. ; Sidorova, N. : Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows. In: *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings* Bd. 5565, Springer, 2009 (Lecture Notes in Computer Science), S. 425–439

[165] Valette, R. : Analysis of Petri nets by stepwise refinements. In: *Journal of Computer and System Sciences* 18 (1979), Nr. 1, S. 35–46

[166] van der Aalst, W. M. P.: Structural Characterizations of Sound Workflow Nets. In: *Computing Science Reports/23* (1996), Nr. 96

[167] van der Aalst, W. M. P.: Verification of Workflow Nets. In: *Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings*, 1997, S. 407–426

[168] van der Aalst, W. M. P.: The Application of Petri Nets to Workflow Management. In: *The Journal of Circuits, Systems and Computers* 8 (1998), Nr. 1, S. 21–66

[169] van der Aalst, W. M. P.: Formalization and verification of event-driven process chains. In: *Information & Software Technology* 41 (1999), Nr. 10, S. 639–650

[170] van der Aalst, W. M. P.: Woflan: a Petri-net-based workflow analyzer. In: *Syst. Anal. Model. Simul.* 35 (1999), Nr. 3, S. 345–357

[171] van der Aalst, W. M. P.: Challenges in Business Process Management: Verification of Business Processing Using Petri Nets. In: *Bulletin of the EATCS* 80 (2003), S. 174–199

[172] van der Aalst, W. M. P. ; Desel, J. ; Kindler, E. : On the semantics of EPCs: A vicious circle. In: *EPK 2004, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, 2002, S. 71–79

[173] van der Aalst, W. M. P. ; Hirnschall, A. ; Verbeek, E. : An Alternative Way to Analyze Workflow Graphs. In: *CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering*. London : Springer, 2002, S. 535–552

[174] van der Aalst, W. M. P. ; Lassen, K. B.: Translating unstructured workflow processes to readable BPEL: Theory and implementation. In: *Inf. Softw. Technol.* 50 (2006), Nr. 3, S. 131–159

[175] van der Aalst, W. M. P. ; ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language / Queensland University of Technology, Brisbane. 2002 (FIT-TR-2002-06). – Forschungsbericht

[176] van der Aalst, W. M. P. ; ter Hofstede, A. H. M. ; Kiepuszewski, B. ; Barros, A. : Workflow Patterns. In: *Distributed and Parallel Databases* 14 (2003), Nr. 3

[177] van Dongen, B. F. ; Jansen-Vullers, M. H.: EPC Verification in the ARIS for MySAP reference model database. In: *BETA Working Paper Series, WP 142, Eindhoven University of Technology, Eindhoven*, 2005

[178] van Dongen, B. F. ; Jansen-Vullers, M. ; Verbeek, E. ; van der Aalst, W. M. P.: Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. In: *Comput. Ind.* 58 (2007), Nr. 6, S. 578–601

[179] van Dongen, B. F. ; van der Aalst, W. M. P. ; Verbeek, E. : Verification of EPCs: Using Reduction Rules and Petri Nets. In: *Proceedings of the International Conference on Advanced Information Systems 2005*, 2005, S. 372–386

[180] Vanderfeesten, I. T. P. ; Reijers, H. A. ; Mendling, J. ; van der Aalst, W. M. P. ; Cardoso, J. : On a Quest for Good Process Models: The Cross-Connectivity Metric. In: *Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings* Bd. 5074, Springer, 2008 (Lecture Notes in Computer Science), S. 480–494

[181] Vanhatalo, J. ; Völzer, H. ; Koehler, J. : The Refined Process Structure Tree. In: *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings* Bd. 5240, Springer, 2008 (Lecture Notes in Computer Science), S. 100–115

[182] Vanhatalo, J. ; Völzer, H. ; Leymann, F. : Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: *Service-Oriented Computing – ICSOC 2007* (2007), S. 43–55

[183] Verbeek, E. ; van der Aalst, W. M. P. ; ter Hofstede, A. H. M.: Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Relaxed Soundness and Invariants. In: *Comput. J.* 50 (2007), Nr. 3, S. 294–314

[184] Volanschi, N. : A Portable Compiler-Integrated Approach to Permanent Checking. In: *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*. Washington, USA : IEEE Computer Society, 2006, S. 103–112

[185] vom Brocke, J. : *Referenzmodellierung: Gestaltung und Verteilung von Konstruktionsprozessen*. 10. Auflage. Logos Verlag, 2003

[186] vom Brocke, J. : *Serviceorientierte Architekturen: Management und Controlling von Geschäftsprozessen*. Vahlen, 2008

[187] von der Tann, M. : *Konzeption eines Repositoriums für Fachkomponenten*, Technische Universität Chemnitz, Diplomarbeit, 2003

[188] Warmer, J. ; Kleppe, A. : *The object constraint language: precise modeling with UML*. Addison-Wesley Longman Publishing Co., Inc. Boston, USA, 1998

[189] Wenzel, P. : *Geschäftsprozessoptimierung mit SAP/R3*. Friedr. Vieweg und Sohn, 1995

[190] Weyuker, E. : Evaluating Software Complexity Measures. In: *IEEE Transactions on Software Engineering* 14 (1988), Nr. 9, S. 1357–1365

[191] White, S. A.: Using BPMN to Model a BPEL Process. (2007). `http://www.bpmn.org/Documents/Mapping BPMN to BPEL Example.pdf`. – Online; zuletzt abgerufen am 3. Juli 2009

[192] Wölfle, R. ; (Hrsgb.), P. S.: *Integrierte Geschäftsprozesse mit Business Software*. München Wien : Carl Hanser Verlag, 2005

[193] Wong, P. Y. ; Gibbons, J. : A Process Semantics for BPMN. In: *Proceedings of 10th International Conference on Formal Engineering Methods*. Bd. 5256, Springer, October 2008 (Lecture Notes in Computer Science)

[194] Woodward, M. R. ; Hennell, M. A. ; Hedley, D. : A measure of control-flow complexity in program text. In: *IEEE Transactions on Software Engineering* SE-5 (1979), Nr. 1, S. 45–50

[195] Wynn, M. T.: *Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins*, Queensland University of Technology Brisbane, Australia, Diss., 2006

[196] Wynn, M. T. ; Edmond, D. ; van der Aalst, W. M. P. ; ter Hofstede, A. H. M.: Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In: *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings* Bd. 3536, Springer, 2005 (Lecture Notes in Computer Science), S. 423–443

[197] Wynn, M. T. ; Verbeek, E. ; van der Aalst, W. M. P. ; Edmond, D. : Business Process Verification - Finally a Reality! In: *Business Process Management Journal* 15 (2009), Nr. 1, S. 74–92

[198] Zellner, G. : *Leistungsprozesse im Kundenbeziehungsmanagement*, Universität St. Gallen, Diss., 2003

[199] Ziemann, J. ; Mendling, J. : Transformation of EPCs to BPEL: A pragmatic approach. In: *7th International Conference on the Modern Information Technology in the Innovation Processes of the industrial enterprises Genoa, Italy*, 2005

Wissenschaftlicher Werdegang des Verfassers

Persönliche Angaben:
Ralf Laue, geboren am 25. April 1968 in Leipzig

Schulbildung und Armeedienst:
1974 - 1984 Polytechnische Oberschule
1984 - 1986 Erweiterte Oberschule Max Klinger in Leipzig
1986 - 1988 Grundwehrdienst

Ausbildung und Beruf:
1988 - 1993 Studium der Mathematik an der Universität Leipzig, Spezialisierungsrichtung „Informatik“
Januar 1994 - April 1999 Netzwerk-Systembetreuer im Rechenzentrum Leipzig der Deutschen Rentenversicherung
April 1999-Juli 2003 Systemadministrator bei der Virbus AG Leipzig (seit 2003 Bereichsleiter für den IT-Betrieb)
ab August 2003 wissenschaftlicher Mitarbeiter am Lehrstuhl für Angewandte Telematik und e-Business der Universität Leipzig

Bibliographische Angaben

Laue, Ralf: Musterbasierte Überprüfung der Qualitätseigenschaften von Geschäftsprozessmodellen. Dissertation, Universität Leipzig, 2009

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 6. Juli 2009

Ralf Laue