Semantikprüfung von Geschäftsprozessmodellen

Studienarbeit

für die

Prüfung zum Diplom-Wirtschaftsinformatiker (BA)

im

Ausbildungsbereich Wirtschaft

Fachrichtung: Handel Kurs: WWI04V3

der Berufsakademie Karlsruhe

von

Name:	Eckleder	Vorname: Andreas
Geburtsort:	Altötting	
Ausbildungsstätte:	Nero AG	
Betreuender Dozent:	Prof. Dr. Thomas Freytag	
Abgabedatum:	06. November 2006	

Note: _____

Inhaltsverzeichnis

1 Einleitung

Der Entwurf und die Analyse von Geschäftsprozessen mit Hilfe spezieller Modellierungssprachen und Tools gewinnt zunehmend an Bedeutung. Was in der Softwareentwicklung längst gängige Praxis ist wird auch im Bereich der Geschäftsprozesse immer wichtiger: Die komplette Modellierung von Geschäftsprozessen von einem groben Entwurf hin zu einem durch Syntax- und Semantikprüfung verifizierbaren Ablauf. Diese Arbeit vergleicht die hierfür angebotenen Tools und Modellierungstechniken, ordnet diese ein und versucht nicht zuletzt einen Bezug zu den Techniken der Softwaremodellierung herzustellen. Im weiteren Verlauf schließlich wird aufgezeigt, wie sich durch den Aufbau einer kompletten Kette von Entwicklungstools für Geschäftsprozesse in Form einer grafischen integrierten Entwicklungsumgebung ungenutzte Potentiale bei der Geschäftsprozessmodellierung nutzen lassen. Die im Verlauf dieser Arbeit an dem Petri-Netz Modellierungstool „WoPeD“ vorgenommenen Erweiterungen stellen einen ersten Schritt in diese Richtung dar.

2 Entwurf und Analyse von Geschäftsprozessmodellen

2.1 Modellierung von Geschäftsprozessen

2.1.1 Der Begriff des Modells

Der Begriff Modell beschreibt eine Abbildung eines Teils der Realität. Diese Abbildung beschränkt sich auf die relevanten Aspekte in Bezug auf ein zu analysierendes Thema.

Ein Modell erlaubt es, sich bei der Analyse von reellen Sachverhalten auf die für ein Thema wesentlichen Aspekte zu konzentrieren. So lassen sich oft aus reellen Sachverhalten nur schwer erschliessbare Rückschlüsse ziehen.

Beim Design von Systemen ist es sinnvoll, zunächst die groben Anforderungen zu definieren und das so entstandene Modell immer weiter zu verfeinern. Diesen Vorgang nennt man „top-down“ Design, also den Entwurf eines Systems durch iteratives Verfeinern des Entwurfs seiner Komponenten. Ein geeignetes Modell erleichtert diese Arbeit erheblich, da unnötige Details durch Wahl geeigneter Modellparameter ausgeblendet werden können. Zudem wird eine Aufteilung der Aufgaben auf mehrere Personen möglich. Jede Person kann innerhalb der durch die Modellparameter festgelegten Zuständigkeitsgrenzen selbständig einen Teilaspekt des Systementwurfes bearbeiten.

2.1.2 Formale Geschäftsprozesse

Ein Geschäftsprozess ist ein in betrieblichem Umfeld ablaufender Vorgang, der sich aus verschiedenen Aufgaben oder auch Arbeitspaketen zusammensetzt. Diese Aufgaben können wiederum in Teilaufgaben zerlegt werden um so eine immer genauer werdende Beschreibung eines Prozesses zu erhalten. Geschäftsprozesse werden also durch die ihnen zu Grunde liegenden Aufgaben beschrieben und nicht etwa durch das von ihnen erwartete Ergebnis. Dennoch dienen Geschäftsprozesse stets direkt oder indirekt der Erstellung von Produkten oder Dienstleistungen. Sie werden durch das Auftreten bestimmter Ereignisse (beispielsweise Kundenanfragen) angestoßen und haben ein definiertes Ende (Abschluss- oder Endereignis).

2.1.3 Aspekte der Modellierung von Geschäftsprozessen

Wie in Kapitel 2.1.1 beschrieben stellt ein Modell eine Vereinfachung der Realität dar. Um Erkenntnisse aus einem Geschäftsprozessmodell gewinnen zu können ist es daher wichtig, die für die Aufgabenstellung relevanten Aspekte eines Geschäftsprozesses zu finden und bei der Modellbildung zu berücksichtigen. Geschäftsprozesse können aus vielen verschiedenen Blickwinkeln, oft auch Sichten genannt, betrachtet werden. Soll z.B. ein Datenbank oder Kommunikationsschema erarbeitet werden so wird sicherlich eine modellhafte Darstellung der Datenflüsse und Speicherordnung eine große Rolle spielen. Diesen Aspekt eines Geschäftsprozess bezeichnet man oft als Datensicht. Liegt der Schwerpunkt auf dem Kontext, in dem Geschäftsprozesse ausgelöst werden, so bietet sich dagegen eher die sog. Ereignissicht an. Sie beschreibt die Ereignisse, die bei Ablauf eines Geschäftsprozesses auftreten rund deren Folgen. In der Gliederungssicht schließlich wird versucht, Teilprozesse hierarchisch Überbegriffen zuzuordnen. Nützlich ist diese Darstellung beispielsweise, um ähnliche Teilprozesse erkennen und ggf. Synergieeffekte nutzen zu können. Auch für die Optimierung der Resourcenzuordnung (z.B. Bildung von Abteilungen) kann diese Information nützlich sein.

Neben dieser Unterteilung nach Aspekten ist es auch wichtig, den Detaillierungsgrad festzulegen. So kann ein Teilvorgang zunächst durch Worte beschrieben werden, um ihn dann in einer späteren Phase des Modellentwurfs genauer darzustellen (siehe Abbildung 1).

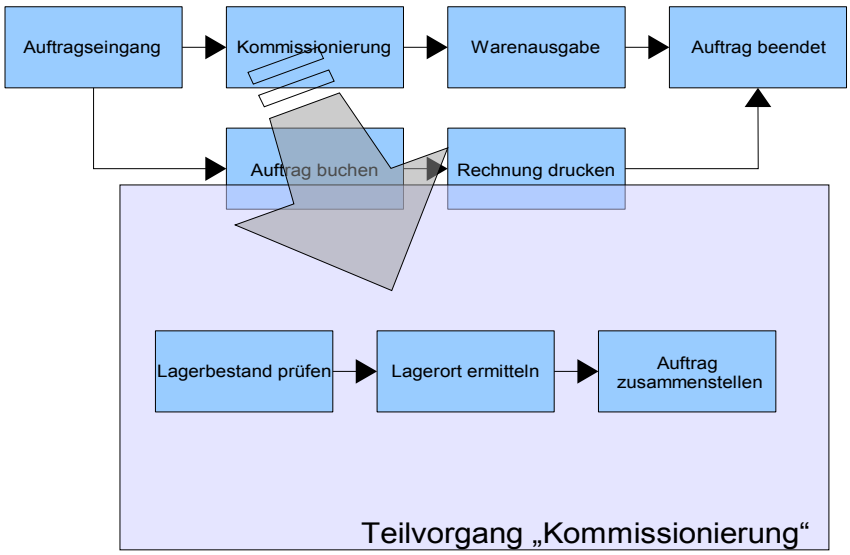

Abbildung 1: Vorgang und Teilvorgang

Soll der Entwurf von mehreren Personen gemeinsam erstellt werden, bietet es sich an, jeder Person einen Teilvorgang zuzuweisen, dessen Systemgrenze durch seine textuelle Beschreibung vorgegeben ist. Eine Person schließlich ist für den Entwurf des Gesamtvorgangs verantwortlich. Für diese Person existieren die Teilvorgänge weiterhin nur als textuelle Beschreibung, sie hat keinen Einfluss auf deren detaillierte Umsetzung.

Die folgenden Ausführungen beziehen sich ausschließlich auf die Ereignissicht. Um den Ablauf einen Geschäftsprozesses modellieren zu können ist es wichtig, mit einem geeigneten Modell alle relevanten, den Geschäftsprozess steuernden Ereignisse darstellen zu können. Eine zentrale Bedeutung kommt dabei einigen wenigen Steuerungselementen zu, welche wiederum kategorisiert werden können:

- Verzweigungen
  - Alternativen/Konflikte: An einem bestimmten Punkt während des Ablaufs eines Geschäftsprozesses wird für die weitere Bearbeitung zwischen zwei Alternativen A und B entschieden, z.B. Bezahlung mit Kredit-Karte oder in Bar
  - Parallelisierungen: Ab einem bestimmten Punkt während des Ablaufs eines Geschäftsprozesses laufen mindestens zwei Vorgänge parallel ab, z.B. Rechnung wird geschrieben während parallel dazu die Waren im Lager kommissioniert werden
- Synchronisation
  - Zusammenführen von Alternativen: Wurde während des Geschäftsprozesses eine Entscheidung zwischen zwei Alternativen getroffen, so ist ein Synchronisationspunkt ein Punkt während der Bearbeitung eines Geschäftsprozesses, ab dem unabhängig von der getroffenen Entscheidung wieder in beiden Fällen gleich weitergearbeitet wird, z.B. nachdem der Kunde sich zwischen Karten- oder Barzahlung entschieden hat, geht er in jedem Fall zur Warenausgabe und holt sich das gekaufte Produkt ab. In diesem Fall ist das Abholen der Ware an der Warenausgabe ein Synchronisationspunkt, da dieser Schritt im Geschäftsprozess stattfindet, egal ob der Kunde in bar oder mit Karte bezahlt hat.
  - Gemeinsames Ergebnis paralleler Vorgänge: Ein Punkt im Ablauf eines Geschäftsprozesses, an dem auf die Beendigung mindestens zweier parallel ablaufender Teilvorgänge gewartet werden muss bevor mit der weiteren Bearbeitung des Geschäftsprozesses fortgefahren werden kann ist ebenfalls ein Synchronisationspunkt. Ab diesem Punkt wird die Abarbeitung des Geschäftsprozesses wieder in einem einzigen Ablauf fortgeführt, z.B. ein Kunde wartet bis die Ware kommissioniert und die Rechnung geschrieben ist, um dann den auf der Rechnung ausgewiesenen Betrag zu bezahlen und die Ware in Empfang zu nehmen.
- Ereignisse
  - Allgemein: Ereignisse sind die Beschreibung von aktuellen Zuständen, in denen sich ein Geschäftsprozess befindet. So ist „Ware erhalten“ ein klarer Zustand, der bei der Modellierung eines Geschäftsprozesses eine oder mehrere Aktionen auslöst.
  - Auslösendes Ereignis: Auch der ursprüngliche Auslöser eines Geschäftsprozesses ist ein Ereignis. Bei der Modellierung eines Geschäftsprozesses ist die Möglichkeit der Beschreibung des den Prozess auslösenden Ereignisses zweifellos von großem Interesse. Es beschreibt die Ausgangsbedingung(en), unter denen der Geschäftsprozess begonnen wird.
  - Ergebnis: Ein Geschäftsprozess arbeitet auf ein gewünschtes Ergebnis, also ein Ziel hin. Das Ziel ist wie die Ausgangssituation ein zentraler Bestandteil der Beschreibung eines Geschäftsprozesses. Denn nur wenn das Ziel klar ist kann der Erfolg eines Geschäftsvorganges bewertet werden.

2.2 Von der Idee zum Geschäftsprozess

2.2.1 Modellbildung

Die Modellierung von Geschäftsprozessen verläuft meist nach der top-down Methode. Als erster Schritt der Modellierung muss zunächst klargestellt werden, welche Ziele (Ergebnisse) mit dem zu entwerfenden Geschäftsprozess erreicht werden sollen. Zudem muss klar sein, welche Ausgangsbedingungen vorliegen. Ein Geschäftsprozess kann hierbei mehrere Ziele gleichzeitig erreichen oder aber auch wahlweise zu einem aus mehreren möglichen Ergebnissen gelangen. In diesem ersten Schritt muss auch überlegt werden, durch welches Ereignis der Geschäftsprozess ausgelöst wird.

Es folgt eine Analyse der zu erledigenden Aufgaben, wobei es sich anbietet, die Abhängigkeiten dieser Aufgaben voneinander festzuhalten. Muss beispielsweise eine Rechnung an einen Kunden verschickt werden, so ist eine der notwendigen Aufgaben sicherlich das Drucken und Verpacken der Rechnung in ein Kuvert sowie dessen Frankierung. Diese Aufgabe kann jedoch erst vorgenommen werden, wenn die Rechnung tatsächlich bereits ausgestellt wurde. Neben der Beschreibung der Aufgaben sollte hier auch bereits eine grobe Festlegung der Ressourcen erfolgen, von denen diese Aufgaben erledigt werden sollen. An Stelle konkreter Ressourcen wird dabei oft mit Ressourcenklassen wie Lager, Versand oder Büro gearbeitet.

Für jede spezifizierte Aufgabe erfolgt nun eine Analyse der zu erledigenden Teilaufgaben. Die Vorgehensweise ist wie bereits beschrieben. Es werden wiederum Teilaufgaben spezifiziert, ihre Abhängigkeiten festgehalten sowie ggf. eine Präzision der Resourcenzuordnung vorgenommen. Der Geschäftsprozess kann durch wiederholte Analyse von Teilaufgaben beliebig verfeinert werden.

Bei der Modellbildung ist zu beachten, dass die sog. ACID Eigenschaften eingehalten werden. ACID ist die Abkürzung für vier Eigenschaften, die ursprünglich für den Umgang mit Datenbanken als wünschenswert erkannt wurden, aber auch bei der korrekten Modellierung von Geschäftsprozessen eine zentrale Bedeutung haben:

- **Atomicity (Atomarität):** Ein Arbeitsschritt im Modell wird entweder komplett ausgeführt oder gar nicht, für die Formulierung von Teilschritten muss der Detailgrad erhöht werden, wobei ein Arbeitsschritt durch mehrere Teilschritte ersetzt wird, die wiederum entweder komplett oder gar nicht ausgeführt werden
- **Consistency (Konsistenz):** Die Durchführung eines Arbeitsschrittes überführt den Geschäftsprozess von einem gültigen Zustand zum nächsten. Es gibt keine „ungültigen“ Zustände, also Zustände, die in der realen Welt nicht vorkommen.
- **Isolation (Isolation):** Parallel ausgeführte Arbeitsschritte beeinflussen sich nicht gegenseitig.
- **Durability (Nachhaltigkeit):** Die Ergebnisse eines einmal durchgeführten Arbeitsschrittes gehen nicht verloren.

2.2.2 Syntaktische Prüfung von Geschäftsprozessen

Merkmale der syntaktischen Analyse

Bei der syntaktischen Analyse eines Geschäftsprozesses werden zunächst dessen Teilabläufe in Beziehung zueinander gebracht. Ein beliebtes Mittel hierzu sind die Darstellung als Graph oder als Tabelle.

Für jedes Darstellungsmittel lassen sich nun Analysemethoden ableiten, die dazu dienen, den Geschäftsprozess auf seine Eigenschaften hin zu untersuchen. Diese Eigenschaften können entweder positiv oder negativ in Bezug auf die in Kapitel 2.1.2 dargestellte Beschreibung eines Geschäftsprozesses sein:

- Hat der Geschäftsprozess einen definierten Anfang, ein definiertes Ende ?
- Stehen alle Teilprozesse in Beziehung zueinander ?
- Ist klar, welches Ereignis diese auslöst ?
- Ist es überhaupt möglich, den kompletten Geschäftsprozess korrekt zu durchlaufen ?

Einschränkungen der syntaktischen Analyse

Das Laufzeitverhalten kann nicht erkannt werden, keine Garantie dafür, dass ein Prozess mit allen Eingangsdaten und allen Situationen zurecht kommt. So mag ein Geschäftsprozess zwar unter bestimmten Bedingungen ausführbar sein, in Anderen aber sind die Eingangsdaten so ungünstig, dass der Prozess z.B. aufgrund gegenseitiger Abhängigkeiten in parallelen Teilvorgängen verklemmt.

Ein Beispiel für eine Verklemmung in einem realen Ablauf ist folgendes Szenario: ein Betrieb erwirbt bei einem Online-Versandhandel einen Gegenstand. Der Workflow des Betriebs sieht vor, die Waren erst nach Eingang im Lager zu bezahlen. Der Workflow des Online-Versandhandels sieht vor, dass Waren erst verschickt werden, wenn die Bezahlung eingegangen ist. Solche Situationen treten nicht nur an den Schnittstellen zwischen Firmen auf sondern oft auch im täglichen Betriebsablauf. Gerade wenn es um das Ausfüllen von Antragsformularen etc. Geht kann es schnell zu Verklemmungen im Ablauf kommen.

2.2.3 Semantische Prüfung von Geschäftsprozessen

Merkmale der semantischen Analyse

Die semantische Analyse untersucht konkrete Abläufe von Geschäftsprozessen, also zustandsbehaftete Instanzen. Nur durch die Analyse aller möglichen Zustände eines Geschäftsprozesses (also der Betrachtung des modellierten Ablaufs unter allen möglichen Umweltbedingungen) kann ausgeschlossen werden, dass ein Geschäftsprozess in endlicher Zeit abgearbeitet werden kann und nicht etwa auf Grund einer Verklemmung nicht weiterbearbeitet wird oder gar für immer weiter läuft ohne jemals zu einem definierten Ende zu gelangen (z.B. ein Auftrag wird dauernd zwischen zwei Abteilungen hin und her geschoben ohne dass sich eine Abteilung wirklich zuständig fühlt). Letztere Situation erkennt man daran, dass sich zwei Zustände gegenseitig bedingen, sie also einen Zyklus bilden.

Gibt es Verklemmungen oder Zyklen in der Abarbeitung, so müssen diese erkannt und als Fehler gemeldet werden.

Schwierigkeiten bei der semantischen Analyse

Die semantische Analyse ist sehr rechenaufwändig, da alle möglichen Zustände eines Geschäftsprozesses daraufhin untersucht werden müssen, ob eine weitere Abarbeitung des Prozesses diesen noch zu seinem definierten Ende bringen kann. Der Rechenaufwand steigt also mit der Anzahl möglicher Zustände eines Geschäftsprozesses. Die Anzahl der möglichen Zustände wiederum ergibt sich aus der Anzahl Variablen, die während der Abarbeitung des Geschäftsprozesses eine Rolle spielen und der Mächtigkeit ihrer jeweiligen Wertebereiche.

Wie viele mögliche Zustände ein Geschäftsprozess annehmen kann ist erkennbar, wenn man sich als Beispiel ein Formular für einen einfachen Bestellvorgang vorstellt:

Bestellschein	
Artikeltyp	{ Zerbrechlich groß, Zerbrechlich klein, Robust groß, Robust klein }
Art der Bezahlung	{ Bar, Kreditkarte, EC-Karte }
Verpackung als Geschenk	Ja / Nein
Versandkosten	{ Deutschland, Europa, Weltweit }
Bestellwert	{ Unter € 20.-, Von € 20.- bis € 30.-, Über € 30.- }
Versicherter Versand	Ja / Nein

Die Anzahl der Möglichkeiten, dieses Formular auszufüllen ergibt sich wie bereits erwähnt aus der Anzahl der Variablen und ihrer Mächtigkeit:

$$|Moeglichkeiten| = \prod_{x=1}^{\text{Anzahl Variablen}} |W(Var_x)|$$

Der Bestellschein aus dem Beispiel enthält folgende Variablen:

- Der „Artikeltyp“: Der Artikeltyp bestimmt die Art der Verpackung, die für den Versand gewählt wird. Es gibt in unserem Beispiel vier Möglichkeiten, die Mächtigkeit des Wertebereichs beträgt also 4.
- Die „Art der Bezahlung“: In unserem Beispiel gibt es drei Arten der Bezahlung, die Mächtigkeit des Wertebereichs beträgt also 3.
- „Verpackung als Geschenk“: Hier kann der Kunde zwischen den Möglichkeiten Ja und Nein wählen, die Mächtigkeit des Wertebereichs beträgt 2.
- „Versandkosten“: Abhängig vom groben Wohnort des Kunden gibt es drei Möglichkeiten, wie hoch die Versandkosten sein können, die Mächtigkeit des Wertebereichs beträgt somit 3.
- „Bestellwert“: Hier gibt es drei Kategorien, nach denen sich z.B. die Übernahme der Versandkosten durch den Händler bestimmt, die Mächtigkeit des Wertebereichs beträgt also 3.
- „Versicherter Versand“: Hier kann der Kunde zwischen den Möglichkeiten Ja und Nein wählen, die Mächtigkeit des Wertebereichs beträgt 2.

Es gibt also 4 * 3 * 2 * 3 * 3 * 2 = 432 Möglichkeiten, das Formular auszufüllen. Für all diese Möglichkeiten muss der Geschäftsprozess (der Bestellvorgang) komplett durchlaufen werden, um Verklemmungen oder Zyklen feststellen zu können.

3 Sprachen und Tools zur Modellierung und Analyse von Geschäftsprozessen

3.1 Ereignisgesteuerte Prozessketten (EPK)

Ereignisgesteuerte Prozessketten sind eine formal ungenaue Variante der in Kapitel 3.2.1 vorgestellen Petri-Netze. Sie wurden 1992 von Prof. A. W. Scheer am Institut für Wirtschaftsinformatik in Saarbrücken entwickelt. Ihre weite Verbreitung in Deutschland verdanken sie dem am gleichen Institut entwickelten ARIS (Architektur integrierter Informationssysteme) Tool zum Entwurf von betrieblichen Informationssystemen sowie der Integration von EPK in SAP R/3.

Durch die formal ungenaue Definition machen viele der bei Petri-Netzen möglichen Analysemethoden für EPK keinen Sinn. Bei einer Top-Down Modellierung stellen sie jedoch ein gutes Werkzeug zum ersten Entwurf von Geschäftsprozessen dar (siehe Kapitel 4.1).

3.1.1 Definition und Syntax von EPK

Ereignisgesteuerte Prozessketten sind gerichtete Graphen, bestehend aus den beiden Knoten Ereignis und Funktion sowie den Verknüpfungsoperatoren XOR, OR und AND. Die Abbildung 2 zeigt die möglichen Knoten eines EPK Diagramms.

Abbildung 2: Knotentypen und Operatoren ereignisgesteuerter Prozessketten

Der Vorbereich eines Knoten bezeichnet im Folgenden die Knoten, die über eine gerichtete Kante zu einem gegebenen Knoten hin verbunden sind. Der Nachbereich eines Knoten bezeichnet die Knoten, die über eine gerichtete Kante vom dem angegebenen Knoten weg verbunden sind.

Für EPK gelten die folgenden bindenden Grundregeln:

- Die Quellen einer EPK sind stets vom Typ Ereignis und werden Startereignis genannt
- Es dürfen nie zwei Ereignisse oder zwei Funktionen unmittelbar oder transitiv über Operatoren aufeinander folgen (siehe Abbildung 3)

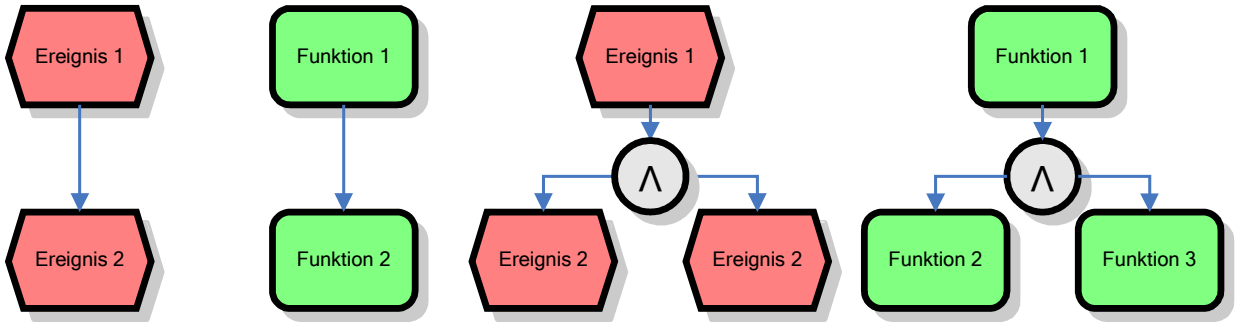

Abbildung 3: Illegale Verknüpfungen von EPK Knoten

- Auf Ereignisse dürfen die Operatoren XOR und OR nicht unmittelbar folgen, da Ereignisse nur beschreiben, nicht aber Entscheidungsfunktionalität besitzen (siehe Abbildung 4)

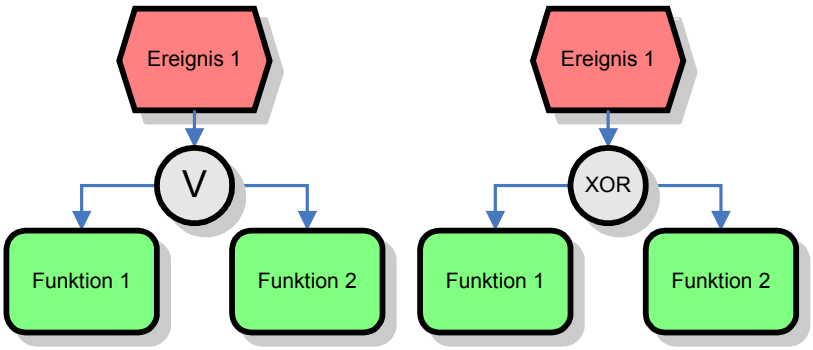

Abbildung 4: Illegale Folgeoperatoren für Ereignisse

- Alle anderen Verknüpfungsoperatoren dürfen sowohl auf Ereignisse als auch auf Funktionen folgen und beschreiben Konflikte, Parallelität sowie Synchronisationspunkte
- Mit Ausnahme des Startereignisse besteht der Vorbereich aller Funktionen und Ereignisse genau aus einem Knoten
- Mit Ausnahme des Endereignisses besteht der Nachbereich aller Funktionen und Ereignisse genau aus einem Knoten
- Der Vorbereich eines rückwärtsverzweigten Operators enthält mindestens zwei Knoten, die auch vom Typ Operator sein dürfen
- Der Nachbereich eines vorwärtsverzweigten Operators enthält mindestens zwei Knoten, die auch vom Typ Operator sein dürfen
- Alle Senken müssen vom Typ Ereignis sein und stellen Endereignisse dar

EPK, die den OR Operator nicht verwenden lassen sich problemlos in Petri-Netze umformulieren. Da der OR Operator beliebige Kombinationen aus seinen möglichen Verzweigungen erlaubt erfordert seine Umsetzung in ein Petri-Netz komplizierte Hilfskonstrukte.

Neben den bindenden Grundregeln existieren für EPK Designrichtlinien, die für eine korrekte Struktur der beschriebenen Geschäftsprozesse sorgen sollen:

- Durch einen vorwärtsverzweigten AND Operator parallelisierte Aktivitäten sollten durch einen rückwärtsverzweigten AND Operator synchronisiert werden
- Durch einen vorärtsverzweigten OR bzw. XOR Operator modellierte Alternativen sollten durch rückwärtsverzweigte OR bzw. XOR Operatoren synchronisiert werden

3.1.2 Tools zur Modellierung von EPK

Das Bekannteste Tool zur Modellierung von EPK ist ARIS (Architektur integrierter Informationssysteme). Neben der Integration in SAP R/3 unterstützen jedoch auch zahlreiche Modellierungstools aus der Softwareentwicklung EPK Diagramme. Unter ihnen sind Microsoft Visio und Eclipse.

Viele dieser Tools unterstützen eine Erweiterung der reinen EPK, die sog. eEPK. Diese lassen zusätzlich Verknüpfungen von Funktionen und Ereignissen zu den drei anderen Sichten des ARIS Tools zu. Eine kurze Darstellung der verschiedenen Sichten in Form des bekannten ARIS-Hauses findet sich in Abbildung 5).

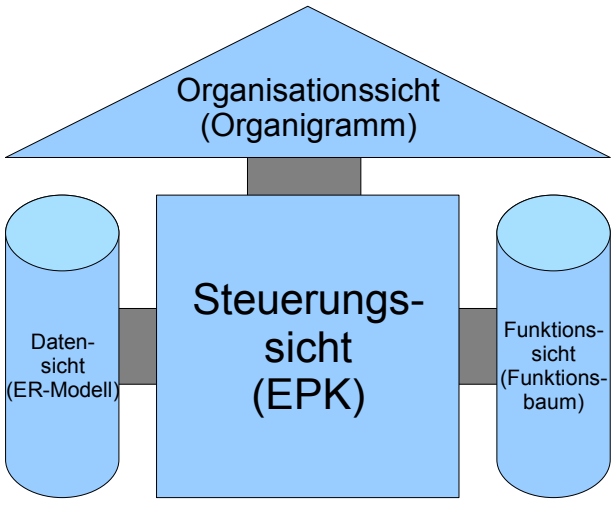

Abbildung 5: Das ARIS Haus mit den Sichten Organisationssicht, Steuerungssicht, Funktionssicht und Datensicht

3.1.3 Syntaktische Überprüfbarkeit von EPK

Die verbindlichen Modellierungsvorschriften aus Kapitel 3.1.1 lassen sich bereits in der grafischen Modellierungsumgebung für EPK durchsetzen, da es sich ausschließlich um Vorschriften für Verbindungen zwischen den Knoten des Graphen handelt.

Eine integrierte Entwicklungsumgebung für EPK kann zusätzlich noch die ebenfalls in Kapitel 3.1.1 dargestellten Designrichtlinien überprüfen. Da diese jedoch nicht zwingend vorgeschrieben sind muss der Überprüfungsvorgang vom Benutzer manuell angestoßen werden.

Weitere strukturelle Prüfungen machen erst nach der Übersetzung in Petri-Netze Sinn und sollen hier nicht weiter behandelt werden, da auf die Strukturanalyse von Petri-Netzen in Kapitel 3.2.3 ausführlich eingegangen wird.

3.1.4 Semantische Überprüfbarkeit von EPK

Da EPK nicht simulierbar ist es nicht ohne weiteres möglich, eine semantische Analyse von EPK durchzuführen. Durch die Möglichkeit der Überführung in Petri-Netze ist es jedoch möglich, für ausreichend definierte EPK semantische Analysemethoden für Petri-Netze und insbesondere für Workflow-Netze anzuwenden (siehe Kapitel 3.2.4). Hinsichtlich der Überführung in Petri-Netze können zahlreiche Probleme auftreten, so dass dieser Vorgang meist nicht automatisiert durchgeführt werden kann.

3.2 Petri-Netze

Petri-Netze sind heute ein wichtiges Hilfsmittel für die Modellierung und die Analyse von Geschäftsprozessen. Petri-Netze haben ihren Ursprung in der Dissertationsschrift „Kommunikation mit Automaten“, die 1962 von Carl Adam Petri verfasst wurde. Sie eignen sich insbesondere zur Beschreibung nebenläufiger Systeme. Erst in jüngeren Jahren wurde das Potential der Petri-Netze zur Modellierung von Geschäftsprozessen erkannt.

Geschäftsprozesse können mit Petri-Netzen mit beliebig vielen Details modelliert werden. Das Ergebnis bleibt stets simulierbar, so dass sowohl syntaktische als auch semantische Analysemethoden angewendet werden können.

3.2.1 Formale Definition von Petri-Netzen

Ein Petri-Netz besteht aus genau zwei Arten von Knoten, die man als Stellen und Transitionen bezeichnet. Die Knoten dürfen über gerichtete Kanten miteinander verbunden werden. Allerdings muss auf eine Stelle stets mindestens eine Transition folgen, auf eine Transition wiederum muss mindestens eine Stelle folgen. Ein Petri-Netz ist somit ein gerichteter, bipartiter Graph. Abbildung 6 zeigt die Darstellung eines typischen Petri-Netzes mit Stellen (dargestellt als Kreise) und Transitionen (dargestellt als Quadrate).

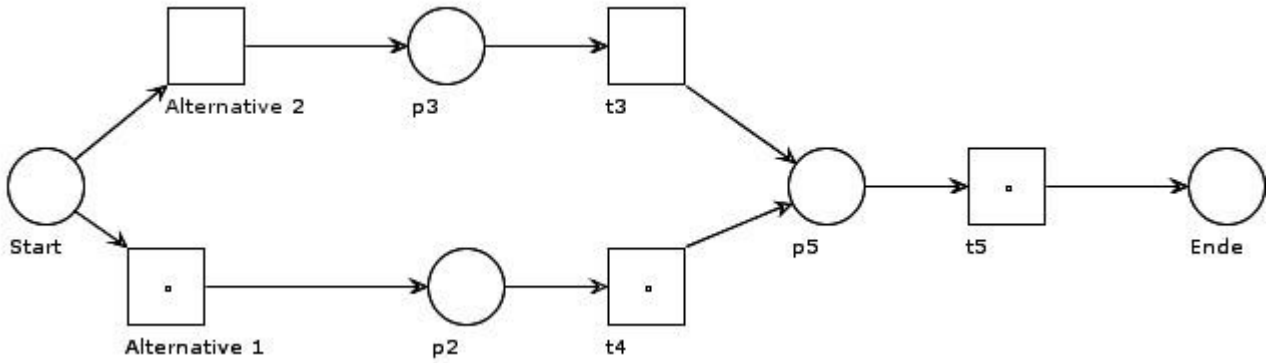

Abbildung 6: Ein typisches Workflow Petri-Netz

Die übliche formale Definition eines Petri-Netzes kann z.B. [richstucbusi2004ubka] entnommen werden: Ein Petri-Netz ist ein Tripel N = (S, T, K), für das gilt:

- Ein Knoten ist entweder Stelle oder Transition: $S \cap T = \emptyset$
- Es muss mindestens eine Stelle oder eine Transition existieren: $S \cup T \neq \emptyset$
- Eine Kante verbindet entweder eine Stelle mit einer Transition oder eine Transition mit einer Stelle, jedoch niemals eine Stelle mit einer Stelle bzw. eine Transition mit einer Transition: $K \subseteq (S \times T) \cup (T \times S)$

Eine Transition gehört zum Vorbereich einer Stelle, wenn eine gerichtete Kante die Transition mit der Stelle verbindet. Ebenso gehört eine Stelle zum Vorbereich einer Transition, wenn diese über eine gerichtete Kante mit der Transition verbunden ist. Allgemein ist der Vorbereich eines Knotens nach [richstucbusi2004ubka] definiert als: $\cdot x = \{ y | (y, x) \in K \}$

Eine Transition gehört zum Nachbereich einer Stelle, wenn eine gerichtete Kante die Stelle mit der Transition verbindet. Ebenso gehört eine Stelle zum Nachbereich einer Transition, wenn die Transition über eine gerichtete Kante mit dieser verbunden ist. Allgemein ist der Nachbereich eines Knotens nach [richstucbusi2004ubka] definiert als: $x \cdot = \{ y | (x, y) \in K \}$

Eine Transition oder Stelle heißt vorwärtsverzweigt, wenn ihr Nachbereich mehr als einen Knoten enthält: $|x \cdot| > 1$

Eine Transition oder Stelle heißt rückwärtsverzweigt, wenn ihr Vorbereich mehr als einen Knoten enthält: $|\cdot x| > 1$

Die Stellen eines Petri-Netzes können beliebig viele Marken enthalten. Die Beschreibung der Marken eines Petri-Netzes heißt Markierung. Sie drückt den Zustand eines Petri-Netzes aus. Die Anfangsmarkierung eines Petri-Netzes beschreibt seinen Ursprungszustand. Das Symbol [m> bezeichnet die Menge aller Folgemarkierungen einer gegebenen Markierung m.

Des weiteren können Markierungen miteinander verglichen werden:

- Eine Markierung m_1 ist genau dann $\geq$ einer Markierung m_2, wenn für jede einzelne Stelle des zugehörigen Petri-Netzes gilt, dass die Anzahl Marken unter m_1 größer oder gleich der unter m^2 ist
- Eine Markierung m_1 ist genau dann > als eine Markierung m_2, wenn $m_1 \geq m_2$ und für mindestens eine Stelle des zugehörigen Petri-Netzes die Anzahl Marken unter m_1 größer ist als unter m_2
- Eine Markierung m_1 ist genau dann $\leq$ einer Markierung m_2, wenn für jede einzelne Stelle des zugehörigen Petri-Netzes gilt, dass die Anzahl Marken unter m_1 kleiner oder gleich der unter m_2 ist
- Eine Markierung m_1 ist genau dann < als eine Markierung m_2, wenn $m_1 \leq m_2$ und für mindestens eine Stelle des zugehörigen Petri-Netzes die Anzahl Marken unter m_1 kleiner ist als unter m_2
- Eine wichtige Beobachtung ergibt sich aus der Definition der Vergleichsoperatoren für Markierungen eines Petri-Netzes: $\neg(m_1 \leq m_2) \neq (m_1 > m_2)$

Die Markierung eines Petri-Netzes spielt bei der semantischen Analyse eine große Rolle. Für die Strukturanalyse wird sie dagegen nicht beachtet bzw. die Ergebnisse der strukturellen Analyse gelten unter jeder beliebigen Markierung eines Petri-Netzes.

Zur schrittweisen Verfeinerung von Modellen bietet es sich an, Teilvorgänge als Subprozesse zu modellieren. Mit ihnen lässt sich der in Kapitel 2.1.3 beschriebene Ansatz der schrittweisen Verfeinerung eines Modells gut umsetzen. Nicht benötigte Details lassen sich bei der Analyse einfach ausblenden. Subprozesse sind stellenberandet, d.h. es existieren keine Transitionen ohne ein- oder ausgehende Kanten. Soll ein Subprozess ausgeblendet werden, wird an seiner Stelle eine Transition mit doppeltem Rand angezeigt (siehe Abbildung 7).

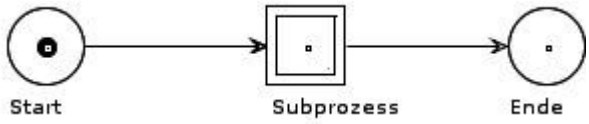

Abbildung 7: Einbindung eines Subprozesses

Um reale Sachverhalte effizienter und ohne unnötige Komplexität durch die Abbildung von Teilaspekten eines Geschäftsprozesses als Workflow-Netz modellieren zu können wurde eine Erweiterung zu den traditionellen Petri-Netzen entwickelt, die sog. High-Level Petri-Netze. So besteht die Möglichkeit, äußere Einflüsse in Form von Triggern zu modellieren und durch Beschriftung von Transitionen ihr Schaltverhalten zu beschreiben (z.B. „if (Anzahl_Artikel > 10)“). Zudem wird es durch Angaben zur Schaltdauer von Transitionen möglich, die Performance modellierter Geschäftsprozesse zu analysieren. Eine weitere nützliche Neuerung stellen die sog. „colored tokens“ dar. Sie erlauben es, Marken voneinander zu unterscheiden um so ihren Verlauf während der Abarbeitung des Workflow-Netzes nachvollziehbar zu machen.

3.2.2 Tools zur Modellierung und Analyse von Petri-Netzen

Die wohl bekannteste Analysesoftware für Workflow-Netze (siehe Kapitel 3.2.3) ist die an der Universität Eindhoven entwickelte Software Woflan. Sie bietet sowohl strukturelle als auch semantische Analysemethoden für Petri-Netze. Die hierfür eingesetzten Algorithmen werden ständig verbessert und weiter optimiert. Die Software Woflan kann von [Woflan] bezogen werden.

Für die Modellierung und Analyse von Petri-Netzen existieren eine ganze Reihe von spezialisierten Tools, die untereinander Modelle von Petri-Netzen austauschen können. Ein grafisches Tool zur Bearbeitung, Analyse und Simulation insbesondere von Workflow-Netzen ist die Java-Anwendung WoPeD, welche an der Berufsakademie Karlsruhe von einer Reihe freiwilliger Programmierer als quelloffene Software entwickelt wird. Durch seine Export-Fähigkeiten in das XML Format PNML sowie das von Woflan verwendete Datenformat TPL (siehe [verbeek]) lassen sich mit WoPeD entworfene Petri-Netz Modelle durch externe Software hervorragend analysieren. Die Analysesoftware WOFLAN kann hierbei sogar aus der grafischen Benutzeroberfläche von WoPeD heraus gestartet werden. Die Software WoPeD kann von [WoPeD] bezogen werden.

Die Software WoPeD wurde im Rahmen dieser Arbeit um eigene Strukturanalysefähigkeiten sowie eine Java Native Interface basierte Direktanbindung an die Woflan Bibliothek erweitert. Dies erlaubt unter Anderem die grafische Darstellung von Struktur- und Semantikverletzungen, die während der sogenannten Soundness Überprüfung gefunden wurden.

Weitere grafische Tools zur Modellierung und Analyse von Petri-Netzen sind das am Institut für Regelungstechnik der TH Aachen entwickelte Programm Netlab (http://www.irt.rwth-aachen.de/typo3/index.php?id=101&type=0) sowie die Java-Software JARP (http://jarp.sourceforge.net/us/index.html).

3.2.3 Syntaktische Überprüfbarkeit von Petri-Netzen

Bei der syntaktischen Überprüfung von Petri-Netzen werden lediglich Stellen, Transitionen und Verbindungen betrachtet, nicht jedoch Markierungen. Im Gegensatz zur Semantikanalyse beschränkt man sich bei der syntaktischen Analyse also auf die Struktur des Petri-Netzes, ohne konkrete Instanzen zu betrachten.

Es existieren eine Reihe formaler Eigenschaften, die von Petri-Netzen erfüllt sein können. Viele dieser Eigenschaften sichern dem Petri-Netz ein bestimmtes Verhalten zu, andere formale Eigenschaften wiederum erlauben eine beschleunigte Semantikanalyse (siehe Kapitel 3.2.4).

Die „Workflow“ Eigenschaft

Die Abbildung eines Geschäftsprozesses in ein Petri-Netz sollte die Vorgaben eines Workflow-Netzes erfüllen. Für Workflow-Netze gelten eine Reihe von Bedingungen, die sich aus der Definition des Begriffs Geschäftsprozess ergeben:

1. Ein Workflow-Netz hat stets genau einen definierten Anfang, also ein Ereignis, das den Geschäftsprozess anstößt
2. Ebenso existiert genau ein definiertes Ende, der Abschluss des Geschäftsprozesses
3. In einem Workflow-Netz dürfen nur Stellen und Transitionen modelliert sein, die auch tatsächlich für die Darstellung des Geschäftsprozesses benötigt werden

Während die ersten beiden Bedingungen sich sehr einfach formal ausdrücken lassen, bedarf die letzte Bedingung weiterer Erläuterung:

Stellen und Transitionen können nur dann tatsächlich Teil des Workflow-Netzes sein, wenn sie von dem auslösenden Ereignis als Zustand auf irgendeinem Weg erreichbar sind. Es muss also zumindest eine Verbindung von jeder Stelle oder Transition zu jeder Anderen existieren. Ein Petri-Netz, welches diese Eigenschaft erfüllt nennt man zusammenhängend. Ein Beispiel für ein nicht zusammenhängendes Petri-Netz gibt Abbildung 8.

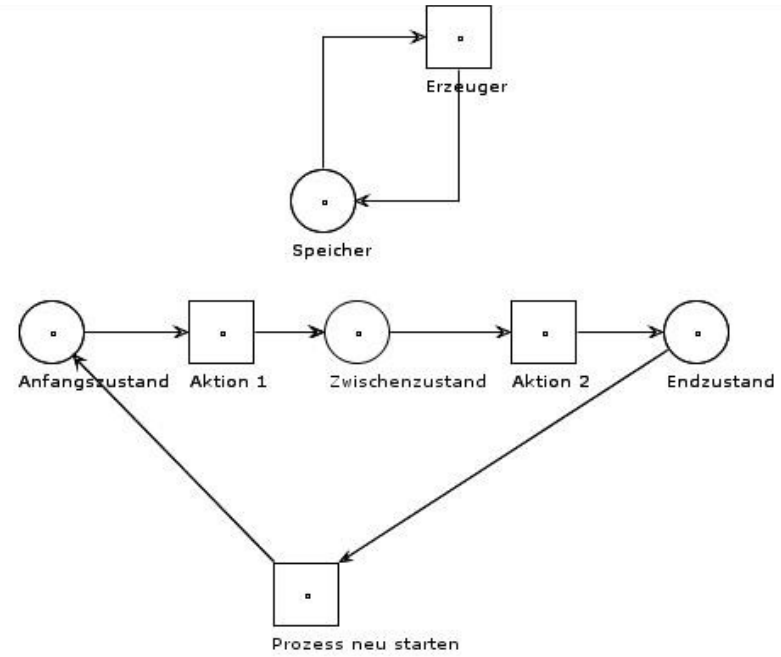

Abbildung 8: Nicht zusammenhängendes Petri-Netz

Formal lässt sich diese Bedingung nach [richstucbusi2004ubka] wie folgt ausdrücken:

Ein Petri-Netz N = (S, T, F) ist genau dann zusammenhängend, wenn keine Zerlegung der Knotenmenge in X_1 und X_2 existiert, so dass gilt:

1. Beide Knotenmengen sind nicht leer: $X_{1,} X_2 \neq \emptyset$
2. Die Summe der beiden Knotenmengen enthält genau alle Stellen und Transitionen des Petri-Netzes: $X_1 \cup X_2 = S \cup T$
3. Die beiden Knotenmengen überschneiden sich nicht: $X_1 \cap X_2 = \emptyset$
4. Die Menge aller Verbindungen enhält keine Verbindungen zwischen Elementen der Knotenmenge X_1 und X_2: $F \subseteq (X_1 \times X_1) \cup (X_2 \times X_2)$

Wie in Kapitel 3.2.1 dargestellt sind Petri-Netze gerichtete Graphen. Daher ist bei einem zusammenhängenden Petri-Netz noch nicht automatisch sichergestellt, dass man von jedem Knoten zu jedem Anderen gelangen kann. Schließlich lasen sich die Verbindungen zwischen zwei Knoten jeweils nur in einer bestimmten Richtung benutzen. Ein Petri-Netz heißt stark zusammenhängend, wenn sich unter Berücksichtigung der jeweiligen Richtung der Verbindungen jeder Knoten des Petri-Netzes von jedem Anderen aus erreichen lässt.

Ein Beispiel für ein zusammenhängendes, aber nicht stark zusammenhängendes Petri-Netz findet sich in Abbildung 9.

Formal ist ein Petri-Netz nach [richstucbusi2004ubka] genau dann stark zusammenhängend, wenn für je zwei Elemente $x, y \in S \cup T$ mit $x \neq y$ eine Sequenz $(z_{1,} z_2), (z_{2,} z3), ..., (z_{n-1}, z_n) \in F$ mit $(n \geq 2)$ existiert, so dass $x = z_1$ und $y = z_n$. Mit anderen Worten: x ist transitiv durch mindestens eine gerichtete Verbindung mit y verbunden.

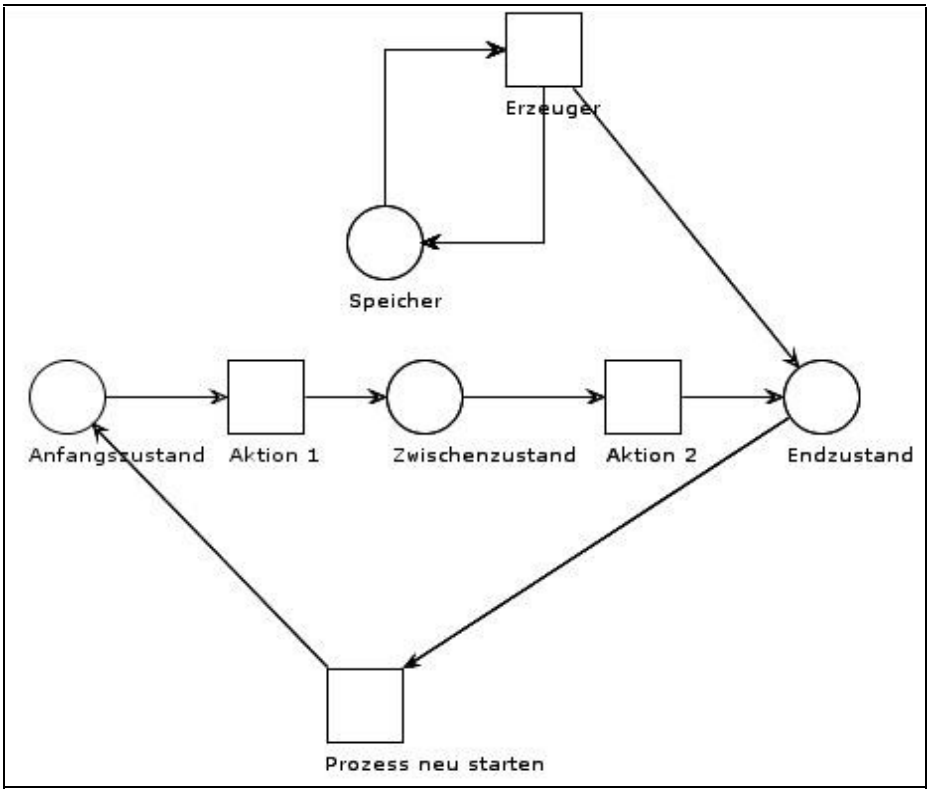

Abbildung 9: Zusammenhängendes, aber nicht stark zusammenhängendes Petri-Netz

Um zeigen zu können, dass jeder Knoten eines Petri-Netzes mit einem anderen transitiv verbunden ist bietet sich eine tabellarische Darstellung der Erreichbarkeit der Knoten eines Petri-Netzes an. Die folgende Tabelle gibt die Erreichbarkeit der Knoten des in Abbildung 9 dargestellten Petri-Netzes wieder. Jede Zelle der Tabelle gibt dabei die kürzeste Entfernung zwischen zwei Knoten wieder. Ist keine Verbindung zwischen zwei Knoten vorhanden, so ist die korrespondierende Zelle mit ∞ markiert.

	Anfangszustand	Aktion 1	Zwischenzustand	Aktion2	Endzustand	Prozess neu starten	Erzeuger	Speicher
Anfangszustand	0	1	2	3	4	5	∞	∞
Aktion 1	5	0	1	2	3	4	∞	∞
Zwischenzustand	4	5	0	1	2	3	∞	∞
Aktion 2	3	4	5	0	1	2	∞	∞
Endzustand	2	3	4	5	0	1	∞	∞
Prozess neu starten	1	2	3	4	5	0	∞	∞
Erzeuger	3	4	5	6	1	2	0	1
Speicher	4	5	6	7	2	3	1	0

Das betrachtete Petri-Netz ist genau dann stark zusammenhängend, wenn keine Zelle der korrekt ausgefüllten Tabelle mit ∞ markiert ist. Ähnlich lässt sich auch zeigen, dass ein Petri-Netz zusammenhängend ist. Hierfür muss lediglich die Richtung der Verbindungen ignoriert werden.

Algorithmisch kann die obige Tabelle T[N,N] beispielsweise unter Zuhilfenahme des Algorithmus von Moore (vgl. [ihridisk1994ubka]) errechnet werden. Hierzu wird der Algorithmus für jede Zeile der Tabelle einmal angewendet. Die Berechnung der Tabelle lässt sich somit mit einer Komplexität von $O(n^2)$ durchführen.

Die üblicherweise mitgeführte Routenmatrix wird für diesen Zweck nicht benötigt, da der kürzeste Pfad zwischen zwei Knoten nicht angegeben werden muss.

Der Wert -1 wird im Folgenden als ∞ interpretiert. Alle Knoten der Petri-Netzes seien ferner von 1 bis N durchnummeriert und die von einem Knoten x direkt erreichbaren Knoten über die Indexliste OUTGOINGARCS[x] bekannt.

Die Tabelle T[N,N] wird also zunächst so ausgefüllt, dass

$$T[I,J]=\begin{cases}0 & \textit{für } (i=j)\\ -1 & \textit{sonst}\end{cases}$$

Dann wird der folgende iterative Algorithmus angewendet:

```
FOR (J=1 TO N)
       CURRENTLIST[] = { J };
       WHILE (CURRENTLIST[].SIZE>0)
       BEGIN
              NEWLIST[] = {};
              FOR (I=1 TO CURRENTLIST[].SIZE)
              BEGIN
                     OUTGOINGARCS[] = OUTGOINGARCS[CURRENTLIST[I]];
                     FOR (K=1 TO OUTGOINGARCS[].SIZE)
                     BEGIN
                            IF (T[J,OUTGOINGARCS[K]]==-1)
                            BEGIN
                                   T[J,OUTGOINGARCS[K]] = CURRENTLIST[I] + 1;
                                   NEWLIST := NEWLIST + {OUTGOINGARCS[K]};
                            END;
                     END;
              END;
              CURRENT := NEWLIST;
       END;
END;
```

Mit den inzwischen gewonnenen Erkenntnissen kann nun eine formale Definition eines Workflow-Netzes nach [richstucbusi2004ubka] wie folgt angegeben werden:

Ein Petrinetz N = (S, T, F) mit zwei Stellen i und o ist genau dann ein Workflow-Netz, wenn

1. die Stelle i eine Quelle ist, also keine Vorstellen hat: $\cdot i = \emptyset$
2. die Stelle o eine Senke ist, also keine Nachstellen hat: $o \cdot = \emptyset$
3. das um eine zusätzliche Transition t* mit $\cdot t^* = o \wedge t^* \cdot = i$ erweiterte Netz stark zusammenhängend ist

Ohne die Erweiterung um die Transition t* ist ein Workflow-Netz zwar zusammenhängend, jedoch niemals stark zusammenhängend, da wegen (1) und (2) keine Verbindung von o zu i existiert.

Gibt es genau eine Quelle und eine Senke, so ist ein um die Transition t* erweitertes Petri-Netz genau dann zusammenhängend bzw. stark zusammenhängend, wenn jeder Knoten von der Transition t* aus erreichbar ist und von jedem Knoten aus die Transition t* erreicht werden kann.

Ein Knoten, der diese Bedingung nicht erfüllt ist mit dem restlichen Netz nicht verbunden bzw. stark verbunden. In diesem Fall ist er entweder in der zu t* gehörenden Zeile oder Spalte der Erreichbarkeitstabelle mit ∞ markiert.

Zu jedem Workflow-Netz lässt sich ein korrespondierendes um die Transition t* mit $\cdot t^* = o \wedge t^* \cdot = i$ erweitertes Netz konstruieren. Im weiteren Verlauf wird dieses Netz als „short-circuited“[1]-Netz bezeichnet.

Zwei weitere nützliche strukturelle Eigenschaften sind die „free-choice“ und die „well-structured“ Eigenschaft. Ist ein Petri-Netz entweder well-structured oder free-choice, so erleichtert dies die dynamische Analyse erheblich (siehe Kapitel 3.2.4). Ist keine der beiden Eigenschaften erfüllt, so ist die Gefahr groß, dass der durch das Petri-Netz modellierte Geschäftsprozess fehlerhaft ist.

1 Englisch: kurzgeschlossen

Die „Free-Choice“ Eigenschaft

Damit ein Petri-Netz die free-choice Eigenschaft erfüllt müssen zwei prinzipielle Bedingungen erfüllt sein:

- Existieren zu einer beliebigen Stelle des Petri-Netzes mehrere mögliche Transitionen (Vorwärtsverzweigung), so darf keine dieser Transitionen von irgendeiner Stelle abhängen, von der nicht auch alle Anderen abhängen. Mit anderen Worten: Ist eine der beiden Transitionen schaltbar, so muss auch die Andere schaltbar sein.
- Ist die Schaltbarkeit einer Transition von mehr als einer Stelle abhängig (Rückwärtsverzweigung), so darf keine dieser Stellen eine weitere Transition freischalten, die nicht auch von allen Anderen Stellen abhängt.

Formal lassen sich die beiden Bedingungen nach [richstucbusi2004ubka] wie folgt ausdrücken:

1. $\forall t_{1,} t_2 \in T : \cdot t_1 \cap \cdot t_2 \neq \emptyset \Rightarrow \cdot t_1 = \cdot t_2$
2. $\forall s_{1,} s_2 \in S : s_1 \cdot \cap s_2 \cdot \neq \emptyset \Rightarrow s_1 \cdot = s_2 \cdot$

Der im folgenden beschriebene Algorithmus mit einer Komplexität von $O(n^2)$ lässt sich leicht aus der formalen Definition der „Free-Choice“ Eigenschaft herleiten. Die Variablen werden vor Ausführung des Algorithmus wie folgt initialisiert:

Variable	Beschreibung	Wert
S[]	Array, das eine Beschreibung aller Stellen enthält.	Jede Stelle wird durch eine Referenz auf die Vorgänger- und Nachfolger-Knoten sowie deren Anzahl beschrieben
NS	Gibt die Anzahl der Stellen des Petri-Netzes an	Anzahl Einträge in S[]
T[]	Array, das eine Beschreibung aller Transitionen enthält.	Jede Transition wird durch eine Referenz auf die Vorgänger- und Nachfolger-Knoten sowie deren Anzahl beschrieben
NT	Gibt die Anzahl der Transitionen des Petri-Netzes an	Anzahl Einträge in T[]

Der Algorithmus liefert einen Wahrheitswert TRUE oder FALSE als Ergebnis. Dieser Wert gibt an, ob es sich bei dem gegebenen Petri-Netz um ein „Free-Choice“ Netz handelt oder nicht:

```
FREECHOICE := TRUE;
X := 1;
WHILE (X<=NS AND FREECHOICE)
BEGIN
       IF (S[X].NUMNACHFOLGER>1)
       BEGIN
              NACHFOLGER := S[X].NACHFOLGER;
              NUMNACHFOLGER := S[X].NUMNACHFOLGER;
              Y := 2;
              WHILE (Y<=NUMNACHFOLGER AND FREECHOICE)
              BEGIN
                     FREECHOICE :=
                            (NACHFOLGER[Y].VORGAENGER[] ==
                             NACHFOLGER[1].VORGAENGER[]);
                     Y := Y + 1;
              END;
       END;
       X := X + 1;
END;
X := 1;
WHILE (X<=NT AND FREECHOICE)
BEGIN
       IF (T[X].NUMVORGAENGER>1)
       BEGIN
              VORGAENGER := T[X].VORGAENGER;
              NUMVORGAENGER := T[X].NUMVORGAENGER;
              Y := 2;
              WHILE (Y<=NUMVORGAENGER AND FREECHOICE)
              BEGIN
                     FREECHOICE :=
                            (VORGAENGER[Y].NACHFOLGER[] ==
                             VORGAENGER[1].NACHFOLGER[]);
                     Y := Y + 1;
              END;
       END;
       X := X + 1;
END;
RETURN FREECHOICE;
```

Die Begriffe „Well-handled“, „Well-Structured“

Workflow-Netze sollten so entworfen werden, dass sowohl bei der Modellierung von Konflikten als auch bei der Modellierung von parallelen Vorgängen (vorwärts verzweigte Stellen bzw. Transitionen) korrespondierende Synchronisationspunkte vorhanden sind.

Im Fall von Konflikten muss der Synchronisationspunkt eine Stelle sein. Sie beschreibt einen Zustand, in dem die Konfliktsituation abgearbeitet wurde, so dass von nun an wieder unabhängig von der vorher gewählten Route weitergearbeitet werden kann. (siehe Abbildung 10). Wird diese Regel verletzt, so führt dies mit einiger Wahrscheinlichkeit zu Dead- oder Live-Locks, da der Synchronisationspunkt nur schaltet, wenn beide alternative Routen abgearbeitet wurden oder von einer anderen Stelle des Petri-Netzes eine Marke „zweckentfremdet“ werden kann. Diese Marke fehlt dann möglicherweise später für einen anderen Schaltvorgang, so dass sich der Fehler im System fortpflanzen und an völlig unerwarteter Stelle zu Problemen führen kann.

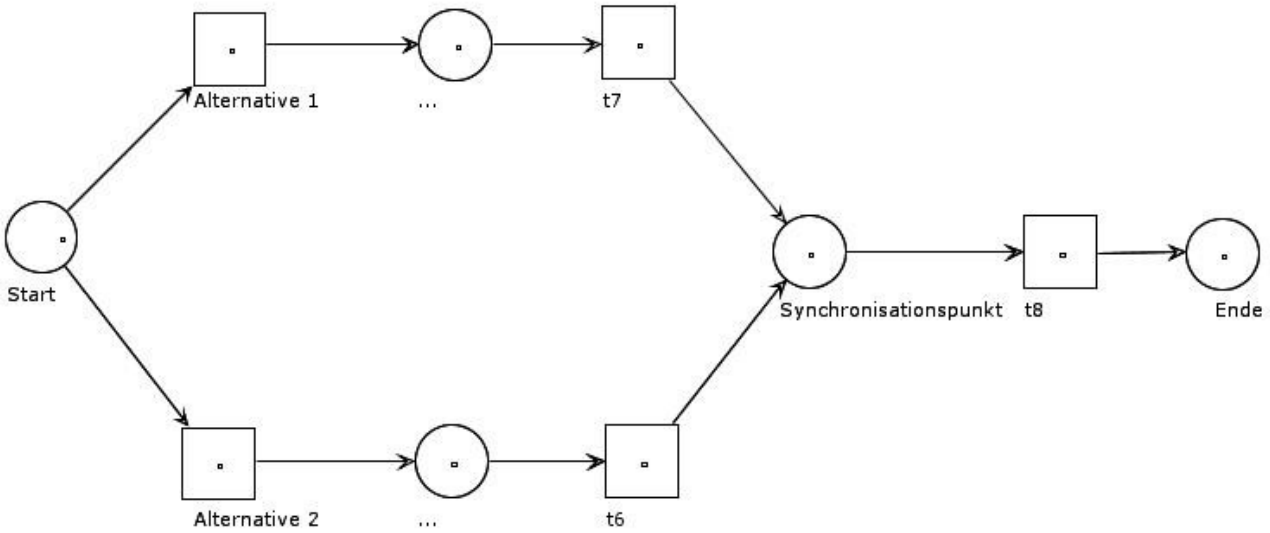

Abbildung 10: Alternative Ausführung (Konflikt) und anschließende Synchronisation

Im Fall von parallelen Vorgängen besteht der Synchronisationspunkt aus einer rückwärtsverzweigten Transition. Sie lässt sich erst schalten, wenn beide parallele Abarbeitungsstränge erfolgreich beendet wurden (siehe Abbildung 11). Wird diese Regel verletzt so findet nie eine saubere Synchronisation zwischen zwei Vorgängen statt, die Bearbeitung der restlichen Aufgaben wird fortgeführt, obwohl möglicherweise ihre Vorbedingungen noch nicht erfüllt wurden.

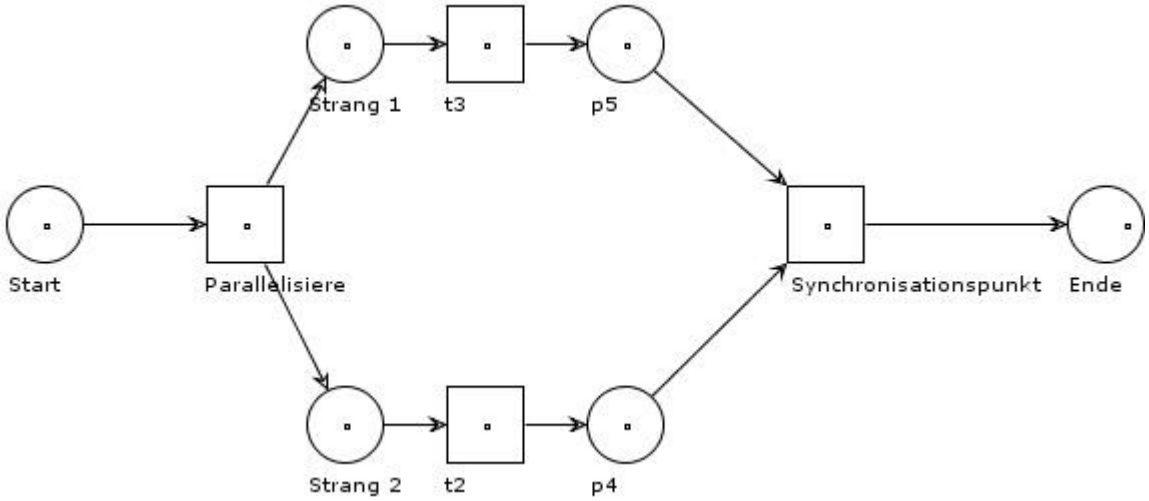

Abbildung 11: Parallele Ausführung und anschließende Synchronisation

Ein Petri-Netz, das unter Beachtung der genannten Synchronisationsregeln modelliert wurde bezeichnet man als „Well-handled“. Bei Workflow-Netzen soll die Synchronisation zusätzlich für beliebig viele sequentielle Abläufe des gleichen Workflows erfüllt sein. Ist dies der Fall, so ist das Workflow-Netz „Well-structured“.

Die Formale Definition der Begriffe well-handled und well-structured ist nach [richstucbusi2004ubka]:

Ein Petrinetz N = (S, T, K) wird genau dann als well-handled bezeichnet, wenn für je zwei Knoten $x \in T \wedge y \in S$ oder $x \in S \wedge y \in T$ keine zwei Pfade von x nach y existieren, die nur x und y gemeinsam haben.

Ein Workflow-Netz ist genau dann well-structured, wenn das korrespondierende „short-circuited“-Netz „well-handled“ ist. Kennt man den Algorithmus zur Ermittlung der well-handled Eigenschaft für ein beliebiges Petri-Netz so ist also auch die well-structured Eigenschaft eines beliebigen Workflow-Netzes leicht zu ermitteln.

Ein Algorithmus zur Überprüfung der well-handled Eigenschaft muss also

- Transitionen finden, die durch mindestens zwei unabhängige Pfade mit einer Stelle verbunden sind
- Stellen finden, die durch mindestens zwei unabhängige Pfade mit einer Transition verbunden sind

Eine einfache Implementation eines solchen Algorithmus sucht zunächst alle in Frage kommenden Konflikte und Parallelisierungen ($(|S\cdot|>1)\vee(|T\cdot|>1)$). Für jeden gefundenen Knoten I werden zunächst alle Knoten als unmarkiert gekennzeichnet. Dann werden die abgehenden Kanten J iteriert. Bei jeder Iteration werden durch Tiefensuche alle über die aktuelle Kante J ohne erneutes Besuchen des aktuellen Knoten I erreichbare Knoten K mit der Kantennummer J markiert. Wird ein bereits über eine andere Kante J besuchter Knoten K gefunden, so wird dessen Typ mit dem Typ von Knoten I verglichen. Stimmen die Typen nicht überein, so wird das Knotenpaar bestehend aus den Knoten I und K in die Liste der nicht „well-handled“ Knoten aufgenommen. Die abgehenden Kanten eines bereits markierten Knotens werden bei der Tiefensuche nicht berücksichtigt.

Der folgende Pseudo-Code verdeutlicht den beschriebenen Algorithmus:

```
NOT_WELLHANDLED := {};
FOR (I = 0 TO ANZAHL_KNOTEN)
BEGIN
	FOR (X = 0 TO ANZAHL_KNOTEN)
	BEGIN
		KNOTEN[X].MARKIERUNG = -1;
	END;
	FOR (J = 0 TO I.ANZAHL_NACHFOLGER)
	BEGIN
		KNOTEN[I].MARKIERUNG = J;
		SUCHSTAPEL = { KNOTEN[I].NACHFOLGER[j] };
		WHILE (SUCHSTAPEL.SIZE > 0)
		BEGIN
			KNOTEN[K] = SUCHSTAPEL.POP();
			IF (KNOTEN[K].MARKIERUNG == -1)
			BEGIN
				KNOTEN[K].MARKIERUNG = J;
				FOR (L = 0 TO KNOTEN[K].ANZAHL_NACHFOLGER)
				BEGIN
					SUCHSTAPEL.PUSH(KNOTEN[K].NACHFOLGER[L]);
				END;
			ELSE IF ((KNOTEN[K].MARKIERUNG != J) AND
					(KNOTEN[K].TYP != KNOTEN[I].TYP))
			BEGIN
				NOT_WELLHANDLED = NOT_WELLHANDLED UNION { { K, I } };
			END;
		END;
	END;
END;
```

Der beschriebene Algorithmus hat eine Komplexität von $O(n^2)$ und liefert eine Liste von Knotenpaaren, die jeweils die well-handled Eigenschaft des Petri-Netzes verletzen.

In [Aal97] wird noch eine weitere Methode beschrieben, wie die „well-handled“ Eigenschaft von Petri-Netzen überprüft werden kann. Es wird hierbei auf den „max-flow min-cut“ Algorithmus zurückgegriffen, der unter anderem in [ihridisk1994ubka] beschrieben ist. Der Algorithmus berechnet den maximalen Fluss zwischen zwei Knoten eines Graphen. Da der maximale Fluss zunächst für Kanten im Graph gilt, werden die Knoten des Netzes aufgeteilt in einen Knoten, in dem die eingehenden Kanten enden und einen, von dem die abgehenden Kanten starten. Diese beiden Knoten werden dann durch eine einzelne Kante verbunden. Für diese Kante lässt sich nun der maximale Fluss berechnen. Damit die „well-handled“ Eigenschaft erfüllt ist muss dieser kleiner oder gleich eins sein. Eine abgewandelte Form dieses Algorithmus kommt in der Analysesoftware Woflan zum Einsatz.

Petri-Netze als Zustandsautomaten

Da Petri-Netze eine Verallgemeinerung von Zustandsautomaten für verteilte Systeme sind lassen sich auch echte Zustandsautomaten als Petri-Netze modellieren. Ein Petri-Netz ist genau dann ein Zustandsautomat, wenn es keine Parallelisierungen und entsprechende Synchronisationspunkte enthält. Konflikte dürfen dagegen modelliert werden. Die formale Definition eines Zustandsautomaten besagt daher, dass sowohl der Vor- als auch der Nachbereich jeder Transition genau einen Knoten enthält: $\forall t \in T : |\cdot t| = |t \cdot| = 1$

Durch das die fehlende Möglichkeit parallele Abläufe zu modellieren sind Zustandsautomaten nur begrenzt zur Modellierung von Geschäftsprozessen geeignet. Deshalb kommt die Eigenschaft „Zustandsautomat“ in der Regel nicht als wünschenswerte Eigenschaft für ein Workflow-Netz in Frage. Statt dessen kann versucht werden, ein Workflow-Netz so zu entwerfen, dass es in mehrere parallel ablaufende (miteinander synchronisierte) Zustandsautomaten zerlegbar ist. Die Ergebnisse einer solchen Zerlegung nennt man S-Komponenten. Eine S-Komponente ist definiert als ein Teilnetz N' = (S', T', F') eines gegebenen Petri-Netzes N = (S, T, F) mit den folgenden Eigenschaften:

- Das mit N' korrespondierende „short-circuited“-Netz ist stark zusammenhängend
- N' ist ein Zustandsautomat, also $\forall t' \in T' : |\cdot t'| = |t' \cdot| = 1$
- Keine zu N' gehörende Stelle ist mit einer Transition verbunden, die nicht zu N' gehört, also $\forall s \in S' : \cdot s \cup s \cdot \subseteq T'$

Sind alle Stellen eines Petri-Netzes N = (S, T, F) in einer seiner S-Komponenten enthalten, so hat das Petri-Netz die Eigenschaft „s-coverable“. Die formale Definition nach [richstucbusi2004ubka] ist: $\forall s \in S \; \exists \text{ S-Komponente } N' = (S', T', F') \; von \; N : s \in S'$

Die Eigenschaft „s-coverable“ ist die Generalisierung der Eigenschaften „well-structured“ und „free-choice“. Diese Erkenntnis ergibt sich aus der semantischen Analyse und der Überprüfung auf „soundness“. Jedoch lassen sich nur die Struktureigenschaften „well-structured“ und „free-choice“ für eine beschleunigte Berechnung der Eigenschaft „soundness“ heranziehen.

Netze die entweder die „well-structured“ oder „free-choice“ Eigenschaft besitzen und zudem „sound“ sind besitzen immer auch die Eigenschaft „s-coverable“. Daraus kann gefolgert werden, dass „well-structured“ und „free-choice“ Netze, die nicht „s-coverable“ sind auch nicht „sound“ sind.

Die Eigenschaft „s-coverable“ lässt sich in polynomieller Zeit ermitteln.

S-Invarianten

S-Invarianten sind eine weitere sehr nützliche Struktureigenschaft. Die Existenz sog. positiver S-Invarianten garantiert, dass ein Petri-Netz unter beliebiger Anfangsmarkierung beschränkt ist. Beschränktheit ist eine der beiden Grundvoraussetzungen für „soundness“ (siehe Kapitel 3.2.4).

Eine S-Invariante besteht aus Gewichtungen oder Skalierungsfaktoren für die Stellen eines Petri-Netzes so, dass für alle möglichen Folgemarkierungen einer beliebigen Anfangsmarkierung die gewichtete Summe der Marken aller Stellen konstant ist.

Formal gilt für S-Invarianten

$\sum_{i=1}^{n} y_i * m(s_i) = constant. \forall m \in [m_0>$ für beliebige Anfangsmarkierungen m_0 mit

- n = Anzahl Stellen des Petri-Netzes
- y_i = Gewichtung / Skalierungsfaktor der Stelle i
- $m(s_i)$ = Anzahl Marken der Stelle i

Eine S-Invariante mit $y_i \geq 1 \forall i$ nennt man positiv. Bei positiven S-Invarianten wird jede Stelle mit einem positiven Faktor in die gewichtete Summe einbezogen. In diesem Fall ist das Petri-Netz unter beliebiger Anfangsmarkierung m_0 beschränkt, da ja die gewichtete Summe aller Stellen konstant ist und

- keine Marken durch Multiplikation mit 0 verloren gehen
- keine unbeschränkten Stellen sich durch zueinander multiplikativ inverse Faktoren gegenseitig ausgleichen können

Nach [aalshee] besteht ein Zusammenhang zwischen den bereits beschriebenen S-Komponenten, der Eigenschaft „s-coverable“ und den in einem Petri-Netz vorhandenen S-Invarianten:

- Für jede S-Komponente existiert eine semi-positive S-Invariante ($y_i \geq 0 \forall i$)
- Erfüllt ein Petri-Netz die „s-coverable“ Eigenschaft, so existiert eine positive S-Invariante, es ist also unter beliebiger Anfangsmarkierung m_0 beschränkt.

Die Berechnung der S-Invarianten geschieht durch das Lösen eines Gleichungssystems, das für jede mögliche unmittelbar durch Aktivierung einer beliebigen Transition erreichbare Folgemarkierung von m_0 die gewichtete Summe aller Marken der Folgemarkierung mit der gewichteten Summe aller Marken von m_0 gleichsetzt.

Für das in Abbildung 12 dargestellte Petri-Netz ergeben sich die folgenden Gleichungen:

$$\begin{aligned}
t_1: \quad & y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 &=& \quad y_1(s_1 - 1) + y_2(s_2 + 1) + y_3(s_3 + 1) + y_4 s_4 \\
t_2: \quad & y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 &=& \quad y_1 s_1 + y_2(s_2 - 1) + y_3(s_3 - 1) + y_4(s_4 + 1) \\
t_3: \quad & y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 &=& \quad y_1(s_1 - 1) + y_2 s_2 + y_3 s_3 + y_4(s_4 + 1) \\
t_4: \quad & y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 &=& \quad y_1(s_1 + 1) + y_2 s_2 + y_3 s_3 + y_4(s_4 - 1)
\end{aligned}$$

Für das Gleichungssystem existieren unendlich viele Lösungen, die allesamt die Gleichungen

$$\begin{aligned}
y_1 &= y_2 + y_3 \\
y_4 &= y_1
\end{aligned}$$

erfüllen. Eine mögliche S-Invariante ist somit $y_1 = 2, y_2 = 1, y_3 = 1, y_4 = 2$.

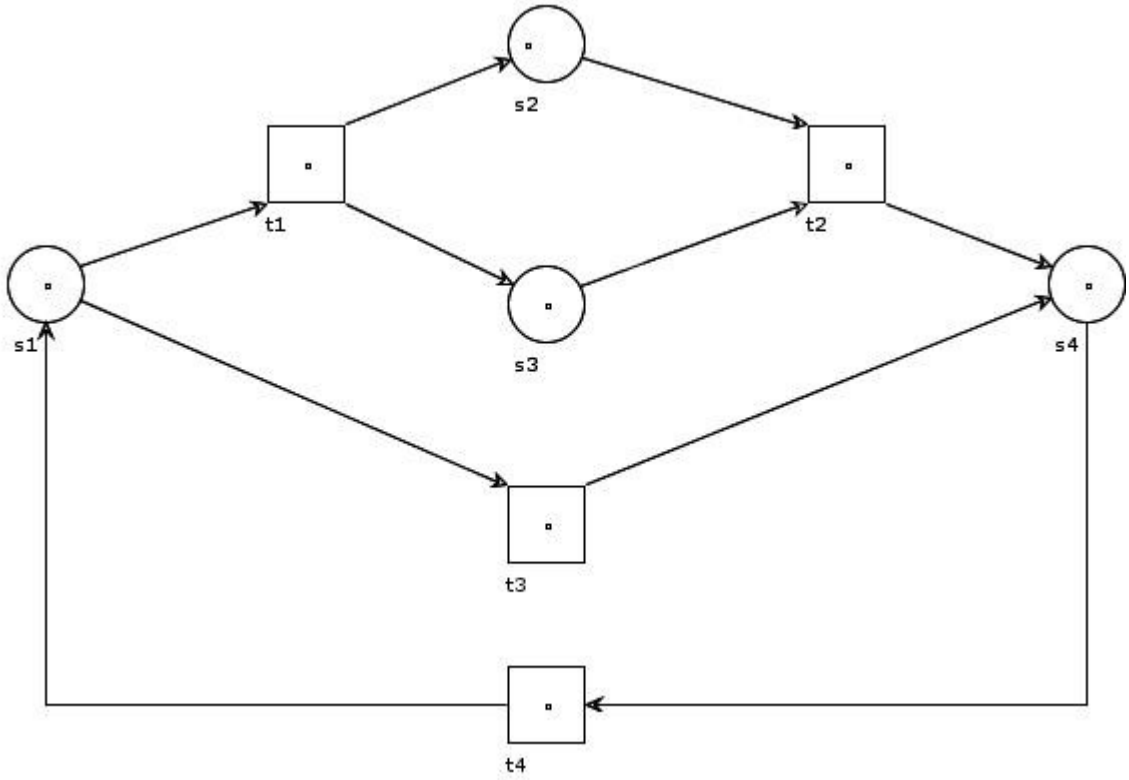

Abbildung 12: Beispiel für die Berechnung der S-Invarianten eines Petri-Netzes

Berechnung der Eigenschaft „s-coverable“ mit Hilfe von S-Invarianten

Die Software Woflan verwendet zur Ermittlung der „s-coverable“ Eigenschaft wie in [Aal97] beschrieben ein Verfahren, bei dem zunächst die S-Invarianten des Petri-Netzes ermittelt werden. Für jede S-Komponente existiert eine korrespondierende S-Invariante, jedoch existiert nicht für jede S-Invariante eine S-Komponente. Von Interesse sind vielmehr nur solche S-Invarianten, die nur 0 oder 1 als Gewichtungsfaktoren haben. Die so gefundenen Stellen werden nun um alle Transitionen erweitert, die die folgende Bedingungen erfüllen:

- Die Transition ist unmittelbar mit einer der gefundenen Stellen verbunden
- Sowohl der Vor- als auch der Nachbereich jeder Transition enthalten genau einen Knoten: $\forall t \in T : |\cdot t| = |t \cdot| = 1$

T-Invarianten

T-Invarianten stellen das „Gegenstück“ zu S-Varianten dar. Auch bei T-Invarianten handelt es sich um ein strukturelles Merkmal. Umgangssprachlich ausgedrückt weist eine T-Invariante jeder Transition eine Gewichtung zu. Wird jede Transition in einer bestimmten, durch die T-Invariante jedoch nicht ausgedrückten Reihenfolge gemäß ihrer Gewichtung mehrmals betätigt, so ergibt sich für eine beliebige Anfangsmarkierung m_0 wieder genau diese Markierung.

Formal gilt für T-Invarianten nach [verbeek]:

$\forall\, p \in P : \sum_{t \in \cdot p} t_i = \sum_{t \in p \cdot} t_i$ für beliebige Anfangsmarkierungen m_0 mit

- P = Die Menge aller Transitionen im Petri-Netz
- t_i = Gewichtung der Transition i

Eine T-Invariante mit $t_i \geq 1 \,\forall\, i$ nennt man positiv. Bei positiven T-Invarianten wird jede Transition mit einem positiven Faktor in die gewichtete Summe einbezogen. Dies bedeutet, dass jede Transition mindestens einmal schalten muss, damit das Petri-Netz wieder die ursprüngliche Markierung hat.

T-Invarianten beschreiben nach [richstucbusi2004ubka] mögliche Zyklen in dem in Kapitel 3.2.4 näher beschriebenen Erreichbarkeitsgraphen eines Petri-Netzes.

Zudem gilt nach [aalshee], dass alle lebendigen Petri-Netze, die die „s-coverable“ Eigenschaft besitzen eine positive T-Invariante besitzen. Folglich kann ein Petri-Netz, das die „s-coverable“ Eigenschaft besitzt, also für alle Anfangsmarkierungen beschränkt ist, aber keine positive T-Invariante besitzt, nicht lebendig sein.

3.2.4 Semantische Überprüfbarkeit von Petri-Netzen

Im Gegensatz zur strukturellen Überprüfung von Petri-Netzen wird bei der semantischen Überprüfung mit markierten Petri-Netzen gearbeitet. Die beiden wichtigsten zu untersuchenden Eigenschaften sind die Beschränktheit und die Lebendigkeit. Sind diese beiden Eigenschaften für das mit einem Workflow-Netz korrespondierende „short-circuited“-Netz erfüllt, so ist das Workflow-Netz „sound“.Der Nachweis hierüber kann für „free-choice“ und „well-structured“ Workflow-Netze in polynomieller Zeit geführt werden.

Die Abbildung 13 zeigt die Abhängigkeiten zwischen den verschiedenen Struktur- und Semantik-Eigenschaften von Workflow-Netzen.

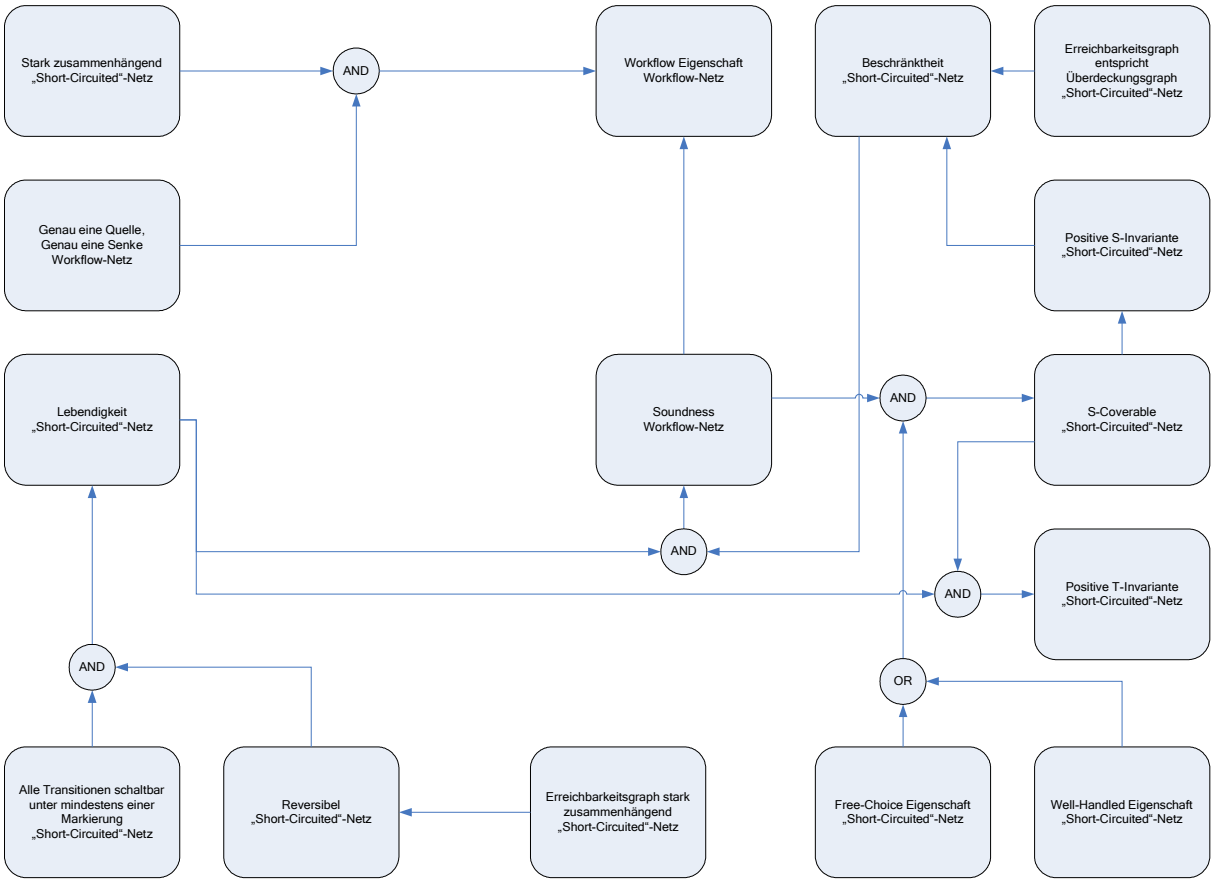

Abbildung 13: Struktur- und Semantikeigenschaften von Workflow-Netzen

Beschränktheit

Ein markiertes Petri-Netz ist dann beschränkt, wenn für jede Stelle ein unter allen Markierungen gültiges Maximum für die Anzahl Marken angegeben werden kann. Diese Beschränktheit ist eine Voraussetzung dafür, dass der durch das Petri-Netz definierte Workflow in endlicher Zeit abgearbeitet werden kann. Ist eine Stelle bezüglich der maximalen Anzahl ihrer Marken nicht beschränkt so gibt es dagegen unendlich viele Markierungen für das gesamte Netz.

Als formale Bedingung formuliert bedeutet Beschränktheit für ein markiertes Petri-Netz N = (N, m_0): $m(s) \leq b \, \forall \, s \in S, \forall \, m \in [m_0>$.

Eine Spezialform der Beschränktheit ist die Eigenschaft „sicher“ oder „1-beschränkt“. Ein Petri-Netz ist „sicher“ oder „1-beschränkt“, wenn gilt: b = 1. Sollen in einem Petri-Netz von alle Stellen nur die beiden Zustände „wahr“ (1) und „falsch“ (0) darstellbar sein, so muss das Petri-Netz „1-beschränkt“ sein, da es sonst für einzelne Stellen zu Werten außerhalb des Wertebereichs kommen kann.

Tot oder lebendig

Petri-Netze oder Teile davon sind entweder tot oder lebendig, oder in einem Zustand dazwischen. Um zu verstehen, wann ein Petri-Netz tot ist, muss zunächst der Begriff tot für Markierungen und einzelne Transitionen definiert werden.

Eine Transition ist tot unter einer bestimmten Markierung, wenn sie weder unter dieser noch unter einer der möglichen Folgemarkierungen schaltbar ist. Eine Markierung heißt dann tot, wenn keine einzige Transition unter dieser Markierung schaltbar ist. Diese Situation bezeichnet man auch als Deadlock. Ein markiertes Petri-Netz heißt dann tot, wenn bereits seine Anfangsmarkierung tot ist.

Ein nicht totes Petri-Netz ist nicht unbedingt lebendig. Genauso wenig trifft dies auf seine möglichen Markierungen und seine Transitionen zu. Wieder muss der Begriff „lebendig“ zunächst für Transitionen, dann für Markierungen und schließlich für markierte Petri-Netze definiert werden. Eine Transition ist lebendig unter einer bestimmten Markierung, wenn sie unter keiner möglichen Folgemarkierung tot ist. Eine Markierung heißt lebendig, wenn alle Transitionen unter dieser Markierung lebendig sind, also alle Transitionen von allen Folgemarkierungen ausgehend irgendwie wieder schaltbar gemacht werden können. Ein markiertes Petri-Netz schließlich heißt dann lebendig, wenn seine Anfangsmarkierung lebendig ist. Ein nicht totes Petri-Netz heißt reversibel, wenn seine Anfangsmarkierung von jeder Folgemarkierung aus wieder erreichbar ist.

Ein Workflow-Netz terminiert, wenn es eine endliche Menge an Schaltfolgen besitzt. Besitzt ein Petri-Netz eine Markierung von welcher aus kein Deadlock, aber auch keine Beendigung mehr möglich ist, so nennt man diese Markierung Livelock.

Der Erreichbarkeitsgraph eines markierten Petri-Netzes

Für jedes markierte Petri-Netz lässt sich der Erreichbarkeitsgraph konstruieren. Der Erreichbarkeitsgraph eines markierten Petri-Netzes mit Anfangsmarkierung m_0 enthält für alle Markierungen $[m_0>$ jeweils einen Knoten. Zwischen zwei Knoten m_x und m_y wird eine gerichtete Kante erzeugt, wenn die Markierung m_y von m_x aus direkt durch Betätigung einer einzigen Transition erreichbar ist. Die Kante wird mit der zu betätigenden Transition beschriftet.

Formal ist ein Erreichbarkeitsgraph $MG(N,m_0) = (M,E)$ nach [richstucbusi2004ubka] für ein markiertes Petri-Netz $N = (S, T, F, m_0)$ wie folgt definiert:

- $M = [m_0>$ (für jede Folgemarkierung aus $[m_0>$ existiert ein Knoten)
- $E \subseteq M \times M \times T$ mit $(m, m', t) \in E \Leftrightarrow m \xrightarrow{t} m'$ (eine Kante verbindet zwei Markierungen m und m' aus M über eine Transition t aus T)

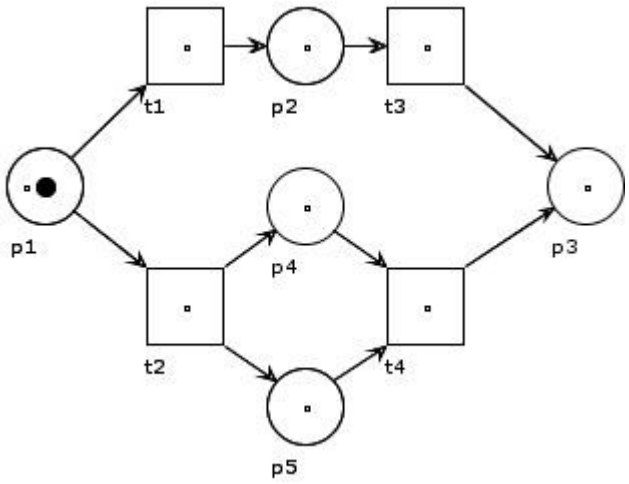

Abbildung 14: Markiertes Petri-Netz für die Erreichbarkeitsanalyse

Die Abbildung 15 zeigt den Erreichbarkeitsgraph des in Abbildung 14 dargestellten markierten Petri-Netzes. Jeder Knoten beschreibt die Anzahl Marken an jeder Stelle des Petri-Netzes.

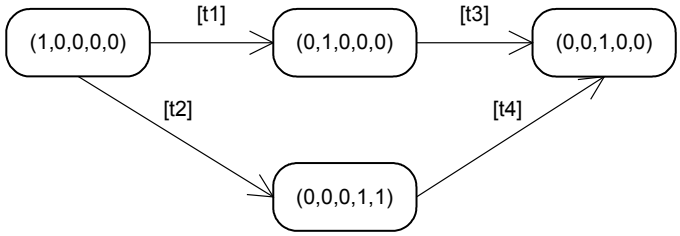

Abbildung 15: Erreichbarkeitsgraph eines markierten Petri-Netzes

Aus dem Erreichbarkeitsgraphen lassen sich nach [richstucbusi2004ubka] die semantischen Eigenschaften eines markierten Petri-Netzes wie folgt ermitteln:

- Ein Erreichbarkeitsgraph ist genau dann endlich, wenn das markierte Petri-Netz beschränkt ist.
- Enthält der Erreichbarkeitsgraph keine Senken, so ist das markierte Petri-Netz verklemmungsfrei, es besitzt also keinen Deadlock. Livelocks sind jedoch weiterhin möglich.
- Ist der Erreichbarkeitsgraph stark zusammenhängend, so ist das Petri-Netz reversibel, d.h. die Anfangsmarkierung m_0 ist von jeder Folgemarkierung aus wieder erreichbar.

Ist ein Petri-Netz unbeschränkt, so ist der Erreichbarkeitsgraph unendlich. Ein unendlicher Graph ist für eine Analyse ungeeignet, da Algorithmen wie der Test auf die Eigenschaft „stark zusammenhängend“ nicht terminieren. Kann man also nicht bereits vor Erstellung des Erreichbarkeitsgraphen eine Aussage über die Beschränktheit des Petri-Netzes treffen (z.B. durch Ermittlung der S-Invarianten oder Feststellung der Eigenschaft „s-coverable“), so muss statt des Erreichbarkeitsgraphen der sog. Überdeckungsgraph aufgebaut werden.

Bei der Erstellung des Überdeckungsgraphen werden streng monoton steigende Folgen von Markierungen ($\forall x \in \mathbb{N}_0 : m_x < m_{x+1}$) durch eine einzige Ersatzmarkierung ersetzt. Die Anzahl der Marken an den unbeschränkten Stellen der Ersatzmarkierung wird als ω notiert.

Der Überdeckungsgraph UEG(NETZ,m_0) = (KNOTEN,KANTEN) lässt sich nach dem folgenden in ähnlicher Form auch in [richstucbusi2004ubka] dargestellten Algorithmus berechnen. Unbeschränkte Stellen werden vom Programm durch -1 markiert.

```
ANFANGSMARKIERUNG.VORGAENGERMARKIERUNG = NULL;
ZUUNTERSUCHEN := { ANFANGSMARKIERUNG };
KNOTEN:= {};
KANTEN := {};
WHILE (ZUUNTERSUCHEN.SIZE > 0)
BEGIN
       MARKIERUNG := ZUUNTERSUCHEN.POP();
       KNOTEN.INSERT(MARKIERUNG);
       FOR (I = 1 TO NETZ.NUMTRANSITIONS)
       BEGIN
              IF (NETZ.TRANSITION[I].ISACTIVE(m))
              BEGIN
                     FOLGEMARKIERUNG := MARKIERUNG.FOLGEMARKIERUNG(NETZ.TRANSITION[i]);
                     UEBERDECKT := MARKIERUNG;
                     WHILE ((UEBERDECKT != NULL) AND !(UEBERDECKT <= FOLGEMARKIERUNG))
                     BEGIN
                            UEBERDECKT := UEBERDECKT.VORGAENGERMARKIERUNG;
                     END;
                     IF (UEBERDECKT != NULL)
                     BEGIN
                            FOLGEMARKIERUNG := FOLGEMARKIERUNG
                                          + (FOLGEMARKIERUNG – UEBERDECKT)*(-1);
                     END;
                     KANTEN.INSERT(ARC(MARKIERUNG,FOLGEMARKIERUNG,NETZ.TRANSITION[i]));
                     FOLGEMARKIERUNG.VORGAENGERMARKIERUNG := m;
                     IF (!(KNOTEN.CONTAINS(FOLGEMARKIERUNG) OR
                            ZUUNTERSUCHEN.CONTAINS(FOLGEMARKIERUNG)))
                     BEGIN
                            ZUUNTERSUCHEN.PUSH(FOLGEMARKIERUNG);
                     END;
              END;
       END;
END;
```

Überprüfung der Eigenschaft „soundness“ durch Analyse des Überdeckungsgraphen

Durch Analyse des Überdeckungsgraphen eines mit einem Workflow-Netz korrespondierenden „short-circuited“-Netzes lässt sich die „soundness“ Eigenschaft für das ursprüngliche Workflow-Netz nachweisen.

Die erste Voraussetzung hierfür, die Beschränktheit, ist immer dann gegeben wenn der Überdeckungsgraph keine Markierungen mit unbeschränkten Stellen enthält. Für einen gegebenen Überdeckungsgraph kann die Beschränktheit somit durch den folgenden Algorithmus nachgewiesen werden:

```
BESCHRAENKT := TRUE;
FOR (I = 1 TO GRAPH.NUMMARKIERUNGEN)
BEGIN
       MARKIERUNG := GRAPH.MARKIERUNG[I];
       FOR (I = 1 TO MARKIERUNG.NUMSTELLEN)
       BEGIN
              IF (MARKIERUNG.STELLE[I]==-1)
              BEGIN
                     BESCHRAENKT := FALSE;
              END;
       END;
END;
```

Bei beschränkten markierten Petri-Netzen entspricht der Überdeckungsgraph dem Erreichbarkeitsgraphen, die über Erreichbarkeitsgraphen gewonnenen Erkenntnisse können in diesem Fall also auch auf den Überdeckungsgraph angewendet werden.

Die zweite Voraussetzung, die Lebendigkeit eines markierten Petri-Netzes, kann aus dem Überdeckungsgraph ebenfalls ermittelt werden. Lebendigkeit ist dann gegeben, wenn

1. das markierte Petri-Netz reversibel ist, also die Anfangsmarkierung von jeder Folgemarkierung aus erreichbar ist.
2. für alle Transitionen des Petri-Netzes im Überdeckungsgraphen mindestens eine korrespondierende Kante existiert, die Transition also unter mindestens einer Markierung des Überdeckungsgraphen schaltbar ist.

Die Reversibilität ist wie bereits erläutert dann gegeben, wenn der Überdeckungsgraph stark zusammenhängend ist. Ein Algorithmus zur Überprüfung der Eigenschaft „stark zusammenhängend“ eines gerichteten Graphen wurde bereits in Kapitel 3.2.3 für Petri-Netze vorgestellt. Er kann auch auf Überdeckungsgraphen unverändert angewendet werden.

Die zweite Bedingung begründet sich wie folgt:

Voraussetzung: Sei N ein markiertes Petri-Netz. Sei M die Menge aller Markierungen (Knoten) des Überdeckungsgraphen von N. Es existiert für alle Transitionen $t \in N$ mindestens eine Markierung $m \in M$ die t aktiviert. Außerdem ist N reversibel.

Behauptung: Das markierte Petri-Netz N ist lebendig.

Beweis:

- Die Markierung m ist durch die Reversibilität von jeder anderen Markierung m aus M erreichbar ($\forall m \in M : M = [m >$).
- Nach Definition ist Transition t folglich unter keiner Markierung $m \in M$ tot, da es durch die Reversibilität mindestens eine Folgemarkierung gibt, die t aktiviert. Damit ist t lebendig.
- Da nach Voraussetzung alle t lebendig sind , sind nach Definition alle Markierungen $m \in M$ lebendig.
- Die Anfangsmarkierung m_0 ist in M enthalten ($m_0 \in M$). Nach Definition ist ein markiertes Petri-Netz dann lebendig, wenn seine Anfangsmarkierung lebendig ist. **q.e.d.**

Unter Zuhilfenahme moderner Rechner lassen sich Überdeckungsgraphen für überschaubare Petri-Netze problemlos erstellen. Komplexere Workflow-Netze lassen sich oft mit Hilfe von Subprozessen auf sehr übersichtliche Petri-Netze abbilden. Um jedoch den Überdeckungsgraphen nicht unnötigerweise aufzubauen bietet es sich dennoch an, durch vorherige Auswertung der Struktureigenschaften mögliche Fehlerquellen, die die „soundness“ beeinträchtigen schon im Vorfeld zu erkennen. In vielen Fällen lässt sich „soundness“ bereits durch die strukturellen Eigenschaften eines Workflow-Netzes ausschließen (siehe Abbildung 13 am Anfang dieses Kapitels).

4 Softwareengineering und Workflowmanagement im Vergleich

Im Vergleich zu ausführbaren Programmen als Endergebnis der Softwareentwicklung sind Geschäftsprozessmodelle zwangsläufig ungenauer. Während der Unterschied in den anfänglichen Stufen der Modellierung noch nicht so deutlich sein mag so ist doch verständlich, dass der Zweck eines Geschäftsprozessmodells, egal wie detailliert, stets eine vereinfachte Darstellung der Realität bleibt. Die Abläufe sind für eine Automatisierung zu komplex und erfordern das Mitwirken von Personal. Das Ziel des Softwareentwicklungsprozesses dagegen ist eine vollständige Beschreibung der in einem Programm stattfindenden Abläufe, aus der sich dann automatisiert mit geeigneten Werkzeugen das lauffähige Produkt erstellen lässt. Dieser Grad an Automatisierung ist bei der Implementation von Geschäftsprozessen nicht erreichbar. Um Fehler bereits vor der Umsetzung eines Geschäftsprozesses in der Realität finden zu können, muss die Analyse des Geschäftsprozesses auf Basis von Modellen erfolgen.

4.1 Vom Entwurf zur Umsetzung

In der Praxis existieren vielfältige Methoden und Ansätze, um Softwareentwicklung und Geschäftsprozessmodellierung zu koordinieren. Während iterative Verfahren wie „Scrum“ in sich ständig wiederholenden Zyklen zwischen den verschiedenen Entwurfsstufen der Modellierung wechseln sind insbesondere bei exakt planbaren Projekten Verfahren, bei denen verschiedene Arbeitsschritte aufeinander aufbauen, weiterhin sehr beliebt. Für den im folgenden angestrebten Vergleich zwischen Geschäftsprozessmodellierung und Softwareentwicklung wird das in der Softwareentwicklung sehr weit verbreitete Wasserfallmodell zur Einteilung des Modellierungsprozesses in verschiedene Phasen herangezogen.

Nach dem Wasserfallmodell unterteilt sich der Entwicklungsprozess in fünf Schritte:

1. **Die Anforderungsanalyse.** Hier werden die Anforderungen an das zu entwickelnde System festgelegt.
2. **Systemdesign.** Es wird zunächst ein High-Level Design des zu entwickelnden Systems modelliert. Das so entstandene Modell wird iterativ verfeinert, indem Teilvorgänge immer detaillierter formuliert werden.
3. **Programmierung und Tests von Teilsystemen.** An dieser Stelle werden die Teilsysteme des Modells in die Praxis umgesetzt und in Programmform konkretisiert. Auch finden in diesem Schritt in der Regel die ersten Tests der Teilsysteme statt.
4. **Integrations- und Systemtests.** Hier wird das Zusammenspiel der Teilkomponenten des Systems gestestet.
5. **Einsatz und Wartung.** Das System wird produktiv eingesetzt und regelmäßig gewartet.

Das Wasserfallmodell in seiner ursprünglichen Form hat seine Schwächen. Insbesondere beim Einsatz in der Softwareentwicklung ist es oft nicht möglich, eine Phase komplett abzuschließen bevor eine neue Phase beginnt. So sieht das Wasserfallmodell keine Testphase für das in Phase 2 entstandene Systemdesign vor. Oftmals lassen sich jedoch in einer vorausgegangenen Phase gemachte Designfehler nicht mehr in der Nachfolgephase ausgleichen sondern erfordern das erneute Durchlaufen der vorausgegangenen Phase.

Bei der Einführung eines neuen Geschäftsprozesses ist die Lage noch deutlich prekärer. Es ist ökonomisch wenig sinnvoll, eine ganze Firma an einem neuen Geschäftsprozessdesign auszurichten, ohne dass dessen Umsetzbarkeit zumindest am Modell nachgewiesen wurde. Das Hinauszögern der ersten Tests bis nach der konkreten Realisierung der jeweiligen Teilsysteme ist bei der Geschäftsprozessmodellierung also nicht machbar. Eine integrierte Entwicklungsumgebung für Geschäftsprozessmodelle macht es also erforderlich, dass eine möglichst genaue Verifikation des zu modellierenden Geschäftsprozesses bereits in den verschiedenen Schritten der Modellierung in Phase 2 des Wasserfallmodells möglich ist. Abbildung 16 stellt diesen Sachverhalt grafisch dar.

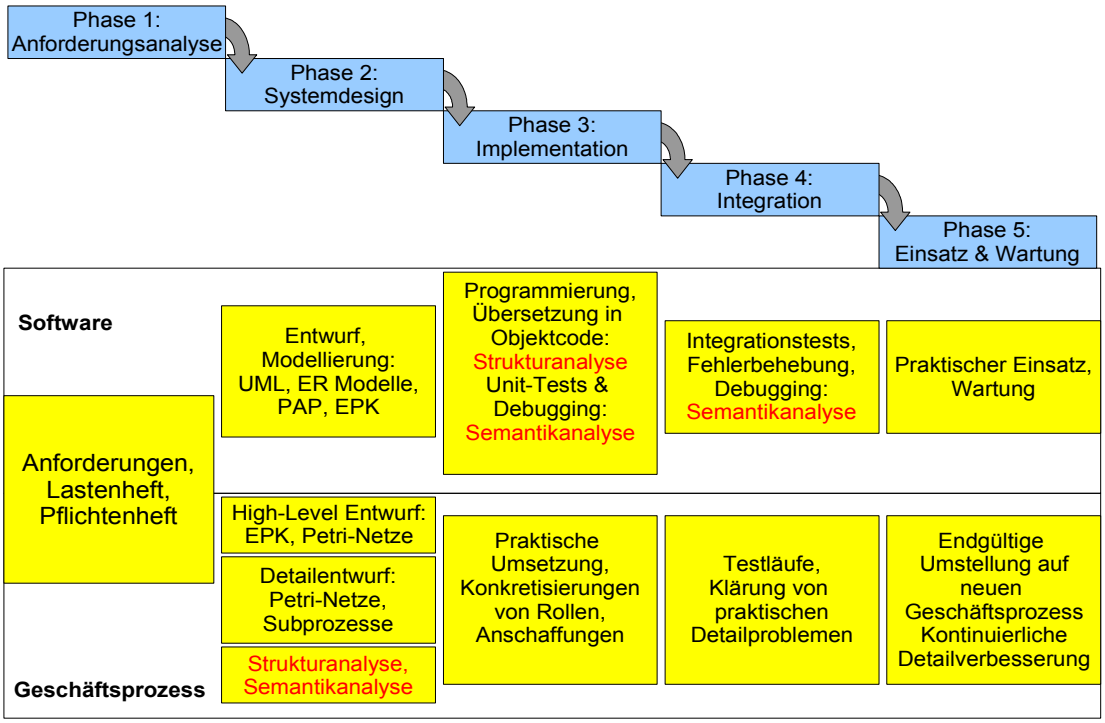

Abbildung 16: Das Wasserfallmodell bei Geschäftsprozessmodellierung und Softwareentwicklung

Der Entwurf komplexer Geschäftsprozesse ist ein iterativer Vorgang, bei dem immer detaillierter modelliert wird. Je detaillierter das Modell wird, desto höher sind die Anforderungen an die formale Korrektheit des Modells. So ist es oftmals sinnvoll, den ersten High-Level Entwurf eines Geschäfts- oder Teilprozesses mit Hilfe der in Kapitel 3.1 vorgestellten EPK zu formulieren. Da EPK jedoch formal ungenau definiert sind ist weder eine strukturelle noch eine semantische Analyse in dieser ersten Entwurfsphase möglich. Im weiteren Verlauf wird daher das EPK-Modell in eine formal eindeutige Form gebracht. Diese lässt sich dann 1:1 in ein Petri-Netz übersetzen, welches später sowohl strukturell als auch semantisch analysiert werden kann.

Einen ähnlichen iterativen Vorgang gibt es auch bei der Softwareentwicklung. Während jedoch die zunehmende Detaillierung bei der Geschäftsprozessmodellierung immer ein Bestandteil der Systemdesignphase ist, wird die zu entwickelnde Software beginnend mit der Entwurfsphase bis zum Ende der Implementationsphase immer weiter konkretisiert. In der Designphase kommen diverse Modelle zum Einsatz, um den Ablauf, das Datenlayout und andere Aspekte der Software zu

beschreiben. Beliebte Modelle hierfür sind die Unified Modelling Language (UML), sowie Datenflussdiagramme und Programmablaufpläne (PAP). Für die konkrete Implementation steht eine Vielzahl an Programmiersprachen zur Verfügung, die in unterschiedlichstem Umfang von der Plattform, auf der die Software eingesetzt werden soll, abstrahieren. In vielen Fällen ist der so entstandene Quelltext nicht direkt einsetzbar, sondern muss zunächst in einem automatisierten Prozess in eine auf der Zielplattform ausführbare Forme gebracht werden. Während dieses Vorgangs, der als Übersetzung oder Compilierung bezeichnet wird, wird die Software oft zum ersten Mal strukturell analysiert. Abhängig von der Zielplattform schließen sich an die Übersetzung weitere Schritte wie z.B. das Linken, also das Zusammenbinden einzelner Programmteile zu einem ausführbaren Programmpaket an. Erst danach kann die Software oder ein Teil davon zur Ausführung gebracht und im weiteren Verlauf semantisch analysiert werden. Sehr oft ist eine vernünftige semantische Analyse sogar erst in Phase 4 des Wasserfallmodells, während der sog. Integrationstests möglich.

4.2 Strukturüberprüfung

Ziel der strukturellen Überprüfung ist es, Fehler zu finden, die unabhängig von konkreten Anfangszuständen eines Systems immer auftreten. Idealerweise werden neben echten Fehlern auch Designschwächen und potentielle Probleme bereits bei der Strukturüberprüfung erkannt. Aus diesem Grund lassen sich die Ergebnisse der strukturellen Überprüfung in Warnungen und Fehler klassifizieren.

4.2.1 Strukturelle Überprüfung von Geschäftsprozessmodellen in der Praxis

Ist ein Geschäftsprozess oder Teilprozess formal ausreichend genau definiert, um als Workflow-Netz darstellbar zu sein, so kann er mittels der in Kapitel 3.2.3 ausführlich beschriebenen strukturellen Analysemethoden evaluiert werden. Eine integrierte Entwicklungsumgebung für Workflow-Netze muss dabei nicht nur auf potentielle und definitive Probleme des Workflow-Netzes hinweisen, sondern nach Möglichkeit auch auf konkrete Stellen innerhalb des Workflow-Netzes verweisen, die eine gewünschte oder notwendige Eigenschaft verletzen.

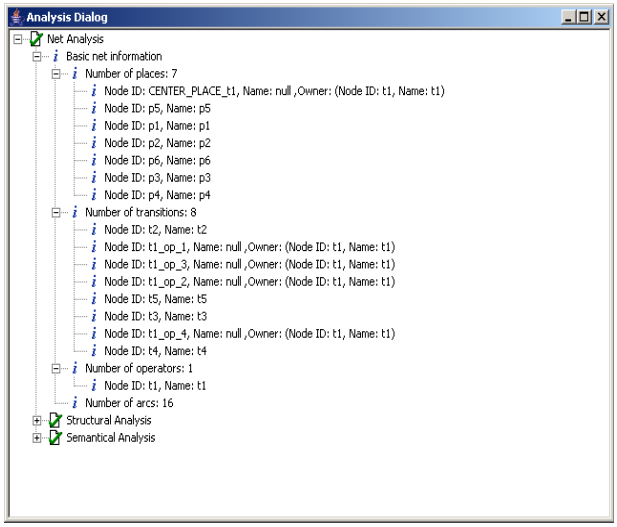

Abbildung 17: Statistische Angaben zum aktuellen Modell

Die in der Software WoPeD implementierten Funktionen zur strukturellen Analyse bieten zunächst eine statistische Übersicht der in einem Workflow-Netz verwendeten Elemente (siehe Abbildung 17).

Bei der strukturellen Analyse wird dann zunächst überprüft, ob das zu analysierende Petri-Netz überhaupt ein Workflow-Netz ist, also einen Geschäftsprozess mit einem definierten Anfangs- und Endereignis, darstellt. (siehe Abbildung 18).

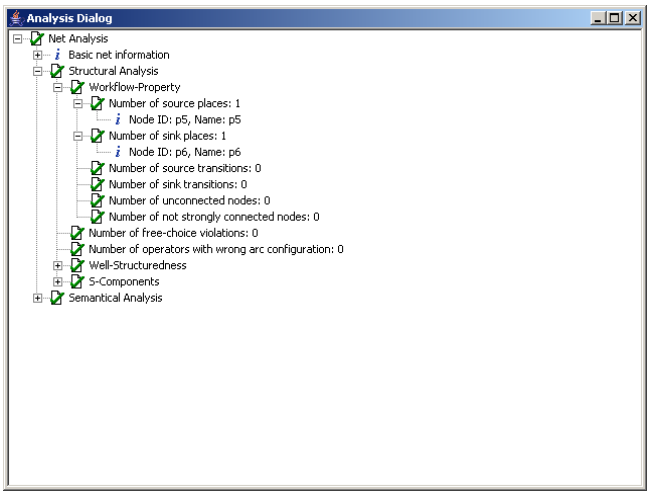

Abbildung 18: Überprüfung der "Workflow" Eigenschaft

Die übrigen strukturellen Eigenschaften sind für die „soundness“ des Workflow-Netzes nicht bindend. Jedoch weist jeder einzelne Analysepunkt auf potentielle Designprobleme des Workflow-Netzes hin. So ist es generell wünschenswert, dass ein Workflow-Netz entweder die „well-structured“ oder die „free-choice“ Eigenschaft besitzt oder zumindest „s-coverable“ ist (siehe Abbildung 19).

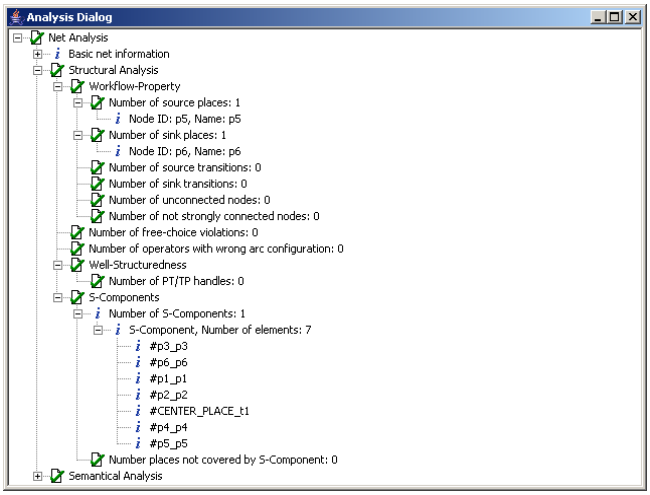

Abbildung 19: "Free-Choice", "Well-Structured" und "S-Coverable"

Die Software WoPeD unterstützt die in [aalshee] vorgeschlagene Alternativnotation für AND-split, XOR-split, AND-join and XOR-join sowie zwei zusammengesetzte Operatoren XOR-split-join und AND-split-join (siehe Abbildung 20). Als Bestandteil der Strukturanalyse eines Workflow-Netzes wird geprüft, ob diese Operatoren eine sinnvolle Anzahl eingehender und abgehender Verbindungen aufweisen.

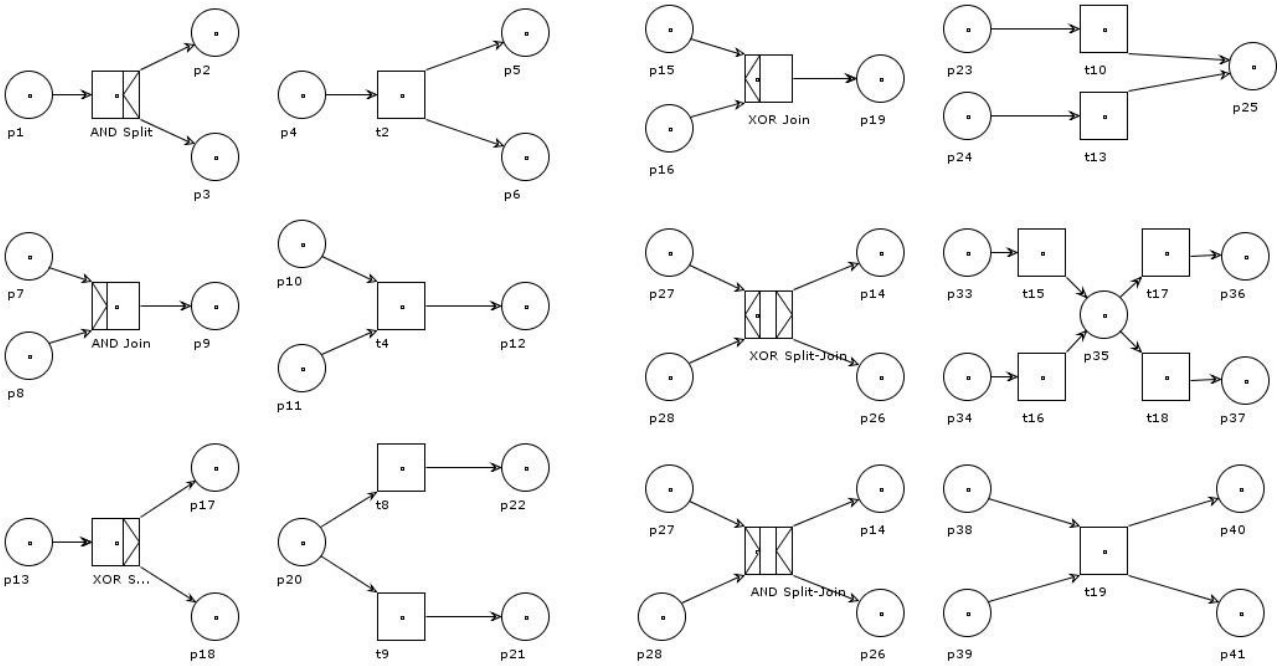

Abbildung 20: Alternativnotationen in WoPeD

Alle Fehler und potentiellen Fehler, die während der syntaktischen Analyse gefunden werden, werden von WoPeD im Workflow-Netz grafisch hervorgehoben. Hierzu werden Gruppen von Transitionen, Stellen und Operatoren gebildet, die mit dem Auftreten des Fehlers in Zusammenhang stehen. Diese werden im Analysedialog dargestellt. Wird ein Knoten oder eine Knotengruppe des Modells im Analysedialog ausgewählt, so werden die korrespondierenden Elemente des Modells grafisch hervorgehoben.

4.2.2 Strukturelle Überprüfung von Computersoftware

Wie bereits erwähnt wird eine erste strukturelle Überprüfung von Computersoftware meist erst durch den Compiler durchgeführt. Compiler übersetzen in einer bestimmten, durch eine sog. Grammatik beschriebenen Programmiersprache verfassten Quelltext in eine auf der Zielplattform ausführbare Form. Die Grammatik einer Programmiersprache gibt die notwendigen Struktureigenschaften des Quelltextes vor. Wird die vorgegebene Struktur nicht eingehalten kann der Quelltext in aller Regel nicht übersetzt werden. Der Compiler verweist in diesem Fall auf den Teil des Quelltextes, der nicht der vorgegebenen Grammatik entspricht und bricht die Übersetzung mit einer Fehlermeldung ab (siehe Abbildung 21).

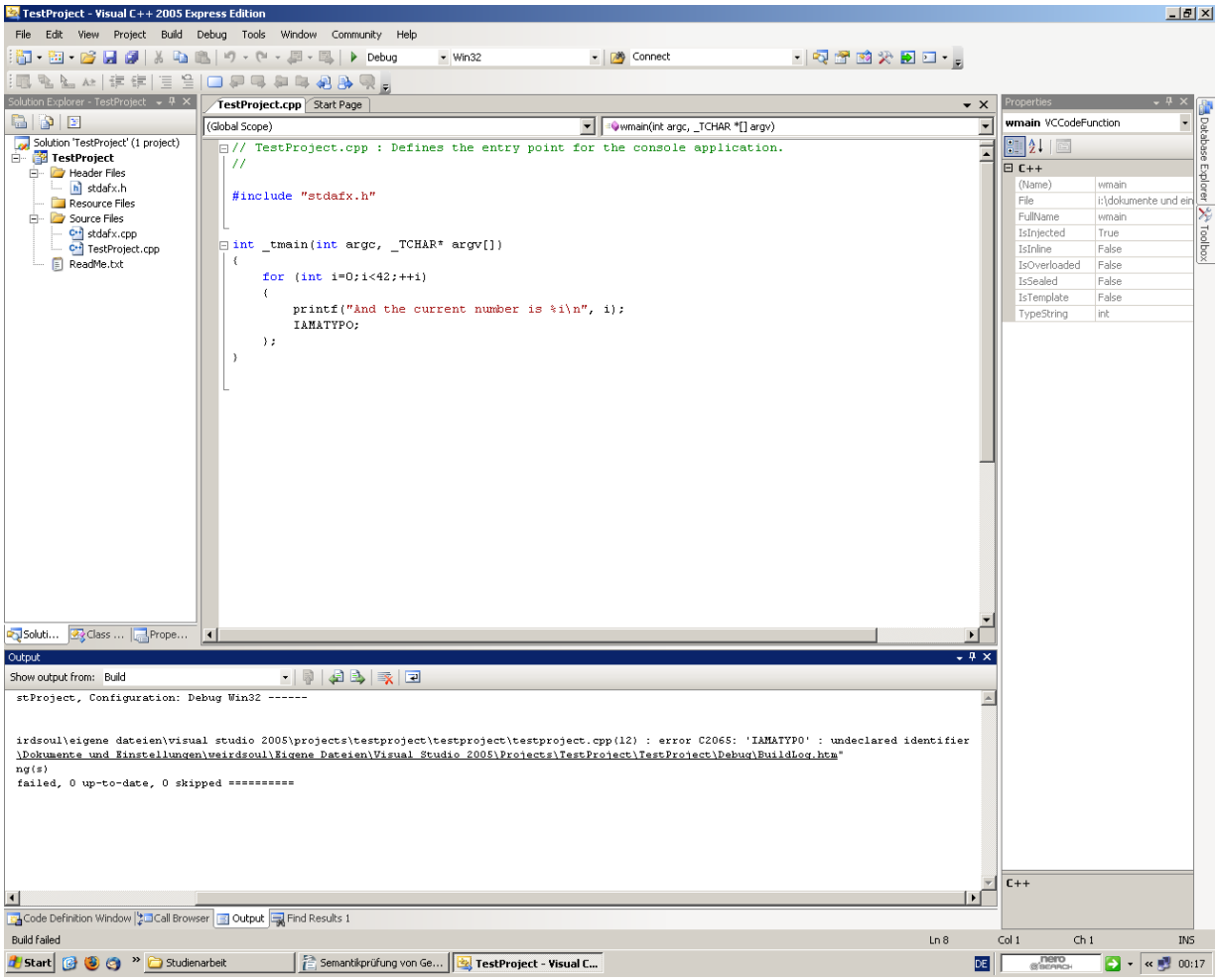

Abbildung 21: Compilerfehlermeldung als Teil der Strukturanalyse von Software Quelltext

Ähnlich der Voraussetzung für die „soundness“-Eigenschaft bei Workflow-Netzen, dass es keine sinnlosen, also nicht benutzte Transitionen in einem Workflow-Netz geben darf, weisen viele Compiler während der Übersetzung mit einer Warnung auf Quelltextteile hin, die den Programmablauf nicht beeinflussen. Hierzu gehören z.B. Variablen, die zwar deklariert, jedoch nie benutzt werden. Oftmals kommen Alternativen nie zur Ausführung, weil die zugehörige Verzweigung des Kontrollflusses unabhängig vom Programmzustand diese Alternative nie berücksichtigt. Entsprechender Programmcode wird von vielen Compilern erkannt und eine entsprechende Warnung generiert. Auf diese Weise können teilweise sogar endlose Rekursionen erkannt werden.

4.3 Debugging

> „Wenn Debugging der Vorgang ist, Fehler aus einem Programm auszubauen, dann ist Programmieren der Vorgang, Fehler einzubauen. „ (Ursprung unbekannt)

Debugging wird stets mit Instanzen eines Programms oder Workflows durchgeführt, also einer konkreten, zustandsbehafteten Ausprägung eines Programms oder Workflows. Der Zustand eines Petri-Netzes wird durch seine Markierung ausgedrückt (siehe Kapitel 3.2.4), während die Instanz eines Programms typischerweise als Prozess bezeichnet wird. Der Zustand eines Prozesses ergibt sich hier durch die im Datenspeicher sowie auf dem Stapel und in den Prozessorregistern hinterlegten Informationen.

Beim Debugging wird ein durch ein Programm oder eine Prozessbeschreibung und einen definierten Ausgangszustand vorgegebener Ablauf analysiert, indem er Schritt für Schritt durchlaufen wird. Dabei ist es vor jedem Schritt möglich, den exakten Zustand des Gesamtsystems zu analysieren und Rückschlüsse zu ziehen. Durch die Steuerung von externen Ereignissen, die den Prozessablauf beeinflussen, lassen sich beispielsweise bekannte Fehlersituationen reproduzieren. Die Rückführung eines Problems in einem realen Ablauf auf seine exakte Ausgangssituation bezeichnet man auch als Reproduzierbarkeit. Sie ist unmittelbare Voraussetzung für die nachhaltige und effektive Problembehebung. Kennt man die Voraussetzungen für ein Problem nicht genau, so ist die Wahrscheinlichkeit groß, dass jeder Versuch, dieses Problem zu beheben nur Symptome bekämpft. Mögliche Folgen sind das erneute Auftreten des Problems in einem leicht veränderten Kontext oder aber die Entstehung von Kollateralschäden durch die zur Problembehebung vorgenommenen Änderungen am Ablauf.

4.3.1 Debugging durch semantische Analyse

Können sowohl das auftretende Problem als auch der Ablauf formal beschrieben werden, so kann es sinnvoll sein, alle möglichen Folgezustände eines definierten Ausgangszustandes und einer Prozessbeschreibung auf das Auftreten des Problems hin zu untersuchen. Diese automatisierte Suche nach Problemen wird im Folgenden als semantische Analyse bezeichnet.

Semantikanalyse von Geschäftsprozessen in der Praxis

Die Möglichkeiten zur semantischen Analyse von Petri-Netzen wurden bereits in Kapitel 3.2.4 beschrieben. Die Aufgabe einer integrierten Entwicklungsumgebung ist es auch hier, dem Benutzer Probleme, die während der Semantikanalyse gefunden wurden, möglichst genau darzustellen. Nach Möglichkeit sollen die Teile des Modells, die für das Problem verantwortlich zeichnen, grafisch hervorgehoben werden. Die Software WoPeD bedient sich für die semantische Analyse von Petri-Netzen einer Hilfsbibliothek, welche die Analysefunktionalität der Software Woflan enthält. Die Rückmeldungen der Bibliothek werden von WoPeD ausgewertet und in einem Analysedialog dargestellt. Die Ermittlung der T- und S-Invarianten gehört hierzu ebenso wie die Überprüfung der Beschränktheit und Lebendigkeit des mit dem modellierten Netz korrespondierenden „short-circuited"-Netzes. Stellen und Transitionen die zu einem durch die Semantikanalyse gefundenen Fehler führen werden im Analysedialog angezeigt. Wie bei der Struktureanalyse bewirkt die Selektion auf diese Weise dargestellter Knoten die Markierung des korrespondierenden Elements im Modell (siehe Abbildung 22).

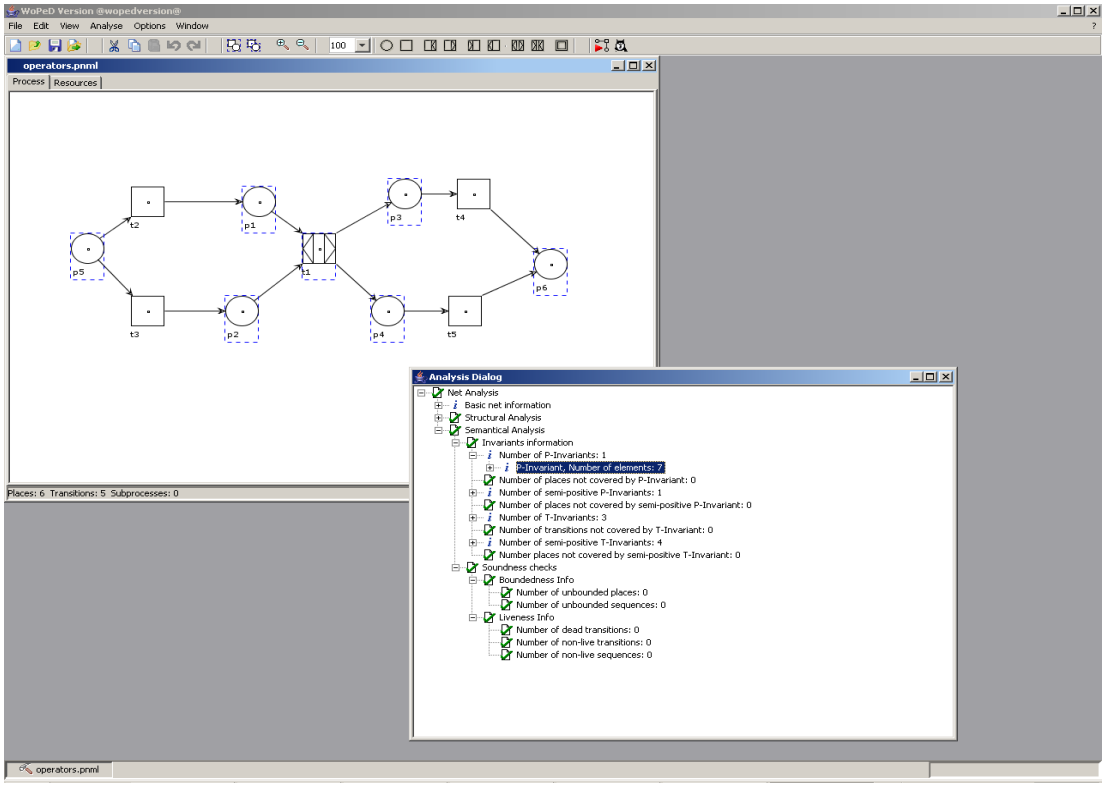

Abbildung 22: Markierung der im Analysedialog ausgewählten Knoten im Modell

Semantikanalyse von Computersoftware

Automatisierte Semantikanalyse ist bei Computersoftware praktisch kaum möglich. Allenfalls lassen sich besonders wichtige Teilaspekte eines Systems auf ihre gewünschte Funktionalität hin überprüfen, indem sog. Unit-Tests durchgeführt werden. Da Unit-Tests jedoch für jedes Modul neu entworfen und umgesetzt werden müssen und oftmals eine manuelle Koordination der Tests erfordern sind diese eher in der Ablaufanalyse anzusiedeln und werden daher in Kapitel 4.3.2 genauer erläutert.

Der Grund dafür, dass eine automatisierte Semantikanalyse von Computersoftware kaum umsetzbar ist, liegt in der hohen Komplexität und dem damit verbundenen enorm großen Zustandsraum von Computersoftware. Schon das einfache Einlesen einer Datei führt zu einer schier unendlich großen Anzahl möglicher Ausgangskombinationen, die sich durch die möglichen in der Datei gespeicherten Bitfolgen ergeben.

Aus diesem Grund ist es bei der Entwicklung von Computersoftware enorm wichtig, Designrichtlinien und sog. Best-practice rules zu beachten, die in vielen Fällen eine korrekte Ausführung von Software in den meisten Situationen sicherstellen. Ein bekannter Vertreter solcher Regeln sind Synchronisationsprotokolle wie das in der Datenbankwelt sehr beliebte Zwei-Phasen Sperrprotokoll.

4.3.2 Debugging durch Ablaufanalyse

Durch schrittweises Ausführen des Ablaufes und Analyse der Folgezustände ist es möglich, Probleme immer weiter einzugrenzen bis ihre Ursache gefunden wurde. Der wohl wichtigste Unterschied zur dynamischen Analyse ist, dass nur ein Teil der möglichen Zustände eines Systems betrachtet wird. Welche Teilzustände betrachtet werden sollen wird oft durch intuitive Entscheidungen des Benutzers bestimmt. Für diese Entscheidungen ist ein gewisses Verständnis des zu analysierenden Systems erforderlich, was eine automatisierte Analyse ausschließt. Oft wird darauf verzichtet, alle Folgezustände einzeln zu analysieren. Durch gezieltes Setzen von Unterbrechungspunkten auf Zustandswechsel kann die Ausführung an einem für die Analyse interessanten Punkt unterbrochen werden.

Beim Debugging von Geschäftsprozessen wird stets auf Basis einer Simulation gearbeitet. Bei der Simulation von Petri-Netzen wird meist im Einzelschrittmodus gearbeitet, da bei mehreren möglichen Folgezuständen ohnehin eine manuelle Entscheidung getroffen werden muss. Prinzipiell ist jedoch auch bei Petri-Netzen das Setzen von Unterbrechungspunkten sowie die automatisierte Ausführung von Teilen des Ablaufes denkbar. So lassen sich für viele Transitionen formale Bedingungen für ihre Schaltbarkeit angeben.

Das Debugging von Computersoftware findet oft unter Realbedingungen statt, also auf Basis echter Eingabedaten. Für die Analyse von Teilsystemen kann es jedoch sinnvoll sein, diese in einer Testumgebung ablaufen zu lassen.

Zerlegung von Systemen in Teilsysteme

Eine der größten Fehlerquellen für Computersoftware ist die Ressourcenverwaltung. Anders als bei Petri-Netzen, deren Zustand stets durch ihre Markierung eindeutig dargestellt werden kann ist es bei Computersoftware möglich, auf externe Ressourcen zuzugreifen. Der Begriff extern muss hierzu erst definiert werden, denn eine Rechnerarchitektur hat viele ineinander verschachtelte Systemgrenzen. Nehmen wir also an, die CPU des Rechners wäre das zu untersuchende System. Eine CPU hat eine feste Menge Speicher, die jedoch durch Interaktion mit den übrigen Rechnerkomponenten stets verändert wird. Es werden Daten ausgelagert, wieder eingelagert und dieser Vorgang erstreckt sich längst nicht mehr nur auf den der CPU angegliederten Hauptspeicher. Durch „Swap“-Bereiche auf persistenten Speichermedien wird dies besonders deutlich: Hier können Teile des Hauptspeichers sogar auf einem externen, über das Internet verbundenen Massenspeicher abgelegt werden. Der korrekte Ablauf eines Systems kann stets nur innerhalb seiner Systemgrenzen sicher überprüft werden. Strategien gegen dieses Problem sind die systematische Einstufung system-externer Daten als unsicher sowie die Definition von klaren Systemschnittstellen.

Je enger gesteckt die Systemgrenzen eines solchen Teilsystems sind, desto wahrscheinlicher wird es, dass sich seine Verhaltenseigenschaften wie in Kapitel 4.3.1 beschrieben durch semantische Analysemethoden ermitteln lassen.

Diese Vorgehensweise ist auch bei Work-Flow Definitionen nicht unüblich. Durch die Unterteilung eines Workflows in Teilaufgaben und deren Modellierung als Sub-Netze (siehe Kapitel 3.2.1) lässt sich eine Hierarchie von Teilsystemen entwerfen. Diese Teilsysteme können dann getrennt untersucht werden.

Untersuchung von Teilsystemen durch Unit-Tests

Die Untersuchung von (Teil-)Systemen auf ihr gewünschtes Verhalten in gegebenen Ausgangssituationen bezeichnet man als „Unit-Test“.

Wie bereits erklärt ist der entscheidende Unterschied der Ablaufanalyse zur semantischen Analyse, dass nur ein Teil des Zustandsraumes des zu untersuchenden Systems betrachtet wird. Es interessiert eine bestimmte Menge an Ausgangssituationen, die einem (Teil-)System Probleme bereiten könnten. Diese Ausgangssituationen zu finden ist nicht trivial. Mögliche Faktoren für die Auswahl zu untersuchender Ausgangsbedingungen sind:

- Bekannte Probleme mit der Komponente (Bug-Reports, User-Feedback, ...)
- Gezielte Untersuchung von Veränderungen am System durch Bug-Fixes, neue Features etc.
- Projekt-unabhängig häufig gemachte Fehler (Falscher Umgang mit Ressourcen, Falsche Wertebereiche für Variablen, ...)
- Randbedingungen und selten auftretende Use-Cases, da diese bei der Entwicklung oft unzureichend berücksichtigt werden

Zu jeder gegebenen Ausgangssituation wird nun der gewünschte Endzustand des Teilsystems festgelegt. Das System wird nun auf korrektes Verhalten überprüft, indem die Endzustände für die im Vorfeld entworfenen Ausgangssituationen untersucht werden. Sind Anfangs- und Endzustand klar definiert, so lässt sich dieser Vorgang automatisieren. Dies erlaubt die kontinuierliche Sicherung der Qualität der (Teil-)Systeme eines komplexen Systems.

5 Ungenutzte Potentiale

Das Design und die Analyse von Geschäftsprozessen unterscheidet sich in vielen Aspekten deutlich von den in der Softwareentwicklung angewandten Methoden. Während bei Software aufgrund der fehlenden semantischen Analysemöglichkeiten die (teilweise halbautomatische) Ablaufanalyse besonders wichtig ist, müssen Geschäftsprozesse bereits während der Modellierungsphase eingehend getestet und analysiert werden.

Dies führt dazu, dass die einzelnen Analysemethoden in entsprechenden Entwicklungsumgebungen unterschiedlich detailliert und umfangreich umgesetzt wurden. So wäre es durchaus auch in der Softwareentwicklung wünschenswert, während der Designphase gemachte Fehler auch schon in der Designphase finden und beheben zu können, da Designfehler oftmals in der Implementationsphase nicht mehr behoben, aber dort erst gefunden werden.

Umgekehrt wäre es wünschenswert, wenn für Geschäftsprozesse noch mehr Hilfsmittel der Ablaufanalyse zur Verfügung stehen würden. Eine integrierte Entwicklungsumgebung könnte hier beispielsweise das Konzept der Unit-Tests auf Geschäftsprozesse und ihre Ergebnisse übertragen oder das Setzen von (bedingten) Unterbrechungspunkten während der Simulation unterstützen.

6 Literaturverzeichnis

[richstucbusi2004ubka] Richter-von Hagen, Cornelia und Stucky, Wolffried, Business-Process- und Workflow-Management, 2004

[Woflan] http://is.tm.tue.nl/research/woflan/, Stand 2006

[verbeek] Verbeek, Henricus M.W., Verification of WF-nets, 2004

[WoPeD] http://www.woped.org/, Stand 2006

[ihridisk1994ubka] Ihringer, Thomas, Diskrete Mathematik, 1994

[Aal97] D. Hauschildt, E. Verbeek, W. van der Aalst, WOFLAN: A Petri-net based Workflow Analyzer, 1997

[aalshee] Aalst, Wil van der and Hee, Kees Max van, Workflow management, 2002