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.................................................................3
2 Entwurf und Analyse von Geschäftsprozessmodellen..................................................4
  2.1 Modellierung von Geschäftsprozessen........................................................................4
  2.1.1 Der Begriff des Modells.........................................................................................4
  2.1.2 Formale Geschäftsprozesse..................................................................................4
  2.1.3 Aspekte der Modellierung von Geschäftsprozessen..............................................5
  2.2 Von der Idee zum Geschäftsprozess...........................................................................7
  2.2.1 Modellbildung......................................................................................................7
  2.2.2 Syntaktische Prüfung von Geschäftsprozessen......................................................8
    Merkmale der syntaktischen Analyse...........................................................................8
    Einschränkungen der syntaktischen Analyse............................................................8
  2.2.3 Semantische Prüfung von Geschäftsprozessen.......................................................9
    Merkmale der semantischen Analyse..........................................................................9
    Schwierigkeiten bei der semantischen Analyse.........................................................9
3 Sprachen und Tools zur Modellierung und Analyse von Geschäftsprozessen............11
  3.1 Ereignisgesteuerte Prozessketten (EPK)..................................................................11
  3.1.1 Definition und Syntax von EPK.............................................................................11
  3.1.2 Tools zur Modellierung von EPK..........................................................................13
  3.1.3 Syntaktische Überprüfbarkeit von EPK.................................................................14
  3.1.4 Semantische Überprüfbarkeit von EPK.................................................................14
  3.2 Petri-Netze.............................................................................................................15
    3.2.1 Formale Definition von Petri-Netzen.................................................................15
    3.2.2 Tools zur Modellierung und Analyse von Petri-Netzen.......................................18
    3.2.3 Syntaktische Überprüfbarkeit von Petri-Netzen.................................................19
      Die „Workflow“ Eigenschaft......................................................................................19
      Die „Free-Choice“ Eigenschaft................................................................................23
      Die Begriffe „Well-handled“, „Well-Structured“......................................................25
      Petri-Netze als Zustandsautomaten......................................................................28
      S-Invarianten...........................................................................................................29
      Berechnung der Eigenschaft „s-coverable“ mit Hilfe von S-Invarianten...............30
      T-Invarianten...........................................................................................................31
  3.2.4 Semantische Überprüfbarkeit von Petri-Netzen....................................................32
      Beschränkheit...........................................................................................................33
      Tot oder lebendig.......................................................................................................33
      Der Erreichbarkeitsgraph eines markierten Petri-Netzes..........................................33
      Überprüfung der Eigenschaft „soundness“ durch Analyse des Überdeckungsgraphen............................................................................................................36
4 Softwareengineering und Workflowmanagement im Vergleich............................38
  4.1 Vom Entwurf zur Umsetzung....................................................................................38
  4.2 Strukturüberprüfung...............................................................................................40
  4.2.1 Strukturelle Überprüfung von Geschäftsprozessmodellen in der Praxis............40
  4.2.2 Strukturelle Überprüfung von Computersoftware..............................................43
  4.3 Debugging.............................................................................................................44
    4.3.1 Debugging durch semantische Analyse............................................................45
      Semantikanalyse von Geschäftsprozessen in der Praxis........................................45
      Semantikanalyse von Computersoftware...............................................................46
    4.3.2 Debugging durch Ablaufanalyse.........................................................................47
      Zerlegung von Systemen in Teilsysteme..................................................................47
      Untersuchung von Teilsystemen durch Unit-Tests...................................................48
5 Ungenutzte Potentiale..............................................................................................49
6 Literaturverzeichnis..................................................................................................50
1 Einleitung

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.


2.1.2 Formale Geschäftsprozesse

2.1.3 Aspekte der Modellierung von Geschäftsprozessen


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).

Semantikprüfung von Geschäftsprozessmodellen

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 eines 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.
  ○ 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


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 Teilblüte in Beziehung zueinander gebracht. Ein beliebtes Mittel hierzu sind die Darstellung als Graph oder als Tabelle.

Für jedes Darstellungsmittel lassen sich nun Analysemethode 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.

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:

<table>
<thead>
<tr>
<th>Bestellschein</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artikeltyp</td>
</tr>
<tr>
<td>Art der Bezahlung</td>
</tr>
<tr>
<td>Verpackung als Geschenk</td>
</tr>
<tr>
<td>Versandkosten</td>
</tr>
<tr>
<td>Bestellwert</td>
</tr>
<tr>
<td>Versicherter Versand</td>
</tr>
</tbody>
</table>

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

\[
|Moeglichkeiten| = \prod_{x = 1}^{Anzahl Variablen} |W(Var_x)|
\]
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 $4 \times 3 \times 2 \times 3 \times 3 \times 2 = 432$ 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)


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](image)

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 Startereignisses 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 kompliziertere 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](image-url)
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

3.2 Petri-Netze


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


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: $K \subseteq (S \times T) \cup (T \times S)$

**Abbildung 6: Ein typisches Workflow Petri-Netz**
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: $|\cdot x| > 1$


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.


*Abbildung 7: Einbindung eines Subprozesses*
3.2.2 Tools zur Modellierung und Analyse von Petri-Netzen


Die Software WoPeD wurde im Rahmen dieser Arbeit um eigene Strukturanalyseschäften 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.

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:


![Abbildung 8: Nicht zusammenhängendes Petri-Netz](image-url)
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 enthält keine Verbindungen zwischen Elementen der Knotenmenge $X_1$ und $X_2$: $F \subseteq (X_1 \times X_1) \cup (X_2 \times X_2)$


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 $(z_1, z_2), (z_2, z_3), \ldots, (z_{n-1}, z_n) \in F$ 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.

<table>
<thead>
<tr>
<th>Anfangszustand</th>
<th>Aktion 1</th>
<th>Zwischenzustand</th>
<th>Aktion2</th>
<th>Endzustand</th>
<th>Prozess neu starten</th>
<th>Erzeuger</th>
<th>Speicher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anfangszustand</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Aktion 1</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Zwischenzustand</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Aktion 2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Endzustand</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Prozess neu starten</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Erzeuger</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Speicher</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

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


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 $\infty$ interpretiert. Alle Knoten der Petri-Netzes seien ferner von 1 bis N durchnumeriert 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

$$T[i,j] = \begin{cases} 
0 & \text{für } (i=j) \\
-1 & \text{sonst}
\end{cases}$$
Dann wird der folgende iterative Algorithmus angewendet:

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

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 \( \cdot t^* = o \land t^* . = i \) 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 \( \infty \) markiert.

Zu jedem Workflow-Netz lässt sich ein korrespondierendes um die Transition \( t^* \) mit \( \cdot t^* = o \land t^* . = i \) erweitertes Netz konstruieren. Im weiteren Verlauf wird dieses Netz als „short-circuited“-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 [richstuebusi2004ubka] wie folgt ausdrücken:

1. \( \forall t_1, t_2 \in T : t_1 \cap t_2 \neq \emptyset \Rightarrow t_1 = t_2 \)
2. \( \forall s_1, s_2 \in S : s_1 \cap s_2 \neq \emptyset \Rightarrow s_1 = s_2 \)

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:

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

Im Fall von parallelen Vorgängen besteht der Synchronisationspunkt aus einer rückwärts verzweigten 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 10: Alternative Ausführung (Konflikt) und anschließende Synchronisation

Abbildung 11: Alternative Ausführung (Parallelität) 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 \( x \in T \land y \in S \) oder \( x \in S \land y \in T \) keine zwei Pfade von \( x \) nach \( y \) existieren, die nur \( x \) und \( y \) gemeinsam haben.


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 (\( (|S| > 1) \lor (|T| > 1) \)). 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 \( I \) erreichbaren 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.

---

**Abbildung 11**: Parallele Ausführung und anschließende Synchronisation
Der folgende Pseudo-Code verdeutlicht den beschriebenen Algorithmus:

```pseudo
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.

**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: \( \forall t \in T : |\cdot t| = |t \cdot| = 1 \)


- 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: \( \forall s \in S \exists S\text{-Komponente } N' = (S', T', F') \text{ von } N : s \in S' \)


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

Semantikprüfung von Geschäftsprozessmodellen Seite 28
**S-Invarianten**


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

\[
\sum_{i=1}^{n} y_i \cdot m(s_i) = constant \quad \forall m \in [m_0] >
\]

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


- 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:

\[ t_1: \quad y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 = y_1 (s_1 - 1) + y_2 (s_2 + 1) + y_3 (s_3 + 1) + y_4 s_4 \]

\[ t_2: \quad y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 = y_1 s_1 + y_2 (s_2 - 1) + y_3 (s_3 - 1) + y_4 (s_4 + 1) \]

\[ t_3: \quad y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 = y_1 (s_1 - 1) + y_2 s_2 + y_3 s_3 + y_4 (s_4 + 1) \]

\[ t_4: \quad y_1 s_1 + y_2 s_2 + y_3 s_3 + y_4 s_4 = y_1 (s_1 + 1) + y_2 s_2 + y_3 s_3 + y_4 (s_4 - 1) \]

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

\[ y_1 = y_2 + y_3 \]
\[ y_4 = y_1 \]

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

$$\forall p \in P: \sum_{i \in p} t_i = \sum_{i \in p} t_i$$

für beliebige Anfangsmarkierungen $m_0$ mit

- $P = \text{Die Menge aller Transitionen im Petri-Netz}$
- $t_i = \text{Gewichtung der Transition } i$

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.


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 Überprüfbarekeit von Petri-Netzen


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 \( N = (N, m_0) \):

\[
m(s) \leq b \quad \forall s \in S, \forall m \in [m_0].
\]


**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.


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 \( m_1 \) und \( m_2 \) wird eine gerichtete Kante erzeugt, wenn die Markierung \( m_2 \) von \( m_1 \) 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 \( (m, m', t) \in E \Rightarrow m \xrightarrow{t} m' \) (eine Kante verbindet zwei Markierungen \( m \) und \( m' \) aus \( M \) über eine Transition \( t \) aus \( T \))


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.

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 \( \omega \) notiert.

Der Überdeckungsgraph \( UEG(NETZ, m_0) = (\text{KNOTEN}, \text{KANTEN}) \) lässt sich nach dem folgenden in ähnlicher Form auch in [richtstucbusi2004ubka] dargestellten Algorithmus berechnen. Unbeschränkte Stellen werden vom Programm durch -1 markiert.

```plaintext
ANFANGSMARKIERUNG.VORGÄNGERMARKIERUNG = 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.VORGÄNGERMARKIERUNG;
            END;
            IF (UEBERDECKT != NULL)
            BEGIN
                FOLGEMARKIERUNG := FOLGEMARKIERUNG + (FOLGEMARKIERUNG - UEBERDECKT)*(-1);
            END;
            KANTEN.INSERT(ARC(MARKIERUNG, FOLGEMARKIERUNG, NETZ.TRANSITION[I]));
            FOLGEMARKIERUNG.VORGÄNGERMARKIERUNG := m;
            IF (!KNOTEN.CONTAINS(FOLGEMARKIERUNG) OR ZUUNTERSUCHEN.CONTAINS(FOLGEMARKIERUNG))
            BEGIN
                ZUUNTERSUCHEN.PUSH(FOLGEMARKIERUNG);
            END;
        END;
    END;
END;
```

Semantikprüfung von Geschäftsprozessmodellen
Überprüfung der Eigenschaft „soundness“ durch Analyse des Überdeckungsgraphen


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:

\[\begin{align*}
\text{BESCHRÄNKT} & := \text{TRUE}; \\
\text{FOR} \ (I = 1 \ \text{TO} \ \text{GRAPH.NUMMARKIERUNGEN}) \\
\text{BEGIN} & \text{MARKIERUNG} := \text{GRAPH.MARKIERUNG}[I]; \\
\text{FOR} \ (I = 1 \ \text{TO} \ \text{MARKIERUNG.NUMSTELLEN}) \\
\text{BEGIN} & \text{IF} \ (\text{MARKIERUNG.STELLE}[I] == -1) \\
& \text{BEGIN} \text{BESCHRÄNKT} := \text{FALSE}; \\
& \text{END}; \\
& \text{END}; \\
\text{END}; \\
\end{align*}\]

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 \( m \in M \) 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.

4 Softwareengineering und Workflowmanagement im Vergleich


4.1 Vom Entwurf zur Umsetzung


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.


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**


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

4.2 Strukturüberprüfung


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


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


Abbildung 20: Alternativnotationen in WoPeD
4.2.2 Strukturelle Überprüfung von Computersoftware


Abbildung 21: Compilerfehlermeldung als Teil der Strukturanalyse von Software Quelltext

Semantikprüfung von Geschäftsprozessmodellen
4.3 Debugging

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


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


Abbildung 22: Markierung der im Analysedialog ausgewählten Knoten im Modell
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.

4.3.2 Debugging durch Ablaufanalyse


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


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.

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


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

[richtucbusi2004ubka] RICHTER-VON HAGEN, CORNELIA UND STUCKY, WOLFFRIED, Business-
Process- und Workflow-Management, 2004

[verbeek] VERBEEK, HENRICUS M.W., Verification of WF-nets, 2004
based Workflow Analyzer, 1997
[aalshee] AALST, WIL VAN DER AND HEE, KEES MAX VAN, Workflow management,
2002