

Diplomarbeit

# Public-View-Generierung

Peter Laufer

4. November 2007



Humboldt-Universität zu Berlin  
Mathematisch-Naturwissenschaftliche Fakultät II  
Institut für Informatik

Gutachter:  
Prof. Dr. Wolfgang Reisig  
Prof. Dr. Karsten Wolf



## Vorwort

Wissenschaftlicher Forschung haftet stets das Risiko an, dass die Suche nach neuen Erkenntnissen nicht immer in dem Maße erfolgreich ist, wie man sich dies zu Beginn der Untersuchung wünscht. Nicht anders verhält es sich mit Diplomarbeiten, die Fragen der aktuellen Forschung zum Thema haben. So kann denn auch trotz professioneller Betreuung nicht garantiert werden, dass jede Arbeit in einem positiven Resultat ihren Abschluss findet. Daher musste diese Diplomarbeit leider zweimal geschrieben werden.

In der ersten Fassung entwickelte ich einen eigenen Ansatz, um einen Public View eines Service automatisch zu berechnen. Leider stellte sich im Laufe der Untersuchung heraus, dass mein Verfahren nicht allgemein anwendbar war. Trotz einiger Einschränkungen in Bezug auf die betrachteten Services, gelang es nicht, die Korrektheit des Verfahrens zu beweisen. Weitere Einschränkungen hätten jedoch ein sinnvolles Resultat der Arbeit unmöglich gemacht.

Parallel zu meinem Bemühen entwickelte Professor Wolf, anhand vergleichbar komplexer Überlegungen, ein allgemein anwendbares Verfahren zur Public-View-Generierung. Aufgrund der praktischen Relevanz der Thematik sollte dieses Verfahren möglichst rasch implementiert werden.

In Anbetracht meiner Situation entschied ich mich daher dafür, den Ansatz von Professor Wolf softwaretechnisch umzusetzen und im Vergleich mit einem existierenden Verfahren zur Public-View-Generierung auf seine praktische Anwendbarkeit hin zu überprüfen. Zusammen mit den von mir im Zuge der Entwicklung meines Ansatzes gesammelten Erfahrungen über Services, ihr Verhalten und dessen abstrakte Beschreibung als Public View, gelang es mir so, eine umfassende Abhandlung zum Thema Public-View-Generierung und ihren praktischen Einsatz zu erarbeiten. Das Resultat dieser Arbeit ist diese nunmehr zweite Fassung meiner Diplomarbeit.



## Zusammenfassung

Die Analyse und Optimierung der *Geschäftsprozesse* ist von enormer Bedeutung für den langfristigen Erfolg eines Unternehmens. Die Märkte sind heutzutage zunehmend global und befinden sich in stetem Wandel, was eine fortwährende Überprüfung und Anpassung der Prozesse innerhalb eines Unternehmens erfordert. Durch den rasanten technologischen Fortschritt und den vermehrten Einsatz von Computern lassen sich immer größere Teile von Prozessen automatisieren. Einen neuen Ansatz zur Realisierung der IT-Infrastruktur in Unternehmen stellt dabei die *service-orientierte Architektur* (SOA) dar. *Services*, die eine wohldefinierte Funktionalität in einem Netzwerk zur Verfügung stellen, lassen sich unter relativ geringem Aufwand zu neuen Prozessen kombinieren und bei Bedarf ersetzen, ohne dass eine kostenintensive Anpassung vorhandener Lösungen erforderlich ist. Über das Internet können Dienste als sog. *Web-Services* zur Verfügung gestellt werden und so in die Abläufe der Prozesse anderer Unternehmen integriert werden. Um Services für potentielle Interessenten bekannt zu machen, muss eine Beschreibung der Funktionalität bei einem Verzeichnisdienst (*service broker*) hinterlegt werden. Im Bereich der Veröffentlichung von Services in Verzeichnisdiensten bieten sich zwei Konzepte zur Abstraktion von unternehmensinternen Informationen der Prozesse an: der *Public View* und die *Bedienungsanleitung*. Während für beliebige Prozesse eine Bedienungsanleitung automatisch berechnet werden kann und es Algorithmen gibt, die auf Basis der Bedienungsanleitungen entscheiden können, ob zwei Prozesse problemlos miteinander interagieren können, muss ein Public View bisher noch von einem Entwickler per Hand modelliert werden. Eine solche Modellierung ist jedoch aufwendig, fehleranfällig und kann ohne eine anschließende Verifikation nicht garantieren, dass sich der ursprüngliche Prozess  $P$  und sein Public View  $P'$  in Bezug auf die Bedienbarkeit äquivalent verhalten. Wir wollen uns in dieser Arbeit daher mit Verfahren zur automatischen Public-View-Generierung befassen. Verschiedene Ansätze werden im Detail vorgestellt und miteinander verglichen. Anhand von Fallstudien werden wir die Stärken und Schwächen der Verfahren näher beleuchten und Empfehlungen zu deren Einsatz ableiten.



# Inhaltsverzeichnis

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Einleitung</b>  | <b>11</b> |
| 1.1      | Problemstellung . . . . .  | 11        |
| 1.2      | Stand der Forschung . . . . .  | 12        |
| 1.3      | Aufbau der Arbeit . . . . .  | 14        |
| <b>2</b> | <b>Hintergrund</b>   | <b>15</b> |
| 2.1      | Geschäftsprozesse . . . . .  | 15        |
| 2.2      | Petrinetze . . . . .   | 16        |
| 2.3      | Offene Workflow-Netze . . . . .  | 18        |
| 2.4      | Bedienungsanleitungen . . . . .  | 22        |
| 2.4.1    | Serviceautomaten . . . . .   | 22        |
| 2.4.2    | Übersetzung von oWFN in Serviceautomaten . . . . .   | 25        |
| 2.4.3    | Situationen und Wissen . . . . .   | 27        |
| 2.4.4    | Generierung von Bedienungsanleitungen . . . . .  | 29        |
| 2.5      | Private und Public View . . . . .  | 33        |
| <b>3</b> | <b>Verhaltensbasierte Public-View-Generierung</b>  | <b>37</b> |
| 3.1      | Über die kausalen Beziehungen zwischen den Nachrichten des Service zum Public View . . . . . | 38        |
| 3.1.1    | Idee des Verfahrens . . . . .  | 38        |
| 3.1.2    | Eine mögliche Umsetzung . . . . .  | 39        |
| 3.1.3    | Probleme eines sprachenbasierten Ansatzes . . . . .  | 44        |
| 3.2      | Durch Spiegelung der Bedienungsanleitung zum Public View . . . . .                           | 46        |
| 3.2.1    | Idee des Verfahrens . . . . .  | 46        |
| 3.2.2    | Der duale Serviceautomat . . . . .   | 47        |
| 3.2.3    | Vom dualen Serviceautomaten zum Public View . . . . .  | 48        |
| 3.2.4    | Stärken und Schwächen des Ansatzes . . . . .   | 52        |
| <b>4</b> | <b>Strukturbasierte Public-View-Generierung</b>  | <b>53</b> |
| 4.1      | Idee des Verfahrens . . . . .  | 53        |
| 4.2      | Transformationsregeln . . . . .  | 54        |
| 4.2.1    | Regeln zur Eliminierung . . . . .  | 54        |
| 4.2.2    | Regeln zur Fusion . . . . .  | 56        |
| 4.2.3    | Transformation am Beispiel . . . . .   | 58        |
| 4.3      | Grenzen der strukturbasierten Transformation . . . . .                                       | 59        |

|   |           |
|---|-----------|
| <b>5 Vergleich der Verfahren</b>                                    | <b>61</b> |
| 5.1 Implementationen . . . . .                                      | 61        |
| 5.1.1 Implementation des verhaltensbasierten Verfahrens . . . . .   | 61        |
| 5.1.2 Implementation des strukturbasierten Verfahrens . . . . .     | 68        |
| 5.2 Fallstudien . . . . .   | 71        |
| 5.2.1 Fallstudie I: Reisebuchung . . . . .                          | 72        |
| 5.2.2 Fallstudie II: Kreditvergabe . . . . .                        | 77        |
| 5.2.3 Fallstudie III: Ausstellung eines Personalausweises . . . . . | 82        |
| 5.3 Auswertung und Empfehlungen zum Einsatz der Verfahren . . . . . | 84        |
| <b>6 Zusammenfassung und Ausblick</b>                               | <b>87</b> |
| 6.1 Zusammenfassung . . . . .                                       | 87        |
| 6.2 Ausblick . . . . .  | 88        |
| <b>Literaturverzeichnis</b>   | <b>91</b> |

# Abbildungsverzeichnis

|      |   |    |
|------|---|----|
| 2.1  | Kinokarten-Service als oWFN $N_{K_P}$ . . . . .   | 19 |
| 2.2  | Ein Partner-oWFN $N_{K_R}$ des Kinokarten-Service $N_{K_P}$ . . . . .   | 20 |
| 2.3  | Komposition $N_{K_P} \oplus N_{K_R}$ der Partner-oWFN $N_{K_P}$ und $N_{K_R}$ . . . . .   | 21 |
| 2.4  | Serviceautomat $A_{K_P}$ des Kinokarten-Service $N_{K_P}$ . . . . .   | 23 |
| 2.5  | Partnerautomaten $A_{K_R}$ (a) und $A_{K_F}$ (b) des Serviceautomaten $A_{K_P}$ . . . . .   | 23 |
| 2.6  | Komposition $A_{K_P} \oplus A_{K_F}$ der Partnerautomaten $A_{K_P}$ und $A_{K_F}$ . . . . .   | 25 |
| 2.7  | Transformation eines oWFN $N$ in ein oWFN $N'$ mit internem Nachrichtenpuffer . . . . .   | 26 |
| 2.8  | Kanonischer Partner $S_K$ des Kinokarten-Service $A_{K_P}$ . . . . .  | 31 |
| 2.9  | Partnerautomaten $A_{K_R}$ (a) und $A_{K_F}$ (b) des Serviceautomaten $A_{K_P}$ und ihr Matching mit $S_K$ . . . . .  | 31 |
| 2.10 | Bedienungsanleitung des Kinokarten-Service $A_{K_P}$ . . . . .  | 32 |
| 2.11 | Private View des Kinokarten-Service (vereinfacht) . . . . .   | 34 |
| 2.12 | Public View des Kinokarten-Service . . . . .  | 35 |
|      |   |    |
| 3.1  | Bedienungsanleitung des Serviceautomaten $A_{K_R}$ . . . . .  | 40 |
| 3.2  | Service $N_M$ mit ambivalenten Nachrichtenbeziehungen . . . . .   | 42 |
| 3.3  | Elementare Teilnetze . . . . .  | 43 |
| 3.4  | Muster zur nebenläufigen Komposition von $n$ Teilnetzen . . . . .   | 43 |
| 3.5  | Services $N_1$ und $N_2$ mit gleicher Kommunikationssprache . . . . .   | 45 |
| 3.6  | Strukturgleiche Bedienungsanleitungen der Services $N_1$ und $N_2$ . . . . .  | 46 |
| 3.7  | Serviceautomat $P$ mit Bedienungsanleitung $OG_P$ und dualer Serviceautomat $P' = \overline{OG_P}$ mit Bedienungsanleitung $OG_{P'}$ . . . . .                  | 48 |
| 3.8  | 1. Transformationsschritt vom dualen Serviceautomaten zum Public View . . . . .   | 49 |
| 3.9  | Serviceautomat $P$ mit Bedienungsanleitung $OG_P$ und dualer Serviceautomat $P'$ nach dem 1. Transformationsschritt mit Bedienungsanleitung $OG_{P'}$ . . . . . | 50 |
| 3.10 | 2. Transformationsschritt vom dualen Serviceautomaten zum Public View . . . . .   | 51 |
|      |   |    |
| 4.1  | Regeln ET1, ET2, EP1 und EP2 zur Eliminierung von Netzelementen . . . . .   | 55 |
| 4.2  | Regel FSeq zur Fusion sequenzieller Transitionen . . . . .  | 56 |
| 4.3  | Regel FAlt zur Fusion alternativer Transitionen . . . . .   | 57 |
| 4.4  | Regel FPar zur Fusion paralleler Transitionen . . . . .   | 58 |
| 4.5  | Private View (a) und minimaler Public View (b) . . . . .  | 60 |
|      |   |    |
| 5.1  | Auszug aus dem Klassendiagramm zur Implementation der verhaltensbasierten Public-View-Generierung . . . . .   | 62 |

|      |  |    |
|------|--|----|
| 5.2  | Auszug aus dem Klassendiagramm zur Implementation der strukturba-<br>sierten Public-View-Generierung . . . . .                       | 69 |
| 5.3  | Strukturmuster der Reduktionsregel zur Fusion gleicher Plätze . . . . .  | 70 |
| 5.4  | Private View des Reisebuchungs-Service . . . . .   | 73 |
| 5.5  | Strukturbasiert generierter Public View des Reisebuchungs-Service . . . . .  | 75 |
| 5.6  | Minimaler Public View des Reisebuchungs-Service . . . . .  | 76 |
| 5.7  | Bedienungsanleitung (a) und verhaltensbasiert generierter Public-View-<br>Serviceautomat (b) des Reisebuchungs-Service . . . . .     | 77 |
| 5.8  | Positiver Kontrollfluss des Kreditvergabe-Service . . . . .  | 78 |
| 5.9  | Positiver Kontrollfluss des Kreditvergabe-Service zugeschnitten auf die<br>Kommunikation mit dem potentiellen Kreditnehmer . . . . . | 80 |
| 5.10 | Bedienungsanleitung (a) und Public-View-Serviceautomat (b) des modifi-<br>zierten Kreditvergabe-Service . . . . .                    | 82 |
| 6.1  | Toolchain zur Analyse und Public-View-Generierung von WS-BPEL-<br>Prozessen . . . . .  | 89 |

# 1 Einleitung

## 1.1 Problemstellung

Der langfristige Erfolg eines Unternehmens hängt wesentlich davon ab, inwieweit es immer wieder gelingt, sich den stetig verändernden Marktbedingungen optimal anzupassen.

Im Zuge der Globalisierung wächst das Feld potentieller Wettbewerber nahezu täglich an. Es ist daher nicht nur erforderlich innovative Produkte und Dienstleistungen zu erbringen, sondern auch bestehende Prozesse innerhalb des Unternehmens stets auf ihre Effizienz hin zu untersuchen und wenn möglich zu verbessern. Der rasante technologische Fortschritt führt dazu, dass immer größere Teile von Prozessen automatisiert und computergestützt durchgeführt werden können. Allerdings sind mit der Entwicklung und Einführung dieser Lösungen oft auch enorme Kosten verbunden. Es ist daher nur folgerichtig, dass ein Unternehmen, das sich flexibel auf die Dynamik des Marktes einstellen will, dieselbe Flexibilität auch von seiner IT-Infrastruktur und deren Softwarelösungen fordert.

In letzter Zeit versucht man vermehrt dieser Forderung mit einer service-orientierten Architektur (SOA) [Got00] zu begegnen. Möglichst kleine eigenständige Prozesse werden dabei als *Service* bereitgestellt und können später zu komplexen Geschäftsprozessen kombiniert werden. Unter einem Service versteht man dabei eine wohldefinierte Funktionalität, die über eine Schnittstelle (*interface*) bereitgestellt wird und über einen Namen (*ID*) identifiziert werden kann. Der Anbieter eines solchen Service (*service provider*) veröffentlicht eine Beschreibung seines Dienstes bei einem Verzeichnisdienst (*service broker*). Mögliche Interessenten (*service requestor*) können dann den Verzeichnisdienst nach der gewünschten Funktionalität durchsuchen und sich auf Wunsch an einen gefundenen Service binden, um diesen zu nutzen. Obwohl Services eigenständige Softwarekomponenten sind, arbeiten sie im Allgemeinen nicht in Isolation. Sie sind in ein Netzwerk von Services eingebettet und kommunizieren untereinander durch Austausch von Nachrichten. Wir gehen hier davon aus, dass dieser Nachrichtenaustausch asynchron erfolgt, Nachrichten sich gegenseitig überholen können, jedoch nicht verloren gehen.<sup>1</sup> Ein Geschäftsprozess kann somit durch das Zusammenspiel verteilt ablaufender Services realisiert werden. Ein Service kann jederzeit durch einen anderen Service ersetzt werden, der über die gleiche oder erweiterte Funktionalität verfügt. So kann durch Ersetzung oder Hinzunahme weiterer Services rasch auf neue Marktsituationen reagiert werden.

Zentrale Fragen zu service-orientierten Architekturen sind jedoch noch ungeklärt. Wie soll z.B. ein Anbieter seinen Service *P* und insbesondere dessen Semantik und Verhalten

---

<sup>1</sup>Eine zuverlässige Zustellung von Nachrichten wird in der Regel durch eine unter der SOA angesiedelte Middleware gewährleistet.

beschreiben, sodass der Verzeichnisdienst entscheiden kann, ob er zu einem Service  $R$  eines Interessenten passt? Die publizierten Informationen müssen ausführlich genug sein, um vorab überprüfen zu können, ob  $P$  und  $R$  problemlos miteinander interagieren können, ohne sich beispielsweise durch unerwartetes Kommunikationsverhalten gegenseitig zu blockieren. Die konkrete innere Struktur von  $P$  kann allerdings nicht veröffentlicht werden, da sie unternehmensinterne Informationen des Anbieters enthält.

Massuthe et al. verfolgen den Ansatz, an Stelle der internen Struktur eine *Bedienungsanleitung* für  $P$  zu veröffentlichen [MS05]. Die Bedienungsanleitung eines Service  $P$  beschreibt das Kommunikationsverhalten aller möglichen Partner von  $P$ . Ein Service  $R$  interagiert dann problemlos mit  $P$ , wenn  $R$  alle Anforderungen der Bedienungsanleitung von  $P$  erfüllt. Die Prüfung auf Erfüllung der Anforderungen wird *Matching* genannt. In [LMW07] wird gezeigt, wie für beliebige Services eine Bedienungsanleitung generiert werden kann und wie das Matching ausgeführt wird.

Leymann et al. und Martens schlagen die Veröffentlichung einer abstrakten Sicht  $P'$  auf den Service  $P$  vor - den sog. *Public View* [LRS02], [Mar03]. Obwohl die Idee des Public View in der Literatur weit verbreitet ist, gibt es bisher nur recht wenige Algorithmen die das Konzept unterstützen. Wir wollen uns daher in dieser Arbeit mit Verfahren beschäftigen, die zu einem gegebenen Service  $P$  bzw. dessen Bedienungsanleitung weitgehend automatisch einen Public View  $P'$  von  $P$  berechnen können. Die verschiedenen Ansätze werden ausführlich vorgestellt, implementiert und anhand von Fallstudien miteinander verglichen. Dabei wollen wir Empfehlungen für den optimalen Einsatz der Verfahren ableiten und abschließend einen Ausblick auf weitere Forschung in diesem Bereich geben.

## 1.2 Stand der Forschung

Wir wollen nun einige Arbeiten vorstellen, die sich bereits mit dem Thema Public View beschäftigt haben. Das Konzept der Bedienungsanleitung werden wir ausführlich in Abschnitt 2.4 vorstellen, da es für unsere Untersuchung eine wesentliche Rolle spielt.

Die Idee des Public View geht auf [LRS02] zurück. Leymann et al. sehen den Public View  $P'$  als eine Version des Prozesses  $P$  an, die von dessen internen Aktivitäten abstrahiert und nur die nach außen sichtbaren Aktivitäten darstellt [LRS02]. In einem Top-Down-Entwicklungsansatz soll ein Public View zu einem komplexen Prozess verfeinert werden können, der die nach außen sichtbaren Aktivitäten in eine interne Kontrollstruktur integriert.

Für die konkrete Umsetzung des Public-View-Ansatzes stellen sich jedoch einige grundlegende Fragen, z.B.: Welche Beziehung besteht zwischen dem Prozess  $P$  und seinem Public View  $P'$ ? Welche Eigenschaften von  $P$  muss auch  $P'$  besitzen, um als Public View von  $P$  gelten zu können? Wie kann zu einem gegebenen Prozess  $P$  ein Public View  $P'$  berechnet werden? Diese Fragen werden in [LRS02] nicht beantwortet.

Martens beschreibt zwei Ansätze zur weitgehend automatischen Generierung eines Public View  $P'$  von  $P$  - die *strukturbasierte Transformation* und die *verhaltensbasierte Transfor-*

tion [Mar03]. Die strukturbasierte Transformation setzt direkt an der inneren Struktur von  $P$  (repräsentiert als *Petrinetz*) an und versucht diese, anhand von Regeln, so weit wie möglich zu vereinfachen. Da wir dieses Verfahren in Kapitel 4 detailliert vorstellen werden, wollen wir hier nicht weiter darauf eingehen. Für Prozesse mit komplexem Kommunikationsverhalten kann allerdings mit der strukturbasierten Transformation meist kein befriedigendes Ergebnis erzielt werden. In [Mar03] wird für solche Prozesse ein verhaltensbasierten Ansatz vorgeschlagen. Bei der verhaltensbasierten Transformation wird ausgehend vom Kommunikationsverhalten des Service  $P$  versucht, einen in Bezug auf die Bedienbarkeit äquivalenten aber weniger komplexen Service  $P'$  zu generieren. Der generierte Service  $P'$  soll dann als Public View für  $P$  verwendet werden. In den meisten Fällen wird die Struktur von  $P'$  keine Ähnlichkeit mit der von  $P$  mehr haben.  $P'$  kann jedoch aufgrund der geringeren Komplexität einem potentiellen Benutzer oft besser darlegen, wie der Service  $P$  zu bedienen ist. Wie sich jedoch später gezeigt hat, kann die in [Mar03] präsentierte verhaltensbasierte Transformation die geforderte Äquivalenz im Verhalten von  $P$  und  $P'$  nicht gewährleisten, so dass dieses Verfahren nicht ohne Weiteres eingesetzt werden kann.<sup>2</sup>

Zhao et al. sehen einen Public-View-Ansatz als zu restriktiv an, um die Zusammenarbeit von beliebigen Geschäftspartnern (bzw. deren Prozessen) zu beschreiben. Sie schlagen vor, anstatt eines Public View  $P'$ , zu einem Prozess  $P$  eine ganze Reihe von sog. *relativen Workflows* zu definieren [ZLY05]. Wobei jeder dieser relativen Workflows die für den jeweiligen Geschäftspartner geeignete Sicht auf  $P$  darstellt. So kann die Sichtbarkeit einzelner Aktivitäten des Geschäftsprozesses für jeden Partner individuell festgelegt werden. In [ZLY05] wird ein Algorithmus angegeben, der, geg. einen Prozess  $P$  (repräsentiert durch einen gerichteten azyklischen Graphen) und einer Menge von Sichtbarkeitsbeschränkungen über den Aktivitäten, einen relativen Workflow  $P'$  berechnet.<sup>3</sup> Das Verfahren ist der strukturbasierten Transformation von [Mar03] nicht unähnlich, da versucht wird, durch Zusammenfassen unsichtbarer Aktivitäten die Komplexität von  $P$  so weit wie möglich zu verringern. Die Betrachtungen in [ZLY05] beschränken sich allerdings auf azyklische Prozesse und sind daher nicht allgemein anwendbar.

Schulz und Orlowska untersuchten die Frage nach den gegenseitigen Abhängigkeiten von Private View und Public View, um eine organisationsübergreifende Ausführung von Geschäftsprozessen auf Basis von Public Views zu unterstützen [SO04]. Für die Kopplung von Public View und Private View entwickelten sie ein petrinetzbasiertes Zustandsübergangsmodell, das die Aktivitäten des Public View auf die entsprechenden Aktivitäten des Private View abbildet und z.B. eingehende Nachrichten an diese vermittelt. Die synchronisierte Ausführung der Prozesse steht dabei im Mittelpunkt. Auf Fragen der eigentlichen Public-View-Generierung gehen die Autoren daher nicht näher ein.

Shan et al. definieren einen Public View als strukturell und syntaktisch korrekte Teilmenge des Private View [SYL<sup>+</sup>06]. Sie schlagen eine Sprache zur Definition von Sichten auf

<sup>2</sup>Ein Kollege von Martens konnte formale Fehler in der Beweisführung zur Korrektheit des Verfahrens aufdecken, die jedoch bisher nicht veröffentlicht wurden.

<sup>3</sup>Dabei werden auch Aspekte der Kommunikation zwischen den Prozessen berücksichtigt, auf die wir hier jedoch nicht näher eingehen wollen.

einen Prozess vor und haben eine Laufzeitumgebung implementiert, in der verschiedene Sichten auf ein und den selben Prozess extern verfügbar gemacht werden können. Dazu verwenden sie, ähnlich wie Schulz und Orłowska, eine zustandsbasierte Abbildung der Aktivitäten eines solchen Public View auf die des Private View. Aspekte der Implementation eines sichtenbasierten Ansatzes zur Zusammenarbeit von Services stehen dabei im Zentrum der Untersuchung, so dass auch in dieser Arbeit Fragen nach einer weitgehend automatischen Generierung des Public View keine Rolle spielen.

Aalst et al. zeigen, wie der Public-View-Ansatz in einem Top-Down-Entwicklungsverfahren eingesetzt werden kann [AW01]. Das Szenario sieht dabei wie folgt aus: Eine Reihe von Organisationen möchte zu einem bestimmten Zweck kooperieren. In gemeinsamer Absprache wird ein *Public Workflow* entworfen, der als Vertrag zwischen den Organisationen dient und die Zusammenarbeit auf globaler Ebene modelliert. Anschließend werden die einzelnen Aktivitäten des Public Workflow auf die entsprechenden Organisationen verteilt. Mit den in [AW01] vorgestellten Verfeinerungsmechanismen kann jede der beteiligten Organisationen ihren Teil des Public Workflow (den Public View auf die von ihr geleisteten Aktivitäten) so durch interne Prozesse realisieren, dass die Korrektheit des Gesamtprozesses gewährleistet bleibt.

Die vorliegende Arbeit soll die Idee einer automatischen Public-View-Generierung durch weitere Untersuchungen und Algorithmen unterstützen. Dabei verfolgen wir im Gegensatz zu [AW01] einen Bottom-Up-Ansatz, d.h. zu einem gegebenen Private View bzw. einer genauen Verhaltensbeschreibung des Service  $P$  soll ein Public View  $P'$  berechnet werden. Wir wollen insbesondere auf dem Gebiet der verhaltensbasierten Public-View-Generierung neue Ansätze vorstellen und aus einem Vergleich mit strukturbasierten Verfahren Empfehlungen für den praktischen Einsatz ableiten.

### 1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Kapitel 2 führt in die Grundlagen zu Geschäftsprozessen, Petrinetzen, offenen Workflow-Netzen und den Begriffen Public View und Private View ein. Außerdem wird das Konzept der Bedienungsanleitung ausführlich vorgestellt. Kapitel 3 beschäftigt sich mit zwei Ansätzen zur verhaltensbasierten Public-View-Generierung. Anhand eines ersten sprachenbasierten Verfahrens wird gezeigt, dass nicht jede Verhaltensbeschreibung ausdrucksstark genug ist, um eine anschließende Public-View-Generierung zu ermöglichen. Danach wird ein zweites allgemein anwendbares Verfahren vorgestellt, das von der Bedienungsanleitung als Repräsentation des Verhaltens ausgeht. Kapitel 4 befasst sich mit dem strukturbasierten Ansatz von Martens aus [Mar03] und demonstriert dessen Anwendung an einem Beispiel. Vor dem Hintergrund einer Reihe von Fallstudien werden die Ansätze in Kapitel 5 miteinander verglichen, die individuellen Vor- und Nachteile der einzelnen Verfahren herausgestellt und Empfehlung für den praktischen Einsatz abgeleitet. Abschließend fassen wir in Kapitel 6 die Resultate der Arbeit zusammen, diskutieren einige offen gebliebene Fragen und geben einen Ausblick auf weitere verwandte Forschungsvorhaben.

## 2 Hintergrund

In diesem Kapitel wollen wir in die für unsere Arbeit relevante Terminologie einführen und die zugrunde liegenden Konzepte erläutern. In Abschnitt 2.1 werden wir definieren, was unter einem *Geschäftsprozess* zu verstehen ist und wie Geschäftsprozesse modelliert werden können. Als eine Modellierungsmethode für Geschäftsprozesse wollen wir in Abschnitt 2.2 *Petrinetze* bzw. in Abschnitt 2.3 die für unsere Untersuchung geeigneteren *offenen Workflow-Netze* vorstellen, die die Basis für unsere weiteren Betrachtungen bilden. Anschließend wird in Abschnitt 2.4 das Konzept der *Bedienungsanleitung* ausführlich vorgestellt, das eine wesentliche Rolle für unser Vorgehen zur Public-View-Generierung spielt. Mit einer Abgrenzung und Erläuterung der Begriffe *Private View* und *Public View* in Abschnitt 2.5 schließen wir unsere Einführung in die Grundlagen ab.

### 2.1 Geschäftsprozesse

Wenn man die Arbeitsweise eines Unternehmen oder einer Organisation verstehen, vermitteln, technisch unterstützen oder effizient gestalten will, so müssen zuerst die innerbetrieblichen Abläufe - *die Geschäftsprozesse* - herausgearbeitet und geeignet dargestellt werden. In der Literatur finden sich zahlreiche Definitionen des Begriffs *Geschäftsprozess*. Wir wollen für diese Arbeit in Anlehnung an [Obe96] die folgende Definition verwenden:

**Definition 2.1 (Geschäftsprozess / Workflow)**

*Ein Geschäftsprozess ist eine Folge von Aktivitäten, die in einem logischen Zusammenhang stehen, inhaltlich abgeschlossen sind und unter Zuhilfenahme von Ressourcen und eingehenden Informationen durch Menschen und/oder Maschinen auf ein Unternehmensziel hin ausgeführt werden.*

*Ein Workflow ist die informationstechnische Realisierung eines Geschäftsprozesses.*

*Aktivitäten* stellen die kleinsten logischen Schritte bei der Abarbeitung des Prozesses dar. Sie werden entweder manuell (durch Menschen) oder automatisch (rechnergestützt bzw. durch Maschinen) erledigt. Ein Geschäftsprozess hat einen klar definierten Anfang und ein klar definiertes Ende.

Als Beispiel für einen Geschäftsprozess wollen wir einen Kinokarten-Service betrachten, der die Reservierung von Kinokarten über das Internet ermöglicht. Ein Kunde initiiert den Prozess, in dem er sich bei dem Service anmeldet - es entsteht ein *Geschäftsvorfall*. Anschließend übermittelt der Kunde Informationen über die gewünschte Vorstellung und

die Anzahl der zu reservierenden Plätze. Der Service prüft die Verfügbarkeit der Plätze und generiert eine Reservierungsbestätigung, die dem Kunden als Beleg zugesandt wird. Mit dem Erhalt der Reservierungsbestätigung ist der Kundenwunsch erfüllt und der Geschäftsvorfall abgeschlossen.

Um Geschäftsprozesse bzw. Workflows einer Analyse zugänglich zu machen, müssen wir sie geeignet formal modellieren können. Erst ein *Geschäftsprozessmodell* kann bearbeitet, optimiert und gegebenenfalls ausgeführt werden. In der Praxis werden dazu häufig *Ereignisgesteuerte Prozessketten (EPK)* [Sch01], die *Unified Modeling Language (UML)* [OMG03], die *Web Service Business Process Execution Language (WS-BPEL)* [AAA<sup>+</sup>07] und *Petrinetze* [Aa198] verwendet. Jeder dieser Formalismen besitzt eine Reihe individueller Vor- und Nachteile und eignet sich daher besonders gut für die Untersuchung bestimmter Aspekte von Geschäftsprozessen.

Wir werden in dieser Arbeit Geschäftsprozessmodelle auf Basis von Petrinetzen betrachten und wollen im Folgenden diese Entscheidung kurz motivieren.

Petrinetze besitzen eine formale Semantik. Ein mit Petrinetzen modellierter Geschäftsprozess ist daher klar und präzise definiert und bietet keinen Interpretationsspielraum in Bezug auf sein Verhalten. Aufgrund ihrer grafischen Natur sind Petrinetze intuitiv anschaulich und leicht erlernbar (vgl. [Aa198]). Sie erlauben sowohl eine zustands- als auch eine ereignisbasierte Modellierung, mit deren Hilfe zwischen dem Zeitpunkt der Aktivierung und der konkreten Ausführung einer Aktivität unterschieden werden kann [JVW00]. Für Petrinetze wurden in den letzten Jahrzehnten zahlreiche Analyseverfahren entwickelt, auf die nun zum Studium von Geschäftsprozesse zurückgegriffen werden kann.

Auch wenn es für UML und WS-BPEL eine vermeintlich größere Anzahl an unterstützenden Werkzeugen gibt, so bedeutet die Wahl von Petrinetzen in dieser Arbeit keine Einschränkung gegenüber den anderen Methoden. Im Gegenteil: Aufgrund der Tatsache, dass den weiteren oben genannten Modellierungstechniken für Geschäftsprozesse keine formale Semantik zugrunde liegt, finden sich in der Literatur zahlreiche Arbeiten zur Übersetzung dieser Modelle in Petrinetze (z.B. [LSW98], [GGW98], [Loh07]). Denn der Nachweis bestimmter Eigenschaften (wie z.B. Bedienbarkeit) lässt sich nur auf formaler Basis betreiben.

## 2.2 Petrinetze

In diesem Abschnitt wollen wir eine kurze Einführung in Petrinetze geben, die wir anschließend für die Definition von *offenen Workflow-Netzen* benötigen.

Die Theorie der Petrinetze nahm mit der Dissertation von Carl Adam Petri ihren Anfang und wurde in den 60er Jahren entwickelt. Petrinetze stellen eine grafische Modellierungsmethode dar, die einen konkreten mathematischen Hintergrund besitzt. Wir wollen zuerst den Begriff *Petrinetz* definieren.

**Definition 2.2 (Petrinetz)**

Ein Petrinetz ist ein Tripel  $N = (P, T, F)$  mit

- $P$  ist eine Menge von Plätzen,
- $T$  ist eine Menge von Transitionen, mit  $P \cap T = \emptyset$  und
- $F \subseteq (P \times T) \cup (T \times P)$  ist eine zweistellige Flussrelation (die Menge der Kanten).

Grafisch werden Plätze durch Kreise und Transitionen durch Rechtecke repräsentiert. Die Flussrelation wird durch gerichtete Kanten zwischen Plätzen und Transitionen dargestellt.

Der Zustand eines Petrinetzes wird über eine *Markierung* von Plätzen festgelegt.

**Definition 2.3 (Markierung)**

Sei  $N = (P, T, F)$  ein Petrinetz. Eine Abbildung  $m : P \rightarrow \mathbb{N}$  heißt *Markierung*.

Die Ausprägung von Marken kann, je nach Netztyp, sehr unterschiedlich sein. Für unsere Betrachtungen ist es ausreichend, sich auf das klassische (*Low-Level-*) Petrinetz [Rei85] zu beschränken, in dem nur ununterscheidbare schwarze Marken (sog. *black token*) auftreten. Auf einem Platz können zu jedem Zeitpunkt beliebig viele Marken liegen.

Ein Petrinetz beschreibt jedoch nicht nur einen Zustand, sondern bildet eine Dynamik ab. Es kommt zu einem Zustandswechsel innerhalb des Netzes, wenn eine Transition entsprechend einer *Schaltregel* schaltet. Durch einen Zustandswechsel kann sich die Anzahl der Marken auf den Plätzen des Netzes ändern.

**Definition 2.4 (Schaltregel)**

Sei  $N = (P, T, F)$  ein Petrinetz und  $m$  eine Markierung von  $N$ .

- $t \in T$  ist aktiviert für die Markierung  $m$ , wenn für jeden Platz  $p$  mit  $(p, t) \in F$ ,  $m(p) \geq 1$ .
- Eine aktivierte Transition  $t$  kann schalten. Das Schalten von  $t$  überführt die Markierung  $m$  in die Markierung  $m'$  mit  $m'(p) = m(p) - 1$ , falls  $(p, t) \in F$  und  $(t, p) \notin F$ ,  $m'(p) = m(p) + 1$ , falls  $(t, p) \in F$  und  $(p, t) \notin F$ , und  $m'(p) = m(p)$  sonst.

Das Schalten einer Transition  $t$  wird als *Schritt* bezeichnet und wie folgt notiert:  $m_1 \xrightarrow{t} m_2$ . Eine endliche, möglicherweise leere, Folge von Schritten  $\sigma = t_1 t_2 \dots t_n$  wird als *Schaltsequenz* bezeichnet und in der Form  $m_1 \xrightarrow{\sigma} m_{n+1}$  geschrieben. Eine solche Schaltsequenz beschreibt den schrittweisen Übergang des Netzes vom Zustand  $m_1$  mittels  $m_1 \xrightarrow{t_1} m_2 \xrightarrow{t_2} \dots \xrightarrow{t_n} m_{n+1}$  in den Zustand  $m_{n+1}$ . Wenn es zu einem Zustand  $m_1$  eine Schaltsequenz  $\sigma = t_1 t_2 \dots t_n$  gibt, so dass  $m_1 \xrightarrow{\sigma} m_{n+1}$  gilt, so nennen wir  $m_{n+1}$  *erreichbar* von  $m_1$  und schreiben  $m_1 \xrightarrow{*} m_{n+1}$ .

Die Menge  $\bullet t = \{p \mid (p, t) \in F\}$  wird als der Vorbereich der Transition  $t$  bezeichnet und die Menge  $t \bullet = \{p \mid (t, p) \in F\}$  als Nachbereich von  $t$ . Analog dazu lassen sich der Vorbereich  $\bullet p$  und Nachbereich  $p \bullet$  für einen Platz  $p$  definieren.

Petrinetze wurden in den letzten Jahrzehnten in den verschiedensten Anwendungsbereichen zur Modellierung eingesetzt, z.B. für Hardware, Protokolle, eingebettete Systeme und zur Modellierung von Fertigungsanlagen (siehe [CKK<sup>+</sup>00], [BGH04], [BSBN05], [MHKRP95]).

Mit klassischen Petrinetzen können Zustände, Ereignisse, Auswahl von Alternativen, parallele und sequentielle Abläufe, Synchronisation und Iteration modelliert werden. Um komplexe Prozesse geeignet abbilden zu können, wurden verschiedene Erweiterungen für Petrinetze entwickelt. Mit den sog. *High-Level-Petrinetzen* ist daher auch eine Modellierung von Daten, Zeit und Hierarchieebenen möglich (siehe [Aal94], [Jen96]).

## 2.3 Offene Workflow-Netze

Nachdem wir nun Petrinetze als formale Modellierungsmethode vorgestellt haben, wollen wir in diesem Abschnitt zeigen, wie Petrinetze zur Modellierung von Services eingesetzt werden können.

Aalst et al. haben gezeigt, wie viele Aspekte von Geschäftsprozessen bzw. Services mit Hilfe von Petrinetzen modelliert werden können [Aal98]. Uns interessiert jedoch insbesondere die Interaktion zwischen verschiedenen Services, die mit den in [Aal98] definierten *Workflow-Netzen* nicht direkt untersucht werden kann. Wir werden daher auf die von Massuthe et al. vorgeschlagenen *offenen Workflow-Netze* zurückgreifen, die wie folgt definiert sind [MRS05].

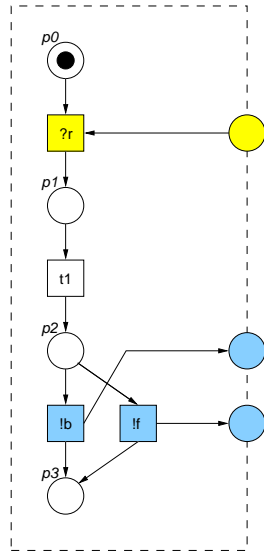
### Definition 2.5 (Offenes Workflow-Netz)

Ein offenes Workflow-Netz (oWFN) ist ein Petrinetz  $N = (P, T, F)$  mit der Anfangsmarkierung  $m_0$ , zusammen mit

- zwei Mengen  $P_i, P_o \subseteq P$ , bezeichnet als Eingabe- und Ausgabeplätze, so dass  $F \cap (P_o \times T) = \emptyset$  und  $F \cap (T \times P_i) = \emptyset$ , wobei  $m_0(p) = 0$  für  $p \in P_i \cup P_o$  und
- einer Menge  $\Omega$  von Markierungen, den Endmarkierungen, von  $N$ , wobei für  $m_f \in \Omega$  und  $p \in P_i \cup P_o$ ,  $m_f(p) = 0$ . Wir verlangen zudem, dass eine Markierung  $m_f$  keine Transition aktiviert.

Die Elemente von  $P_i$  repräsentieren Kanäle für den Empfang von Nachrichten und die Elemente von  $P_o$  repräsentieren Kanäle für das Senden von Nachrichten. Die Menge  $P_i \cup P_o$  wird als die *Schnittstelle* von  $N$  bezeichnet. Die geforderte Einschränkung der Flussrelation  $F$  stellt sicher, dass das Netz  $N$  nicht mit sich selbst über die Schnittstelle kommunizieren kann.

Abbildung 2.1 zeigt das oWFN  $N_{K_P}$ , das den in Abschnitt 2.1 vorgestellten Geschäftsprozess eines Kinokarten-Service modelliert. Zu Beginn übermittelt der Kunde Informationen über die gewünschte Vorstellung und die Anzahl der zu reservierenden Plätze (Nachricht  $r$ ). Der Service prüft die Verfügbarkeit der Plätze und generiert eine Reservierungsbestätigung (Transition  $t1$ ), die dem Kunden als Beleg (Nachricht  $b$ ) zugesandt wird.

Abbildung 2.1: Kinokarten-Service als oWFN  $N_{KP}$ 

Zusätzlich haben wir noch den Fall modelliert, dass die gewünschte Anzahl an Plätzen nicht reserviert werden konnte (z.B. wegen Ausbuchung der Vorstellung). In diesem Fall antwortet der Kinokarten-Service mit einer Fehlermeldung (Nachricht  $f$ ) auf die Anfrage des Kunden.

Mathematisch betrachtet sieht das Modell  $N_{KP} = (P, T, F)$  wie folgt aus:

- $P = \{p0, p1, p2, p3, r, b, f\}$ ,  $P_i = \{r\}$ ,  $P_o = \{b, f\}$ ,
- $T = \{?r, t1, !b, !f\}$ ,
- $F_1 = \{(p0, ?r), (r, ?r), (?r, p1), (p1, t1), (t1, p2)\}$ ,
- $F_2 = \{(p2, !b), (p2, !f), (!b, b), (!b, p3), (!f, f), (!f, p3)\}$ ,
- $F = F_1 \cup F_2$ ,
- $m_0(p0) = 1$ ,  $m_0(p) = 0$  für alle  $p \in P$  mit  $p \neq p0$ ,
- $\Omega = \{m_f\}$ , wobei  $m_f(p3) = 1$ ,  $m_f(p) = 0$  für alle  $p \in P$  mit  $p \neq p3$ .

Wir haben gesagt, dass Geschäftsprozesse durch das Zusammenspiel verteilt ablaufender Services realisiert werden können. Dieses Zusammenspiel wollen wir nun durch die Komposition von oWFN modellieren. Es ist klar, dass die Interaktion zweier beliebiger Services nicht per se problemlos ablaufen muss. Die Services müssen zueinander „passen“. Eine erste Bedingung für eine problemlose Interaktion stellt die Kompatibilität der Schnittstellen der jeweiligen Services dar. Wir definieren dafür den Begriff *Partner-oWFN*.

**Definition 2.6 (Partner-oWFN)**

Zwei oWFN  $N$  und  $N'$  sind *Partner-oWFN*, wenn  $P_i = P'_o$  und  $P_o = P'_i$ . Alle anderen Bestandteile von  $N$  und  $N'$  seien disjunkt.

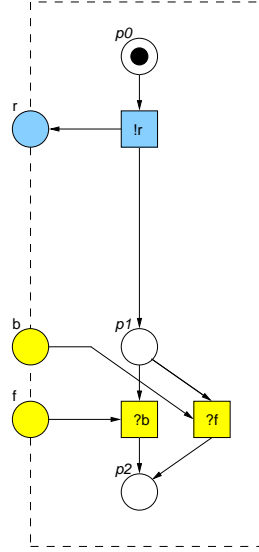


Abbildung 2.2: Ein Partner-oWFN  $N_{KR}$  des Kinokarten-Service  $N_{KP}$

Abbildung 2.2 zeigt ein Partner-oWFN  $N_{KR}$  des Kinokarten-Service  $N_{KP}$ .  $N_{KR}$  sendet zunächst eine Reservierungsanfrage (Nachricht  $r$ ) und empfängt anschließend eine Reservierungsbestätigung (Nachricht  $b$ ) oder eine Fehlermeldung (Nachricht  $f$ ).

Eine Partnerschaft zwischen oWFN ist keine hinreichende Bedingung, um eine problemlose Interaktion der repräsentierten Services zu gewährleisten. Trotz kompatibler Schnittstellen könnten sich die Services Nachrichten senden, die der Partner nicht erwartet, oder in eine Situation gelangen, in der beide Services gegenseitig auf eine Nachricht des jeweils anderen Partners warten. Wir wollen daher nun die Komposition von oWFN definieren, um anschließend das Zusammenspiel der Services besser untersuchen zu können. Die Komposition besteht im Wesentlichen aus der Verschmelzung der Schnittstellen-Plätze und ist wie folgt definiert.

**Definition 2.7 (Komposition von oWFN)**

Seien  $N$  und  $N'$  zwei bis auf die Schnittstellen-Plätze disjunkte oWFN. Die Komposition  $m \oplus m' : P \cup P' \rightarrow \mathbb{N}$  zweier Markierungen  $m$  von  $N$  und  $m'$  von  $N'$  ist definiert als  $(m \oplus m')(p) = m(p)$ , falls  $p \in P$  und  $(m \oplus m')(p) = m'(p)$ , falls  $p \in P'$ . Die Komposition von  $N$  und  $N'$  ist das oWFN  $N \oplus N'$  und ist wie folgt definiert:

- $P_{N \oplus N'} = P \cup P'$ ,  $T_{N \oplus N'} = T \cup T'$ ,  $F_{N \oplus N'} = F \cup F'$ ,  $m_{0_{N \oplus N'}} = m_0 \oplus m'_0$ .
- $P_{i_{N \oplus N'}} = P_{o_{N \oplus N'}} = \emptyset$ .
- $\Omega_{N \oplus N'} = \{m \oplus m' \mid m \in \Omega, m' \in \Omega'\}$ .

Abbildung 2.3 zeigt die Komposition  $N_{KP} \oplus N_{KR}$  der Partner-oWFN  $N_{KP}$  und  $N_{KR}$ .

Die Komposition zweier Partner-oWFN  $N$  und  $N'$  führt zu einem oWFN  $N \oplus N'$  mit leerer Schnittstelle.  $N \oplus N'$  stellt dann in gewisser Weise ein geschlossenes System dar, das

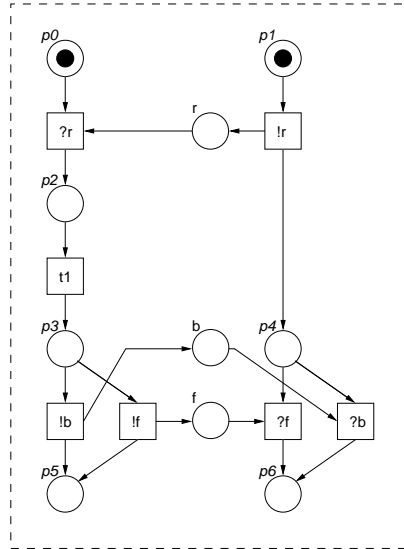


Abbildung 2.3: Komposition  $N_{KP} \oplus N_{KR}$  der Partner-oWFN  $N_{KP}$  und  $N_{KR}$ .

nicht mit seiner Umwelt kommuniziert. Erst dies ermöglicht eine sinnvolle Analyse, da nun alle Aspekte des Verhaltens vollständig im komponierten System  $N \oplus N'$  formuliert sind. Anhand des Erreichbarkeitsgraphen von  $N \oplus N'$  kann verifiziert werden, ob es während der Interaktion von  $N$  und  $N'$  nie einen Zustand gibt, in dem einer der beiden Services unendlich lange auf eine Nachricht des Partners warten muss. Dies ist genau dann der Fall, wenn jede im komponierten System  $N \oplus N'$  erreichbare Markierung, in der keine Transition aktiviert ist, eine Endmarkierung von  $N \oplus N'$  ist. Damit haben wir neben der Partnerschaft die zweite Bedingung für eine problemlose Interaktion auf Basis von oWFN formuliert. Einen Partner  $N'$  von  $N$ , der diese Anforderung erfüllt, werden wir im Folgenden als *Strategie von  $N$*  bezeichnen.

Um auch Services in Isolation untersuchen zu können, wollen wir das *Innere* eines oWFN betrachten:

**Definition 2.8 (Inneres eines oWFN)**

*Sei  $N$  ein oWFN. Wir erhalten das Innere von  $N$ , geschrieben als  $inner(N)$ , durch Entfernen aller Plätze  $P_i$  und  $P_o$  und deren angrenzenden Kanten. Anfangs- und Endmarkierungen werden dabei entsprechend angepasst.*

Das Innere eines oWFN ist ebenfalls ein oWFN mit leerer Schnittstelle, so dass auch hier Erkenntnisse durch die Berechnung des Erreichbarkeitsgraphen gewonnen werden können.

## 2.4 Bedienungsanleitungen

Im Folgenden wollen wir uns der Definition der Bedienungsanleitung nähern. Die Bedienungsanleitung eines Service  $P$  soll das Verhalten aller problemlos mit  $P$  interagierenden Partner charakterisieren.

Bedienungsanleitungen setzen auf dem Konzept von *Serviceautomaten* auf, die wir in Abschnitt 2.4.1 als ein formales Modell für Services einführen werden. Anschließend kann die Interaktion von Services auf Basis dieser Serviceautomaten definiert werden. Um den Bezug zu oWFN herzustellen, wird in Abschnitt 2.4.2 gezeigt, wie ein oWFN in einen Serviceautomaten überführt werden kann.

Wir wollen Aussagen über die Bedienbarkeit eines Service  $P$  treffen können, ohne dabei die Interaktion mit einem konkreten Partner  $R$  zu untersuchen. Dazu führen wir in Abschnitt 2.4.3 die Begriffe *Situation* und *Wissen* ein. Anschließend wird dann in Abschnitt 2.4.4 gezeigt, wie Bedienungsanleitungen für Serviceautomaten generiert werden können.

Sämtliche hier verwendeten Definitionen sowie die Vorgehensweise zur Konstruktion von Bedienungsanleitungen sind aus [LMW07] entnommen und wurden - soweit für die Arbeit erforderlich - angepasst.

### 2.4.1 Serviceautomaten

Für die Berechnung der Bedienungsanleitung werden wir die interne Kontrollstruktur und das Kommunikationsverhalten eines Service im Folgenden durch einen nichtdeterministischen Automaten modellieren. Wir gehen davon aus, dass Nachrichten über unidirektionale Nachrichtenkanäle übertragen werden. Das Senden und Empfangen von Nachrichten wird dabei über Beschriftungen an den Zustandsübergängen des Automaten dargestellt. Für das Senden einer Nachricht auf dem Kanal  $x$  schreiben wir  $!x$  und für das Empfangen einer Nachricht von Kanal  $x$  schreiben wir  $?x$ . Zustandsübergänge bei denen keine Kommunikation mit der Umgebung erfolgt beschriften wir mit  $\tau$ .

Mit  $C$  bezeichnen wir die endliche Menge aller Nachrichtenkanäle, wobei  $\tau \notin C$ . Mit  $\text{bags}(C)$  bezeichnen wir die Menge aller Multimengen über  $C$ . Eine Multimenge über  $C$  beschreibt den Zustand der Nachrichtenkanäle, d.h. für jeden Kanal  $c \in C$  die Menge der Nachrichten, die sich aktuell auf dem Kanal befinden. Die Abbildung  $m : C \rightarrow \mathbb{N}$  liefert für jeden Kanal  $c \in C$  die Anzahl der noch ausstehenden Nachrichten.

#### Definition 2.9 (Serviceautomat)

Ein Serviceautomat  $A = (Q, I, O, \delta, q_0, F)$  besteht aus einer Menge von Zuständen  $Q$ , einer Menge  $I \subseteq C$  von Eingabekanälen, einer Menge  $O \subseteq C$ ,  $I \cap O = \emptyset$  von Ausgabekanälen, einer nichtdeterministischen Zustandsübergangsrelation  $\delta \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$ , einem Anfangszustand  $q_0 \in Q$  und einer Menge von Endzuständen  $F \subseteq Q$ , so dass gilt: Wenn  $q \in F$  und  $[q, x, q'] \in \delta$ , dann  $x \in I$ .  $A$  ist endlich, wenn die Menge  $Q$  seiner Zustände endlich ist.

Abbildung 2.4 zeigt ein Modell des Kinokarten-Service aus Abschnitt 2.1 als Serviceautomaten. Der Anfangszustand des Automaten wird dabei im Folgenden durch einen initialen Pfeil ausgezeichnet und die Endzustände werden durch eine doppelte Umrandung markiert.

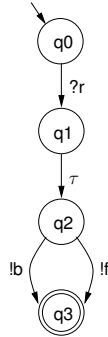


Abbildung 2.4: Serviceautomat  $A_{K_P}$  des Kinokarten-Service  $N_{K_P}$ .

Wir wollen unsere zwei Mindestanforderungen an eine problemlose Interaktion auf Serviceautomaten übertragen und definieren dafür zuerst, analog zur Partnerschaft von oWFN, die Partnerschaft zwischen Serviceautomaten.

**Definition 2.10 (Partnerautomaten)**

Zwei Serviceautomaten  $P$  und  $R$  sind Partnerautomaten, wenn  $I_P = O_R$  und  $I_R = O_P$ .

Zwei Partnerautomaten teilen sich Nachrichtenkanäle derart, dass jeder Kanal einen gerichteten Kommunikationsweg darstellt.

Abbildung 2.5 zeigt zwei mögliche Partnerautomaten  $A_{K_R}$  und  $A_{K_F}$  des Serviceautomaten  $A_{K_P}$ . Der Serviceautomat  $A_{K_R}$  (Abbildung 2.5 (a)) modelliert das selbe Verhalten wie das oWFN  $N_{K_R}$  (Abbildung 2.2). Der Serviceautomat  $A_{K_F}$  (Abbildung 2.5 (b)) modelliert einen Benutzer des Kinokarten-Service, der stets eine Reservierungsbestätigung auf seine Anfrage erwartet und nicht mit einer Fehlermeldung rechnet.

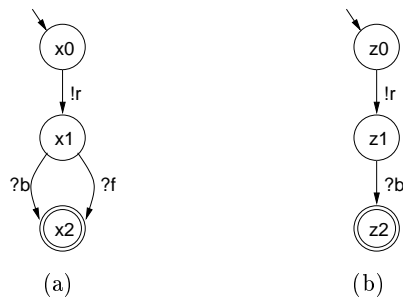


Abbildung 2.5: Partnerautomaten  $A_{K_R}$  (a) und  $A_{K_F}$  (b) des Serviceautomaten  $A_{K_P}$ .

Analog zur Komposition von oWFN werden wir jetzt das Zusammenspiel der beiden Services durch die Komposition ihrer Serviceautomaten beschreiben. Wie bei der Kom-

position von Partner-oWFN, hat auch der aus den Partnerautomaten  $P$  und  $R$  komponierte Serviceautomat  $P \oplus R$  weder Eingabe- noch Ausgabekanäle und kann daher als abgeschlossenes Transitionssystem betrachtet werden. Ein Zustand  $[q_P, q_R, m]$  des Transitionssystems  $P \oplus R$  repräsentiert einen Zustand des Serviceautomaten  $P$ , einen Zustand des Serviceautomaten  $R$  und eine Multimenge  $m$  von Nachrichten auf den Kommunikationskanälen. Im Anfangszustand ist  $m$  leer. Sendet einer der Serviceautomaten eine Nachricht über einen Kanal  $x$ , so wird  $m$  ein Element  $x$  hinzugefügt. Analog dazu wird aus  $m$  ein  $x$  entfernt, wenn einer der Serviceautomaten eine Nachricht vom Kanal  $x$  konsumiert.

**Definition 2.11 (Komposition von Partnerautomaten)**

Die Komposition  $P \oplus R$ , zweier Partnerautomaten  $P$  und  $R$ , ist definiert als Automat mit den folgenden Bestandteilen:  $Q_{P \oplus R} = Q_P \times Q_R \times \text{bags}(C)$ ,  $I_{P \oplus R} = \emptyset$ ,  $O_{P \oplus R} = \emptyset$ ,  $q_{0P \oplus R} = [q_{0P}, q_{0R}, []]$  und  $F_{P \oplus R} = F_P \times F_R \times \{[]\}$ . Die Zustandsübergangsrelation  $\delta_{P \oplus R}$  enthält die folgenden Elemente:

- (interner Übergang in  $P$ )  $[[q_P, q_R, m], \tau, [q'_P, q_R, m]]$  gdw.  $[q_P, \tau, q'_P] \in \delta_P$ ,
- (interner Übergang in  $R$ )  $[[q_P, q_R, m], \tau, [q_P, q'_R, m]]$  gdw.  $[q_R, \tau, q'_R] \in \delta_R$ ,
- (Nachrichtenempfang durch  $P$ )  $[[q_P, q_R, m], \tau, [q'_P, q_R, m - [x]]]$  gdw.  $[q_P, x, q'_P] \in \delta_P$ ,  $x \in I_P$  und  $m(x) > 0$ ,
- (Nachrichtenempfang durch  $R$ )  $[[q_P, q_R, m], \tau, [q_P, q'_R, m - [x]]]$  gdw.  $[q_R, x, q'_R] \in \delta_R$ ,  $x \in I_R$  und  $m(x) > 0$ ,
- (Senden einer Nachricht durch  $P$ )  $[[q_P, q_R, m], \tau, [q'_P, q_R, m + [x]]]$  gdw.  $[q_P, x, q'_P] \in \delta_P$ ,  $x \in O_P$ ,
- (Senden einer Nachricht durch  $R$ )  $[[q_P, q_R, m], \tau, [q_P, q'_R, m + [x]]]$  gdw.  $[q_R, x, q'_R] \in \delta_R$ ,  $x \in O_R$ ,

und keine weiteren Elemente.

Das Transitionssystem  $P \oplus R$  besitzt nur dann einen erreichbaren Endzustand, wenn es eine Folge von Zustandsübergängen in den Serviceautomaten  $P$  und  $R$  gibt, so dass  $P$  und  $R$  jeweils einen Endzustand erreichen und die Multimenge  $m$  wieder leer ist.

Abbildung 2.6 zeigt die Komposition  $A_{K_P} \oplus A_{K_F}$  der Partnerautomaten  $A_{K_P}$  (siehe Abbildung 2.4) und  $A_{K_F}$  (siehe Abbildung 2.5 (b)).

Auch wenn das komponierte Transitionssystem  $P \oplus R$  einen oder mehrere Endzustände besitzt, so muss die Interaktion der beiden Serviceautomaten nicht immer problemlos verlaufen (siehe Abbildung 2.6). Wie bereits erwähnt, kann es z.B. zu einem Zustand kommen, in dem beide Services unendlich lange auf eine Nachricht des Partners warten. Wir wollen daher nun genau die Zustände charakterisieren, in denen ein Partner auf eine Nachricht des anderen angewiesen ist.

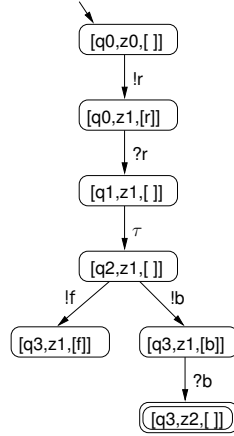


Abbildung 2.6: Komposition  $A_{K_P} \oplus A_{K_F}$  der Partnerautomaten  $A_{K_P}$  und  $A_{K_F}$ .

### Definition 2.12 (Wartezustand, Deadlock)

Der Zustand  $q$  eines Serviceautomaten  $A$  wird Wartezustand genannt, wenn aus  $[q, x, q'] \in \delta$  stets folgt, dass  $x \in I$ . Das heißt, der Zustand  $q$  kann ohne Hilfe der Umgebung nicht verlassen werden. Sei  $q$  ein Wartezustand, dann ist  $\text{wait}(q) = \{x \in C \mid \exists q' \in Q : [q, x, q'] \in \delta\}$ . Einen Wartezustand  $q$  bezeichnen wir als Deadlock, gdw.  $q \notin F$  und  $\text{wait}(q) = \emptyset$ .

Ein Wartezustand kann, wenn überhaupt, nur durch den Empfang einer vom Partner gesendeten Nachricht verlassen werden.  $\text{wait}(q)$  bezeichnet genau die Menge der Nachrichten, die helfen könnten den Zustand  $q$  zu verlassen. Ist keine solche Nachricht vorhanden und  $q$  kein Endzustand, so ist die Interaktion in einen Deadlock geraten, also einen Zustand, der nicht mehr verlassen werden kann.

Der Zustand  $q0$  des Serviceautomaten  $A_{K_P}$  aus Abbildung 2.4 ist ein Wartezustand mit  $\text{wait}(q0) = \{r\}$ , der durch den Empfang einer Nachricht  $r$  aus der Umgebung verlassen werden kann. Der Zustand  $[q3, z1, [f]]$  des Serviceautomaten  $A_{K_P} \oplus A_{K_F}$  aus Abbildung 2.6 ist ein Deadlock.

Mit dem Begriff des Deadlock lässt sich nun auch unsere zweite Bedingung an eine problemlose Interaktion auf Serviceautomaten übertragen. Danach interagieren die Serviceautomaten  $P$  und  $R$  genau dann problemlos, wenn sie Partnerautomaten sind und das Transitionssystem  $P \oplus R$  keinen Deadlock besitzt. Wir werden auch in diesem Fall den Serviceautomaten  $R$  als Strategie von  $P$  bezeichnen.

## 2.4.2 Übersetzung von oWFN in Serviceautomaten

Da wir für die Generierung der Bedienungsanleitung Services durch Serviceautomaten modellieren, wollen wir in diesem Abschnitt zeigen, wie ein beliebiges oWFN in einen Serviceautomaten überführt werden kann.

Ein Unterschied im Verhalten von oWFN und Serviceautomaten besteht darin, dass ein oWFN mit einer einzelnen Transition mehrere Nachrichten senden und/oder empfangen kann. Wir werden der Einfachheit halber nur eine Übersetzung von oWFN in Serviceautomaten für solche oWFN  $N$  angeben, für die jede Transition von  $N$  mit höchstens einem Schnittstellen-Platz verbunden ist. Dies stellt jedoch keine Einschränkung dar, da jedes oWFN  $N$  leicht in ein oWFN  $N'$  transformiert werden kann, das diese Bedingung erfüllt. Abbildung 2.7 zeigt eine solche in [LMW07] vorgeschlagene Transformation. Dabei werden die Schnittstellen-Plätze von  $N$  mit Kapazität Eins versehen (siehe Anschriften an den internen Plätzen von  $N'$ ) und jedem ein neuer Schnittstellen-Platz mit eigener Sende- bzw. Empfangstransition vorgeschaltet. Die Schnittstellen-Plätze von  $N$  werden dadurch zu internen Plätzen von  $N'$ .



Abbildung 2.7: Transformation eines oWFN  $N$  in ein oWFN  $N'$  mit internem Nachrichtenpuffer

Unter Annahme der oben formulierten Beschränkung auf oWFN, deren Transitionen mit höchstens einem Schnittstellen-Platz verbunden sind, kann die Abbildung  $oWFNtoService$  eines oWFN in einen Serviceautomaten wie folgt definiert werden.

**Definition 2.13 (Abbildung oWFN nach Serviceautomat)**

Sei  $N = (P, T, F)$  ein oWFN, in dem jede Transition mit höchstens einem Schnittstellen-Platz verbunden ist. Dann ist  $oWFNtoService(N)$  der Serviceautomat  $A = (Q_A, I_A, O_A, \delta_A, q_{0_A}, F_A)$  mit den folgenden Bestandteilen:

- $Q_A$  ist die Menge der erreichbaren Markierungen von  $inner(N)$ .
- $I_A = P_i, O_A = P_o$ .
- $[m, a, m'] \in \delta_A$  gdw. es eine Transition  $t$  in  $N$  gibt, so dass  $[m, t, m']$  eine Transition innerhalb des Erreichbarkeitsgraphen von  $inner(N)$  ist und entweder ein Schnittstellen-Platz  $p$  mit  $t$  verbunden ist und  $a = p$ , oder  $t$  mit keinem Schnittstellen-Platz verbunden ist und  $a = \tau$ .
- $q_{0_A}$  ist der Anfangszustand von  $inner(N)$ ,  $F_A$  ist die Menge der Endzustände von  $inner(N)$ .

Für zwei beliebige oWFN  $N$  und  $N'$ , deren Transitionen mit höchstens einem Schnittstellen-Platz verbunden sind, ist der Erreichbarkeitsgraph von  $N \oplus N'$  isomorph zu dem Graphen, der durch die Zustände und Transitionen von  $oWFNtoService(N) \oplus oWFNtoService(N')$  definiert wird [LMW07]. Wir können also unsere weiteren Betrachtungen auf Basis von Serviceautomaten führen, ohne die Klasse der repräsentierten oWFN dabei einzuschränken.

### 2.4.3 Situationen und Wissen

In Abschnitt 2.4.1 haben wir festgestellt, dass eine problemlose Interaktion der beiden Services  $P$  und  $R$  nicht immer gewährleistet ist, wenn das Transitionssystem  $P \oplus R$  einen oder mehrere Deadlocks besitzt. Besitzt  $P \oplus R$  jedoch einen Endzustand, so existiert zumindest ein Szenario, in dem ein erfolgreiches Zusammenspiel möglich ist. Möglicherweise kann auch der Service  $R$  durch sein Verhalten bzw. die von ihm gesendeten Nachrichten einen Deadlock in der Interaktion mit  $P$  verhindern. Wenn  $R$  Informationen über die Struktur von  $P$  und die Nachrichten auf den Nachrichtenkanälen vorliegen, kann  $R$  Rückschlüsse darauf ziehen, in welchen Zuständen sich  $P$  gerade befindet und welche Nachrichten  $P$  gerade erwartet. Wir wollen daher nun die Interaktion der Services  $P$  und  $R$  aus der Sicht des Service  $R$  etwas näher betrachten.

#### Definition 2.14 (K, Situation)

Seien  $P$  und  $R$  Partner. Dann sei  $K : Q_R \rightarrow 2^{Q_P \times bags(C)}$  definiert durch  $K(q_R) = \{[q_P, m] \mid [q_P, q_R, m] \text{ ist erreichbar vom Anfangszustand von } Q_{P \oplus R}\}$ . Die Elemente von  $2^{Q_P \times bags(C)}$  heißen Situationen.

Eine Situation umfasst alle Aspekte eines Zustands von  $P \oplus R$ , außer dem Zustand von  $R$  selbst.  $K(q_R)$  kann als das Wissen von  $R$  darüber betrachtet werden, welche Situationen in  $P \oplus R$  in Verbindung mit  $q_R$  auftreten können, d.h. in welchen Zuständen sich  $P$  gerade befinden kann und welche Nachrichten dabei auf den Nachrichtenkanälen liegen.

Beispiele für Werte von  $K$ , bezogen auf den Serviceautomaten  $A_{K_P} \oplus A_{K_F}$  aus Abbildung 2.6, sind:  $K(z_0) = \{[q_0, []]\}$ ,  $K(z_1) = \{[q_0, [r]], [q_1, []], [q_2, []], [q_3, [f]], [q_3, [b]]\}$  und  $K(z_2) = \{[q_3, []]\}$ .

Innerhalb einer Menge  $M$  von Situationen, kann zwischen transienten und stabilen Situationen unterschieden werden. Eine Situation aus  $M$  ist transient, wenn ein Zustandsübergang von  $P$  in dieser Situation zu einer anderen Situation aus  $M$  führt. Anderenfalls ist sie stabil.

#### Definition 2.15 (Transiente, stabile Situation)

Sei  $M$  eine Menge von Situationen.  $[q_P, m]$  ist transient in  $M$ , gdw. ein  $[q'_P, x, q'_P] \in \delta_P$  existiert, so dass gilt:

- $x = \tau$  und  $[q'_P, m] \in M$  oder
- $x \in I_P$ ,  $m(x) > 0$  und  $[q'_P, m - [x]] \in M$  oder
- $x \in O_P$  und  $[q'_P, m + [x]] \in M$ .

Sonst ist  $[q_P, m]$  stabil.

Bezogen auf den Serviceautomaten  $A_{K_P} \oplus A_{K_F}$  aus Abbildung 2.6 ist die Situation  $[q_0, [r]]$  transient in der Menge von Situationen  $K(z_1)$ . Im Gegensatz dazu ist die Situation  $[q_3, [b]]$  stabil in  $K(z_1)$ .

Eine stabile Situation kann von einem Service nur durch Interaktion mit seiner Umgebung verlassen werden. Unter der *abgeschlossenen Hülle* einer Menge  $M$  von Situationen verstehen wir all die Situationen, die ausgehend von einer Situation aus  $M$  ohne Einflussnahme der Umgebung (des Partners), d.h. durch „Überspringen“ allen transienten Situationen, erreicht werden können. Die *abgeschlossene Hülle* ist wie folgt definiert.

**Definition 2.16 (Abgeschlossene Hülle)**

Für eine Menge  $M$  von Situationen, sei die abgeschlossene Hülle, geschrieben als  $cl(M)$ , wie folgt induktiv definiert:

Basis:  $M \subseteq cl(M)$ , Schritt: Wenn  $[q_P, m] \in cl(M)$  und  $[q_P, x, q'_P] \in \delta_P$ , dann

- $[q'_P, m] \in cl(M)$ , falls  $x = \tau$ ,
- $[q'_P, m + [x]] \in cl(M)$ , falls  $x \in O_P$ ,
- $[q'_P, m - [x]] \in cl(M)$ , falls  $x \in I_P$  und  $m(x) > 0$ .

Für den Serviceautomaten  $A_{K_P} \oplus A_{K_F}$  aus Abbildung 2.6 erhalten wir z.B.  $cl(\{[q_0, [r]]\}) = \{[q_0, [r]], [q_1, []], [q_2, []], [q_3, [f]], [q_3, [b]]\}$ .

Das Senden und Empfangen von Nachrichten hat stets einen Effekt auf eine Menge  $M$  von Situationen. Sendet z.B. der Partner  $R$  dem Prozess  $P$  eine Nachricht über den Nachrichtenkanal  $x$  (wir schreiben:  $send(M, x)$ ), dann ergibt sich für  $P$  ausgehend von der aktuellen Situationsmenge  $M$  eine neue Menge an Situationen, in denen sich  $P$  unmittelbar nach diesem Sendeereignis befinden kann. Dazu wird jeder Situation aus  $M$  eine Nachricht  $x$  zum Nachrichtenkanal hinzugefügt. Analog dazu lassen sich Empfangs- und interne Ereignisse definieren, mit dem Unterschied, dass bei einem Empfangsereignis eine Nachricht vom Nachrichtenkanal konsumiert wird und bei einem internen Ereignis ein interner Schritt ohne Kommunikation durchgeführt wird. Um diese Effekte modellieren zu können, wollen wir nun formal definieren, was unter einem Sende-, Empfangs- und internen Ereignis zu verstehen ist.

**Definition 2.17 (Sendeereignis, Empfangsereignis, internes Ereignis)**

Sei  $M \subseteq Q_P \times bags(C)$ . Wenn  $x \in O_P$ , dann ist das Sendeereignis  $x$ ,  $send(M, x)$ , definiert als  $send(M, x) = \{[q, m + [x]] \mid [q, m] \in M\}$ . Wenn  $x \in I_P$ , dann ist das Empfangsereignis  $x$ ,  $receive(M, x)$ , definiert als  $receive(M, x) = \{[q, m - [x]] \mid [q, m] \in M, m(x) > 0\}$ . Das interne Ereignis  $\tau$ ,  $internal(M, \tau)$ , ist definiert als  $internal(M, \tau) = M$ . Da die Ausprägung eines Ereignisses durch  $I_P$  und  $O_P$  bestimmt ist, definieren wir das Ereignis  $x$ ,  $event(M, x)$ , als  $receive(M, x)$ , falls  $x \in I_P$ ,  $send(M, x)$ , falls  $x \in O_P$ , und  $internal(M, x)$ , falls  $x = \tau$ .

Beispiele für Sende- und Empfangsereignisse des komponierten Serviceautomaten  $A_{K_P} \oplus A_{K_F}$ , bezogen auf Abbildung 2.6, sind  $send(\{[q0, []]\}, r) = \{[q0, [r]], [q1, []], [q2, []], [q3, [f]], [q3, [b]]\}$  und  $receive(\{[q0, [r]], [q1, []], [q2, []], [q3, [f]], [q3, [b]]\}, b) = \{[q3, []]\}$ .

Mit Hilfe von Situationen und dem Wissen über den Einfluss von Kommunikationsereignissen auf eine Menge von Situationen, sind wir in der Lage, die Frage nach der Bedienbarkeit des durch  $P$  repräsentierten Service unabhängig eines konkreten Partners  $R$  beantworten zu können.

### 2.4.4 Generierung von Bedienungsanleitungen

Eine Bedienungsanleitung für einen Service  $P$  soll alle möglichen Strategien von  $P$  charakterisieren, d.h. genau solch ein Verhalten, dass Deadlocks im komponierten System  $P \oplus R$  sicher vermeidet. Aus der Sicht von  $R$  stellt sich ein Deadlock von  $P \oplus R$  dabei wie folgt dar.

**Lemma 2.1 ([LMW07])**

$[q_P, q_R, m]$  ist ein Deadlock von  $P \oplus R$ , gdw. alle der folgenden Bedingungen erfüllt sind:

- $q_P \notin F_P$  oder  $q_R \notin F_R$  oder  $m \neq []$ ,
- $q_R$  ist ein Wartezustand von  $R$ ,
- $[q_P, m]$  ist stabil in  $K(q_R)$  und  $m(x) = 0$  für alle  $x \in wait(q_R)$ .

Betrachten wir noch einmal den in Abschnitt 2.4.1 gezeigten Deadlock  $[q3, z1, [f]]$  des Serviceautomaten  $A_{K_P} \oplus A_{K_F}$  aus Abbildung 2.6 unter Berücksichtigung der Bedingungen aus Lemma 2.1. Erstens ist  $[f] \neq []$ , zweitens ist  $z1$  ein Wartezustand von  $A_{K_F}$  mit  $wait(z1) = \{b\}$  und drittens ist  $[q3, [f]]$  stabil in  $K(z1)$  und  $[f](b) = 0$ .  $[q3, z1[f]]$  erfüllt also alle Bedingungen des Lemmas und ist daher ein Deadlock.

Die Bedingungen von Lemma 2.1 können durch boolesche Formeln  $\phi_1(q_P, m)$ ,  $\phi_2$  und  $\phi_3(m)$  beschrieben werden, mit deren Hilfe sich Deadlocks der Form  $[\cdot, q_R, \cdot]$  in  $P \oplus R$  ausschließen lassen. Das heißt, dass nur anhand der für den Zustand  $q_R$  konstruierten Formel und den von  $q_R$  ausgehenden Transitionen entschieden werden kann, ob  $q_R$  ein Deadlock von  $P \oplus R$  ist. Die Formeln nutzen die Menge von Aussagen  $C \cup \{\tau, final\}$  (mit  $final \notin C$ ). Aussagen aus  $C \cup \{\tau\}$  repräsentieren Beschriftungen der Transitionen, die  $q_R$  verlassen, und  $final$  repräsentiert die Tatsache, ob  $q_R \in F_R$ .

**Definition 2.18 (Annotation, R-Zuweisung)**

Seien  $P$  und  $R$  Partner. Wir definieren, für jedes  $q_R \in Q_R$ , die Annotation  $\phi(q_R)$  von  $q_R$  als boolesche Formel über den Aussagen  $C \cup \{\tau, final\}$  wie folgt:

$$\phi(q_R) = \bigwedge_{[q_P, m] \text{ ist stabil in } K(q_R)} (\phi_1(q_P, m) \vee \phi_2 \vee \phi_3(m))$$

wobei

- $\phi_1(q_P, m) = \begin{cases} \text{final}, & \text{falls } q_P \in F_P \text{ und } m = [], \\ \text{falsch}, & \text{sonst}, \end{cases}$
- $\phi_2 = \tau \vee \bigvee_{x \in O_R} x,$
- $\phi_3(m) = \bigvee_{x \in I_R, m(x) > 0} x.$

Die  $R$ -Zuweisung  $ass_R(q_R) : C \cup \{\tau, \text{final}\} \rightarrow \{\text{wahr}, \text{falsch}\}$  weist einer Aussage  $x \in C \cup \{\tau\}$  wahr zu, gdw. es ein  $q'_R$  gibt, so dass  $[q_R, x, q'_R] \in \delta_R$  und sie weist final wahr zu, gdw.  $q_R \in F_R$ .

Da die booleschen Formeln eine Kodierung der Bedingungen aus Lemma 2.1 darstellen, kann gezeigt werden, dass das Transitionssystem  $P \oplus R$  genau dann frei von Deadlocks ist, wenn der Wert von  $\phi(q_R)$  mit der  $R$ -Zuweisung  $ass_R(q_R)$  für jedes  $q_R \in Q_R$  wahr ist [LMW07].

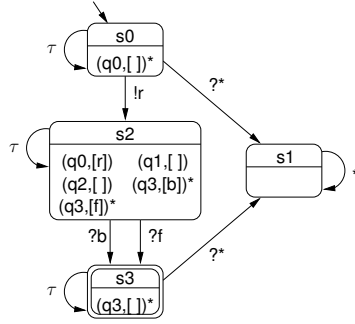
Wenn es für den Service  $P$  eine Strategie gibt, so muss mindestens einen Partner  $S$  existieren, für den das Transitionssystem  $P \oplus S$  frei von Deadlocks ist. Wir versuchen  $S$  zu konstruieren und legen dabei die folgenden Überlegungen zu Grunde: Ein Zustand von  $S$  ist eine Menge von Situationen von  $P$ . Zustände und Zustandsübergänge in  $S$  sind so angelegt, dass für alle Zustände  $q$  von  $S$ ,  $K(q) = q$  gilt. Jeder Zustand ist so über die Menge von Situationen definiert, in deren Verbindung er auftreten kann. Zustandsübergänge werden durch Anwendung der Operationen *event* und *cl* bestimmt. Ausgehend von einem Service  $S_0$ , der all diese Zustände und Zustandsübergänge enthält, ist es möglich, dass  $S_0 \oplus P$  Deadlocks besitzt. Allerdings kann durch Entfernen aller Zustände  $q$ , für die  $\phi(q)$  den Wert falsch liefert, ein neuer Service  $S_1$  gewonnen werden, der eben diese Deadlocks nicht mehr enthält. Durch wiederholtes Anwenden dieses Verfahrens erhalten wir entweder einen Service, der alle Annotationen erfüllt und damit problemlos mit  $P$  interagieren kann - eine Strategie von  $P$  -, oder wir erreichen ein Punkt, an dem die Menge der Zustände leer ist. Es gibt in diesem Fall keinen Partner, mit dem  $P$  problemlos interagieren kann.

**Definition 2.19 (Kanonischer Partner S)**

Sei  $P$  ein Serviceautomat. Wir definieren eine Sequenz von Serviceautomaten  $S_i = (Q_i, I_i, O_i, \delta_i, q_{0i}, F_i)$  wie folgt induktiv:  
 Sei  $Q_0 = 2^{Q_P \times \text{bags}(C)}$ . Für alle  $i$ , sei  $I_i = O_P$ ,  $O_i = I_P$ ,  $[q, x, q'] \in \delta_i$  gdw.  $q, q' \in Q_i$  und  $q' = cl(\text{event}(q, x))$ ,  $q_{0i} = cl(\{[q_{0P}, []]\})$ , und  $F_i = \{q \in Q_i \mid q \text{ ist Wartezustand von } S_i\}$ .  
 Sei, für alle  $i$ ,  $Q_{i+1} = \{q \mid q \in Q_i, \phi(q) \text{ hat den Wert wahr für die Zuweisung } ass_{S_i}(q)\}$ .  
 Sei  $S$  gleich  $S_i$ , für das kleinste  $i$ , das die Bedingung  $S_i = S_{i+1}$  erfüllt.

$S$  ist ein Serviceautomat (und Partner von  $P$ ), gdw.  $q_{0S} \in Q_S$ .

Abbildung 2.8 zeigt den kanonischen Partner  $S_K$  des Kinokarten-Service  $A_{K_P}$ . Jeder Zustand des kanonischen Partners hat eine mit  $\tau$  beschriftete Kante zu sich selbst, d.h. er kann beliebig viele interne Schritte zwischen den einzelnen Kommunikationsaktivitäten vollziehen. Um die Übersichtlichkeit zu erhöhen, haben wir einige Kanten zusammengefasst. Dabei steht  $?*$  für ein beliebiges Empfangsereignis, so z.B. sowohl für  $?b$  als auch


 Abbildung 2.8: Kanonischer Partner  $S_K$  des Kinokarten-Service  $A_{K_P}$ 

für  $?f$ , und  $*$  für ein beliebiges Sende-, Empfangs- oder internes Ereignis. Im Folgenden werden wir die Zustände eines annotierten Serviceautomaten in den Abbildungen als abgerundete Rechtecke darstellen, um den Unterschied zu reinen Serviceautomaten deutlich zu machen.

Mit der Konstruktion von  $S$  haben wir einen großen Schritt in Richtung der Bedienungsanleitung für  $P$  getan. Zum einen ist  $S$  eine Strategie von  $P$  und zum anderen können wir  $S$  mit den booleschen Formeln aus Definition 2.18 beschriften und dadurch alle weiteren Strategien von  $P$  charakterisieren. Um anschließend überprüfen zu können, ob ein Partner  $R$  von  $P$  den Anforderungen der Bedienungsanleitung von  $P$  genügt, müssen wir  $R$  mit der Bedienungsanleitung von  $P$  in Beziehung setzen. Dieser Vorgang wird als *Matching* bezeichnet und ist wie folgt definiert.

**Definition 2.20 (Matching)**

Seien  $R_1 = (Q_1, I_1, O_1, \delta_1, q_{0_1}, F_1)$  und  $R_2 = (Q_2, I_2, O_2, \delta_2, q_{0_2}, F_2)$  zwei Serviceautomaten. Dann ist das Matching von  $R_1$  und  $R_2$  als Relation  $L_{R_1, R_2} \subseteq Q_1 \times Q_2$  wie folgt induktiv definiert: Sei  $(q_{0_1}, q_{0_2}) \in L_{R_1, R_2}$ . Wenn  $(q_1, q_2) \in L_{R_1, R_2}$ ,  $(q_1, x, q'_1) \in \delta_1$  und  $(q_2, x, q'_2) \in \delta_2$  ist, dann ist auch  $(q'_1, q'_2) \in L_{R_1, R_2}$ .

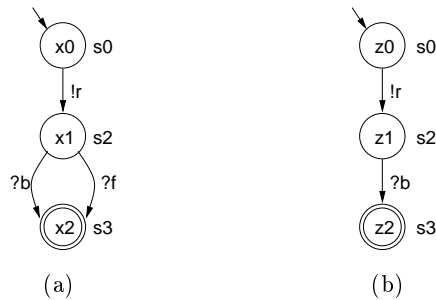

 Abbildung 2.9: Partnerautomaten  $A_{K_R}$  (a) und  $A_{K_F}$  (b) des Serviceautomaten  $A_{K_P}$  und ihr Matching mit  $S_K$

Abbildung 2.9 zeigt das Matching der Partnerautomaten  $A_{K_R}$  und  $A_{K_F}$  des Serviceautomaten  $A_{K_P}$  mit dem kanonischen Partner  $S_K$ . Die Zustände der Automaten  $A_{K_R}$  und  $A_{K_F}$  wurden dafür mit den in der Matching-Relation stehenden Zuständen von  $S_K$  annotiert.

Nun können wir uns der formalen Definition der Bedienungsanleitung widmen.

**Definition 2.21 (Bedienungsanleitung)**

Sei  $P$  ein Serviceautomat, für den es mindestens einen Partner  $R$  gibt, so dass  $P \oplus R$  frei von Deadlocks ist. Dann bezeichnen wir jeden Automaten  $S^*$ , der isomorph zum kanonischen Partner  $S$  von  $P$  unter dem Isomorphismus  $h$  ist, zusammen mit einer Annotation  $\Phi$  mit  $\Phi(q_{S^*}) = \phi(h(q_S))$ , für alle  $q_{S^*}$ , als Bedienungsanleitung für  $P$ .

**Satz 2.1 ([LMW07])**

$R$  ist ein Partner von  $P$ , so dass  $P \oplus R$  frei von Deadlocks ist, gdw. die folgenden Bedingungen für jedes  $[q_R, q_{S^*}] \in L_{R,S^*}$  erfüllt sind:

*(Topologie)* Für jedes  $x \in C \cup \{\tau\}$  gilt, wenn es einen  $x$ -Zustandsübergang gibt, der  $q_R$  in  $R$  verlässt, dann gibt es einen  $x$ -Zustandsübergang, der  $q_{S^*}$  in  $S^*$  verlässt.

*(Annotation)* Die Zuweisung  $ass_R(q_R)$  erfüllt  $\Phi(q_{S^*})$ .

Abbildung 2.10 zeigt eine Bedienungsanleitung des Kinokarten-Service, die auf Basis des Serviceautomaten  $A_{K_P}$  berechnet wurde. Die Formeln wurden soweit möglich vereinfacht und die  $\tau$ -Kanten weggelassen. Der Serviceautomat  $A_{K_R}$  aus Abbildung 2.5 (a) erfüllt alle Bedingungen der Bedienungsanleitung und kann daher problemlos mit  $A_{K_P}$  interagieren. Der Serviceautomat  $A_{K_F}$  aus Abbildung 2.5 hingegen erfüllt die Topologie-Bedingung nicht, da ihm im Zustand  $z1$  eine mit  $?f$  beschriftete Kante fehlt, und ist daher keine Strategie von  $A_{K_P}$ .

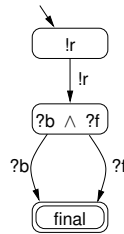


Abbildung 2.10: Bedienungsanleitung des Kinokarten-Service  $A_{K_P}$

Die Bedienungsanleitung eines Service  $P$  wird häufig auch als OG von  $P$  oder  $OG_P$  (von engl. *operating guideline*) bezeichnet.

Im Gegensatz zur Bedienungsanleitung von  $P$ , die das von möglichen Partnern erwartete bzw. tolerierte Verhalten beschreibt, steht beim *Public* und *Private View* der Prozess  $P$  selbst, d.h. seine interne Struktur bzw. sein Verhalten, im Vordergrund. Mit diesen von Partnerschaft und Interaktion vorerst völlig unabhängigen Begriffen wollen wir uns im folgenden Abschnitt beschäftigen.

## 2.5 Private und Public View

In diesem Abschnitt wollen wir noch einmal auf den Begriff *Public View* zurückkommen und ihn in Abgrenzung zum Begriff *Private View* etwas genauer erläutern.

*Private View* und *Public View* bezeichnen zwei unterschiedliche Sichten auf ein und den selben Prozess. Allerdings unterscheiden sich diese Sichtweisen in dem Zweck auf den sie ausgerichtet sind. Im Folgenden wollen wir die beiden Begriffe einmal gegenüber stellen.

Der *Private View* stellt die detailreichste Sicht auf den gesamten Prozess dar und enthält alle Aktivitäten des Prozesses, sowie Informationen über die an die Aktivitäten gebundenen Akteure, Ressourcen und Daten. Alle datenabhängigen Entscheidungen des Prozesses (z.B. Rabattklassen entsprechend dem Umsatz pro Kunde) sind präzise formuliert. Der *Private View* ist die Grundlage der rechnergestützten Ausführung, Verwaltung und Überwachung des Prozesses. Aufgrund der Informationsfülle kann der *Private View* ohne entsprechende Aufbereitung durch eine Software-Lösung kaum mehr von Menschen in seiner Gesamtheit aufgefasst werden.

Abbildung 2.11 zeigt einen vereinfachten *Private View* auf den Kinokarten-Service aus Abschnitt 2.1. Dieser *Private View* wurde als oWFN modelliert und enthält daher keine Informationen über mögliche Akteure, Ressourcen und Daten. Aufgrund der Beschriftung der Transitionen vermittelt das Modell jedoch einen relativ ausführlichen Einblick in den Ablauf des Prozesses.

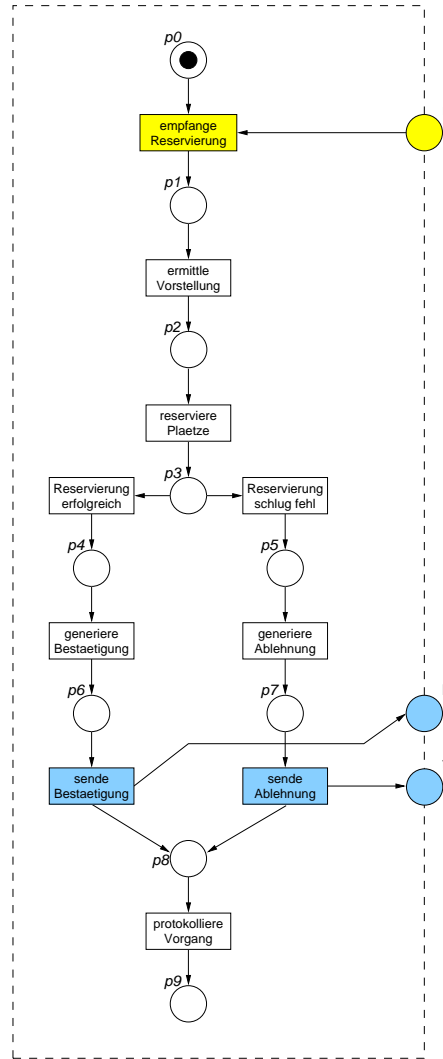


Abbildung 2.11: Private View des Kinokarten-Service (vereinfacht)

Ein *Public View* stellt eine abstrakte vereinfachte Sicht auf den Prozess dar. Zweck eines Public View ist es zumeist, einem Menschen den Ablauf des Prozesses darzulegen und dabei die für das Verständnis irrelevanten bzw. vertraulichen Details zu verbergen. Es gibt daher nicht *den* Public View eines Prozesses, sondern eine Menge an Public Views, die auf bestimmte Adressaten ausgerichtet sind. So wird das Management eines Unternehmens von einem Public View andere Informationen erwarten, als sie externen Kooperationspartner bereitgestellt werden. Jeder Public View muss die Zusammenhänge zwischen den Aktivitäten des Prozesses respektieren, es können aber z.B. Entscheidungen, auf die ein Kooperationspartner bzw. Benutzer keinen Einfluss hat, als nichtdeterministisch dargestellt und so von der konkreten Entscheidungslogik abstrahiert werden.

Abbildung 2.12 zeigt einen möglichen Public View auf den Kinokarten-Service aus Abschnitt 2.1. Dieser Public View wurde als oWFN modelliert und ähnelt daher dem Netz  $N_{K_P}$  aus Abschnitt 2.3. Aufgrund der Beschriftung der Transitionen ist dieser Public View jedoch anschaulicher als das Netz  $N_{K_P}$ . Obwohl es sich hier um ein sehr einfaches Beispiel handelt, ist sofort ersichtlich, dass der Public View aufgrund seiner geringeren Komplexität weitaus besser geeignet ist, einem potentiellen Nutzer die Bedienung des Prozesses zu erklären als der Private View aus Abbildung 2.11.

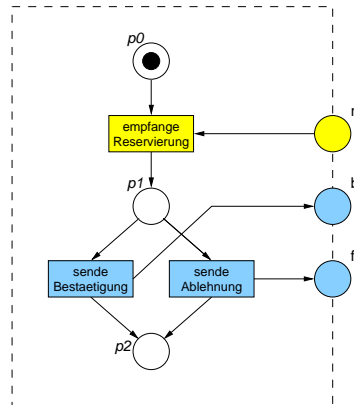


Abbildung 2.12: Public View des Kinokarten-Service

In einer service-orientierten Umgebung bezeichnen wir mit dem Begriff Public View häufig die Sicht auf den Prozess, die zum Zweck des Matchings des Prozesses eines Anbieters mit dem eines Interessenten bei einem Verzeichnisdienst hinterlegt wird. Anders als z.B. in einem Top-Down-Entwicklungsverfahren, bei dem ausgehend von einem Public View durch Verfeinerung der Aktivitäten ein Private View erzeugt wird, spielt die strukturelle Ähnlichkeit in diesem Fall keine Rolle. Es geht einzig und allein um das Kommunikationsverhalten, wenn eine problemlose Interaktion zwischen den Prozessen eines Anbieter und eines Interessenten sicherzustellen ist. Dieser Umstand rechtfertigt auch unsere verhaltensbasierte Herangehensweise, die möglicherweise einen Public View  $P'$  erzeugt, der rein strukturell kaum Ähnlichkeiten mit dem Prozess  $P$  aufweist.

Private View und Public View müssen keineswegs in der selben Sprache definiert sein. Es gibt allerdings einige Sprachen bzw. Methoden, die sich sowohl zur Modellierung des Private View als auch des Public View eignen. So sieht z.B. die zurzeit recht populäre Sprache WS-BPEL [AAA<sup>+</sup>07] dafür zwei Klassen von Prozessen vor - *Abstract* und *Executable Processes*. Auch mit Petrinetzen ist man durch die Kombination von Low-Level- (z.B. oWFNs) und High-Level-Netzen (z.B. XML-Netze [LO02]) in der Lage, beide Sichten auf einen Prozess mit der selben Technik zu modellieren.

Damit haben wir die erforderliche Terminologie für unser weiteres Vorgehen geklärt und können uns nun dem Problem der Public-View-Generierung zuwenden.



## 3 Verhaltensbasierte Public-View-Generierung

In diesem Kapitel wollen wir zwei Verfahren zur verhaltensbasierten Public-View-Generierung vorstellen. Um Aussagen über das Verhalten eines Service treffen zu können, muss dessen Zustandsraum berechnet werden. Anschließend kann dann untersucht werden, welche Aktionen zu welchen Zuständen führen und welche weiteren Aktionen von diesen Zuständen aus möglich sind. Da der Zustandsraum eines Service im Allgemeinen sehr groß ist, wünschen wir uns einen Formalismus, der eine möglichst einfache Darstellung des Verhaltens liefern kann. Insbesondere interessieren wir uns dabei für die Sequenzen von Aktionen, die in einen Endzustand des Service führen und damit eine vernünftige Interaktion mit einem Partner repräsentieren. Im Gegensatz dazu können wir für unsere Betrachtungen auf die Untersuchung von Folgezuständen aller Zustände verzichten, von denen aus ohnehin kein Endzustand des Service mehr erreicht werden kann.

Außer mit dem in Kapitel 2 erläuterten Konzept der Bedienungsanleitung lässt sich das Verhalten eines Service z.B. auch durch die von ihm erzeugte Sprache, seinen Interaktionsgraphen [Wei04], mit Hilfe temporaler Logik (z.B. TLDA [Ale06]), Prozessalgebren (z.B. CSP [Hoa85], CCS [Mil80]) oder mittels Ereignisstrukturen [NPW79] modellieren. Liegt eine vollständige Verhaltensbeschreibung für einen Service vor, sind für die Public-View-Generierung keine weiteren Informationen über die innere Struktur des Service mehr nötig. Wir werden jedoch zeigen, dass nicht alle dieser Modellierungstechniken das Verhalten derart beschreiben können, dass sich ein Public View aus der Beschreibung generieren lässt.

In Abschnitt 3.1 stellen wir einen Ansatz vor, der die Idee einer Public-View-Generierung auf Basis der kausalen Beziehungen zwischen den Nachrichten eines Service beschreibt. Eine mögliche Umsetzung dieser Idee, bei der die kausale Beziehung aus der vom Service erzeugten Sprache extrahiert werden, präsentieren wir in Abschnitt 3.1.2. Allerdings gibt es generelle Probleme mit einer sprachbasierten Umsetzung dieses Ansatzes, die wir in Abschnitt 3.1.3 diskutieren. Dabei kommen wir zu dem Schluss, dass allein anhand der Sprache eines Service kein Public View generiert werden kann.

In Abschnitt 3.2 präsentieren wir daher einen weiteren Ansatz zur verhaltensbasierten Public-View-Generierung, der auf der Bedienungsanleitung als Repräsentation des Verhaltens des Service aufsetzt. Dieser Ansatz ist allgemein anwendbar und immer erfolgreich, so dass mit Hilfe dieser Methode für jeden wie auch immer modellierten Service  $P$  ein Public View generiert werden kann, solange eine Bedienungsanleitung von  $P$  vorliegt.

## 3.1 Über die kausalen Beziehungen zwischen den Nachrichten des Service zum Public View

Wir wollen uns nun mit einem ersten Ansatz zur verhaltensbasierten Public-View-Generierung beschäftigen. Ausgehend von einer Analyse des Nachrichtenverkehrs sollen dabei die kausalen Beziehungen zwischen den Nachrichten des Service  $P$  ermittelt werden. Im Anschluss daran wird ein Service  $P'$  konstruiert, der die selben kausalen Beziehungen zwischen den Nachrichten implementiert wie  $P$ . Damit besitzt  $P'$  das gleiche Kommunikationsverhalten wie  $P$  und ist im Allgemeinen von erheblich geringer Komplexität, da von den internen Aktivitäten von  $P$  weitestgehend abstrahiert wird.

In Abschnitt 3.1.1 werden wir die allgemeine Idee des Verfahrens skizzieren. Im Anschluss daran zeigen wir in Abschnitt 3.1.2, wie eine konkrete Umsetzung dieser Idee auf Basis der Sprache des Service aussehen könnte. Abschließend diskutieren wir in Abschnitt 3.1.3 die generellen Probleme einer sprachbasierten Umsetzung und demonstrieren anhand eines Beispiels, dass ein solches Vorgehen nicht ohne Weiteres gelingen kann.

### 3.1.1 Idee des Verfahrens

Die Abarbeitung der einzelnen Aktivitäten innerhalb eines Service kann von der Umwelt im Allgemeinen nicht beobachtet werden. Von außen lässt sich das Verhalten des Service daher nur über die von ihm gesendeten und empfangenen Nachrichten beschreiben. Allerdings ist es ja gerade ein entscheidendes Ziel der Public-View-Generierung, das interne Verhalten des Service weitestgehend zu verbergen. Wir wollen also versuchen, von der möglichen Reihenfolge der kommunizierten Nachrichten Rückschlüsse auf die sie sendenden bzw. empfangenden Aktivitäten zu ziehen.

So stellen wir z.B. bei der Beobachtung der Interaktion des Kinokarten-Service  $N_{K_P}$  aus Abbildung 2.1 fest, dass Nachricht  $b$  stets erst nach Nachricht  $r$  auftritt. Nachricht  $r$  ist also gewissermaßen ein Vorgänger von Nachricht  $b$ . Dies muss folglich auch für die  $r$  empfangenden bzw.  $b$  sendenden Aktivitäten gelten.

Generell können wir die folgenden Beziehungstypen zwischen zwei Nachrichten  $a$  und  $b$  unterscheiden:

- $a$  ist Vorgänger von  $b$  bzw.  $b$  ist Nachfolger von  $a$
- $a$  ist nebenläufig/parallel zu  $b$
- $a$  steht im Konflikt zu  $b$

Zudem lässt sich beobachten, welche Nachrichten wiederholt während einer Interaktion auftreten können und daraus folgern, dass die entsprechenden Aktivitäten Teil eines Zyklus innerhalb des Service sind. Weitere Informationen können wir auch aus dem Vergleich

verschiedener Kommunikationsszenarien gewinnen: Tritt z.B. eine Nachricht  $a$  während einer Interaktion auf und während einer weiteren nicht, obwohl keine Nachricht zu erkennen ist, die im Konflikt zu  $a$  steht, so steht die  $a$  kommunizierende Aktivität offenbar im Konflikt zu einer internen Aktivität. Wir bezeichnen eine solche Nachricht  $a$  dann als optionale Nachricht.

Haben wir die kausalen Beziehungen zwischen den von  $P$  kommunizierten Nachrichten und Informationen über Zyklen und optionale Nachrichten zur Verfügung, sollten wir in der Lage sein, mit einem Synthesalgorithmus die innere Struktur von  $P$ , zumindest in Bezug auf die Kommunikationsaktivitäten, rekonstruieren zu können. Ein so generierter Service  $P'$  besitzt dann das gleiche Kommunikationsverhalten wie  $P$ , da er die selben kausalen Beziehungen zwischen den Nachrichten implementiert, und beschränkt sich zudem auf die Darstellung der kommunizierenden Aktivitäten. Wir vermuten daher, dass  $P'$  demnach einen sehr guten Kandidaten für einen Public View von  $P$  darstellt.

#### 3.1.2 Eine mögliche Umsetzung

Für eine Umsetzung der oben skizzierten Idee stellt sich zuerst die Frage, wie sich die kausalen Beziehungen zwischen den Nachrichten eines Service berechnen lassen. Wir wollen im Folgenden versuchen, diese kausalen Beziehungen aus der Sprache des Service zu extrahieren.

#### Berechnung der kausalen Beziehungen

Die Grundidee der formalen Sprachtheorie ist die folgende: Gegeben eine Menge von Symbolen - ein Alphabet -, lassen sich durch Verkettung Worte über diesem Alphabet bilden. Eine Menge solcher Worte zusammen mit ihrem Alphabet wird dann als Sprache bezeichnet. Dies lässt sich auf die Welt der Services übertragen, indem die Menge der kommunizierten Nachrichten als Alphabet und Sequenzen von diesen Nachrichten als Worte aufgefasst werden. Da wir uns für das erfolgreiche Zusammenspiel von Services interessieren, wollen wir im Folgenden nur solche Sequenzen von Nachrichten (Worte) betrachten, die ausgehend vom Anfangszustand einen Service  $P$  und einen Partner  $R$  in einen Endzustand überführen. Damit können wir für jeden Service  $P$  seine spezifische Sprache  $L(P)$  angeben. Da die Worte der Sprache  $L(P)$  Sequenzen von zwischen  $P$  und  $R$  ausgetauschten Nachrichten entsprechen, müssen sich aus der Reihenfolge der Nachrichten in den einzelnen Worten Beziehungen zwischen den Nachrichten bzw. den sie versendenden und empfangenden Aktivitäten folgern lassen.

Die in Abschnitt 3.1.1 vorgestellten Beziehungstypen lassen sich aus der Sprache  $L(P)$  des Service  $P$  wie folgt ablesen:

- $a$  ist Vorgänger von  $b$  bzw.  $b$  ist Nachfolger von  $a$ , wenn  $a$  in den Worten von  $L(P)$  stets vor  $b$  auftritt.

- $a$  ist nebenläufig/parallel zu  $b$ , wenn  
 $a$  in den Worten von  $L(P)$  sowohl vor als auch nach  $b$  auftreten kann.
- $a$  steht im Konflikt zu  $b$ , wenn  
es kein Wort in  $L(P)$  gibt, in dem sowohl  $a$  als auch  $b$  vorkommen.

Des Weiteren müssen wir uns nun Klarheit darüber verschaffen, was wir denn konkret unter der Sprache  $L(P)$  eines Service  $P$  verstehen wollen und wie diese berechnet werden kann. Angenommen, wir modellieren  $P$  als Serviceautomaten  $A_P$ . Dann können wir die von  $A_P$  akzeptierte Sprache  $L(A_P)$  in Anlehnung an die klassische Automatentheorie [HMU07] wie folgt definieren.

**Definition 3.1 (akzeptierte Sprache eines Serviceautomaten)**

Sei  $A = (Q, I, O, \delta, q_0, F)$  ein Serviceautomat. Sei  $\Sigma_A = I \cup O \cup \{\tau\}$  das Alphabet von  $A$  und  $\Sigma_A^* = \{x_1 \dots x_n \mid n \geq 0 \text{ und } x_i \in \Sigma_A \text{ für } i = 1, \dots, n\}$  die Menge aller Worte über dem Alphabet  $\Sigma_A$ . Dann ist die von  $A$  akzeptierte Sprache  $L(A)$  wie folgt definiert:

$$L(A) = \{x_1 \dots x_n \in \Sigma_A^* \mid \exists q_1, \dots, q_{n-1} \in Q, \exists q_n \in F : [q_i, x_{i+1}, q_{i+1}] \in \delta \text{ für } i = 0, \dots, n-1\}$$

Die Worte der Sprache  $L(A)$  entsprechen dann Sequenzen von Sende-, Empfangs- und internen Ereignissen, die den Serviceautomaten  $A$  vom Anfangszustand  $q_0$  in einen Endzustand  $q_n$  überführen. Eine Beschränkung auf das nach außen sichtbare Verhalten des Service können wir dadurch erreichen, indem wir die internen Ereignisse, repräsentiert durch  $\tau$ , aus allen Worten von  $L(A)$  streichen.

Beispielhaft betrachten wir noch einmal den Partnerautomaten  $A_{K_R}$  (siehe Abbildung 2.5 (a)) des Kinokarten-Service  $A_{K_P}$ . Nach Definition 3.1 akzeptiert  $A_{K_R}$  die Sprache  $L(A_{K_R}) = \{!r?b, !r?f\}$ , was dem Senden einer Reservierungsanfrage und dem anschließenden Empfang einer Bestätigung bzw. Fehlermeldung entspricht. Ein Blick auf die Bedienungsanleitung von  $A_{K_R}$  (siehe Abbildung 3.1) zeigt jedoch, dass auch ein Partner, der zuerst Nachricht  $b$  sendet und anschließend Nachricht  $r$  empfängt, problemlos mit  $A_{K_R}$  interagieren kann. Demzufolge sollte also auch das Wort  $?b!r$  in der vom Serviceautomaten  $A_{K_R}$  akzeptierten Sprache liegen.

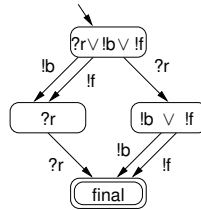


Abbildung 3.1: Bedienungsanleitung des Serviceautomaten  $A_{K_R}$

Der Grund dafür, dass  $?b!r$  (genauso wie  $?b!f$ ) kein Wort der Sprache  $L(A_{K_R})$  ist, liegt darin, dass diese klassische Definition der akzeptierten Sprache eines Automaten nicht die

Natur des Nachrichtenaustauschs berücksichtigt. Da wir in dieser Arbeit von asynchroner Kommunikation zwischen Services ausgehen wollen, ist es für einen Partner durchaus zulässig, Nachrichten quasi im Voraus zu versenden. Empfängt ein Service eine Nachricht, die er im aktuellen Zustand nicht verwerten kann, so verbleibt sie auf dem entsprechenden Nachrichtenkanal, bis zu dem Zeitpunkt, an dem sie tatsächlich konsumiert wird. Daher wird der Service  $A_{K_R}$  im Anfangszustand durch die eingehende Nachricht  $b$  nicht blockiert, sondern sendet zuerst Nachricht  $r$  und wartet anschließend auf Nachricht  $b$  oder  $f$ , wobei in diesem Fall Nachricht  $b$  direkt von Nachrichtenkanal konsumiert werden kann.

Offensichtlich reicht also die vom Serviceautomaten akzeptierte Sprache nicht aus, um das Kommunikationsverhalten eines Service vollständig abzubilden. Glücklicherweise haben wir aber bereits ein Konzept vorgestellt, das eine hinreichend genaue Verhaltensbeschreibung eines Service liefern kann - die Bedienungsanleitung. Die Bedienungsanleitung wird auf Basis der asynchronen Kommunikation zwischen Serviceautomaten berechnet und charakterisiert daher auch alle Sequenzen von Nachrichten, die über die akzeptierte Sprache des Serviceautomaten hinaus eine problemlose Interaktion ermöglichen. Wir wollen daher anstatt der akzeptierten Sprache des Serviceautomaten selbst, die akzeptierte Sprache seiner Bedienungsanleitung als Beschreibung seines Kommunikationsverhaltens nutzen.

**Definition 3.2 (Kommunikationssprache eines Serviceautomaten)**

Sei  $A = (Q, I, O, \delta, q_0, F)$  ein Serviceautomat und der Serviceautomat  $S_A$  zusammen mit einer Annotation  $\Phi$  eine Bedienungsanleitung für  $A$ . Dann bezeichnen wir die von  $S_A$  akzeptierte Sprache  $L(S_A)$  als die Kommunikationssprache des Serviceautomaten  $A$ .

Liegt uns für einen Service  $P$  seine Bedienungsanleitung vor, so können wir die Kommunikationssprache  $L(P)$  von  $P$  also daraus ablesen. Folgen wir der in Abschnitt 3.1.1 skizzierten Idee, müssen nun die kausalen Beziehungen zwischen den Nachrichten aus den Worten der Sprache  $L(P)$  extrahiert werden. Dem Autor ist allerdings kein Verfahren bekannt, mit dem sich die kausalen Beziehungen zwischen den Nachrichten aus einer regulären Sprache, wie sie von einer Bedienungsanleitung akzeptiert wird, automatisch berechnen ließen. Zudem lassen sich die kommunizierten Nachrichten nicht immer eindeutig den Kommunikationsaktivitäten des Service zuordnen. Wenn es zwei verschiedene Aktivitäten im Service gibt, die die selbe Nachricht produzieren bzw. konsumieren, können diese von einem Beobachter (bzw. in den Worten der Sprache) nicht mehr voneinander unterschieden werden. Abbildung 3.2 zeigt einen Service als oWFN  $N_M$ , bei dem die kausalen Beziehungen zwischen den Nachrichten ambivalent sind. Die beiden Aktivitäten, die die Nachricht  $c$  produzieren, können nicht anhand ihres Nachrichtentyps unterschieden werden. Es ergeben sich die folgenden kausalen Beziehungen:  $a$  ist Vorgänger von  $c$  und  $a$  ist nebenläufig zu  $c$ . Solche ambivalenten Beziehungen erschweren die anschließende Synthese des Public View erheblich.

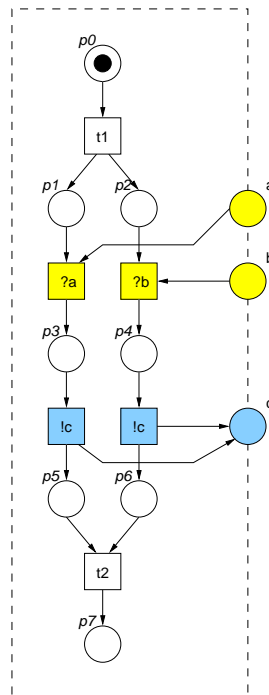


Abbildung 3.2: Service  $N_M$  mit ambivalenten Nachrichtenbeziehungen

### Synthese des Public View

Angenommen, die kausalen Beziehungen zwischen den Nachrichten ließen sich berechnen und es wurden auch Informationen über Zyklen und optionale Nachrichten aus der Kommunikationssprache des Service gewonnen, könnten wir uns nun mit der Synthese des Public View beschäftigen.

Möglich wäre ein Synthesearchiv nach einem Bottom-Up-Ansatz, d.h. wir könnten iterativ einen Prozess erzeugen, der aus Aktivitäten besteht, die das Kommunikationsverhalten des Service nachbilden. Für einen Public View auf Basis von oWFN hieße das: Aus elementaren Teilnetzen, die einzelne kommunizierende Aktivitäten repräsentieren, wird sukzessive ein komplexes Netz komponiert, das die kausalen Beziehungen der Nachrichten modelliert. Ein solcher Ansatz beruht im Grunde auf der Komposition von Petrinetzen. In der Literatur finden sich zahlreiche Arbeiten zum Thema Komposition, Dekomposition, Synthese und Transformation verschiedener Klassen von Petrinetzen, auf die man zur Synthese des Public View zugreifen könnte. Eine ausführliche Übersicht dieser Arbeiten bietet [RM97].

Abbildung 3.3 zeigt eine mögliche Modellierung elementarer Teilnetze als oWFN. Im Wesentlichen besteht ein elementares Teilnetz aus genau einem Start- und Endplatz und nur einer Transition, die entweder eine Nachricht produziert oder konsumiert oder, im Falle

einer internen Aktivität, einen sog.  $\tau$ -Übergang vollzieht. An  $\tau$ -Übergänge sind dabei keine Bedingungen geknüpft, d.h. sie können spontan nach ihrer Aktivierung schalten.

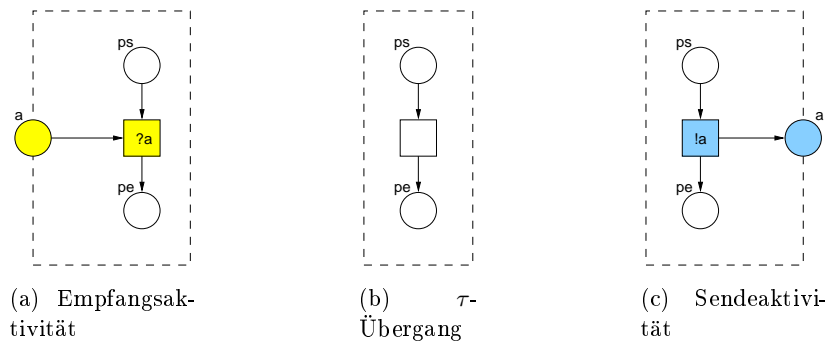


Abbildung 3.3: Elementare Teilnetze

Zur Komposition der elementaren Teilnetze muss dann für jeden der in Abschnitt 3.1.1 vorgestellten Beziehungstypen ebenfalls ein oWFN angegeben werden, das die entsprechende Nachrichtenbeziehung mit Teilnetzen modelliert. Als Beispiel wollen wir an dieser Stelle das Muster zur nebenläufigen Komposition von  $n$  Teilnetzen vorstellen (siehe Abbildung 3.4). Durch das Schalten von Transition  $t1$  werden alle Startplätze der parallel komponierten Teilnetze zeitgleich mit einer Marke belegt und können dann abgearbeitet werden. Die Transition  $t(n+2)$  synchronisiert die parallel komponierten Teilnetze und stellt dadurch sicher, dass die Abarbeitung eventuell folgender Teilnetze nicht eher beginnt, als dass alle parallel komponierten Teilnetze abgearbeitet wurden.

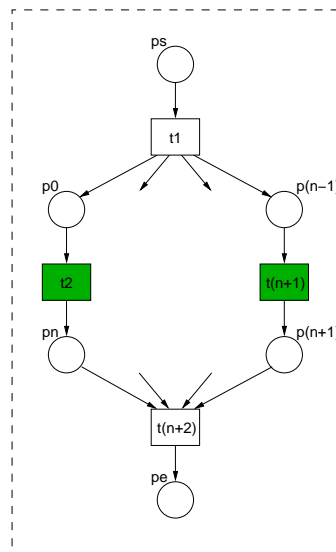


Abbildung 3.4: Muster zur nebenläufigen Komposition von  $n$  Teilnetzen

Das Ergebnis einer Komposition muss stets wieder ein Teilnetz sein, das (analog zu den elementaren Teilnetzen) genau einen Start- und Endplatz besitzt. Dann können durch wiederholtes Anwenden der Kompositionsregeln ausgehend von elementaren Teilnetzen komplexe Teilnetze gewonnen werden. Technisch lässt sich eine solche Komposition über eine Verfeinerung realisieren. Bestimmte Transitionen eines Teilnetzes (in Abbildung 3.4 grün gekennzeichnet), werden durch weitere Teilnetze ersetzt. Dabei werden Vor- und Nachbereich der Transition mit dem Start- bzw. Endplatz des ersetzenden Teilnetzes verschmolzen. Man erhält so mit jedem Verfeinerungsschritt ein komplexeres, aber stets zusammenhängendes und syntaktisch korrektes Petrinetz.

Da keines der vorgestellten Teilnetze eine Markierung enthält, ergibt ihre Komposition kein korrektes oWFN. Wir müssen also die generierte Netzstruktur noch mit Anfangs- und Endmarkierungen anreichern, um ein oWFN (und damit einen Public View) zu erhalten. Aufgrund der Vorgehensweise muss es genau einen Start- und Endplatz im komponierten Netz geben, der nicht im Zuge einer Verfeinerung mit einem weiteren Platz verschmolzen wurde. Dabei handelt es sich um die „äußerste“ Komposition. Bezeichnet man den entsprechenden Start- bzw. Endplatz o.B.d.A. mit  $ps$  bzw.  $pe$ , dann ist die Anfangsmarkierung  $m_0$  wie folgt definiert:  $m_0(ps) = 1$  und für alle  $p \in P$ , mit  $p \neq ps$  ist  $m_0(p) = 0$ . Die Markierung  $m_f$ , mit  $m_f(pe) = 1$  und für alle  $p \in P$ , mit  $p \neq pe$  ist  $m_f(p) = 0$ , ist eine der Endmarkierungen. Es kann durchaus mehrere Endmarkierungen geben, wie diese ermittelt werden können soll uns an dieser Stelle jedoch nicht weiter beschäftigen.

Wir haben damit gezeigt, wie die Synthese eines Public View auf Basis von kausalen Beziehungen aussehen könnte. Warum eine sprachenbasierte Umsetzung des Ansatzes im Allgemeinen jedoch nicht praktikabel ist, wollen wir im nächsten Abschnitt klären.

#### 3.1.3 Probleme eines sprachenbasierten Ansatzes

Die im vorigen Abschnitt besprochene mögliche Umsetzung einer sprachenbasierten Public-View-Generierung hat bereits einige Probleme des Ansatzes gezeigt. Insbesondere die Berechnung der kausalen Beziehungen zwischen den Nachrichten aus einer Menge von Worten stellt ein gravierendes Problem dar. Es stellt sich jedoch auch generell die Frage, ob mit der Kommunikationssprache eines Serviceautomaten dessen Kommunikationsverhalten tatsächlich vollständig beschrieben werden kann. Dies ist nur dann der Fall, wenn es keine zwei Services  $P_1$  und  $P_2$  gibt, die zwar die selbe Kommunikationssprache  $L(P_1) = L(P_2)$  aber trotzdem unterschiedliche Strategien besitzen.

Abbildung 3.5 zeigt zwei Services als oWFN  $N_1$  und  $N_2$ . Der Service  $N_1$  entscheidet zuerst intern, ob eine Nachricht  $a$  gesendet wird oder nicht und wartet anschließend auf eine Nachricht  $b$ . Der Service  $N_2$  sendet entweder eine Nachricht  $a$  und wartet anschließend auf Nachricht  $b$  oder wartet bereits im Anfangszustand auf eine Nachricht  $b$  und entscheidet sich anschließend, ob Nachricht  $a$  gesendet wird oder nicht. In der Anfangsmarkierung von  $N_2$  steht das Senden von Nachricht  $a$  im Konflikt zum Warten auf Nachricht  $b$ . Solange kein  $b$  auf dem entsprechenden Nachrichtenkanal liegt, wird dieser Konflikt früher oder





Abbildung 3.6: Strukturgleiche Bedienungsanleitungen der Services  $N_1$  und  $N_2$

### 3.2 Durch Spiegelung der Bedienungsanleitung zum Public View

In diesem Abschnitt wollen wir einen allgemein anwendbaren und erfolgreichen Ansatz zur verhaltensbasierten Public-View-Generierung vorstellen. Das Verfahren wurde von Wolf entwickelt und generiert für einen Service  $P$  seinen Public View  $P'$  auf Basis eines Serviceautomaten [Wol07].

In Kapitel 2 haben wir motiviert, warum wir uns in dieser Arbeit mit Geschäftsprozessmodellen auf Basis von Petrinetzen bzw. oWFN beschäftigen. Um das Konzept der Bedienungsanleitung einführen zu können, haben wir dann Serviceautomaten vorgestellt und u.a. gezeigt, wie oWFN in Serviceautomaten übersetzt werden können. Es ist daher zulässig, an dieser Stelle einen Ansatz zur Public-View-Generierung zu präsentieren, der einen Public View auf Basis eines Serviceautomaten berechnet. Nichtsdestotrotz wünschen wir uns konsequenter Weise für einen als oWFN modellierten Service einen als oWFN modellierten Public View. Wir wollen daher darauf hinweisen, dass Serviceautomaten mit Hilfe der Regionentheorie [BD98] in oWFN übersetzt werden können, uns aber hier mit den Details dieser Übersetzung nicht näher beschäftigen.

In Abschnitt 3.2.1 skizzieren wir die Idee des Ansatzes, die im Wesentlichen auf der Spiegelung der Bedienungsanleitung beruht. Die Grundlage des Public View ist der gespiegelten bzw. *duale Serviceautomat*, den wir in Abschnitt 3.2.2 erläutern. Da der duale Serviceautomat einer Bedienungsanleitung allein noch kein Public View ist, diskutieren wir in Abschnitt 3.2.3 die Schritte, die nötig sind, um aus einem dualen Serviceautomaten einen korrekten Public View zu generieren. Abschließend werfen wir in Abschnitt 3.2.4 einen kritischen Blick auf das Verfahren und zeigen die Stärken und Schwächen des Ansatzes auf.

#### 3.2.1 Idee des Verfahrens

In Kapitel 2 haben wir gezeigt, dass die Bedienungsanleitung  $OG_P$  eines Service  $P$  alle möglichen Strategien von  $P$  charakterisiert. Insbesondere ist  $OG_P$  ein beschrifteter Serviceautomat und, unter Vernachlässigung der Annotationen, selbst eine Strategie von  $P$ .

Da alle weiteren Strategien von  $P$  in  $OG_P$  enthalten sind, nennen wir den der  $OG_P$  zugrunde liegenden Serviceautomaten die allgemeinste Strategie von  $P$ . Die Tatsache, dass die allgemeinste Strategie das gesamte Verhalten von  $P$  abdeckt, legt die Vermutung nahe, dass sich aus ihr ein Public View für  $P$  generieren lässt.

Ein wichtiger Unterschied zwischen der Bedienungsanleitung und einem Public View ist der Blickwinkel in Bezug auf die Interaktion der Services. Während die Bedienungsanleitung eine Handlungsanweisung für mögliche Partner von  $P$  repräsentiert, ist der Public View im Gegensatz dazu ein Stellvertreter von  $P$  und repräsentiert somit das Verhalten von  $P$  selbst. Um aus der allgemeinsten Strategie von  $P$  einen Public View zu konstruieren, muss daher dieser Blickwinkel verändert werden. Dies kann durch Vertauschen der Ein- und Ausgabekanäle von  $OG_P$  erreicht werden. Wolf bezeichnet den aus dieser Spiegelung des Kommunikationsverhaltens resultierenden Automaten als den *dualen Serviceautomaten* von  $OG_P$  [Wol07].

Der duale Serviceautomat der allgemeinsten Strategie von  $P$  allein ist jedoch noch kein Public View für  $P$ . Der Grund dafür liegt darin, dass eine Bedienungsanleitung nicht nur das erwünschte Verhalten charakterisiert. Sie drückt zugleich auch aus, welches Verhalten verboten ist, wenn die Interaktion der beiden Services zu einem vernünftigen Abschluss gelangen soll. Um diesem Umstand Rechnung zu tragen, sind daher im Anschluss an die Generierung des dualen Serviceautomaten zwei weitere Transformationsschritte nötig, die u.a. die in den Annotationen der Bedienungsanleitung enthaltenen Informationen ausnutzen, um den dualen Serviceautomaten von  $OG_P$  in einen korrekten Public View von  $P$  zu überführen.

### 3.2.2 Der duale Serviceautomat

Die Bedienungsanleitung  $OG_P$  eines Service  $P$  beschreibt die Kommunikation zwischen  $P$  und einem Partner  $R$  aus der Sicht von  $R$ . Das heißt, dass eine Nachricht, die von  $P$  über einen Kanal  $x$  gesendet wird, ein Empfangsereignis für  $R$  darstellt. Dementsprechend ist  $x$  ein Eingabekanal des annotierten Serviceautomaten  $OG_P$ . In einem Public View von  $P$  wird die Kommunikation jedoch aus der Sicht von  $P$  modelliert. Wir werden daher im ersten Schritt der Public-View-Generierung einen Serviceautomaten konstruieren, der die gleiche Gestalt wie  $OG_P$  besitzt, aber das gespiegelte Kommunikationsverhalten aufweist. Dies kann dadurch erreicht werden, indem die Menge der Ein- und Ausgabekanäle von  $OG_P$  vertauscht wird. Das Resultat dieser Spiegelung nennen wir den zu  $OG_P$  *dualen Serviceautomaten*, der wie folgt definiert ist.

#### Definition 3.3 (dualer Serviceautomat)

Sei  $A = (Q, I, O, \delta, q_0, F)$  ein Serviceautomat. Dann besitzt der zu  $A$  duale Serviceautomat  $\overline{A} = (\overline{Q}, \overline{I}, \overline{O}, \overline{\delta}, \overline{q_0}, \overline{F})$  die folgenden Bestandteile:  $\overline{Q} = Q$ ,  $\overline{I} = O$ ,  $\overline{O} = I$ ,  $\overline{\delta} = \delta$ ,  $\overline{q_0} = q_0$  und  $\overline{F} = F$ .

Wenn wir vom dualen Serviceautomaten  $\overline{OG_P}$  einer Bedienungsanleitung  $OG_P$  sprechen, soll damit implizit der duale Serviceautomat des  $OG_P$  zugrunde liegenden Serviceautomaten gemeint sein. Der Serviceautomat  $\overline{OG_P}$  ist keine Bedienungsanleitung für den

Service  $P$  und bedarf daher auch keiner Annotationen. Das  $\overline{OG_P}$  allein jedoch auch noch keinen Public View von  $P$  darstellt, wollen wir nun an einem kleinen Beispiel deutlich machen.

Abbildung 3.7 (a) zeigt einen Serviceautomaten  $P$ , der eine Nachricht  $a$  sendet und anschließend eine Nachricht  $b$  erwartet. Sendet ein Partner  $R$  von  $P$  die Nachricht  $b$  bereits bevor er Nachricht  $a$  erhalten hat, kann es zu einem Deadlock in  $P$  und damit auch in der Interaktion von  $P$  und  $R$  kommen. Dies ist dann der Fall, wenn  $P$  im Zustand  $q_0$  durch das Konsumieren der Nachricht  $b$  in den Zustand  $q_3$  übergeht, von dem aus kein Endzustand von  $P$  mehr erreichbar ist. Die Bedienungsanleitung  $OG_P$  (siehe Abbildung 3.7 (b)) charakterisiert daher das Verhalten einer jeden Strategie für  $P$  wie folgt: Warte auf Nachricht  $a$  und sende anschließend Nachricht  $b$ . Durch Spiegelung der Bedienungsanleitung von  $P$ , was der Berechnung des zu  $OG_P$  dualen Serviceautomaten  $\overline{OG_P}$  entspricht, erhalten wir den Serviceautomaten  $P'$  aus Abbildung 3.7 (c). Ein Vergleich der Bedienungsanleitungen  $OG_P$  und  $OG_{P'}$  (siehe Abbildung 3.7 (d)) zeigt jedoch, dass  $P$  und  $P'$  nicht die selbe Menge an Strategien besitzen. Ein Service der zuerst Nachricht  $b$  sendet und anschließend auf Nachricht  $a$  wartet ist eine Strategie für  $P'$ , aber nicht für  $P$ , da  $OG_P$  im Anfangszustand keine Transition  $!b$  enthält und damit das Senden von Nachricht  $b$  in diesem Zustand verbietet. Folglich kann  $P'$  kein Public View für  $P$  sein.

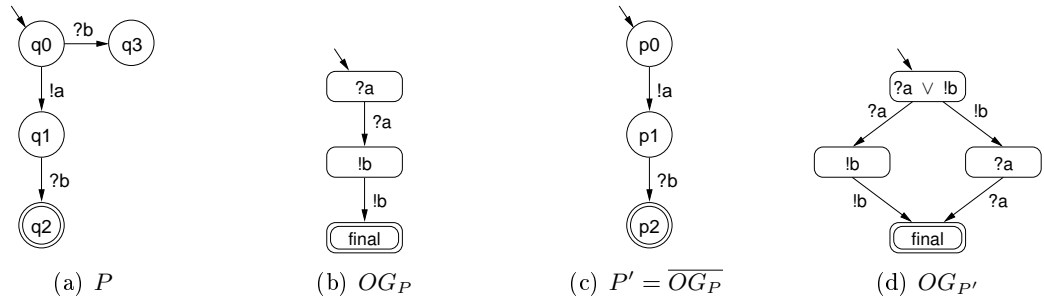


Abbildung 3.7: Serviceautomat  $P$  mit Bedienungsanleitung  $OG_P$  und dualer Serviceautomat  $P' = \overline{OG_P}$  mit Bedienungsanleitung  $OG_{P'}$

Offensichtlich sind weitere Schritte nötig, um aus dem dualen Serviceautomaten von  $OG_P$  einen Public View für  $P$  zu generieren. Wie diese Schritte im Detail aussehen, wollen wir im nächsten Abschnitt klären.

### 3.2.3 Vom dualen Serviceautomaten zum Public View

Der duale Serviceautomat einer Bedienungsanleitung  $OG_P$  modelliert noch nicht alle Aspekte des Verhaltens des Service  $P$ . Wir wollen daher die zwei in [Wol07] formulierten Transformationsschritte vorstellen, durch deren Anwendung der duale Serviceautomat um weitere Zustände und Transitionen angereichert wird, bis das gesamte Verhaltensspektrum von  $P$  abgebildet ist.

Wie eingangs erwähnt, charakterisiert die Bedienungsanleitung eines Service mehr als nur das vom Partner erwartete Verhalten. In jedem Zustand der Bedienungsanleitung wird über die ausgehenden Kanten beschrieben, welche Aktionen (Senden oder Empfangen einer Nachricht) an diesem Punkt für eine sinnvolle Fortsetzung der Interaktion möglich sind. Darüber hinaus bedeutet jedoch die Abwesenheit der Kante für eine Aktion, dass diese Aktion in diesem Zustand nicht durchgeführt werden darf!

Für den dualen Serviceautomaten einer Bedienungsanleitung  $OG_P$  bedeutet dies, dass in jedem Zustand  $q$ , für den die Bedienungsanleitung  $OG_P$  das Senden einer Nachricht  $x$  durch den Partner nicht vorsieht (und damit implizit verbietet), der Empfang einer Nachricht  $x$  zu einem Deadlock führen kann. Da diese Tatsache allein durch Spiegelung der Bedienungsanleitung keine Berücksichtigung findet, formulieren wir einen ersten Transformationsschritt, der den dualen Serviceautomaten um einen expliziten Deadlock erweitert und die durch die Bedienungsanleitung verbotenen Aktionen durch entsprechende Zustandsübergänge zu diesem Deadlock modelliert.

**Definition 3.4 (1. Transformationsschritt)**

*Wenn in einem Zustand  $q$  von  $OG_P$  für eine Nachricht  $a \in O_{OG_P}$  keine ausgehende Transition existiert, dann füge in  $\overline{OG_P}$  die Transition  $[q, ?a, d]$  zu einem Deadlock  $d$  ein (ggf. muss  $\overline{OG_P}$  dafür um  $d$  erweitert werden).*

Abbildung 3.8 verdeutlicht die Anwendung dieses Transformationsschrittes an einem kleinen Ausschnitt aus einer Bedienungsanleitung  $OG_P$ . Vom Zustand  $q$  aus gibt es nur einen Zustandsübergang, der über  $?x$  in den Zustand  $q1$  führt. Ein Partner von  $P$  muss an dieser Stelle also in der Lage sein, eine Nachricht  $x$  zu empfangen, um der Bedienungsanleitung zu genügen. Zusätzlich verbietet die Bedienungsanleitung implizit jedoch auch das Senden jeder beliebigen Nachricht im Zustand  $q$ . Angenommen, die Menge der Ausgabekanäle von  $OG_P$  beschränkt sich allein auf einen Kanal  $a$ , so hat nach Anwenden des ersten Transformationsschrittes der entsprechende Ausschnitt des dualen Serviceautomaten  $\overline{OG_P}$  die auf der rechten Seite der Abbildung gezeigte Gestalt. Besitzt die Menge der Ausgabekanäle von  $OG_P$  eine höhere Kardinalität, so wird durch den Transformationsschritt für jeden Kanal ein separater Zustandsübergang zum Deadlock in  $\overline{OG_P}$  angelegt.

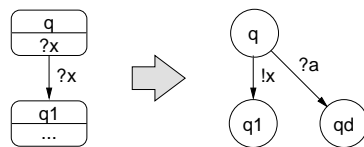


Abbildung 3.8: 1. Transformationsschritt vom dualen Serviceautomaten zum Public View: Links Ausschnitt aus einer Bedienungsanleitung  $OG_P$ , rechts resultierender Ausschnitt aus  $\overline{OG_P}$  nach dem Transformationsschritt.

Allerdings reicht dieser erste Transformationsschritt allein noch nicht aus, um jeden dualen Serviceautomaten in einen korrekten Public View zu überführen. Abbildung 3.9 (a) zeigt einen einfachen Serviceautomaten  $P$ , der zuerst entscheidet, ob eine Nachricht  $a$  gesendet wird oder nicht (interner  $\tau$ -Übergang) und anschließend eine Nachricht  $b$  erwartet. Die Bedienungsanleitung  $OG_P$  von  $P$  (siehe Abbildung 3.9 (b)) charakterisiert daher die möglichen Strategien für  $P$  wie folgt: Sende Nachricht  $b$  und sei anschließend in der Lage, Nachricht  $a$  zu empfangen. Alternativ dazu kann zuerst Nachricht  $a$  empfangen und anschließend Nachricht  $b$  gesendet werden. Es ist jedoch keine Strategie für  $P$ , im Anfangszustand allein auf Nachricht  $a$  zu warten. Dies wird in der Bedienungsanleitung von  $P$  dadurch ausgedrückt, dass es zwar einen Zustandsübergang für den Empfang von Nachricht  $a$  im Anfangszustand von  $OG_P$  gibt, die Annotation aber, die ja jede Strategie von  $P$  erfüllen muss, verlangt, dass ein Partner in diesem Zustand Nachricht  $b$  senden können muss. Nur so kann ein Deadlock in der Kommunikation mit  $P$  verhindert werden. Abbildung 3.9 (c) zeigt den zu  $OG_P$  dualen Serviceautomaten  $P'$  nach Anwendung des ersten Transformationsschritts und Abbildung 3.9 (d) dessen Bedienungsanleitung  $OG_{P'}$ . Obwohl die Bedienungsanleitungen von  $P$  und  $P'$  strukturgleich sind, charakterisieren sie doch unterschiedliche Strategien. Aufgrund der Annotation  $?a \vee !b$  ist es für den Service  $P'$ , ganz im Gegensatz zu  $P$ , eine Strategie, zuerst auf Nachricht  $a$  zu warten und anschließend Nachricht  $b$  zu senden.  $P'$  kann also keineswegs als Public View von  $P$  gelten.

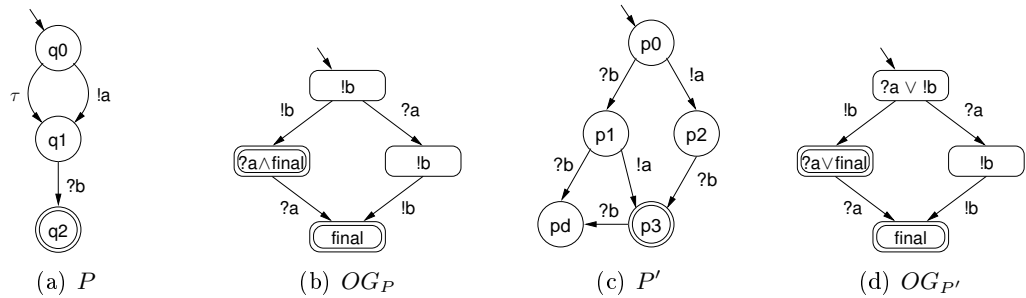


Abbildung 3.9: Serviceautomat  $P$  mit Bedienungsanleitung  $OG_P$  und dualer Serviceautomat  $P'$  nach dem 1. Transformationsschritt mit Bedienungsanleitung  $OG_{P'}$

Die Ursache für diese Verhaltensunterschiede von  $P$  und  $P'$  liegt darin, dass wir bis zu diesem Punkt die Public-View-Generierung allein auf Basis der Struktur der Bedienungsanleitung  $OG_P$  von  $P$  betrieben haben. Wollen wir eine Äquivalenz im Verhalten von  $P$  und  $P'$  herstellen, so müssen also auch die Annotationen von  $OG_P$  einen Einfluss auf die Generierung von  $P'$  haben (siehe Abschnitt 3.1.3). Diese spielen insbesondere dann eine Rolle, wenn über zusätzliche Zustandsübergänge mögliches Verhalten eines Partners von  $P$  beschrieben wird, das jedoch nicht in jedem Fall notwendig ist. Wolf formuliert daher einen zweiten Transformationsschritt, der auch die Annotationen von  $OG_P$  in die Public-View-Generierung mit einbezieht [Wol07].

**Definition 3.5 (2. Transformationsschritt)**

Wenn in einem Zustand  $q$  von  $OG_P$  eine Transition für eine Nachricht  $a \in I_{OG_P}$  existiert, die nicht in der Annotation  $\phi(q)$  vorkommt, dann füge in  $q$  von  $\overline{OG_P}$  eine  $\tau$ -Transition zu einem neuen Zustand  $q'$  ein und übernehme anschließend alle Transitions von  $q$ , bis auf die mit  $?a$  beschriftete, für  $q'$ .

Abbildung 3.10 verdeutlicht die Anwendung des zweiten Transformationsschrittes an einem kleinen Ausschnitt aus einer Bedienungsanleitung. Im Zustand  $q$  gibt es drei mögliche Zustandsübergänge. Unter anderem führt das Konsumieren einer Nachricht  $a$  in  $q$  zum Zustand  $q_3$ . Nachricht  $a$  tritt jedoch nicht in der Annotation von  $q$  auf, d.h. es ist möglich in  $q$  Nachricht  $a$  zu empfangen, es darf allerdings nicht unendlich lange auf Nachricht  $a$  gewartet werden. Anhand des dualen Kommunikationsverhaltens stellt sich die Situation für den Public View wie folgt dar: Im Zustand  $q$  gibt es einen Konflikt zwischen dem Senden von Nachricht  $a$  und dem Empfang einer der Nachrichten  $x$  und  $y$ . Da ein Partner nicht durch Abwarten erzwingen kann, dass dieser Konflikt früher oder später durch das Senden von Nachricht  $a$  aufgelöst wird, fügen wir gemäß des zweiten Transformationsschrittes einen  $\tau$ -Übergang von  $q$  zu  $q'$  ein. Im Zustand  $q'$  reduziert sich der Konflikt auf die beiden Nachrichten  $x$  und  $y$  und kann daher nur vom Partner durch das Senden einer Nachricht aufgelöst werden. Der resultierende Automat ist in Abbildung 3.10 rechts zu sehen. Ein Partner kann einen internen Zustandsübergang nicht beobachten und sich deshalb nicht sicher sein, ob sich der Service in Zustand  $q$  oder  $q'$  befindet. Dadurch ist Abwarten keine Strategie mehr und der Partner muss selbst aktiv werden, um einen Deadlock sicher vermeiden zu können. Damit wird das durch die Bedienungsanleitung charakterisierte Verhalten korrekt modelliert.

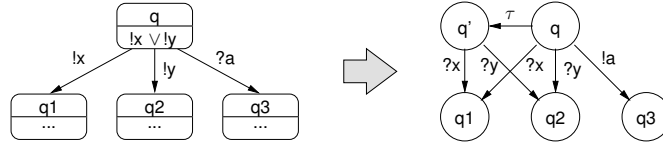


Abbildung 3.10: 2. Transformationsschritt vom dualen Serviceautomaten zum Public View: Links Ausschnitt aus einer Bedienungsanleitung  $OG_P$ , rechts resultierender Ausschnitt aus  $\overline{OG_P}$  nach dem Transformationsschritt.

In [Wol07] wird gezeigt, dass der zu einer Bedienungsanleitung  $OG_P$  duale Serviceautomat  $P'$  nach Anwendung der beiden oben vorgestellten Transformationsschritte die gleiche Bedienungsanleitung wie der ursprüngliche Prozess  $P$  besitzt, d.h. es gilt:  $OG_P = OG_{P'}$ .<sup>1</sup> Damit ist  $P'$  stets ein Public View von  $P$  und diese verhaltensbasierte Public-View-Generierung immer erfolgreich.

<sup>1</sup>In jedem Zustand von  $\overline{OG_P}$  kann jeder Transformationsschritt maximal einmal angewendet werden, da nach erfolgter Transformation die Bedingungen zur Anwendung der entsprechenden Regel in diesem Zustand nicht mehr erfüllt sind.

### 3.2.4 Stärken und Schwächen des Ansatzes

Da die Korrektheit des Verfahrens von Wolf bereits bewiesen worden ist, können wir davon ausgehen, dass mit der hier vorgestellten Methode zu jeder beliebigen Bedienungsanleitung (und damit jedem bedienbaren Service) ein Public View berechnet werden kann [Wol07]. Damit ist dieser Ansatz sowohl dem in Abschnitt 3.1 vorgestellten sprachensbasierten Verfahren als auch der von Martens in [Mar03] vorgeschlagenen verhaltensbasierten Public-View-Generierung überlegen.

Zu den Vorteilen dieses Verfahrens zählt, neben der allgemeinen Anwendbarkeit, die Tatsache, dass die Public-View-Generierung allein auf Basis der Bedienungsanleitung durchgeführt wird. Da die Bedienungsanleitung auf Grundlage einer Verhaltensanalyse konstruiert wird und daher von vertraulichen und betriebsinternen Informationen im Private View weitestgehend abstrahiert, besteht auch für einen aus ihr generierten Public View keine Gefahr des Verrats von Betriebsgeheimnissen, die sich nicht ohnehin aus dem Verhalten des Service ablesen lassen. Damit wird eine der wichtigsten Anforderungen an einen Public View erfüllt. Des Weiteren ist zur Public-View-Generierung nicht zwingend ein Private View erforderlich. Es kann somit für jeden Prozess  $P$ , wie auch immer er modelliert worden ist, ein Public View konstruiert werden, solange eine Bedienungsanleitung von  $P$  vorliegt.

Die Generierung eines minimalen Public View kann allerdings mit diesem Verfahren nur in seltenen Fällen gelingen. Mit einem Public View als Serviceautomaten können parallel ablaufende Aktivitäten, anders als in Petrinetzen, nicht elegant modelliert werden. Betrachten wir z.B. einen Service  $P$ , der nichts weiter tut, als vier die Nachrichten  $a$ ,  $b$ ,  $c$  und  $d$  zu verschicken. Mit vier hintereinander geschalteten Transitionen und fünf internen Plätzen lässt sich dieser Service leicht als oWFN  $N_P$  modellieren. Da wir von einem asynchronen Nachrichtenaustausch ausgehen, in dem sich Nachrichten auch gegenseitig überholen können, muss ein Partner von  $P$  die Nachrichten  $a$ ,  $b$ ,  $c$ ,  $d$  in jeder beliebigen Reihenfolge erwarten. Demzufolge besitzt die Bedienungsanleitung  $OG_P$  des Service  $P$  16 annotierte Zustände und 32 Zustandsübergänge. Der aus  $OG_P$  generierte Public View hat dann 17 Zustände und 64 Transitionen! Auch aus einer sehr geschickten Übersetzung des Public-View-Serviceautomaten in ein oWFN wird vermutlich ein deutlich größeres Netz resultieren als  $N_P$ . Inwieweit sich die Komplexität des Public View tatsächlich durch Übersetzung des Serviceautomaten in ein oWFN mit Hilfe der Regionentheorie [BD98] verringern lässt, ist zurzeit Gegenstand weiterer Forschung am Lehrstuhl. Allerdings zeigt dieses Beispiel, wie schnell ein mit diesem Verfahren generierter Public View eine enorme Größe erreichen kann. Zudem sinkt natürlich mit steigender Komplexität des Public View auch die Wahrscheinlichkeit, dass das Modell am Ende intuitiv verständlich und menschenlesbar ist.

## 4 Strukturbasierte Public-View-Generierung

In diesem Kapitel wollen wir einen strukturbasierten Ansatz zur Public-View-Generierung vorstellen. Der Ansatz stammt aus [Mar03] und wurde später im Zuge der Entwicklung von BPEL2oWFN [LMSW06] um einige Transformationsregeln erweitert. Die hier verwendeten Definitionen und Regeln (sowie deren Abbildungen) wurden aus [Mar03] entnommen und - soweit für die Arbeit erforderlich - angepasst.

In Abschnitt 4.1 werden wir die Idee des Verfahrens erläutern und anschließend in Abschnitt 4.2 einige Transformationsregeln aus [Mar03] vorstellen und ihre Anwendung demonstrieren. Am Ende des Kapitels wollen wir in Abschnitt 4.3 einen kritischen Blick auf das Verfahren werfen und die Grenzen der strukturbasierten Public-View-Generierung aufzeigen.

### 4.1 Idee des Verfahrens

Im Gegensatz zu den von uns in Kapitel 3 vorgestellten verhaltensbasierten Ansätzen, nehmen wir bei diesem strukturbasierten Verfahren den Service  $P$  und seine interne Kontrollstruktur als gegeben an. In dieser Arbeit gehen wir davon aus, dass uns der Private View von  $P$  als Petrinetz bzw. als oWFN vorliegt. Naturgemäß enthält  $P$  viele Details über unternehmensinterne Abläufe und Entscheidungen, die für den Public View  $P'$  nicht relevant bzw. zwingend zu verbergen sind. Um vertrauliche Informationen zu schützen und die Komplexität von  $P'$  nach Möglichkeit zu minimieren, zielt das Verfahren darauf ab, die Struktur von  $P$  weitestgehend auf die Aktivitäten zu reduzieren, die für die Kommunikation mit der Umgebung von Nöten sind.

Für die Reduktion von  $P$  können wir uns das Lokaliätsprinzip von Petrinetzen zu Nutze machen. Die Bedingungen für die Aktivierung einer Transition (Aktivität) des Netzes werden durch ihr direktes Umfeld definiert. Und auch das Schalten der Transition (Durchführung der Aktivität) hat nur Einfluss auf ihre direkte Umgebung. Wir können daher einzelne Teile des Netzes in Isolation betrachten und zum Zwecke der Reduktion manipulieren. Die Reduktion erfolgt anhand von Transformationsregeln. Eine Transformationsregel wird dabei im Grunde durch zwei Strukturmuster definiert. Das erste Muster dient, zusammen mit einigen verbal formulierten Bedingungen, als Beschreibung der Voraussetzung zur Anwendung der Regel. Kann dieses Muster, das i.d.R. mehrere Plätze und Transitionen umfasst, innerhalb der Struktur von  $P$  identifiziert werden und sind die

Bedingungen zur Anwendung der Regel erfüllt, so wird es durch das zweite Muster ersetzt. Das zweite Muster weist im Allgemeinen eine geringere Komplexität auf, wodurch die Anwendung der Transformationsregeln zu einer Reduktion der Komplexität von  $P$  führt. Es wird dabei zwischen zwei Sorten von Regeln unterschieden: Regeln zur *Eliminierung* von Netzelementen und Regeln zur *Fusion* von Netzelementen. In der Literatur findet sich ein reichhaltiger Schatz an Regeln zur Manipulation von Petrinetzen (z.B. in [CTSH02]), auf die in [Mar03] für die Reduktion zurückgriffen wird.

Natürlich führt eine Reduktion von  $P$  stets zu einer Änderung im Verhalten von  $P$ . Eine eliminierte Transition kann nicht mehr schalten und taucht folglich nicht mehr in den Schaltsequenzen von  $P$  auf. Allerdings fordern wir auch keine Äquivalenz im Verhalten von  $P$  und seinem Public View  $P'$ . Es reicht aus, wenn die Abstraktion auf  $P'$  das nach außen sichtbare, d.h. für die Kommunikation und Kooperation maßgebliche, Verhalten von  $P$  besitzt. Es muss daher für jedes Muster nur gezeigt werden, dass die entscheidenden Eigenschaften, wie Lebendigkeit, Verklemmungsfreiheit und Bedienbarkeit, durch die Anwendung bewahrt werden.

## 4.2 Transformationsregeln

In diesem Abschnitt werden wir einige Transformationsregeln im Detail vorstellen. Für jede Regel beschreiben wir Voraussetzung und Resultat der Anwendung verbal und formulieren einige Einschränkungen bzgl. der Verbindungen der entsprechenden Elemente zu den Schnittstellen des Netzes.

Für den Beweis, dass die hier präsentierten Regeln auch die von uns geforderten Eigenschaften erhalten, wird der interessierte Leser auf [Mar03] verwiesen. Auch wenn wir den Begriff der Bedienbarkeit eines Netzes, anders als [Mar03], nicht über Kommunikations- bzw. Bediengraphen sondern über die Bedienungsanleitung definieren, so ist die Beweisidee der folgenden Sätze doch stets die selbe und lässt sich leicht auch auf Basis der Bedienungsanleitung führen.

### 4.2.1 Regeln zur Eliminierung

Wir werden im Folgenden einige Regeln zur Eliminierung von Netzelementen vorstellen. Diese Regeln zielen auf die Entfernung von Plätzen und Transitionen ab, die auf das nach außen sichtbare Verhalten des Netzes keinen Einfluss haben. Die jeweiligen Strukturmuster der einzelnen Transformationsregeln verdeutlichen Abbildung 4.1 (a)-(d).

#### **Definition 4.1 (Regel ET1 - Eliminieren einer Transition)**

Voraussetzung: *Das Muster besteht aus einer Transition  $t$  mit genau einem Platz  $p1$  im Vorbereich und genau einem Platz  $p2$  im Nachbereich. Der Platz im Nachbereich darf beliebige Verbindungen zu weiteren Transitionen haben. Der Platz im Vorbereich darf nicht verzweigen, d.h., der Nachbereich von  $p1$  umfasst nur die Transition  $t$ .*

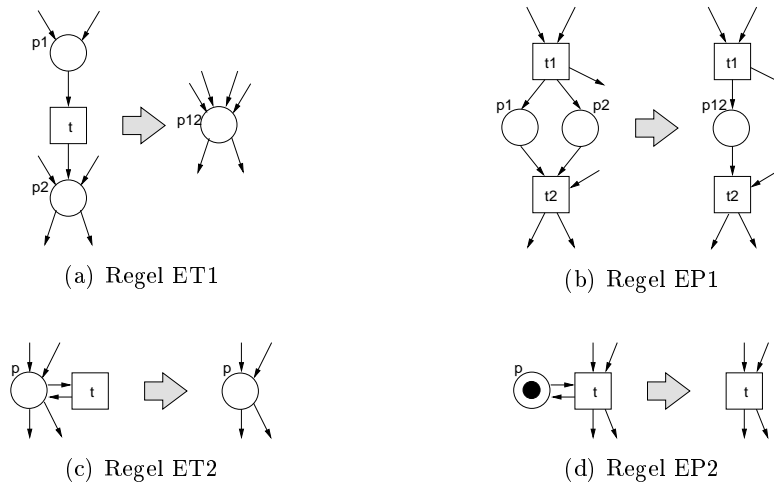


Abbildung 4.1: Regeln ET1, ET2, EP1 und EP2 zur Eliminierung von Netzelementen

Resultat: Die Transition  $t$  wird aus dem Netz entfernt und die Plätze  $p1$  und  $p2$  fusionieren zum Platz  $p12$ .

**Satz 4.1 ([Mar03])**

Wenn die Transition  $t$  nicht mit der Umgebung kommuniziert, dann haben das durch Anwenden der Regel ET1 entstehende  $oWFN N'$  und das  $oWFN N$  die gleiche Bedienungsanleitung.

**Definition 4.2 (Regel ET2 - Eliminieren einer Schleife)**

Voraussetzung: Das Muster besteht aus einer Transition  $t$ , deren Vorbereich und Nachbereich identisch sind und jeweils nur einen Platz umfassen. Der Platz darf beliebige Verbindungen zu weiteren Transitionen haben.

Resultat: Die Transition  $t$  wird aus dem Netz entfernt.

**Satz 4.2 ([Mar03])**

Wenn die Transition  $t$  nicht mit der Umgebung kommuniziert, dann haben das durch Anwenden der Regel ET2 entstehende  $oWFN N'$  und das  $oWFN N$  die gleiche Bedienungsanleitung.

**Definition 4.3 (Regel EP1 - Eliminieren eines Platzes)**

Voraussetzung: Das Muster besteht aus zwei Plätzen  $p1$  und  $p2$  mit dem gleichen Vorbereich  $t1$  und dem gleichen Nachbereich  $t2$ . Der Vor- und Nachbereich darf jeweils auch aus mehr als einer Transition bestehen.

Resultat: Einer der beiden Plätze wird entfernt. Um den Bezug zum Original zu erhalten, wird der verbleibende Platz in  $p12$  umbenannt.

**Satz 4.3 ([Mar03])**

Das durch Anwenden der Regel EP1 entstehende  $oWFN N'$  und das  $oWFN N$  haben die gleiche Bedienungsanleitung.

**Definition 4.4 (Regel EP2 - Eliminieren einer Ressource)**

Voraussetzung: *Das Muster besteht aus einem markierten Platz  $p$ , deren Vorbereich und Nachbereich identisch sind und jeweils nur eine Transition  $t$  umfassen. Diese Transition darf beliebige Verbindungen zu weiteren Plätzen haben.*

Resultat: *Der Platz  $p$  wird aus dem Netz entfernt.*

**Satz 4.4 ([Mar03])**

*Das durch Anwenden der Regel EP2 entstehende  $oWFN N'$  und das  $oWFN N$  haben die gleiche Bedienungsanleitung.*

Mit den in diesem Abschnitt vorgestellten Transformationsregeln lassen sich die meisten Netze jedoch nur sehr begrenzt strukturell vereinfachen, da sie ausschließlich auf Teile des Netzes angewendet werden können, die für das nach außen sichtbare Verhalten irrelevant sind. Im folgenden Abschnitt werden wir daher Transformationsregeln vorstellen, die auch kommunizierende Transitionen in die strukturelle Reduktion mit einbeziehen.

**4.2.2 Regeln zur Fusion**

Das kommunizierende Transitionen nicht einfach eliminiert werden können, ohne das nach außen sichtbare Verhalten eines Netzes maßgeblich zu beeinflussen, ist offensichtlich. Allerdings bietet die Fusion von kommunizierenden und internen, nicht-kommunizierenden Transitionen sowie die Fusion von Transitionen des gleichen Kommunikationstyps (Senden bzw. Empfangen) erhebliches Reduktionspotential. Die folgenden Transformationsregeln sollen uns helfen, dieses Potential für die strukturbasierte Public-View-Generierung zu nutzen. Die entsprechenden Strukturmuster der einzelnen Regeln verdeutlichen Abbildung 4.2, 4.3 und 4.4.

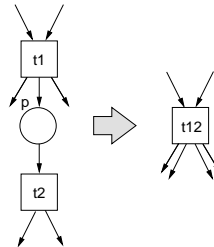


Abbildung 4.2: Regel FSeq zur Fusion sequenzieller Transitionen

**Definition 4.5 (Regel FSeq - Fusion sequenzieller Transitionen)**

Voraussetzung: *Das Muster besteht aus zwei Transitionen  $t1$  und  $t2$  und einem Platz  $p$ , deren Vorbereich nur aus Transition  $t1$  besteht und deren Nachbereich nur aus der Transition  $t2$  besteht. Die Transition  $t1$  darf beliebige Verbindungen zu weiteren Plätzen haben. Die Transition  $t2$  hingegen darf keine weiteren eingehenden Kanten haben.*

Resultat: Der Platz  $p$  wird aus dem Netz entfernt und die Transitionen  $t1$  und  $t2$  fusionieren zur Transition  $t12$ .

**Satz 4.5 ([Mar03])**

Wenn die Transition  $t2$  nicht mit der Umgebung kommuniziert oder die Transition  $t2$  Nachrichten an die Umgebung sendet, die Transition  $t1$  jedoch keine Nachrichten der Umgebung empfängt, dann hat das durch Anwendung der Regel FSeq entstehende oWFN  $N'$  die gleiche Bedienungsanleitung wie das oWFN  $N$ .

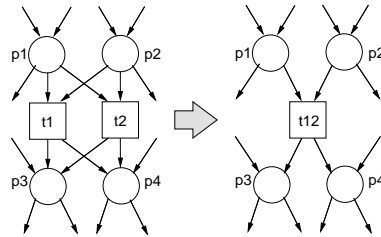


Abbildung 4.3: Regel FAlt zur Fusion alternativer Transitionen

**Definition 4.6 (Regel FAlt - Fusion alternativer Transitionen)**

Voraussetzung: Das Muster besteht aus zwei Transitionen  $t1$  und  $t2$ , die einen identischen Vorbereich (die Plätze  $p1$  und  $p2$ ) und einen identischen Nachbereich (die Plätze  $p3$  und  $p4$ ) haben. Vor- und Nachbereich dürfen jeweils aus beliebig vielen Plätzen bestehen, wobei diese Plätze wiederum beliebige Verbindungen zu weiteren Transitionen haben können.

Resultat: Die Transitionen  $t1$  und  $t2$  fusionieren zur Transition  $t12$ .

**Satz 4.6 ([Mar03])**

Wenn die Transitionen  $t1$  und  $t2$  vom selben Typ sind, d.h. entweder beide Transitionen eine Nachricht aus der Umgebung empfangen bzw. eine Nachricht an die Umgebung senden oder beide nicht mit der Umgebung kommunizieren, dann ist das aus Anwendung der Regel FAlt resultierende oWFN  $N'$  bedienungsäquivalent zum oWFN  $N$ .

**Definition 4.7 (Regel FPar - Fusion paralleler Transitionen)**

Voraussetzung: Das Muster besteht aus vier Transitionen  $t1$  bis  $t4$ . Die Transition  $t1$  hat die Plätze  $p1$  und  $p2$  im Nachbereich und ist ansonsten beliebig mit anderen Plätzen verbunden. Die Transition  $t4$  hat die Plätze  $p3$  und  $p4$  im Vorbereich und ansonsten ebenfalls beliebige Verbindungen. Die Transitionen  $t2$  und  $t3$  haben genau einen Platz im Vorbereich ( $p1$  bzw.  $p2$ ) und genau einen Platz im Nachbereich ( $p3$  bzw.  $p4$ ). Die Plätze  $p1$  bis  $p4$  haben jeweils genau eine Transition im Vorbereich und genau eine Transition im Nachbereich.

Resultat: Paarweise fusionieren die Plätze  $p1$  und  $p2$  zum Platz  $p12$ , die Plätze  $p3$  und  $p4$  zum Platz  $p34$  und die Transitionen  $t2$  und  $t3$  zur Transition  $t23$ .

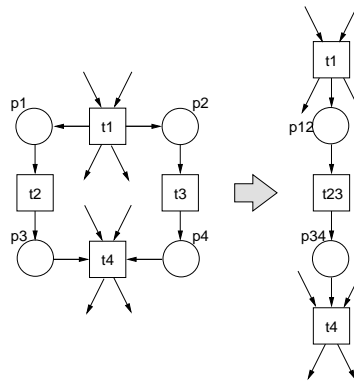


Abbildung 4.4: Regel FPar zur Fusion paralleler Transitionen

**Satz 4.7 ([Mar03])**

Wenn eine der beiden Transitionen  $t2$  und  $t3$  nicht mit der Umgebung kommuniziert oder beide Transitionen vom selben Typ sind, dann haben das durch Anwendung der Regel FPar entstehende  $oWFN N'$  und das  $oWFN N$  die gleiche Bedienungsanleitung.

Durch sukzessives Anwenden der Fusions- und Eliminierungsregeln reduziert sich die Struktur des Netzes und macht die Anwendung weiterer Transformationsregeln möglich. Wie das im Einzelnen aussehen kann, wollen wir in dem nun folgenden Abschnitt an einem Beispiel veranschaulichen.

**4.2.3 Transformation am Beispiel**

Um die Anwendung der im vorigen Abschnitt definierten Regeln zu demonstrieren, wollen wir für den in Kapitel 2 vorgestellten Kinokarten-Service eine strukturbasierte Public-View-Generierung durchführen. Der Private View aus Abbildung 2.11 enthält zehn Transitionen von denen nur drei mit der Umgebung kommunizieren. Die übrigen sieben Transitionen repräsentieren interne Aktivitäten, die für das Verständnis des Prozesses nicht zwingend erforderlich sind und daher nicht im Public View publiziert werden sollten.

Wir führen die Transformation wie folgt durch:

*Schritt 1* Eliminierung der Transitionen *ermittle Vorstellung*, *reserviere Plätze*, *generiere Bestätigung*, *generiere Ablehnung* und *protokolliere Vorgang* nach Regel *ET1*.

*Schritt 2* Fusionieren der Transition *Reservierung erfolgreich* mit der Transition *sende Bestätigung* nach Regel *FSeq*.

*Schritt 3* Fusionieren der Transition *Reservierung schlug fehl* mit der Transition *sende Ablehnung* nach Regel *FSeq*.

Der Ergebnis der Reduktion ist der Public View des Kinokarten-Service, wie in Abbildung 2.12 dargestellt. Die Bezeichnung der Transitionen, die durch eine Fusion entstanden sind, wurde dabei nach dem folgenden Schema gewählt:

- Wird eine kommunizierende Transition (z.B. *sende Bestaetigung*) mit einer internen Transition (z.B. *Reservierung erfolgreich*) fusioniert, so wird die Bezeichnung der kommunizierenden Transition übernommen.
- Anderenfalls werden beide Bezeichnungen der Transitionen übernommen und durch ein Trennzeichen verbunden (für das Beispiel irrelevant).

Vor der Veröffentlichung des Public View sollten gegebenenfalls durch einen Modellierer einige manuelle Anpassungen der Bezeichnungen vorgenommen werden, um die Lesbarkeit und Verständlichkeit des Modells zu optimieren.

In diesem Fall kann mit dem strukturbasierten Verfahren ein minimaler Public View generiert werden, der nur noch kommunizierende Transitionen enthält. Das Ergebnis der Transformation ist ein Netz mit drei Transitionen. Der Private View konnte also um 70% reduziert werden. Warum ein derart gutes Resultat nicht immer zu erreichen ist, soll der folgende Abschnitt klären.

### 4.3 Grenzen der strukturbasierten Transformation

In diesem Abschnitt wollen wir auf einige Probleme der strukturbasierten Transformation eingehen und die Grenzen der Anwendbarkeit des Verfahrens aufzeigen.

Das Ergebnis der strukturbasierten Transformation ist vor allem vom verwendeten Regelwerk abhängig. Trotz einer Vielzahl von Regeln lassen sich immer wieder Strukturen erkennen, die unnötig komplex sind, sich jedoch vom verwendeten Regelwerk nicht weiter reduzieren lassen. In diesem Bereich sind weitere Forschungen nötig, um das Regelwerk so weit wie möglich auszubauen. Nur so kann man sich dem Ziel eines minimalen Public View per strukturbasierter Transformation nähern. Bisher haben wir auch keine Reihenfolge zur Anwendung der einzelnen Regeln definiert. Möglicherweise ist die Anwendung der Regeln eines komplexen Regelwerks nicht kommutativ. Demzufolge liefert das Verfahren bei unterschiedlicher Anwendungsreihenfolge der Regeln verschiedene Ergebnisse, die sich auch in der Komplexität unterscheiden können. Um dann stets einen Public View minimaler Komplexität zu erhalten, müsste der Raum der durch Reduktion erreichbaren Netze vollständig durchsucht oder eine Heuristik zur Anwendung der Regeln entwickelt werden.

Generell gestaltet sich die Reduktion von Prozessen mit komplexem Kommunikationsverhalten recht schwierig. Besitzt ein Netz relativ viele kommunizierende Transitionen, lassen sich oft nur wenige der Regeln anwenden. Zudem kann eine ungeschickte Modellierung des Private View ein befriedigendes Resultat der strukturbasierten Public-View-Generierung,

unabhängig vom verwendeten Regelwerk, verhindern. Abbildung 4.5 (a) zeigt ein Netz  $N$ , das mit Hilfe der strukturbasierten Transformation nicht reduziert werden kann. Obwohl in jedem Fall die Nachricht  $c$  gesendet wird, wurde dies durch zwei separate Transitionen modelliert. Abbildung 4.5 (b) zeigt den minimalen Public View  $N'$  von  $N$ , der z.B. mit Hilfe der verhaltensbasierten Public-View-Generierung konstruiert werden kann.

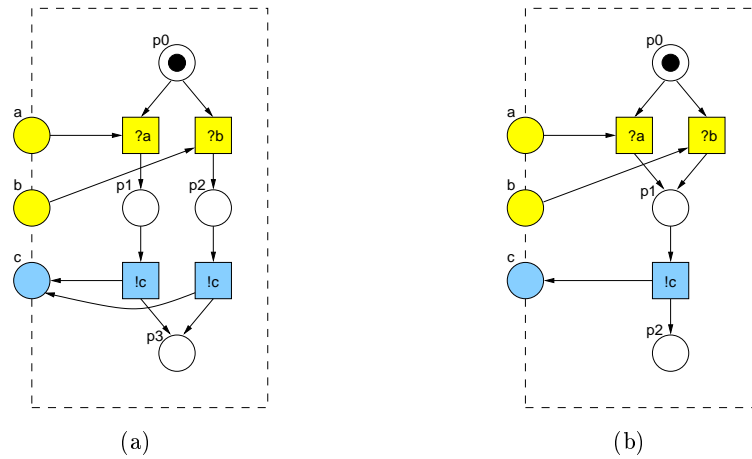


Abbildung 4.5: Private View (a) und minimaler Public View (b)

Schlussfolgernd lässt sich feststellen, dass es sowohl Netze gibt, für die mit einer strukturbasierten Public-View-Generierung ein deutlich besseres Ergebnis erzielt werden kann als mit einem verhaltensbasierten Ansatz, als auch solche, bei denen das Umgekehrte der Fall ist. Die Fragen danach, wann welches Verfahren die größeren Erfolgchancen bietet und woran sich dies im Voraus erkennen lässt, werden uns im nächsten Kapitel beschäftigen.

# 5 Vergleich der Verfahren

Nachdem wir in den Kapiteln 3 und 4 sehr unterschiedliche Ansätze zur Public-View-Generierung vorgestellt haben, wollen wir uns in diesem Kapitel mit der konkreten Anwendung der Verfahren an Beispielen aus der Praxis beschäftigen.

Obwohl wir bereits jeweils die individuellen Stärken und Schwächen der Verfahren diskutiert haben, sind wir bisher noch nicht in der Lage, klare Empfehlungen zu deren Einsatz zu formulieren. Wir wollen daher nun die beiden Ansätze von Wolf (Abschnitt 3.2) und Martens (Kapitel 4) anhand zweier Implementierungen, die wir in Abschnitt 5.1 kurz vorstellen werden, in einer Reihe von Fallstudien direkt vergleichen. Dazu werden wir in Abschnitt 5.2 für jede der Fallstudien, die aus unterschiedlichen Anwendungsbereichen stammen, sowohl eine verhaltensbasierte als auch eine strukturbasierte Public-View-Generierung vornehmen und die Resultate gegenüberstellen. Anschließend geben wir in Abschnitt 5.3 eine Zusammenfassung der Ergebnisse und leiten Empfehlungen für den praktischen Einsatz der Verfahren ab.

## 5.1 Implementierungen

### 5.1.1 Implementation des verhaltensbasierten Verfahrens

Um die verhaltensbasierte Public-View-Generierung von Wolf (siehe Abschnitt 3.2) zu implementieren, haben wir das am Lehrstuhl entwickelte Werkzeug FIONA [LMSW06] erweitert. FIONA kann u.a. zur Analyse von Services auf Basis von oWFN eingesetzt werden und ist in der Lage oWFNs einzulesen und den Interaktionsgraphen [Wei04] sowie die Bedienungsanleitung für das gegebene Netz zu berechnen. Außerdem können mit FIONA Aussagen über die Austauschbarkeit von Services getroffen werden. Mittels der am Lehrstuhl entwickelten Petrinetz-Semantik für WS-BPEL [Loh07] und dem Werkzeug BPEL2oWFN [LMSW06] können mit FIONA auch in WS-BPEL spezifizierte Prozesse analysiert werden.

FIONA arbeitet auf Basis der Kommandozeile, d.h. zur Umsetzung des in Abschnitt 3.2 vorgestellten Verfahrens muss eine zuvor berechnete Bedienungsanleitung aus einer Datei oder direkt von der Eingabe eingelesen und intern repräsentiert werden. Anhand der internen Repräsentation wird der duale Serviceautomat erzeugt und die Transformationsregeln angewandt. Der so generierte Public-View-Serviceautomat wird anschließend wieder als Datei ausgegeben.

Die in C++ entwickelten Programm-Bibliotheken von FIONA enthalten bereits alle Datenstrukturen, die wir zur Repräsentation von Bedienungsanleitungen und dualen Serviceautomaten benötigen. Wir bedienen uns dabei der folgenden Klassen: *OGFromFile* (zur Repräsentation der aus einer Datei eingelesenen Bedienungsanleitung), *OGFromFileNode* (zur Repräsentation eines Knotens/Zustands der Bedienungsanleitung) und *OGFromFileTransition* (zur Repräsentation eines Zustandsübergangs innerhalb der Bedienungsanleitung). Abbildung 5.1 zeigt einen Auszug aus dem Klassendiagramm der Implementation. Um die Anschaulichkeit des Diagramms zu erhöhen, beschränken wir uns auf die Darstellung der für die Public-View-Generierung benötigten Funktionalität. Insbesondere die Klasse *OGFromFile* besitzt eine Vielzahl weiterer Operationen, z.B. zur Berechnung von Produktbedienungsanleitungen zur Charakterisierung austauschbarer Services [Bre07], auf die wir hier nicht näher eingehen werden.

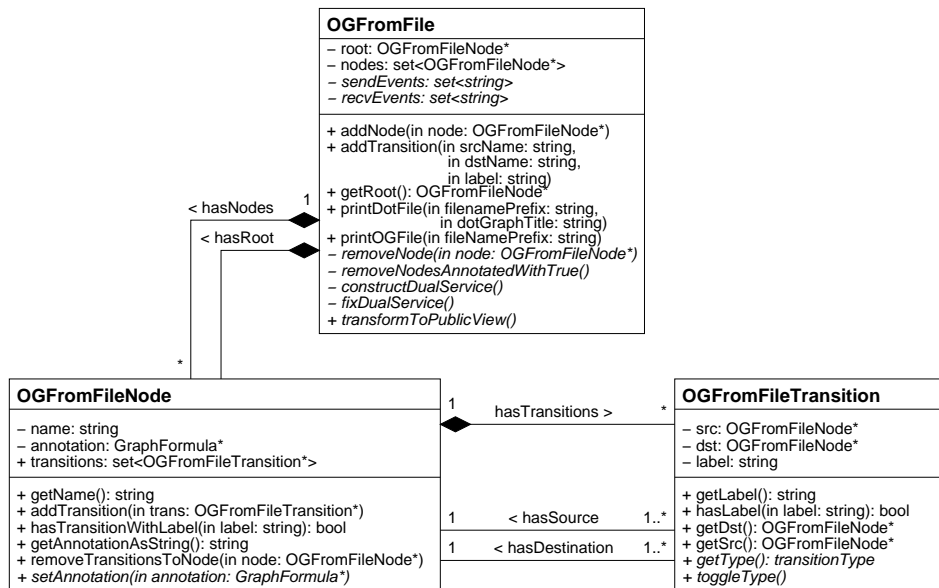


Abbildung 5.1: Auszug aus dem Klassendiagramm zur Implementation der verhaltensbasierten Public-View-Generierung

Eine Instanz der Klasse *OGFromFile* beschreibt eine Bedienungsanleitung über eine Menge von Zuständen (*nodes*) mit einem ausgezeichneten Anfangszustand (*root*). Ein Zustand der Bedienungsanleitung wird als Instanz der Klasse *OGFromFileNode* durch einen Bezeichner (*name*), eine Annotation (*annotation*) und eine Menge von ausgehenden Transitionen (*transitions*) charakterisiert. Transitionen wiederum werden als Instanzen der Klasse *OGFromFileTransition* verwaltet, die für jede Transition einen Ausgangszustand (*src*), einen Zielzustand (*dst*) und eine Beschriftung (*label*) erfasst. Bei der modellierten Datenstruktur handelt es sich also im Grunde um einen beschrifteten Graphen.

Die Operationen der Klassen bieten einen kontrollierten Zugriff auf die Daten der Objekte und ermöglichen die Konstruktion von und Navigation durch Bedienungsanleitungen.

## Algorithmus 5.1: Generierung des Public View aus einer Bedienungsanleitung

---

```

procedure transformToPublicView( $OG_P$ )
   $OG_P \leftarrow$  removeNodesAnnotatedWithTrue( $OG_P$ )
   $\overline{OG_P}$ , RecvEvents, SendEvents  $\leftarrow$  constructDualService( $OG_P$ )
   $P' \leftarrow$  fixDualService( $\overline{OG_P}$ , RecvEvents, SendEvents)
  return  $P'$ 
endprocedure

```

---

Die in Abbildung 5.1 kursiv gesetzten Operationen wurden vom Autor als Erweiterung zur vorhandenen Funktionalität der Klassen implementiert, um die verhaltensbasierte Public-View-Generierung nach Abschnitt 3.2 umsetzen zu können. Die ursprüngliche Funktionalität des Werkzeugs FIONA bleibt davon unberührt, da keine der vorhandenen Operationen modifiziert wurden.

Im Folgenden wollen wir etwas näher auf die Implementation des Ansatzes eingehen, indem wir die von uns entwickelten Algorithmen zur Umsetzung der einzelnen Transformationsschritte von der Bedienungsanleitung zum Public View vorstellen.

Die verhaltensbasierte Public-View-Generierung soll sich auf Basis der Klassen *OGFromFile*, *OGFromFileNode* und *OGFromFileTransition* als Teilfunktion von FIONA nutzen lassen. Dazu wird eine zuvor erzeugte Bedienungsanleitung  $OG_P$  eingelesen, mittels der vom Autor implementierten Operation *transformToPublicView* der Klasse *OGFromFile* in einen Public-View-Serviceautomaten  $P'$  transformiert und über die vorhandenen Operationen *printOGFile* bzw. *printDotFile* ausgegeben.<sup>1</sup>

Algorithmus 5.1 beschreibt das Vorgehen zur Transformation einer Bedienungsanleitung in einen Public-View-Serviceautomaten, wie es als Operation der Klasse *OGFromFile* implementiert wurde. Die Eingabe des Algorithmus ist eine Bedienungsanleitung  $OG_P$ . Die Ausgabe ist der generierte Public View  $P'$ . Der Algorithmus verwendet die Hilfsfunktion *removeNodesAnnotatedWithTrue*( $OG_P$ ), die alle mit *true* annotierten Zustände vor Beginn der Transformation aus der Bedienungsanleitung  $OG_P$  entfernt. Dies ist zulässig, da solche Zustände mögliches beliebiges zusätzliches Verhalten der Partner beschreiben, das auf die Bedienung des Prozesses keinen Einfluss haben darf und dessen Ausführung im Public View ausgeschlossen werden sollte. Die entsprechende Operation *removeNodesAnnotatedWithTrue*, genauso wie die Operationen *constructDualService* und *fixDualService* als Umsetzung der Algorithmen 5.2 und 5.3, wurden vom Autor als private Operationen der Klasse *OGFromFile* implementiert.

Algorithmus 5.2 verdeutlicht das Vorgehen zur Konstruktion des dualen Serviceautomaten aus einer Bedienungsanleitung. Die Eingabe des Algorithmus ist eine Bedienungsan-

---

<sup>1</sup>Eine saubere Abgrenzung von Bedienungsanleitung und Public-View-Serviceautomat, z.B. über eine Klasse zur Repräsentation eines Serviceautomaten und eine davon abgeleitete Klasse zur Repräsentation eines annotierten Serviceautomaten (Bedienungsanleitung), wäre sicherlich wünschenswert, sollte aber nicht Aufgabe unserer prototypischen Implementation des Verfahrens sein.

Algorithmus 5.2: Konstruktion des dualen Serviceautomaten

---

```
procedure constructDualService( $P$ )
   $\bar{P} \leftarrow P$ 
  SendEvents  $\leftarrow \emptyset$ 
  RecvEvents  $\leftarrow \emptyset$ 
  /* iterate over all nodes of the Service Automaton / OG */
  do forall node : node  $\in$  nodes( $\bar{P}$ )
    /* iterate over all transitions of the current node */
    do forall transition : transition  $\in$  transitions(node)
      /* mirror communication behavior */
      choose (type(transition))
        sending:
          type(transition)  $\leftarrow$  receiving
          RecvEvents  $\leftarrow$  RecvEvents  $\cup$  {label(transition)}
        receiving:
          type(transition)  $\leftarrow$  sending
          SendEvents  $\leftarrow$  SendEvents  $\cup$  {label(transition)}
        internal:
          /* do nothing */
      endchoose
    enddo
  enddo
  return  $\bar{P}$ , RecvEvents, SendEvents
endprocedure
```

---

leitung. Die Ausgabe ist der duale Serviceautomat sowie die Mengen seiner Empfangs- und Sendeereignisse. Für die Konstruktion des dualen Serviceautomaten werden sukzessive alle Zustände der Bedienungsanleitung betrachtet und für jede ausgehende Transition deren Kommunikationstyp gespiegelt, d.h. jedes Sende- bzw. Empfangsereignis der Bedienungsanleitung wird zum Empfangs- bzw. Sendeereignis des dualen Serviceautomaten. In Vorbereitung auf die nächsten Transformationsschritte erfassen wir dabei zugleich die Sende- und Empfangsereignisse des dualen Serviceautomaten in den Mengen *SendEvents* und *RecvEvents*. Der Algorithmus verwendet die folgenden Hilfsfunktionen, die wir hier nicht im Detail erörtern werden: *nodes(P)* gibt die Menge der Zustände des Serviceautomaten *P* zurück, *transitions(node)* liefert die Menge der vom Zustand *node* ausgehenden Transitionen, *type(transition)* bezeichnet den Kommunikationstyp der Transition *transition*, wobei für Empfangstransitionen *receiving*, für Sendetransitionen *sending* und für interne Transitionen *internal* verwendet wird und *label(transition)* liefert die Beschriftung des Zustandsübergangs *transition*. Die zur Umsetzung des Algorithmus benötigte Funktionalität zum Auslesen und Spiegeln des Kommunikationstyps einer Transition wurden vom Autor als Operationen *getType* und *toggleType* in der Klasse *OGFromFileTransition* implementiert.

Die zwei von Wolf formulierten Transformationsschritte wurden anhand von Algorithmus 5.3 umgesetzt [Wol07]. Die Eingabe des Algorithmus ist der duale Serviceautomat sowie die Mengen seiner Empfangs- und Sendeereignisse. Die Ausgabe ist der generierte Public View. Um die Effizienz der Implementation zu erhöhen, betrachten wir jeden Zustand des dualen Serviceautomaten nur einmal und wenden, falls nötig, beide Transformationsschritte direkt hintereinander an. Dies ist zulässig, da die beiden Transformationsschritte in ihrer Anwendung unabhängig voneinander sind. Das heißt, die Anwendung des ersten Transformationsschrittes hängt in einem Zustand allein von der Menge der ausgehenden Transitionen ab, die ein Empfangsereignis repräsentieren, wohingegen die Anwendung des zweiten Transformationsschrittes allein von der Menge der ausgehenden Transitionen abhängt, die ein Sendeereignis repräsentieren. Außerdem hat die Anwendung einer der beiden Transformationsschritte in einem Zustand des Serviceautomaten keinen Einfluss auf die Erfüllbarkeit der Bedingungen zur Anwendung des anderen Transformationsschrittes in diesem Zustand.

Für die Anwendung des ersten Transformationsschrittes (siehe Definition 3.4) müssen zuerst alle im aktuell betrachteten Zustand des dualen Serviceautomaten (*node*) verbotenen Empfangsereignisse ermittelt werden. Diese erfassen wir in der Menge *ForbiddenRecvEvents*. Ausgehend von der Menge aller Empfangsereignisse (*RecvEvents*) reduzieren wir die Menge *ForbiddenRecvEvents* um die Ereignisse, für die es bereits einen Zustandsübergang im aktuellen Zustand des dualen Serviceautomaten gibt. Für die verbleibenden Ereignisse legen wir im Public View einen neuen Zustandsübergang zu einem expliziten Deadlock-Zustand (*deadlock*) an.

Für die Umsetzung des zweiten Transformationsschrittes (siehe Definition 3.5) gehen wir wie folgt vor: Für jede ausgehende Transition eines Zustands, die ein Sendeereignis repräsentiert, prüfen wir, ob dieses in der Annotation des Zustands in der Bedienungsanleitung vorkommt. Ist dies für mindestens ein Sendeereignis nicht der Fall, so müssen

Algorithmus 5.3: Transformation des dualen Serviceautomaten zum Public View

---

```
procedure fixDualService( $\overline{P}$ , RecvEvents, SendEvents)
   $P' \leftarrow \overline{P}$ 
  /* create explicit deadlock node */
  addNode( $P'$ , deadlock)
  /* iterate over all nodes of the Service Automaton */
  do forall node : node  $\in$  nodes( $\overline{P}$ )
    ForbiddenRecvEvents  $\leftarrow$  RecvEvents
    apply2ndFix  $\leftarrow$  false
    NewNodesTransitions  $\leftarrow$  transitions(node)
    /* iterate over all transitions of the current node */
    do forall transition : transition  $\in$  transitions(node)
      choose (type(transition))
        receiving:
          /* prepare 1st fix */
          ForbiddenRecvEvents  $\leftarrow$  ForbiddenRecvEvents  $\setminus$  {label(transition)}
        sending:
          if ( $\neg$ (label(transition) occurs in annotation( $OG_P$ , node)))
            /* we have to apply the 2nd fix at this node */
            apply2ndFix  $\leftarrow$  true
            NewNodesTransitions  $\leftarrow$  NewNodesTransitions  $\setminus$  {transition}
          endif
        internal:
          /* do nothing */
      endchoose
    enddo
    /* apply 1st fix (if necessary) */
    do forall event : event  $\in$  ForbiddenRecvEvents
      addTransition( $P'$ , node, deadlock, event)
    enddo
    /* apply 2nd fix (if necessary) */
    if (apply2ndFix)
      addNode( $P'$ , newNode)
      addTransition( $P'$ , node, newNode,  $\tau$ )
      do forall transition : transition  $\in$  NewNodesTransitions
        addTransition( $P'$ , newNode, dest(transition), label(transition))
      enddo
      /* add forbidden receive events from 1st fix to new node */
      do forall event : event  $\in$  ForbiddenRecvEvents
        addTransition( $P'$ , newNode, deadlock, event)
      enddo
    endif
    if (node is annotated with final  $\wedge$  node has no outgoing sending transitions)
      markFinal( $P'$ , node)
    endif
  enddo
  return  $P'$ 
endprocedure
```

---

wir in diesem Zustand den zweiten Transformationsschritt anwenden, um im Public View einen internen Übergang anzulegen, der das nicht geforderte Ereignis „überspringt“. In der Menge *NewNodesTransitions* vermerken wir dafür vorsorglich alle vom aktuellen Zustand ausgehenden Transitionen und entfernen während der Untersuchung all die Transitionen, die ein Sendeereignis repräsentieren, das nicht in der Annotation vorkommt. Sollten die Bedingungen zur Durchführung des zweiten Transformationsschrittes erfüllt sein, legen wir dann einen neuen Zustand (*newNode*) im Public View an und fügen einen  $\tau$ -Übergang vom aktuellen Zustand zum neuen Zustand ein. Anschließend versehen wir den neuen Zustand mit den in der Menge *NewNodesTransitions* verbliebenen Zustandsübergängen. Damit ist die Anwendung des zweiten Transformationsschrittes abgeschlossen. Allerdings ist dieser neu angelegte Zustand im Public View nicht Teil der weiteren Untersuchung, da diese sich nur über die Zustände des ursprünglichen dualen Serviceautomaten erstreckt. Es muss also geprüft werden, in wieweit sich die Transformationsschritte auch auf den neuen Zustand anwenden lassen. Der zweite Transformationsschritt lässt sich auf den neu angelegten Zustand nicht anwenden, da dieser keine Entsprechung in der Bedienungsanleitung und damit keine Annotation besitzt. Der erste Transformationsschritt muss jedoch auch im neuen Zustand angewendet werden, da wir die Bedingungen zur Anwendung der einzelnen Transformationsschritte in einem Zustand quasi parallel untersucht haben und die Menge *NewNodesTransitions* daher noch nicht die durch den ersten Transformationsschritt hinzugefügten Transitionen beinhaltet. Da es sich bei dem neu angelegten Zustand nahezu um eine Kopie des untersuchten Zustands handelt, können wir den ersten Transformationsschritt durchführen, in dem wir für den neuen Zustand ebenfalls für die in der Menge *ForbiddenRecvEvents* verbliebenen Ereignisse Zustandsübergänge zum expliziten Deadlock-Zustand im Public View einfügen.

Nach Anwendung der Transformationsschritte muss zudem geprüft werden, ob der aktuelle Zustand ein Endzustand des generierten Public-View-Serviceautomaten ist. Dies ist der Fall, wenn der Zustand mit *final* annotiert ist und keine Sendeereignisse im Zustand möglich sind.<sup>2</sup>

Zusätzlich zu den für Algorithmus 5.2 vorgestellten Hilfsfunktionen kommen hier einige weitere Funktionen vor, die wir nur kurz erwähnen wollen: *addNode(P, node)* fügt dem Serviceautomaten *P* den Zustand *node* hinzu, *annotation(OG, node)* gibt die Annotation des Zustands *node* der Bedienungsanleitung *OG* zurück, *addTransition(P, 1stNode, 2ndNode, label)* fügt im Serviceautomaten *P* einen neuen Zustandsübergang vom Zustand *1stNode* zum Zustand *2ndNode* mit der Beschriftung *label* ein, *dest(transition)* liefert den Zielzustand des Zustandsübergangs *transition* und *markFinal(P, node)* fügt den Zustand *node* in die Menge der Endzustände des Serviceautomaten *P* ein. Die für die Implementation des Algorithmus benötigten Hilfsfunktionen waren bereits als Operationen der Klassen *OGFromFile*, *OGFromFileNode* und *OGFromFileTransition* vorhanden.

In Abschnitt 3.2 haben wir bereits erwähnt, dass Serviceautomaten mit Hilfe der Re-

<sup>2</sup>Ein im Zuge der Anwendung des zweiten Transformationsschrittes im Public View neu angelegter Zustand kann kein Endzustand des Public-View-Serviceautomaten sein. Ist der zweite Transformationsschritt in einem Zustand *q* des dualen Serviceautomaten anwendbar, dann ist in *q* mindestens ein Sendeereignis möglich. Somit kann *q* kein Endzustand des dualen Serviceautomaten sein.

gionentheorie in oWFN übersetzt werden können [BD98]. Für die Anwendung der Regionentheorie kann es von Vorteil sein, wenn es anstatt des einen expliziten Deadlocks, wie er von Algorithmus 5.3 angelegt wird, für jedes Empfangsereignis einen separaten Deadlock gibt. Das heißt, dass es für jeden Zustand, in dem das Empfangen einer Nachricht  $a$  verboten ist, einen Zustandsübergang zu einem Zustand  $deadlock_a$  und für jeden Zustand, in dem das Empfangen einer Nachricht  $b$  verboten ist, einen Zustandsübergang zu einem Zustand  $deadlock_b$  usw. geben würde. Die Anzahl der Deadlocks entspricht dann der Kardinalität der Menge *RecvEvents*. Wir ermöglichen ein solches Vorgehen, in dem wir eine leicht modifizierte Version des Algorithmus 5.3 implementiert und für das Werkzeug FIONA eine neue Option `-M` bzw. `--multipledeadlocks` definiert haben, mit der sich das Anlegen mehrerer Deadlocks im Zuge der Public-View-Generierung erzwingen lässt.

### 5.1.2 Implementation des strukturbasierten Verfahrens

Für die Implementation der strukturbasierten Public-View-Generierung greifen wir auf die Petrinetz-Bibliotheken von BPEL2oWFN zurück, einem ebenfalls am Lehrstuhl entwickelten Werkzeug, das die Übersetzung von in WS-BPEL spezifizierten Prozessen in oWFN ermöglicht [LMSW06].<sup>3</sup>

Die strukturbasierte Public-View-Generierung mit Hilfe der Petrinetz-Bibliotheken läuft, wie die verhaltensbasierte Public-View-Generierung mit FIONA, auf Basis der Kommandozeile ab. Die den Fallstudien zugrunde liegenden bzw. aus WS-BPEL Prozessen übersetzten oWFN müssen aus einer Datei eingelesen, intern repräsentiert und strukturell reduziert werden. Anschließend wird das reduzierte oWFN wieder als Datei ausgegeben. Da uns für den Vergleich der Resultate an sich nur die Komplexität des generierten Public View interessiert, haben wir auf die in Kapitel 4 angemahnte manuelle Anpassung zur Optimierung der Anschaulichkeit des Public View in den meisten Fällen verzichtet.

Im Folgenden wollen wir auf einige Details der Implementation eingehen, die insbesondere an den technisch interessierten Leser gerichtet sind.

Abbildung 5.2 zeigt einen Auszug aus dem Klassendiagramm der Petrinetz-Bibliotheken, die wie FIONA in C++ implementiert wurden. Die Klasse *PetriNet* dient zur Repräsentation eines Petrinetzes. Instanzen der Klasse *Place* beschreiben die Plätze des Netzes mitsamt ihrer Markierung. Transitionen werden durch Instanzen der Klasse *Transition* repräsentiert. Die Flussrelation des Netzes wird über Instanzen der Klasse *Arc* beschrieben, die eine Verbindung zwischen zwei Netzelementen modelliert.

Mit den Operationen der Klasse *PetriNet* kann ein Petrinetz erzeugt, manipuliert und in einer Vielzahl von Formaten (z.B. oWFN, PNML [BCH<sup>+</sup>03]) ausgegeben werden. Operationen zur Simulation des Schaltverhaltens, zur Berechnung von Invarianten oder zur Verifikation bestimmter Eigenschaften sind nicht Bestandteil des Projektes und gehören auch nicht zu den weiteren Zielen der Entwicklung. Für diese Aufgaben steht bereits

---

<sup>3</sup>Tatsächlich werden die Petrinetz-Bibliotheken mittlerweile als eigenständiges Projekt weiterentwickelt.

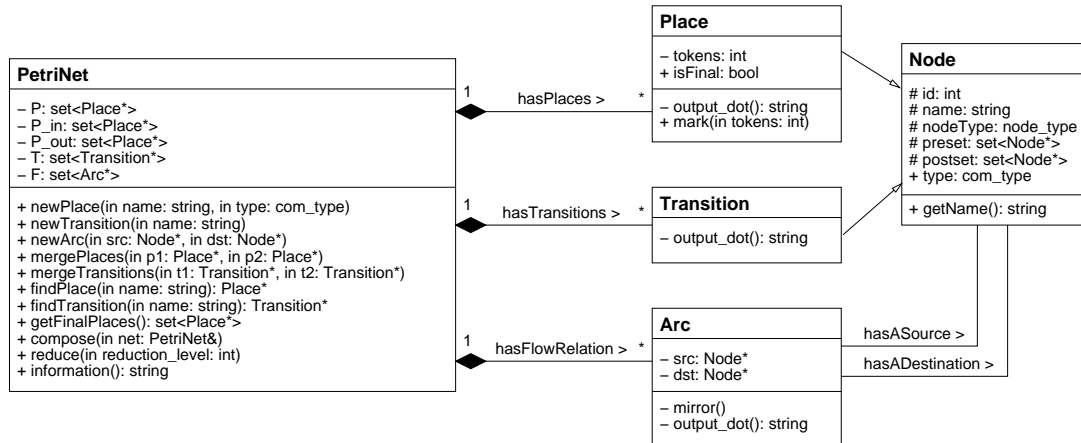


Abbildung 5.2: Auszug aus dem Klassendiagramm zur Implementation der strukturbasierten Public-View-Generierung

eine Reihe von Werkzeugen zur Verfügung (z.B. LoLA [Sch00]), deren gebräuchlichste Ein- und Ausgabeformate mit Hilfe der Petrinetz-Bibliotheken ineinander umgewandelt werden können.

Da aus der Übersetzung von BPEL-Prozessen in oWFN oft sehr große Netze resultieren, hat man sich am Lehrstuhl im Zuge Entwicklung von BPEL2oWFN erneut mit dem Thema der strukturellen Reduktion von Petrinetzen beschäftigt und u.a. die von Martens [Mar03] verwendeten Transformationsregeln in den Petrinetz-Bibliotheken implementiert. Die Operation *reduce(reduction\_level)* der Klasse *PetriNet* führt eine strukturelle Reduktion durch, die sich über den numerischen Parameter *reduction\_level* steuern lässt. Mögliche Werte für *reduction\_level* und die entsprechend zur Anwendung gebrachten Reduktionsregeln zeigt Tabelle 5.1. Je höher der Wert des Parameters *reduction\_level*, um so mehr Reduktionstechniken und -regeln werden angewendet. Für die Public-View-Generierung haben wir die maximale strukturelle Reduktion durchgeführt (*reduction\_level=6*).

Wir wollen nun kurz auf die in Tabelle 5.1 aufgeführten Reduktionsregeln eingehen, die noch nicht im Rahmen von Kapitel 4 besprochen wurden.

Unter der Eliminierung toter Netzelemente verstehen wir z.B. das Entfernen von Plätzen auf denen keine Marke liegt und die aufgrund eines leeren Vorbereichs nie durch das Schalten des Netzes markiert werden können. Des Weiteren werden zugleich alle Transitionen im Nachbereich dieser Plätze eliminiert, da diese niemals aktiviert werden können. Aber auch die Entfernung nicht verwendeter Schnittstellen-Plätze ist Teil einer solchen Reduktion.

Die BPEL2oWFN-spezifischen Reduktionsregeln wollen wir an dieser Stelle nicht näher besprechen, da diese im Allgemeinen nur eine Reduktion auf den Netzen bewirken, die aus

| <i>reduction_level</i> | strukturelle Reduktion          |
|------------------------|---------------------------------|
| 0                      | keine                           |
| 1                      | Eliminierung toter Netzelemente |
| 2                      | BPEL2oWFN-spezifische Reduktion |
| 3                      | Reduktion nach EP1 und FAlt     |
| 4                      | Reduktion nach ET1 und FSeq     |
| 5                      | Reduktion nach ET2 und EP2      |
| 6                      | Fusion gleicher Plätze          |

Tabelle 5.1: Grad der strukturellen Reduktion der Operation  $reduce(reduction\_level)$  in Abhängigkeit des Parameters  $reduction\_level$

der Übersetzung eines BPEL-Prozesses mit dem Werkzeug BPEL2oWFN entstanden sind.

Zusätzlich zu den in Kapitel 4 erläuterten Transformationsregeln wollen wir abschließend auch die in den Petrinetz-Bibliotheken implementierte Reduktionsregel zur Fusion gleicher Plätze kurz vorstellen. Die Strukturmuster der Transformationsregel verdeutlicht dabei Abbildung 5.3.

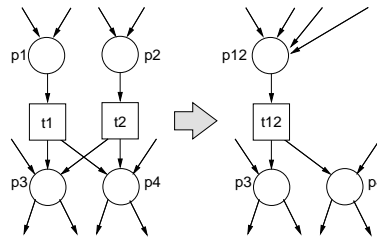


Abbildung 5.3: Strukturmuster der Reduktionsregel zur Fusion gleicher Plätze

**Definition 5.1 (Regel zur Fusion gleicher Plätze)**

Voraussetzung: *Das Muster besteht aus zwei verschiedenen Plätzen  $p1$  und  $p2$ , deren Nachbereich jeweils nur genau eine Transition  $t1$  bzw.  $t2$  umfasst. Die Transitionen  $t1$  und  $t2$  sind verschieden und besitzen bis auf  $p1$  und  $p2$  den selben Vor- und Nachbereich.*

Resultat: *Die Plätze  $p1$  und  $p2$  fusionieren zum Platz  $p12$  und die Transitionen  $t1$  und  $t2$  fusionieren zur Transition  $t12$ .*

Diese Regel wird, wie aus Tabelle 5.1 ersichtlich, nur auf der höchsten Reduktionsstufe angewendet mit der wir unsere Fallstudien durchführen werden.

Damit haben wir die Implementierungen der erfolgreichen Verfahren zur Public-View-Generierung aus Abschnitt 3.2 und Kapitel 4 ausführlich diskutiert und können die beiden Ansätze nun anhand einer Reihe von Fallstudien direkt miteinander vergleichen.

## 5.2 Fallstudien

In diesem Abschnitt wollen wir die Public-View-Generierung für verschiedene Geschäftsprozesse aus der Praxis betreiben. Für jede der präsentierten Fallstudien führen wir sowohl eine strukturbasierte als auch eine verhaltensbasierte Public-View-Generierung durch und vergleichen die beiden Verfahren direkt anhand der erzielten Ergebnisse. Die Erkenntnisse, die wir aus diesen Fallstudien gewinnen können, stellen dabei die Grundlage für unsere anschließende Auswertung und Empfehlung zum praktischen Einsatz der Verfahren in Abschnitt 5.3 dar.

Die erste Fallstudie dient vor allem dazu, das Verständnis des Lesers in Bezug auf die Anwendung der Verfahren zu vertiefen. Sie ist daher bewusst einfach gewählt. In der zweiten Fallstudie untersuchen wir einen Service, der im Sinne der SOA die Funktionalität weiterer Services nutzt, um einen Geschäftsprozess zu realisieren. Durch die Erweiterung der Interaktion von zwei auf mehrere Partner ergeben sich eine Reihe interessanter Aspekte, die wir im Rahmen dieser Fallstudie diskutieren wollen. In der dritten und letzten Fallstudie beschäftigen wir uns mit einem relativ komplexen Prozess, der ein umfangreiches Verhaltensspektrum besitzt und daher sowohl für den strukturbasierten als auch für den verhaltensbasierten Ansatz eine Herausforderung darstellt. Anhand dieser Fallstudie wollen wir die Grenzen der Leistungsfähigkeit der beiden Verfahren abstecken.

Um die Ergebnisse verschiedener Verfahren zur Public-View-Generierung vergleichen zu können, müssen wir neben einer qualitativen Beurteilung des Public View auch ein quantitatives Maß der Güte des generierten Public View einführen. Dem liegen folgende Überlegungen zugrunde: Ein optimaler Public View enthält möglichst nur kommunizierende Aktivitäten. Die Güte eines Public View ist demnach umso höher, je besser das Verhältnis der Anzahl der kommunizierenden Aktivitäten zur Anzahl aller Aktivitäten des Prozessmodells ist. In Abschnitt 4.3 haben wir ein Beispiel dafür gezeigt, wie die Komplexität eines Prozessmodells durch ungeschickte Modellierung unnötig in die Höhe getrieben werden kann. Durch eine verhaltensbasierte Public-View-Generierung ließen sich in diesem Beispiel redundante Kommunikationsaktivitäten zusammenfassen, so dass der generierte Public View eine geringere Komplexität aufwies. Wir stellen also fest, dass ein optimaler Public View zudem möglichst wenige redundante Kommunikationsaktivitäten enthält. Für die Güte eines Public-View-Modells heißt das, dass sie umso höher ist, je besser das Verhältnis der Anzahl der Nachrichtenkanäle zur Anzahl der kommunizierenden Aktivitäten des Prozessmodells ist. Für einen Public View  $N = (P, T, F)$  auf Basis von oWFN lassen sich die Gütekriterien durch die folgende Formel ausdrücken, wenn  $T_{komm}$  die Menge der kommunizierenden Transitionen von  $N$  ist:

$$\begin{aligned} Q_N &= Q_T * Q_K \\ Q_T &= |T_{komm}|/|T| \\ Q_K &= |P_i \cup P_o|/|T_{komm}| \end{aligned}$$

Ein optimaler Public View, der keine redundanten und nur kommunizierende Transitionen enthält, besitzt folglich mindestens eine Güte von 1.

Leider lässt sich diese Formel nicht ohne Weiteres auch auf die aus der verhaltensbasierten Public-View-Generierung resultierenden Serviceautomaten anwenden. Die kanonische Übersetzung eines Serviceautomaten in ein oWFN, bei der jeder Zustand des Automaten durch einen Platz im oWFN und jeder Zustandsübergang durch eine Transition im oWFN repräsentiert werden, ist selten optimal, kann aber als untere Schranke für die Güte eines verhaltensbasiert generierten Public View betrachtet werden. Der  $Q_T$ -Wert des aus der kanonischen Übersetzung resultierenden oWFN wird in Regel sehr gut sein, da die Anzahl der nicht-kommunizierenden Transitionen der Anzahl der  $\tau$ -Übergänge des Serviceautomaten entspricht, von denen es im Allgemeinen nur sehr wenige geben wird. Hingegen wird aufgrund der Sequentialität des Automaten, bei der eine Reihe von nebenläufig kommunizierten Nachrichten durch die Vielzahl möglicher Sequenzen modelliert werden muss, der  $Q_K$ -Wert für Services mit komplexem Kommunikationsverhalten sehr gering sein. Wie gut eine Übersetzung des Public-View-Serviceautomaten in ein oWFN mit Hilfe der Regionentheorie ist, wird sich folglich daran erkennen lassen, inwieweit der  $Q_K$ -Wert im Vergleich zur kanonischen Übersetzung verbessert werden konnte. Da uns jedoch keine Abschätzung der durchschnittlichen Steigerung des  $Q_K$ -Wertes möglich ist, müssen wir in dieser Arbeit allein auf die Güte des kanonisch übersetzten Serviceautomaten zurückgreifen.

### 5.2.1 Fallstudie I: Reisebuchung

In unserer ersten Fallstudie wollen wir uns mit dem Prozess einer Reisebuchung beschäftigen. Dabei handelt es sich um das Workflow-Modul „TripReservation“ aus [Mar03].

Mit Hilfe des Service kann eine Reise über das Internet gebucht werden. Um ein anschauliches Modell zu erhalten, beschränken wir uns allerdings auf folgendes einfaches Szenario: Ein Nutzer des Service übermittelt allein den gewünschten Zielort und den Zeitpunkt sowie die Dauer des Aufenthaltes. Der Service bucht einen passenden Hin- und Rückflug und arrangiert eine Hotelreservierung. Bezahlt wird die Reise per Kreditkarte. Sind die angegebenen Kreditkartendaten ungültig oder lässt sich kein Flug bzw. Hotel buchen, erhält der Nutzer eine Fehlermeldung vom Service. Anderenfalls wird dem Nutzer die Buchungsbestätigung zugesandt, die Informationen über den gebuchten Flug und die Hotelreservierung beinhaltet.

Um insbesondere das Ergebnis des strukturbasierten Verfahrens besser einschätzen zu können, wollen wir dem Leser in dieser Fallstudie den direkten Vergleich zum Private View des Service ermöglichen (siehe Abbildung 5.4). Aufgrund der geringen Komplexität des Private View können wir an dieser Stelle zudem die einzelnen Schritte der strukturbasierten Public-View-Generierung noch einmal explizit angeben (siehe Abschnitt 4.2 zur Erläuterung der Transformationsregeln):

- Schritt 1* Eliminierung der Transitionen *Check Credit Card*, *Flight Reservation*, *Hotel Reservation* und *Bill Credit Card* nach Regel ET1.
- Schritt 2* Fusionieren der Transitionen *t1* mit *Reject Reservation*, *t2* mit *Confirm Reception*, *t10* mit *Build Trip Schedule* und *Build Trip Schedule* mit *Send Trip Schedule* nach Regel FSeq.
- Schritt 3* Eliminierung von Platz *p17* nach Regel EP1.
- Schritt 4* Eliminierung der Transition *Accounting* nach Regel ET1.

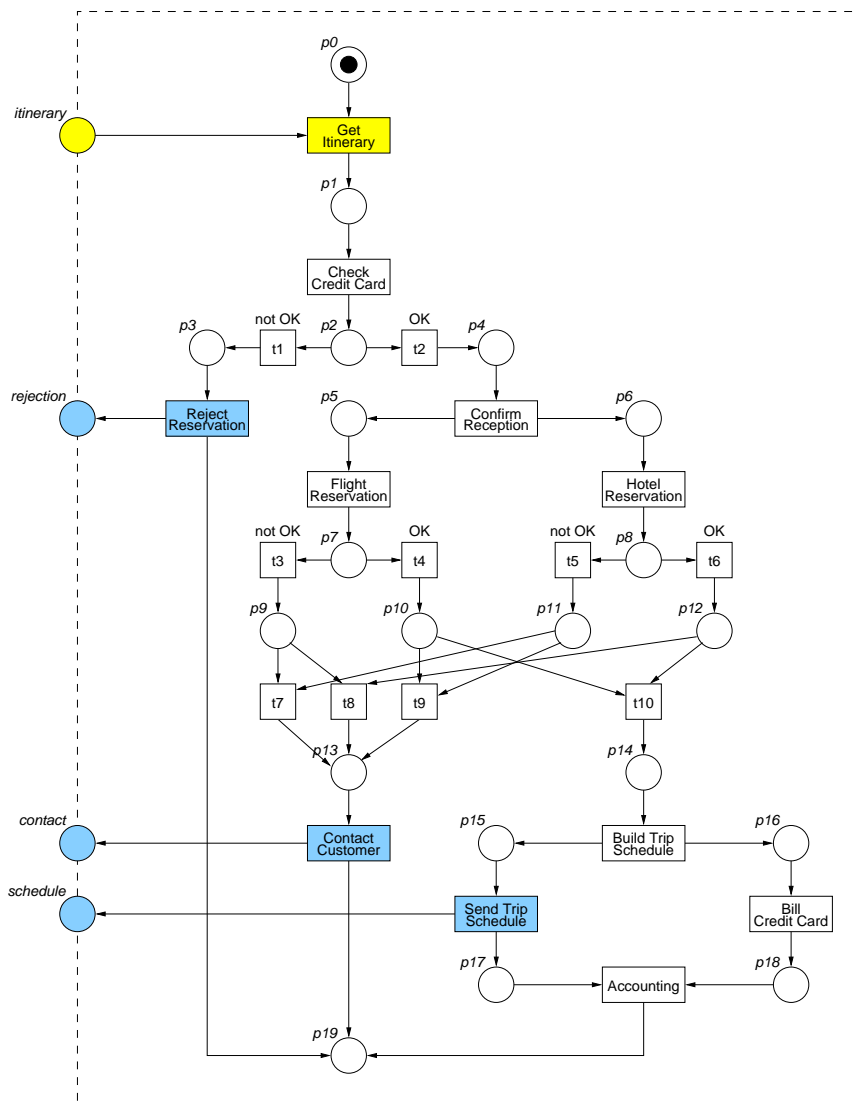


Abbildung 5.4: Private View des Reisebuchungs-Service

Den jeweils erreichten Abstraktionsgrad der Public-View-Generierung haben wir in Tabelle 5.2 zusammengefasst.  $|P|$  steht dabei für die Anzahl der Plätze des Netzes (bzw. Anzahl der Zustände der Bedienungsanleitung),  $|T|$  für die Anzahl der Transitionen des Netzes (bzw. Anzahl der Zustandsübergänge der Bedienungsanleitung),  $|P_i|$  und  $|P_o|$  für die Anzahl der Ein- bzw. Ausgabekanäle und  $Q_N$  für die Güte des Modells in Bezug auf den erreichten Abstraktionsgrad. Zum Vergleich werden wir für alle Fallstudien auch die Güte des Private-View-Prozessmodells mit angeben.

| <i>Reisebuchung</i>                      | $ P $ | $ T $ | $ P_i $ | $ P_o $ | $Q_N$ |
|--|-------|-------|---------|---------|-------|
| Private View (Abb. 5.4)                  | 24    | 21    | 1       | 3       | 0,200 |
| Bedienungsanleitung                      | 3     | 4     | 1       | 3       | -     |
| Public View strukturbasiert (Abb. 5.5)   | 14    | 12    | 1       | 3       | 0,333 |
| Public View verhaltensbasiert (Abb. 5.7) | 4     | 6     | 1       | 3       | 0,571 |
| Public View minimal (Abb. 5.6)           | 3     | 4     | 1       | 3       | 1,000 |

Tabelle 5.2: Abstraktionsgrad der Public-View-Generierung für den Reisebuchungs-Service

Bei der strukturbasierten Public-View-Generierung konnte durch die Anwendung der Transformationsregeln eine Reduktion des Netzes um mehr als 40% erreicht werden. Das generierte oWFN (siehe Abbildung 5.5) ist mit 14 Plätzen und 12 Transitionen nicht minimal, kann aber als menschenlesbar und intuitiv verständlich aufgefasst werden. Ein Vergleich mit dem minimalen Public View des Reisebuchungs-Service (siehe Abbildung 5.6) zeigt jedoch deutlich, dass noch erhebliches Reduktionspotential im strukturbasiert generierten Public View steckt. Insbesondere die Entscheidungslogik (Plätze  $p5$  bis  $p14$  und Transitionen  $t3$  bis  $t10$  im Private View), die das Ergebnis der Buchungsvorgänge auswertet, kann durch das in Kapitel 4 vorgestellte Regelwerk, sowie die Regel zur Fusion gleicher Plätze (siehe Definition 5.1) nicht wesentlich reduziert werden. Obwohl, wie aus dem minimalen Public View ersichtlich, gerade von diesen internen Aktivitäten weitestgehend abstrahiert werden sollte. Während der minimale Public View eine Güte von 1 besitzt, erreicht der strukturbasiert generierte Public View lediglich eine Güte von  $\frac{1}{3}$ .



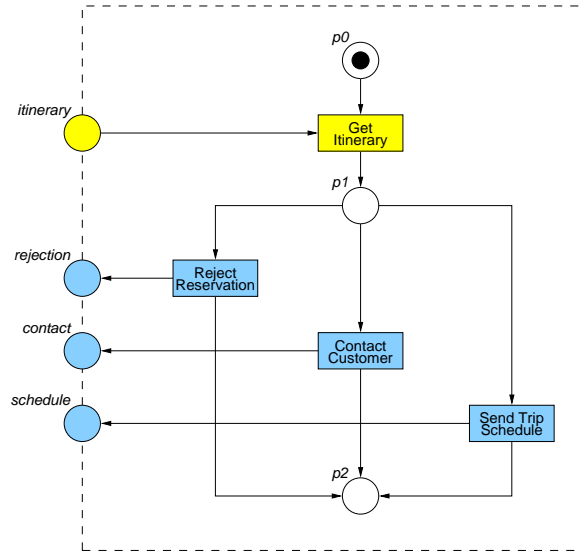


Abbildung 5.6: Minimaler Public View des Reisebuchungs-Service

Zustand  $pd$  des Serviceautomaten. Das mehrfache Senden eines Reisewunsches innerhalb eines Geschäftsvorfalles ist nicht gestattet, daher führt der Empfang einer zweiten Nachricht auf dem Kanal *itinerary* in jedem Zustand zum Deadlock  $pd$ . Im oWFN wird dieser explizite Deadlock zur Modellierung des Verhaltens allerdings nicht benötigt, da hier der Zustand der Nachrichtenkanäle bereits in den Terminierungskriterien berücksichtigt wird. Laut Definition 2.5 dürfen in einer Endmarkierung eines oWFN nämlich keine Marken mehr auf den Schnittstellen-Plätzen liegen. Eine zweite Nachricht vom Typ *itinerary* kann vom oWFN nicht konsumiert werden und macht damit das Erreichen einer Endmarkierung unmöglich. In diesem Fall könnte man also den Zustand  $pd$  vor der Übersetzung des Serviceautomaten in ein oWFN aus dem Automaten entfernen und dann tatsächlich einen Public View mit einer Güte von 1 generieren. Unter welchen Bedingungen ein explizit modellierter Deadlock im Serviceautomaten vor der Übersetzung in ein oWFN entfernt werden kann, weil das durch ihn modellierte Verhalten durch die Terminierungskriterien des oWFN ohnehin ausgeschlossen wird, lässt sich allerdings nicht so leicht beantworten und sollte daher im Rahmen weiterer Forschung untersucht werden.

Auch wenn beide Verfahren in dieser Fallstudie ein nach unseren Kriterien gutes bis sehr gutes Ergebnis erzielen, so ist der verhaltensbasierte Ansatz aufgrund des besseren Resultates hier doch klar zu favorisieren.



Abbildung 5.7: Bedienungsanleitung (a) und verhaltensbasiert generierter Public-View-Serviceautomat (b) des Reisebuchungs-Service

### 5.2.2 Fallstudie II: Kreditvergabe

Bei dieser Fallstudie handelt es sich um einen einfachen Web-Service zur Kreditvergabe. Der Service wurde als BPEL-Prozess modelliert und ist Teil einer Reihe von Beispielen aus der Spezifikation der Sprache WS-BPEL, die zunehmend zur Beschreibung von Geschäftsprozessen auf Basis der SOA eingesetzt wird [AAA<sup>+</sup>07]. Mit Hilfe des Werkzeugs BPEL2oWFN haben wir den Prozess in ein oWFN übersetzt und anschließend sowohl eine strukturbasierte als auch eine verhaltenbasierte Public-View-Generierung mit den in Abschnitt 5.1 vorgestellten Implementationen durchgeführt.

Ein Geschäftsvorfall läuft hier wie folgt ab: Ein potentieller Kreditnehmer sendet eine Anfrage an den Service, die seine persönlichen Daten und den angeforderten Kreditrahmen umfasst. In Abhängigkeit seiner Kreditwürdigkeit und des angeforderten Betrags wird dann entschieden, ob der Kredit gewährt wird oder nicht. Der Service beantwortet eine Kreditanfrage anschließend lediglich mit einer Nachricht, die die Information darüber enthält, ob der Kreditantrag genehmigt oder abgelehnt wurde.

Bei einem Kreditrahmen von unter 10.000 Euro und geringem Risiko (kreditwürdiger Antragsteller) wird der Kreditantrag im Schnellverfahren genehmigt. Bei einem größeren Betrag oder mittlerem bzw. hohem Risiko wird der Antrag genauer geprüft. Gemäß dem Konzept der SOA nutzt der Kreditvergabe-Service dafür die Funktionalität zweier weiterer Services. Einer dieser Services ermöglicht eine schnelle Bemessung der Kreditwürdigkeit des Antragstellers und wird im Schnellverfahren für kleinere Kreditrahmen eingesetzt. Der andere Service realisiert eine umfangreiche Prüfung des Kreditantrages (möglicherweise unter Einbeziehung eines Experten), falls das Schnellverfahren nicht angewendet werden kann.

Zum besseren Verständnis haben wir den positiven Kontrollfluss des Service als oWFN modelliert (siehe Abbildung 5.8), werden aber für die Public-View-Generierung auf den vollständigen Prozess (als oWFN 95 Plätze und 112 Transitionen) zurückgreifen.

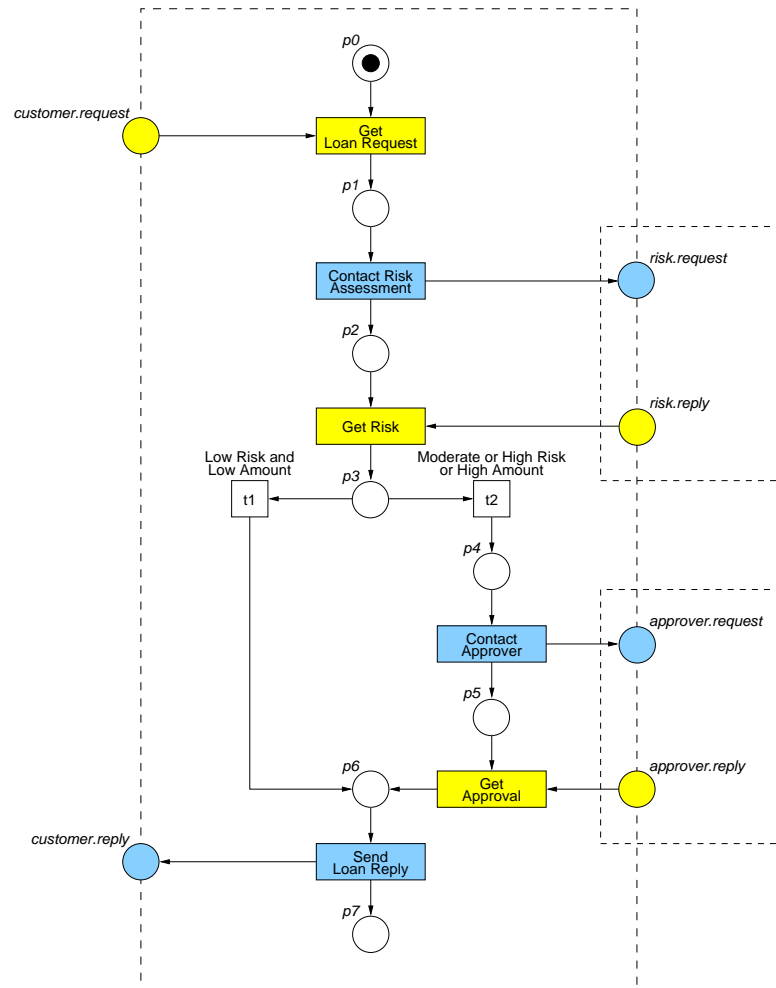


Abbildung 5.8: Positiver Kontrollfluss des Kreditvergabe-Service

Tabelle 5.3 zeigt den jeweils erreichten Abstraktionsgrad der Public-View-Generierung für den vollständigen Kreditvergabe-Service.

| <i>Kreditvergabe</i>          | $ P $ | $ T $ | $ P_i $ | $ P_o $ | $Q_N$ |
|-------------------------------|-------|-------|---------|---------|-------|
| Private View                  | 95    | 112   | 3       | 3       | 0,054 |
| Bedienungsanleitung           | 20    | 36    | 3       | 3       | -     |
| Public View strukturbasiert   | 75    | 87    | 3       | 3       | 0,069 |
| Public View verhaltensbasiert | 21    | 78    | 3       | 3       | 0,077 |

Tabelle 5.3: Erreichter Abstraktionsgrad der Public-View-Generierung für den Kreditvergabe-Service

Durch die Anwendung der Transformationsregeln konnte mit dem strukturbasierten Ver-

fahren eine Reduktion des Netzes um ca. 21% erreicht werden. Damit liegt der Reduktionsgrad deutlich unter dem in der ersten Fallstudie erreichten Wert. Der strukturbasiert generierte Public View ist mit 75 Plätzen und 87 Transitionen zu groß, um als menschenlesbar und intuitiv verständlich gelten zu können. Dies spiegelt sich auch in der geringen Güte von nur 0,069 wider. Wir verzichten daher an dieser Stelle auf eine Abbildung des erzeugten oWFN.

Die kanonische Übersetzung des verhaltensbasiert generierten Public-View-Serviceautomaten kann, mit 21 Plätzen und 78 Transitionen, in Anbetracht der äußerst geringen Komplexität des Kreditvergabe-Service ebenfalls nicht als optimaler Public View angesehen werden. Die kanonische Übersetzung des Public-View-Serviceautomaten besitzt eine Güte von 0,077 und liegt damit leicht über dem mit dem strukturbasierten Verfahren erreichten Wert.

Auf den ersten Blick sind die Resultate beider Verfahren also eher enttäuschend. Es lohnt sich daher, der Frage nachzugehen, warum der jeweils generierte Public View derart komplex ist.

Offensichtlich konnten die Transformationsregeln des strukturbasierten Verfahrens nicht in dem Maße angewendet werden, wie man dies vermutet hätte. Das erhärtet den Verdacht, dass das verwendete Regelwerk noch nicht umfangreich genug ist, um eine weitestgehende Reduktion der Netzstruktur zu ermöglichen. Denkbar ist zudem, dass der Private View des Kreditvergabe-Service eines der Beispiele darstellt, die einer strukturellen Reduktion besonders unzugänglich sind.

In unserer Beschreibung des Kreditvergabe-Service zu Beginn dieses Abschnitts stellte sich dessen Verhalten als recht einfach dar. Es überrascht daher um so mehr, dass der verhaltensbasierte Ansatz zur Public-View-Generierung nicht ein deutlich besseres Resultat erzielen konnte.

Betrachten wir die Bedienungsanleitung des Kreditvergabe-Service, so stellen wir fest, dass diese aus der Sicht eines potentiellen Kreditnehmers unerwartet komplex ist. Bisher sind wir in unserer Untersuchung immer von einer 1-zu-1 Beziehung zwischen dem Service und einem Partner ausgegangen. Wie bereits eingangs erwähnt, stellt sich in dieser Fallstudie die Situation jedoch etwas anders dar. Da der Kreditvergabe-Service die Funktionalität zweier weiterer Services in Anspruch nimmt, und daher mit diesen Nachrichten austauschen muss, beinhaltet die Bedienungsanleitung nicht nur das vom Antragsteller erwartete Verhalten, sondern auch das der beiden genutzten Services. Der generierte Public View enthält daher wesentlich mehr Informationen, als sie für einen potentieller Kreditnehmer zum Verständnis des Service erforderlich wären.

In Abschnitt 2.5 haben wir bereits darauf hingewiesen, dass es nicht immer *den* Public View an sich gibt, sondern möglicherweise mehrere Public Views, die auf verschiedene Adressaten hin ausgerichtet sind. Ein Nutzer des Kreditvergabe-Service muss z.B. keine Kenntnis über die weiteren vom Service genutzten Dienste haben, da er ohnehin keinen Einfluss auf deren Verhalten ausüben kann. Der an einen potentiellen Kreditnehmer gerichtete Public View sollte daher von diesen Services und insbesondere deren Kommunikationsverhalten abstrahieren.

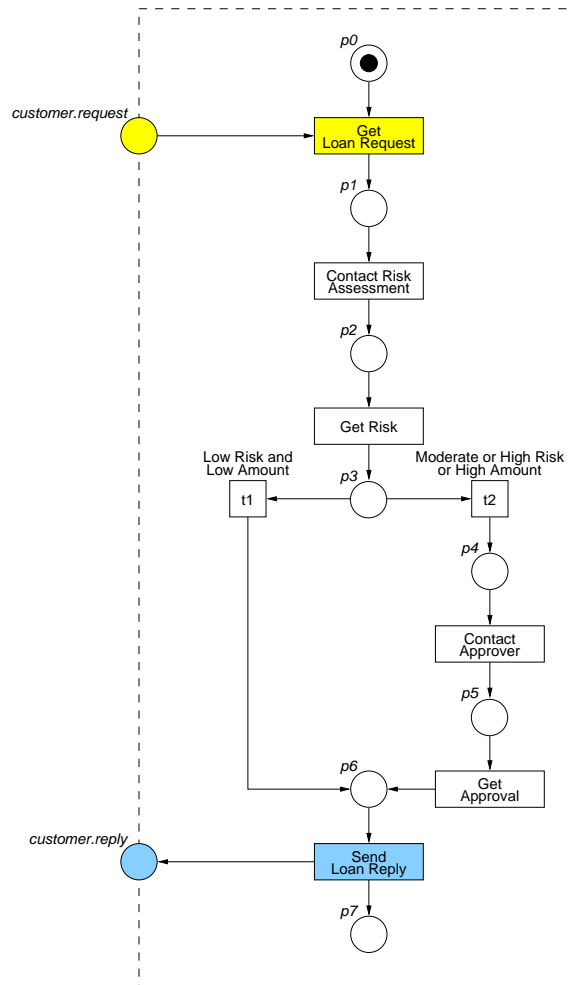


Abbildung 5.9: Positiver Kontrollfluss des Kreditvergabe-Service zugeschnitten auf die Kommunikation mit dem potentiellen Kreditnehmer

Setzt man die Kommunikation zwischen dem Kreditvergabe-Service und den beiden weiteren Services als problemlos voraus, so kann man das aus der Übersetzung des BPEL-Prozesses entstandene oWFN um die entsprechenden Schnittstellen-Plätze reduzieren.<sup>4</sup> Abbildung 5.9 veranschaulicht dieses Vorgehen am positiven Kontrollfluss des Kreditvergabe-Service aus Abbildung 5.8. Die Transitionen *Contact Risk Assessment*, *Get Risk*, *Contact Approver* und *Get Approval* wurden hierbei in interne Transitionen umgewandelt und die entsprechenden Schnittstellen-Plätze *risk.request*, *risk.reply*, *approver.request* und *approver.reply* entfernt. Die Bedienungsanleitung dieses leicht modi-

<sup>4</sup>Bei komplexem Kommunikationsverhalten zwischen dem zu untersuchenden Service und den von ihm verwendeten Services ist dies allerdings nicht ohne Weiteres möglich.

fizierten Kreditvergabe-Service beschreibt dann allein das vom Antragsteller erwartete Verhalten in der Interaktion der Services.

| <i>Kreditvergabe (Kundensicht)</i> | $ P $ | $ T $ | $ P_i $ | $ P_o $ | $Q_N$ |
|------------------------------------|-------|-------|---------|---------|-------|
| Private View                       | 95    | 112   | 1       | 1       | 0,021 |
| Bedienungsanleitung                | 3     | 2     | 1       | 1       | -     |
| Public View strukturbasiert        | 66    | 75    | 1       | 1       | 0,030 |
| Public View verhaltensbasiert      | 4     | 4     | 1       | 1       | 0,500 |

Tabelle 5.4: Erreichter Abstraktionsgrad der Public-View-Generierung für den modifizierten Kreditvergabe-Service

Tabelle 5.4 fasst die Ergebnisse der erneuten Public-View-Generierung für den modifizierten Kreditvergabe-Service zusammen. Durch die Modifikation des Prozesses verliert die Bedienungsanleitung erheblich an Komplexität und besitzt nur noch 3 Zustände und 2 Transitionen (siehe Abbildung 5.10 (a)). Dementsprechend reduziert sich auch der verhaltensbasiert generierte Public-View-Serviceautomat auf 4 Zustände und 4 Transitionen (siehe Abbildung 5.10 (b)). Die kanonische Übersetzung der Public-View-Serviceautomaten besitzt dann eine Güte von 0,5. Das heißt, dass durch eine Adressierung des Public View dessen Güte um über 600% gesteigert werden konnte. Dies verdeutlicht, welchen Einfluss eine automatische Ausrichtung des Public View auf die Public-View-Generierung ausüben kann. Der strukturbasierte Ansatz kann in dieser Fallstudie offensichtlich nicht von einer Adressierung des Prozessmodells profitieren. Der strukturbasiert generierte Public View des modifizierten Kreditvergabe-Service besitzt nur eine Güte von 0,030.<sup>5</sup>

Damit kann in dieser Fallstudie, nach einer kleinen Anpassung des oWFN, mit Hilfe des verhaltensbasierten Verfahrens ein deutlich besserer Public View generiert werden als dies mit dem strukturbasierten Verfahren möglich ist.

<sup>5</sup>Die geringere Güte im Vergleich zum strukturbasiert generierten Public View des nicht-modifizierten Prozessmodells liegt an einer redundanten Kommunikationsaktivität im Private View des Kreditvergabe-Service. Während der  $Q_K$ -Wert des nicht-modifizierten oWFN bei ca. 0,86 liegt, drückt die redundante Kommunikationsaktivität bei nur zwei Nachrichtenkanälen im modifizierten oWFN dessen  $Q_K$ -Wert auf ca. 0,67 und begründet damit die insgesamt schlechtere Güte des zweiten Public View.



Abbildung 5.10: Bedienungsanleitung (a) und Public-View-Serviceautomat (b) des modifizierten Kreditvergabe-Service

### 5.2.3 Fallstudie III: Ausstellung eines Personalausweises

Bei unserer letzten Fallstudie handelt es sich um einen Service aus dem Bereich des eGovernment, der das Vorgehen zur Ausstellung eines Personalausweises innerhalb eines Einwohnermeldeamtes modelliert. Das Geschäftsprozessmodell wurde dem Lehrstuhl im Rahmen des BMBF-Projektes „Tools4BPEL“ von der Firma Gedilan zur Verfügung gestellt.

Das Szenario sieht wie folgt aus: Ein Bürger beantragt über ein Internet-Portal die Ausstellung eines Personalausweises und zahlt die entsprechende Gebühr per Überweisung oder Lastschrift. Der Service ermittelt das zuständige Einwohnermeldeamt und leitet die Daten an das Amt weiter. Sobald das Amt den Zahlungseingang der Gebühr festgestellt hat, wird dem Bürger eine Nachricht zugesendet, dass der Antrag bearbeitet wird. Sind die vom Antragsteller übermittelten Daten korrekt, bekommt er regelmäßig eine Nachricht über den aktuellen Stand des Vorgangs: z.B. Antrag genehmigt, Personalausweis ausgestellt, Personalausweis versandt. Treten Probleme während der Abarbeitung des Prozesses auf, wird der Bürger ebenfalls benachrichtigt und gebeten, fehlerhafte Daten zu korrigieren oder offene Gebühren zu bezahlen.

Der Service lag, wie in der in Abschnitt 5.2.2 besprochenen Fallstudie, als BPEL-Prozess vor und wurde mit dem Werkzeug BPEL2oWFN in ein oWFN übersetzt, auf dem anschließend sowohl die strukturbasierte als auch die verhaltensbasierte Public-View-Generierung vorgenommen wurde.

Tabelle 5.5 zeigt den jeweils erreichten Abstraktionsgrad der Public-View-Generierung für den Service der Personalausweis-Ausstellung.

Mit dem strukturbasierten Verfahren konnte durch die Anwendung der Transformationsregeln eine Reduktion des Netzes um ca. 25% erreicht werden. Das Ergebnis der strukturellen Reduktion ist demnach nicht wesentlich besser als in der Fallstudie des Kreditvergabe-Service. Das generierte oWFN ist mit einer Güte von 0,062 in jedem Fall

| <i>Personalausweis-Ausstellung</i> | $ P $ | $ T $ | $ P_i $ | $ P_o $ | $Q_N$ |
|------------------------------------|-------|-------|---------|---------|-------|
| Private View                       | 224   | 237   | 2       | 9       | 0,046 |
| Bedienungsanleitung                | 1536  | 7936  | 2       | 9       | -     |
| Public View strukturbasiert        | 171   | 177   | 2       | 9       | 0,062 |
| Public View verhaltensbasiert      | 1537  | 9728  | 2       | 9       | 0,001 |

Tabelle 5.5: Erreichter Abstraktionsgrad der Public-View-Generierung für den Service der Personalausweis-Ausstellung

deutlich zu komplex, um als Public View dienen zu können, der dem Bürger das intuitive Verständnis des Prozesses ermöglicht. Auch hier zeigt sich, dass das Regelwerk des strukturbasierten Verfahrens ausgebaut werden muss, um eine weiter gehende Reduktion erreichen zu können.

Aufgrund des komplexen Kommunikationsverhaltens des Service besitzt dieser eine sehr umfangreiche Bedienungsanleitung. In dieser Fallstudie lässt sich die Anzahl der Schnittstellen-Plätze des Private View zudem nicht durch eine Adressierung des Public View verringern. Die relativ große Zahl an Nachrichtenkanälen und die damit einhergehende Vielzahl nebenläufiger Kommunikationsaktivitäten lassen die Größe des Zustandsraum des Service enorm anwachsen. Daher hat die kanonische Übersetzung des Public-View-Serviceautomaten 1537 Plätze und 9728 Transitionen. Der berechnete Public View besitzt damit eine Güte von 0,001 und ist demnach noch weniger als Public View geeignet als der Private View selbst. Hier zeigt sich die bereits in Abschnitt 3.2.4 diskutierte Schwäche des verhaltensbasierten Ansatzes, dass für recht umfangreiche Bedienungsanleitungen mit diesem Verfahren meist kein menschenlesbarer Public View generiert werden kann. Inwieweit eine geschickte Übersetzung des Serviceautomaten die Güte des resultierenden Prozessmodells verbessern könnte, lässt sich hier in dieser Arbeit leider nicht abschätzen.

Trotz der verhältnismäßig geringen Reduktion des Netzes erzielt also der strukturbasierte Ansatz in dieser Fallstudie das deutlich bessere Ergebnis.

## 5.3 Auswertung und Empfehlungen zum Einsatz der Verfahren

Nachdem wir den Einsatz der Verfahren nun anhand einer Reihe von Geschäftsprozessen aus der Praxis erprobt haben, wollen wir in diesem Abschnitt eine allgemeine Auswertung der Ergebnisse vornehmen und Empfehlung zur Anwendung der Verfahren ableiten.

Es ist deutlich geworden, dass mit keinem der vorgestellten Verfahren eine automatische Public-View-Generierung für beliebige Prozesse möglich ist, die alle der von uns in Abschnitt 2.5 gestellten Anforderungen an einen Public View erfüllt. Wir wollen daher diese Anforderungen noch einmal gesondert betrachten und diskutieren, welches der Verfahren die jeweilige Anforderung in welchem Maße erfüllen kann.

### Äquivalenz im Verhalten von Private View und Public View

Die Äquivalenz im Verhalten von Private View und Public View stellt die wichtigste Anforderung an die Public-View-Generierung dar. Kann sich ein potentieller Nutzer eines Service nicht an dessen Bedienungsanleitung orientieren, so muss er alle Informationen, die nötig sind, um mit dem Service problemlos zu interagieren, aus dessen Public View ableiten. Die eigentliche Kommunikation zwischen den Services findet später jedoch mit dem vom Private View repräsentierten Prozess statt und kann daher nur dann problemlos verlaufen, wenn das vom Public View modellierte Verhalten mit dem tatsächlichen Verhalten des Prozesses übereinstimmt.<sup>6</sup>

Sowohl der verhaltensbasierte Ansatz von Wolf aus Abschnitt 3.2 als auch das struktur-basierte Vorgehen von Martens aus Kapitel 4 gewährleisten nachweislich die Äquivalenz im Verhalten von Private View und generiertem Public View, so dass diese Anforderung von beiden Verfahren vollständig erfüllt wird.

### Abstraktion von vertraulichen Informationen / Betriebsgeheimnissen

Im Zuge der Public-View-Generierung soll von datenabhängigen Entscheidungen und anderen konkreten Details der Prozessausführung weitgehend abstrahiert werden. Dies dient jedoch nicht allein der Reduzierung der Komplexität des Private View, um die Verständlichkeit des Modells zu erhöhen. Vielmehr sollen bestimmte Informationen gezielt vor dem Nutzer des Public View verborgen werden. Bei diesen Informationen handelt es sich dann zumeist um vertrauliche Informationen bzw. sogenannte Betriebsgeheimnisse, durch deren Offenlegung dem Unternehmen mitunter ein erheblicher wirtschaftlicher Schaden drohen könnte.

Die von uns in dieser Arbeit verwendeten Prozessmodelle auf Basis von oWFN besitzen keine Ausdrucksmittel, um vertrauliche Informationen im Private-View-Prozessmodell

---

<sup>6</sup>Mit anderen Worten: Private View und Public View müssen die selbe Bedienungsanleitung besitzen.

gesondert auszuzeichnen. Die vorgestellten Verfahren zur Public-View-Generierung zielen daher auf den größtmöglichen Grad der Abstraktion ab, mit dem das Verhalten des Private View gerade noch hinreichend genau beschrieben werden kann.

Dem verhaltensbasierten Ansatz liegt die Bedienungsanleitung zugrunde, zu deren Berechnung nicht die Struktur des Service sondern sein Zustandsraum herangezogen wird. Die Bedienungsanleitung enthält daher keine Informationen, die nicht zwingend zur Beschreibung des Kommunikationsverhaltens des Service erforderlich wären. Möglicherweise repräsentieren aber gerade Teile des Kommunikationsprotokolls vertrauliche Informationen des Serviceanbieters. In diesem Fall muss die Bedienungsanleitung vor der verhaltensbasierten Public-View-Generierung auf einen bestimmten Partner hin ausgerichtet und vom Kommunikationsverhalten der weiteren Partnern „bereinigt“ werden. Dies kann z.B. mit einem Ansatz geschehen, wie wir ihn in der Analyse der Fallstudie des Kreditvergabe-Service in Abschnitt 5.2.2 vorgestellt haben. Kann selbst auf diese Weise nicht vom zu schützenden Verhalten abstrahiert werden, so ist es nicht möglich, einen verhaltensäquivalenten Public View zu generieren. In allen anderen Fällen ist mit einer verhaltensbasierten Public-View-Generierung eine weitestgehende Abstraktion von vertraulichen Informationen gewährleistet.

Inwieweit mit dem strukturbasierten Verfahren von vertraulichen Informationen abstrahiert werden kann, hängt maßgeblich vom verwendeten Transformationsregelwerk ab. Das mit dem in dieser Arbeit vorgestellten Regelwerk eine weitestgehende Abstraktion noch nicht möglich ist, ist in Abschnitt 5.2.1 im Rahmen der Fallstudie des Reisebuchungs-Service deutlich geworden. Aber selbst mit einem sehr umfangreichen Regelwerk kann eine strukturbasierte Public-View-Generierung vertrauliche Informationen nicht in dem Maße verbergen, wie dies mit dem verhaltensbasierten Verfahren möglich ist. Prinzipiell birgt jeder Bereich des Private View, der nicht mit Hilfe einer Transformationsregel reduziert werden kann, die Gefahr des Verrats von Betriebsgeheimnissen im strukturbasiert generierten Public View.

#### **Intuitive Verständlichkeit und Menschenlesbarkeit**

Auf Basis der SOA finden die meisten Interaktionen zwischen Services - also zwischen Maschinen - statt. Die intuitive Verständlichkeit und Menschenlesbarkeit eines Public View spielt in dieser Situation kaum eine entscheidende Rolle. Soll ein Prozess entwickelt werden, der die Funktionalität eines vorhandenen Service nutzt, so kann mit Hilfe der Bedienungsanleitung der kanonische Partner für diesen Service berechnet und eine seiner Strategien in einem Top-Down-Verfahren so zu einem neuen Service verfeinert werden, dass die Interaktion der beiden Services stets problemlos verläuft. Im Grunde kann ein Modellierer eine solche Verfeinerung durchführen, ohne dafür einen Public View des Service studieren zu müssen. Allerdings ist es für Prozesse an denen Menschen beteiligt sind oft hilfreich, wenn ein anschauliches Modell zur Grundlage der Arbeit genommen werden kann. Sollen z.B. Geschäftsprozesse im Zuge einer Optimierung oder Evolution der Geschäftspraktiken diskutiert werden, fällt dies anhand eines intuitiven Public-View-Modells wesentlich leichter als mit dem detailreichen Private View.

Die Beurteilung der intuitiven Verständlichkeit und Menschenlesbarkeit eines Prozessmodells kann je nach Betrachter mitunter stark variieren. Für komplexe Prozesse mit einem großem Verhaltensspektrum wird sich womöglich gar keine Abstraktion finden lassen, die das Verhalten hinreichend genau beschreibt und dabei trotzdem intuitiv verständlich bleibt. Wir nehmen für unsere Untersuchung die Größe des Prozessmodells (bei oWFN die Anzahl der Plätze und Transitionen und bei Serviceautomaten die Anzahl der Zustände und Zustandsübergänge) bzw. das in Abschnitt 5.2 vorgestellte Gütekriterium als Maß für die intuitive Verständlichkeit und Menschenlesbarkeit.

Die Fallstudien haben gezeigt, dass weder mit der verhaltensbasierten noch mit der strukturbasierten Public-View-Generierung die intuitive Verständlichkeit des generierten Public View garantiert werden kann. Mit den in dieser Arbeit vorgestellten Verfahren sind wir in der Lage, für Services mit kleinen Bedienungsanleitungen oder sehr einfacher interner Struktur einen intuitiv verständlichen und menschenlesbaren Public View zu generieren. Wie in der Fallstudie des Kreditvergabe-Service aus Abschnitt 5.2.2 deutlich geworden ist, kann durch eine Fokussierung der Betrachtung des Kommunikationsverhaltens auf 1-zu-1 Beziehungen teilweise auch für umfangreiche Bedienungsanleitungen eine Reihe von intuitiv verständlichen Public Views generiert werden. Insbesondere zur Beschreibung des Verhaltens eines Netzwerks von Services, die durch ihr Zusammenspiel einen komplexen Geschäftsprozess realisieren, stellt die Generierung von auf einzelne Partner hin ausgerichtete Public Views ein probates Mittel dar, um die Komplexität des generierten Prozessmodells möglichst gering zu halten (vgl. auch [ZLY05]).

### **Fazit**

Zusammenfassend lässt sich sagen, dass weitere Bemühungen auf dem Gebiet der Public-View-Generierung nötig sind, um insbesondere die letzte der genannten Anforderungen im Allgemeinen besser zu erfüllen. Geht es vor allem um die Generierung eines maschinenlesbaren Public View, dann ist der verhaltensbasierte Ansatz von Wolf, wie in Abschnitt 3.2 beschrieben, derzeit die beste Wahl, da er nicht nur die Äquivalenz im Verhalten von Private View und Public View sondern auch die weitestgehende Abstraktion von vertraulichen Informationen garantiert. Das strukturbasierte Verfahren erzielt mit dem hier vorgestellten Regelwerk nur für Prozesse mit komplexem Kommunikationsverhalten das bessere Ergebnis. Wobei anschließend immer geprüft werden muss, ob tatsächlich von allen vertraulichen Informationen abstrahiert werden konnte.

## 6 Zusammenfassung und Ausblick

In diesem Kapitel wollen wir abschließend die Ergebnisse der Arbeit zusammenfassen und einen Ausblick auf weitere und verwandte Forschungsvorhaben geben.

### 6.1 Zusammenfassung

Ziel der Arbeit war es, die Idee einer weitgehend automatischen Public-View-Generierung anhand weiterer Untersuchungen und Algorithmen zu unterstützen. Dafür haben wir in Kapitel 3 zwei neue verhaltensbasierte Verfahren vorgestellt. Wir haben die Idee einer verhaltensbasierten Public-View-Generierung auf Basis der kausalen Beziehungen zwischen den Nachrichten eines Services skizziert und gezeigt, dass eine Umsetzung dieses Ansatzes allein auf Grundlage der Kommunikationssprache eines Service nicht möglich ist. Anschließend haben wir ein weiteres Verfahren zur verhaltensbasierten Public-View-Generierung vorgestellt, das die Bedienungsanleitung als Repräsentation des Verhaltens eines Service nutzt und für jedes so repräsentierte Service-Verhalten einen Public View automatisch berechnet [Wol07]. Um einen Vergleich mit existierenden Ansätzen zu ermöglichen, haben wir in Kapitel 4 exemplarisch das Verfahren zur strukturbasierten Public-View-Generierung von Martens präsentiert [Mar03].

Um die Praxistauglichkeit der Verfahren unter Beweis zu stellen und die Resultate der verschiedenen Ansätze zur Public-View-Generierung anhand von Fallstudien direkt zu vergleichen, wurden die vorgestellten Verfahren in C++ implementiert. Die Architektur und die den Implementationen zugrunde liegenden Algorithmen haben wir in Abschnitt 5.1 ausführlich diskutiert. Im Rahmen der Fallstudien haben wir dann untersucht, inwieweit mit den verschiedenen Ansätzen die von uns formulierten Anforderungen an einen Public View erfüllt werden können. Im Zuge der Auswertung der erzielten Resultate in Abschnitt 5.3 wurde deutlich, dass keines der Verfahren in der Lage ist, stets den optimalen Public View  $P'$  eines Service  $P$  zu generieren. Während bei der verhaltensbasierten Public-View-Generierung für Prozesse mit komplexem Kommunikationsverhalten kein menschenlesbarer und intuitiv verständlicher Public View generiert werden kann, liegen die Probleme des strukturbasierten Verfahrens neben der Komplexität des generierten Prozessmodells auch im Bereich der sicheren Abstraktion von vertraulichen Informationen des Private View.

Sowohl bei der Vorstellung der einzelnen Ansätze als auch in der Auswertung der Fallstudien haben wir auf das Potential zur Optimierung der Verfahren hingewiesen. Im Hinblick auf die Verringerung der Komplexität des generierten Public View ist für die

verhaltensbasierte Public-View-Generierung insbesondere die Frage nach einer geschickten Übersetzung des generierten Serviceautomaten in ein oWFN zu klären. Beim strukturbasierten Verfahren sollten sich weitere Bemühungen auf den Ausbau des Transformationsregelwerks konzentrieren. Damit könnte nicht nur die Komplexität des generierten Public View verringert sondern auch die Abstraktion von vertraulichen Informationen verbessert werden.

### 6.2 Ausblick

Zusätzlich zu den oben formulierten Bemühungen zur Optimierung der Verfahren zur Public-View-Generierung, die bereits Teil aktueller Forschung des Lehrstuhls sind, gibt es einige weitere interessante Fragen und Arbeiten, auf die wir an dieser Stelle hinweisen wollen.

Die Analyse der Fallstudie des Kreditvergabe-Service in Abschnitt 5.2.2 hat gezeigt, dass für einen Geschäftsprozess, der durch das Zusammenspiel mehrerer Services realisiert wird, die Fokussierung der Betrachtung auf die Kommunikation zwischen den einzelnen Services die Resultate der Public-View-Generierung erheblich verbessert. Für jeden der beteiligten Services lässt sich so ein speziell auf ihn zugeschnittener Public View auf den gesamten Geschäftsprozess generieren, der das Kommunikationsverhalten zwischen den weiteren beteiligten Services weitestgehend verbirgt. In der Fallstudie konnte ein solcher Zuschnitt durch einen manuellen Eingriff in das Private-View-Prozessmodell erreicht werden. An dieser Stelle ist weitere Forschungsarbeit nötig, um einen allgemein anwendbaren Mechanismus zu entwickeln, der den gezielten Zuschnitt des Public View auf einzelne beteiligte Services ermöglicht.

Gibt es in der Interaktion der Services, anders als in der Fallstudie, keine koordinierende Instanz sondern müssen mindestens zwei der Services untereinander kommunizieren, um erfolgreich mit einem dritten Service zusammenzuarbeiten, sprechen wir von verteilter Bedienbarkeit. Unter verteilter Bedienbarkeit ist insbesondere der hier präsentierte verhaltensbasierte Ansatz zur Public-View-Generierung nicht länger ohne Weiteres einsetzbar, da die zugrunde liegende Definition der Bedienungsanleitung auf einer 1-zu-1 Beziehung zwischen Service und Partner beruht. Es gilt also die Frage zu klären, wie sich eine verhaltensbasierte Public-View-Generierung unter verteilter Bedienbarkeit realisieren lässt.

In der Einleitung haben wir motiviert, warum wir in dieser Arbeit Geschäftsprozesse mit Petrinetzen modellieren wollen. Wir haben mit BPEL2oWFN ein Werkzeug vorgestellt, mit dem sich Prozessmodelle aus der derzeit sehr populären Sprache WS-BPEL, dem offiziellen Industriestandard für Geschäftsprozessmodellierung auf Basis von Services, in oWFN übersetzen und anschließend z.B. mit FIONA analysieren lassen. Um die Ergebnisse dieser Analyse oder einen auf Basis von oWFN generierten Public View auch den Entwicklern zugänglich zu machen, die ihre Prozesse mit WS-BPEL modellieren, hat sich Kleine mit der Transformation von oWFN zu abstrakten WS-BPEL-Prozessen

beschäftigt [Kle07]. Mit dem im Rahmen der Arbeit von Kleine entwickelten Werkzeug oWFN2BPEL lässt sich nun eine sog. Toolchain aufbauen, die eine durchgehend petri-netzbasierte Analyse und Public-View-Generierung für WS-BPEL-Prozesse ermöglicht (siehe Abbildung 6.1). Einzig ein Werkzeug zur effektiven Übersetzung eines Public-View-Serviceautomaten in ein oWFN gilt es noch zu implementieren, um auch einen verhaltensbasiert generierten Public View vollautomatisch in einen WS-BPEL-Prozess überführen zu können.

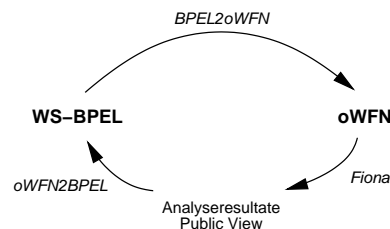


Abbildung 6.1: Toolchain zur Analyse und Public-View-Generierung von WS-BPEL-Prozessen

Die von uns in dieser Arbeit verwendete Definition der Bedienungsanleitung berücksichtigt keine kausalen Beziehungen zwischen den einzelnen Aktivitäten eines Prozesses, sondern spiegelt das theoretisch mögliche Kommunikationsverhalten des Prozesses wider. So haben wir z.B. in Abschnitt 3.1.2 bei der Untersuchung der Kommunikationssprache des Partnerautomaten  $A_{K_R}$  (siehe Abbildung 2.5 (a)) des Kinokarten-Service  $A_{K_P}$  festgestellt, dass das Senden einer Nachricht  $b$  und anschließendes Empfangen einer nach Nachricht  $r$  eine Strategie für  $A_{K_R}$  ist. Bezogen auf den Verkauf von Kinokarten hieße dies, dass zuerst die Reservierungsbestätigung versandt wird und erst anschließend Informationen darüber entgegen genommen werden, welche Vorstellung und wieviele Plätze gebucht werden sollen. Dies ist jedoch praktisch unmöglich. Um solche kausalen Abhängigkeiten in der Bedienungsanleitung zu berücksichtigen und damit auszuschließen, dass Services, die aufgrund natürlicher Beschränkungen nicht problemlos miteinander interagieren können, einander vermittelt werden, hat Kerlin ein Konzept der kausalen Bedienungsanleitung entwickelt [Ker07]. Eine Bedienungsanleitung lässt sich durch Berücksichtigung kausaler Beschränkungen oft erheblich reduzieren. Die verhaltensbasierte Public-View-Generierung auf Basis kausaler Bedienungsanleitungen wird damit in der Regel die Anforderung an einen menschenlesbaren und intuitiv verständlichen Public View deutlich besser erfüllen können. Es sollte daher näher untersucht werden, ob sich das in dieser Arbeit präsentierte Verfahren von Wolf für kausale Bedienungsanleitungen anpassen lässt.

Abschließend wollen wir darauf hinweisen, dass die Verfahren zur strukturbasierten und verhaltensbasierten Public-View-Generierung keineswegs als rein alternativ zu betrachten sind. Beispielsweise kann durch die strukturelle Reduktion eines Netzes dessen Zustandsraum im Vorfeld einer verhaltensbasierten Public-View-Generierung mitunter erheblich

reduziert werden, wobei die für die Bedienbarkeit des Prozesses entscheidenden Eigenschaften erhalten bleiben. Dadurch wird die Berechnung der Bedienungsanleitung des Service vereinfacht oder, für sehr komplexe Prozesse, gar erst ermöglicht. Denkbar ist auch, dass aus einem verhaltenbasiert generierten Public View Erkenntnisse gewonnen werden können, aus denen sich globale Reduktionsregeln, etwa für die Eliminierung redundanter Kommunikationsaktivitäten, für den strukturbasierten Ansatz ableiten lassen. All dies zeigt, dass auf dem Gebiet der automatischen Public-View-Generierung noch viele interessante Fragen zu klären sind.

# Literaturverzeichnis

- [AAA<sup>+</sup>07] ALVES, Alexandre ; ARKIN, Assaf ; ASKARY, Sid ; BARRETO, Charlton ; BLOCH, Ben ; CURBERA, Francisco ; FORD, Mark ; GOLAND, Yaron ; GUIZAR, Alejandro ; KARTHA, Neelakantan ; LIU, Canyang K. ; KHALAF, Rania ; KÖNIG, Dieter ; MARIN, Mike ; MEHTA, Vinkesh ; THATTE, Satish ; RIJN, Danny van d. ; YENDLURI, Prasad ; YIU, Alex: *Web Services Business Process Execution Language Version 2.0 / OASIS*. 2007. – Standard
- [Aal94] AALST, W.M.P. van d.: Putting Petri nets to work in industry. In: *Computers in Industry* 25 (1994), Nr. 1, S. 45–54
- [Aal98] AALST, W.M.P. van d.: The Application of Petri Nets to Workflow Management. In: *The Journal of Circuits, Systems and Computers* 8 (1998), Nr. 1, S. 21–66
- [Ale06] ALEXANDER, Adrianna: *Komposition temporallogischer Spezifikationen*. Cuvillier Verlag, 2006
- [AW01] AALST, W.M.P. van d. ; WESKE, Mathias: The P2P Approach to Interorganizational Workflows. In: *Lecture Notes in Computer Science* 2068 (2001), S. 140–156
- [BCH<sup>+</sup>03] BILLINGTON, J. ; CHRISTENSEN, S. ; HEE, K. v. ; KINDLER, E. ; KUMMER, O. ; PETRUCCI, L. ; POST, R. ; STEHNO, C. ; WEBER, M.: The Petri Net Markup Language: Concepts, technology and tools. In: *Proc. 24th Int. Conf. Application and Theory of Petri Nets (ICATPN 2003)* Bd. 2679, Springer-Verlag, Juni 2003 (Lecture Notes in Computer Science), S. 483–505
- [BD98] *Kapitel* Theory of regions. In: BADOUEL, E. ; DARONDEAU, P.: *Lecture Notes in Computer Science*. Bd. 1491: *Lectures on Petri Nets I: Basic Models*. Berlin : Springer-Verlag, 1998, S. 529–586
- [BGH04] *Kapitel* A Coloured Petri Net Approach to Protocol Verification. In: BILLINGTON, Jonathan ; GALLASCH, Guy E. ; HAN, Bing: *Lecture Notes in Computer Science*. Bd. 3098: *Lectures on Concurrency and Petri Nets*. Berlin : Springer-Verlag, 2004, S. 210–290
- [Bre07] BRETSCHNEIDER, Jan: *Produktbedienungsanleitungen zur Charakterisierung austauschbarer Services*, Humboldt-Universität zu Berlin, Diplomarbeit, 2007

- [BSBN05] *Kapitel* Modeling NoC Architectures by Means of Deterministic and Stochastic Petri Nets. In: BLUME, H. ; SYDOW, T. v. ; BECKER, D. ; NOLL, T.G.: *Lecture Notes in Computer Science*. Bd. 3553: *Embedded Computer Systems: Architectures, Modeling and Simulation*. Berlin : Springer-Verlag, 2005, S. 374–383
- [CKK<sup>+</sup>00] *Kapitel* Hardware and Petri Nets: Application to Asynchronous Circuit Design. In: CORTADELLA, Jordi ; KISHINEVSKY, Michael ; KONDRATYEV, Alex ; LAVAGNO, Luciano ; YAKOVLEV, Alex: *Lecture Notes in Computer Science*. Bd. 1825: *Application and Theory of Petri Nets*. Berlin : Springer-Verlag, 2000, S. 1–15
- [CTSH02] COLOM, J.M. ; TEUREL, E. ; SILVA, M. ; HADDAD, S.: Structural Methods. In: GIRAULT, C. (Hrsg.) ; VALK, R. (Hrsg.): *Petri Nets for System Engineering*, Springer-Verlag, 2002, S. 277–317
- [GGW98] GEHRKE, Thomas ; GOLTZ, Ursula ; WEHRHEIM, Heike: The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process / Technische Universität Braunschweig. 1998 (11/98). – Forschungsbericht
- [Got00] GOTTSCHALK, Karl: Web Services Architecture Overview / IBM developerWorks. Version: September 2000. <http://ibm.com/developerWorks/web/library/w-ovr/>. 2000. – IBM whitepaper
- [HMU07] HOPCROFT, John E. ; MOTWANI, Rajeev ; ULLMAN, Jeffrey D.: *Introduction to Automata Theory, Languages and Computation (3rd Edition)*. Addison-Wesley, 2007
- [Hoa85] HOARE, C. A. R.: *Communicating Sequential Processes*. Prentice Hall, 1985
- [Jen96] JENSEN, Kurt: *Coloured Petri Nets. Basic concepts, analysis methods and practical use*. Berlin : Springer-Verlag, 1996 (EATCS monographs on Theoretical Computer Science)
- [JVW00] JANSSENS, Gerrit K. ; VERELST, Jan ; WEYN, Bart: Techniques for Modelling Workflows and Their Support of Reuse. In: *Lecture Notes in Computer Science: Business Process Managements - Models, Techniques and Empirical Studies* 1806 (2000), S. 1–15
- [Ker07] KERLIN, Andreas: *Bedienbarkeit unter Kausalität*, Humboldt-Universität zu Berlin, Diplomarbeit, Januar 2007
- [Kle07] KLEINE, Jens: *Transformation von offenen Workflow-Netzen zu abstrakten WS-BPEL-Prozessen*, Humboldt-Universität zu Berlin, Diplomarbeit, Juli 2007

- [LMSW06] LOHMANN, Niels ; MASSUTHE, Peter ; STAHL, Christian ; WEINBERG, Daniela: Analyzing Interacting BPEL Processes. In: DUSTDAR, Schahram (Hrsg.) ; FIADEIRO, José L. (Hrsg.) ; SHETH, Amit (Hrsg.): *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings* Bd. 4102, Springer-Verlag, September 2006 (Lecture Notes in Computer Science), S. 17–32
- [LMW07] LOHMANN, Niels ; MASSUTHE, Peter ; WOLF, Karsten: *Operating Guidelines for Finite-State Services*. 2007
- [LO02] LENZ, K. ; OBERWEIS, A.: Interorganizational Business Process Management with XML Nets. In: EHRIG (Hrsg.) ; REISIG, W. (Hrsg.) ; ROZENBERG, G. (Hrsg.): *Advances in Petri Nets 2002*, Springer-Verlag, 2002 (Lecture Notes in Computer Science 2472)
- [Loh07] LOHMANN, Niels: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: DUMAS, Marlon (Hrsg.) ; HECKEL, Reiko (Hrsg.): *Web Services and Formal Methods*. Berlin : Springer-Verlag, 2007 (Lecture Notes in Computer Science)
- [LRS02] LEYMANN, F. ; ROLLER, D. ; SCHMIDT, M.-T.: Web services and business process management. In: *IBM Systems Journal* 41 (2002), Nr. 2, S. 198–211
- [LSW98] LANGNER, Peter ; SCHNEIDER, Christoph ; WEHLER, Joachim: *Lecture Notes in Computer Science*. Bd. 1420/1998: *Petri Net Based Certification of Event-Driven Process Chains*. Berlin : Springer-Verlag, 1998
- [Mar03] MARTENS, Axel: *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Dissertation, 2003. – erschienen in WiKi: Stuttgart, Berlin & Paris
- [MHKRP95] *Kapitel* An Inspection Model with Minimal and Major Maintenance for a Flexible Manufacturing Cell Using Generalized Stochastic Petri Nets. In: MOLLA-HOSSEINI, M. ; KERR, R. M. ; RANDALL, R. B. ; PLATFOOT, R. B.: *Lecture Notes in Computer Science*. Bd. 935: *Application and Theory of Petri Nets*. Berlin : Springer-Verlag, 1995, S. 335–356
- [Mil80] MILNER, Robin: *Lecture Notes in Computer Science*. Bd. 92: *A Calculus of Communicating Systems*. Springer-Verlag, 1980
- [MRS05] MASSUTHE, Peter ; REISIG, Wolfgang ; SCHMIDT, Karsten: An Operating Guideline Approach to the SOA. In: *Annals of Mathematics, Computing & Teleinformatics* 1 (2005), Nr. 3, S. 35–43
- [MS05] MASSUTHE, Peter ; SCHMIDT, Karsten: Operating Guidelines - an Alternative to Public View / Humboldt-Universität zu Berlin. 2005 (189). – Informatik-Berichte

- [NPW79] NIELSEN, Mogens ; PLOTKIN, Gordon D. ; WINSKEL, Glynn: Petri Nets, Event Structures and Domains. In: *Proceedings of the International Symposium on Semantics of Concurrent Computation*, Springer-Verlag, 1979, S. 266–284
- [Obe96] OBERWEIS, Andreas: *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Stuttgart : B.G. Teubner Verlag, 1996
- [OMG03] OBJECT MANAGEMENT GROUP (Hrsg.): *OMG Unified Modeling Language Specification*. Object Management Group, März 2003
- [Rei85] REISIG, Wolfgang: *Systementwurf mit Netzen*. Springer-Verlag, 1985. – ISBN 3–540–13786–6
- [RM97] ROJAS M., Isabel C.: *Compositional construction and analysis of Petri net systems*, University of Edinburgh, Diss., 1997
- [Sch00] SCHMIDT, Karsten: LoLA: A Low Level Analyser. In: *ICATPN 2000* Bd. 1825, Springer-Verlag, 2000 (Lecture Notes in Computer Science), S. 465–474
- [Sch01] SCHEER, August-Wilhelm: *Modellierungsmethoden, Metamodelle, Anwendungen/ARIS*. Berlin : Springer-Verlag, 2001
- [SO04] SCHULZ, Karsten A. ; ORLOWSKA, Maria E.: Facilitating cross-organisational workflows with a workflow view approach. In: *Data and Knowledge Engineering* 51 (2004), Nr. 1, S. 109–147
- [SYL<sup>+</sup>06] *Kapitel A Light-Weighted Approach to Workflow View Implementation*. In: SHAN, Zhe ; YANG, Yu ; LI, Qing ; LUO, Yi ; PENG, Zhiyong: *Lecture Notes in Computer Science*. Bd. 3841: *Frontiers of WWW Research and Development - APWeb 2006*. Berlin : Springer-Verlag, 2006, S. 1059–1070
- [Wei04] WEINBERG, Daniela: *Analyse der Bedienbarkeit*, Humboldt-Universität zu Berlin, Diplomarbeit, October 2004
- [Wol07] WOLF, Karsten: *Reversing the construction of an operating guideline*. 2007. – noch nicht erschienen
- [ZLY05] ZHAO, Xiaohui ; LIU, Chengfei ; YANG, Yun: An Organisational Perspective on Collaborative Business Processes. In: *Lecture Notes in Computer Science* 3649 (2005), S. 17–31

# Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den

Datum, Unterschrift