Verteilte Geschäftsprozesse
– Modellierung und Verifikation mit Hilfe von Web Services –

Axel Martens
Humboldt-Universität zu Berlin
E-Mail: martens@informatik.hu-berlin.de

Abstract: Web Services sind derzeit eines der dominierenden Themen in der IT-Branche, denn sie versprechen ein einheitliches Komponentenkonzept für verteilte, heterogene Systeme. Dadurch wird die Integration von Anwendungen vereinfacht, insbesondere bei Geschäftsprozessen über die Grenzen einzelner Unternehmen hinweg. Führende Software-Hersteller arbeiten mit Hochdruck an der Umsetzung der Konzepte und beginnen, darauf basierende Produkte auszuliefern. Trotzdem befinden sich derzeit noch viele der Web-Service-Technologien im Stadium der Standardisierung. Die vorliegende Arbeit betrachtet die grundlegenden Eigenschaften verteilter Geschäftsprozesse abgestützt auf die formale Modellierung mit Petrinetzen, ohne sich dabei auf eine konkrete Syntax festzulegen. Durch das gewählte Abstraktionsniveau lassen sich die Ergebnisse einfach auf jede konkrete Sprache übertragen.

1 Einleitung

Das Internet hat von sich einem reinen Kommunikationsmedium zu einer Umgebung für verteilte Systeme aller Art entwickelt. Dadurch ist es Unternehmen möglich, ihre bisher verteilt ablaufenden Geschäftsprozesse zu *einem* verteilten Geschäftsprozess zu kombinieren. Diese Arbeit befasst sich mit der Modellierung und Analyse solcher, verteilter Geschäftsprozesse.

Für dieses Anwendungsgebiet scheinen *Web Services* [CKM02] die geeignete technologische Grundlage zu sein, denn sie bieten ein standardisiertes und plattform-unabhängiges Komponentenkonzept. Ein Web Service ist eine modulare Software-Komponente mit einer wohl definierten Schnittstelle zur Außenwelt. Alle zur Benutzung notwendigen Informationen über einen Web Service sind in der öffentlich zugänglichen Beschreibung seiner Schnittstellen enthalten. Jeder lokale Geschäftsprozess kann durch einen oder mehrere Web Services implementiert werden, die Komposition von Web Services realisiert den verteilten Prozess.

Mit dem Web-Service-Ansatz entsteht in absehbarer Zeit eine gemeinsame *Syntax* zur Beschreibung der Struktur und Funktionalität verteilter Systeme. Weiterhin offen sind hingegen *semantische* Fragen: Passen zwei Web Services zusammen, so dass ihre Komposition ein fehlerfreies System ergibt? – die Frage nach *Kompatibilität*. Kann ein Web Service in einem komponierten System durch einen anderen ersetzt werden, ohne dass das System verändert werden muss? – die Frage nach *Äquivalenz*. Kann ein gegebener Web

Service überhaupt von einer anderen Komponente fehlerfrei verwendet werden? – die Frage nach *Bedienbarkeit*. Nach Einschätzung der Gartner Group [Ho02] hängt der Erfolg der Web Services maßgeblich an der methodischen Unterstützung nicht-trivialer Entwicklungsschritte. Das Ziel der Arbeit ist es, formale Methoden für die Beantwortung der o. g. Fragestellungen und damit für den systematischen Entwurf verteilter Geschäftsprozesse zu entwickeln und in ein durchgängiges Vorgehensmodell zu integrieren.

Die Erfahrung lehrt, dass in den nächsten zwei bis fünf Jahren der Weiterentwicklung viele Konzepte in der Web-Service-Architektur geändert, verfeinert oder ausgetauscht werden. Aus diesem Grund löst sich die vorliegende Arbeit von der konkreten Spezifikation und betrachtet die zugrunde liegenden Konzepte. In dieser Arbeit finden Petrinetze zur Modellierung von Web Services Verwendung. Petrinetze sind eine etablierte Methode zur Modellierung in sich geschlossener Geschäftsprozesse. Sie gestatten es, die syntaktischen Strukturen existierender Technologien (z. B. BPEL4WS, WSCI) präzise und anschaulich zu unterlegen. Damit wird eine größere Robustheit gegen syntaktische Änderungen bei gleichzeitig enger semantischer Bindung an die Thematik verteilter Geschäftsprozesse erreicht.

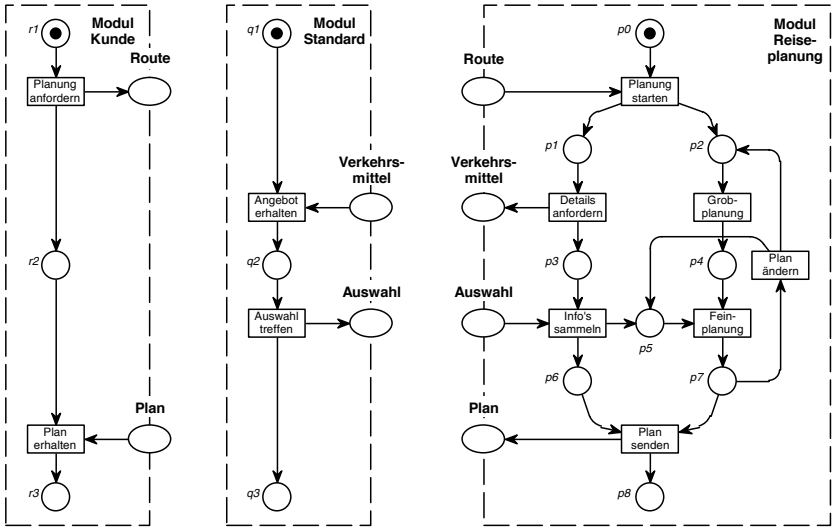
Die Handhabbarkeit der in dieser Arbeit entwickelten Methode zur Modellierung und Verifikation verteilter Geschäftsprozesse ist durch ihre prototypische Implementierung belegt. Für den kommerziellen Einsatz versteht der Autor seine Methode als theoretische Grundlage, die in weiteren Projekten an eine existierende Syntax angebunden sowie in eine reale Entwicklungsumgebung integriert werden soll, so dass die Ergebnisse der Analyse dem Anwender in einer verständlichen Form präsentiert werden und der Formalismus weitgehendst verborgen bleibt.

Diese Kurzfassung der Arbeit präsentiert einen intuitiven Überblick der Methode: Abschnitt 2 zeigt an einem kleinen Beispiel die Modellierung mit Petrinetzen, die Komposition von Workflow-Modulen und diskutiert die wesentlichen Eigenschaften. Der Nachweis der Bedienbarkeit wird im Abschnitt 3 behandelt. Dazu wird anhand eines etwas komplexeren Beispiels die algorithmische Idee skizziert. Abschnitt 4 belegt die Bedeutung der Bedienbarkeit mit dem Verweis auf weitere praktische Anwendungen der Methode und fasst die Ergebnisse zusammen.

2 Modellierung

Dieser Abschnitt beschreibt die Modellierung eines Web Service und die Komposition von mehreren Web-Service-Modellen auf Basis von Petrinetzen. Dazu betrachten wir das Zusammenspiel von drei Akteuren: einem *Anbieter*, einem *Vermittler* und einem *Kunden*. Abbildung 1 zeigt die Modelle ihrer Web Services. Der Web Service des Anbieters. erstellt eine Reiseplanung und kommuniziert zu diesem Zweck mit seiner Umgebung über vier Kanäle: zwei eingehende (Route, Auswahl) und zwei ausgehende (Verkehrsmittel, Plan).

Abbildung 1(c) zeigt ein *Petrinetz* als Modell dieses Web Service. Dabei repräsentieren ein Rechteck eine Aktivität (genannt *Transition*) und ein Kreis bzw. eine Ellipse einen Kanal zwischen den Aktivitäten (genannt *Stellen*). Eine Kante stellt die Beziehung zwischen einer Stelle und einer Transition dar. Eine Nachricht in einem Kanal wird durch eine *Marke* auf der Stelle repräsentiert. Der gewählte Ansatz abstrahiert vom Inhalt der Nachricht,



(a) Benutzer (b) Vermittler (c) Anbieter

Abbildung 1: Modellierung mit Workflow-Modulen

das heißt, zwei Marken auf einer Stelle können nicht voneinander unterschieden werden. Man nennt ein solches Modell ein *einfaches (low-level) Petrinetz* – eine grundlegende Einführung findet sich in [Re85]. In einem aktuellen Sammelband [GV02] wird der Einsatz von Petrinetzen in der ingenieurmäßigen Entwicklung komplexer Systeme umfassend beleuchtet.

Ein Petrinetz beschreibt neben der Struktur auch das *Verhalten* eines Systems. Im Beispiel aus Abbildung 1(c) kann die Transition Planung starten erst *schalten* (d. h. die Aktivität stattfinden), wenn die Anfrage eines Kunden über den Kanal Route eintrifft. Beim Schalten der Transition wird diese Marke sowie die Marke von p0 entfernt und jeweils eine Marke auf p1 und p2 erzeugt. Damit stößt die Transition Planung starten zwei parallele Handlungsstränge an: Einerseits wird dem Kunden eine Liste der möglichen Verkehrsmittel gesendet und seine Auswahl erwartet. Andererseits findet bereits eine Grobplanung statt. Die Feinplanung kann jedoch erst stattfinden, wenn die Auswahl des Kunden verarbeitet wurde, d. h. eine Marke auf p5 liegt. Nach der Feinplanung besteht die Möglichkeit, den Prozess durch die Transition Plan senden zu beenden oder die Arbeitsschritte Grob- und Feinplanung erneut durchzuführen. Das Petrinetz beschreibt hier ein *nichtdeterministisches* Verhalten, das heißt, die Entscheidung für eine der Möglichkeiten ist offen gelassen.

Ein Web Service realisiert einen lokal abgeschlossenen Geschäftsprozess, der mit seiner Umwelt über Nachrichtenaustausch kommuniziert. Aus diesem Grund besteht ein *Workflow-Modul* – das Petrinetzmodell eines Web Service – aus einem *Workflow-Netz* [vdA98], welches den internen Prozess des Web Service modelliert und einem *Interface*

– einer Menge von (Schnitt-)Stellen: Eine *Input-Stelle* repräsentiert dabei einen Nachrichtenkanal von der Umgebung zum Web Service, analog ist eine *Output-Stelle* ein Kanal vom Modul zu seiner Umgebung. Aus Gründen der Einfachheit fordern wir, dass jede Transition mit höchstens einer Sorte Schnittstellen verbunden ist.

Komposition von Workflow-Modulen

Der Zweck eines Web Services ist die Komposition mit anderen Komponenten. Deshalb müssen auch Workflow-Module miteinander komponiert werden können. Dies geschieht über das Zusammenstecken (d. h. *Verschmelzen*) der Schnittstellen. Wir betrachten die paarweise Komposition, denn der allgemeine Fall der Komposition beliebig vieler Module kann auf diesen zurückgeführt werden.

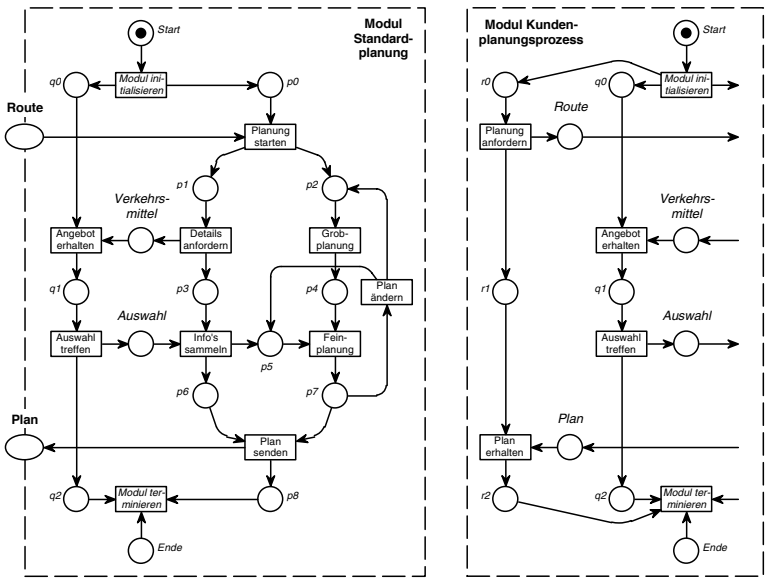
Notwendige Voraussetzung für die Komposition zweier Module ist die *syntaktische Kompatibilität* der Schnittstellen. Das bedeutet, dass jede *gemeinsame* Schnittstelle beider Module in einem Modul eine Input-Stelle und in dem anderen eine Output-Stelle sein muss. Betrachten wir die Module aus Abbildung 1. Das Modul Standard ist kompatibel zum Modul Reiseplanung, ebenso das Modul Kunde. Wie gesehen, müssen syntaktisch kompatible Module keinesfalls vollständig überdeckende Interfaces besitzen. Die Schnittstellen des einen Moduls müssen auch keine Teilmenge der Schnittstellen des anderen sein, im Extremfall haben zwei kompatible Module überhaupt keine gemeinsame Schnittstelle (z. B. Modul Kunde und Modul Standard). Dann ist jedoch zumindest der Sinn der Komposition fraglich.

Betrachten wir nun die Komposition der Module Standard und Reiseplanung. Der Zweck des Moduls Standard ist es, eine Vorauswahl abzubilden. Dazu erhält dieses Modul die Liste der verfügbaren Verkehrsmittel und trifft eine geeignete Auswahl anstelle des Kunden. Im Ergebnis der Komposition soll daher wiederum ein Workflow-Modul entstehen, das dem Kunden ein einfacheres Interface bietet. Aus diesem Grund wird das Netz nach dem Verschmelzen der Schnittstellen um eine *Initialisierungs-* und eine *Terminierungskomponente* ergänzt. Abbildung 2(a) zeigt das Modul Standardplanung – das Ergebnis der Komposition Standard $\oplus$ Reiseplanung. Das neue Interface besteht aus den offen gebliebenen Schnittstellen beider Module. Ein solches Modul nennen wir Modell eines *verteilten Web Service*.

Das Modul Kunde aus Abbildung 1(a) ist syntaktisch kompatibel zu diesem neu entstandenen Modul. Darüber hinaus überdecken sich die Schnittstellen beider Module vollständig. Wir nennen deshalb das Modul Kunde eine *Umgebung* des Moduls Standardplanung (und umgekehrt). Das Ergebnis der Komposition ist ein Workflow-Modul mit einem leeren Interface – angedeutet in Abbildung 2(b). Ein solches Modul nennen wir Modell eines *verteilten Geschäftsprozesses*. Dabei ist anzumerken, dass das Modul Standardplanung bereits aus syntaktischen Gründen um eine *Initialisierungs-* und *Terminierungskomponente* ergänzt wurde. Daher werden bei der weiteren Komposition mit dem Modul Kunde lediglich neue Kanten und keine neuen Transitionen eingefügt. Auf diese wird die Assoziativität der Komposition erreicht (d. h. $(A \oplus B) \oplus C = A \oplus (B \oplus C)$).

Eigenschaften von Workflow-Modulen

Ein Schwerpunkt der Arbeit ist die Modellierung verteilter Geschäftsprozesse. Ein Grund



(a) Komponierter Service (b) Komponierter Prozess

Abbildung 2: Komposition von Workflow-Modulen

für den Erfolg von Petrinetzen im Bereich Geschäftsprozessmodellierung liegt in dem einfachen und zweckmäßigen Kriterium für „vernünftige“ Geschäftsprozesse (siehe *Soundness*-Kriterium [vdA98]):

1. Jeder begonnene Prozess muss auch terminieren können,
2. keine Nachricht darf in einem Kanal vergessen werden und
3. jede Aufgabe ist relevant, d. h. kann durchgeführt werden.

Mit dem Begriff der Umgebung ist es nun möglich, die Qualität eines einzelnen Workflow-Moduls zu untersuchen und den zentralen Begriff der *Bedienbarkeit* eines Workflow-Moduls herzuleiten: Ein Workflow-Modul heißt *bedienbar*, wenn dieses Modul Bestandteil eines „vernünftigen“ verteilten Geschäftsprozesses sein kann. Da es sich bei einem Workflow-Modul um ein Modell eines Web Service handelt, der i. Allg. mehr Verhalten anbieten kann als ein konkreter Kunde nachfragt, sind in diesem Kontext nur die ersten beiden Forderungen relevant (d. h. *Weak-Soundness*):

Definition 2.1 (Bedienbarkeit)
Sei M ein Workflow-Modul. Eine Umgebung U *bedient* das Modul M, wenn das komponierte System $M \oplus U$ weak-sound ist. Das Modul M heißt *bedienbar*, wenn es mindestens eine Umgebung U gibt, die M bedient. $\star$

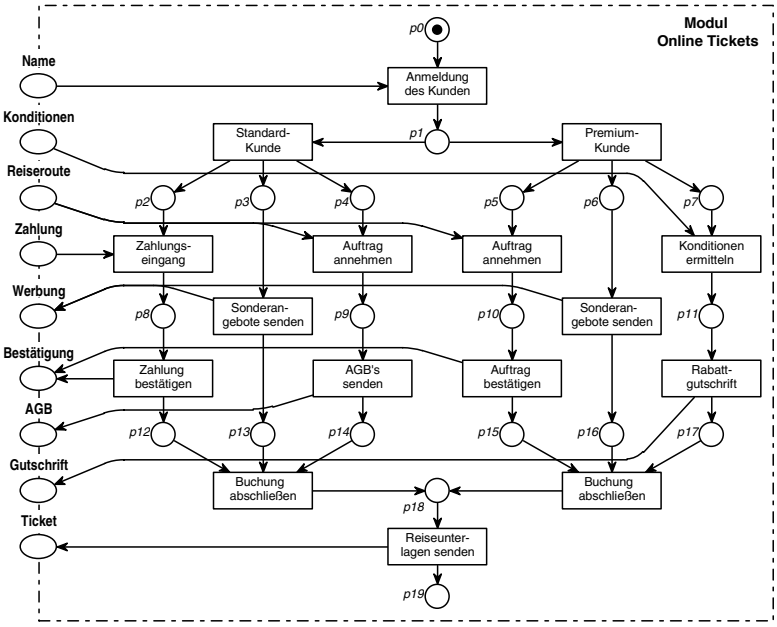


Abbildung 3: Komplexes Modul eines Online Ticket Service

Es lässt sich leicht zeigen, dass das Modul aus Abbildung 2(b) ein weak-sound Workflow-Netz ist. Damit sind die Module Standardplanung und Kunde jeweils bedienende Umgebungen für einander und selbst bedienbar. Wir nennen diese beiden Module daher zueinander *semantisch kompatibel*. Eine ausführliche Diskussion dieser und weiterer Eigenschaften findet sich in [Ma04]. Der folgende Abschnitt beschäftigt sich mit dem Nachweis der Bedienbarkeit für beliebige Workflow-Module.

3 Verifikation

Dieser Abschnitt betrachtet ein Workflow-Modul mittlerer Größe: zum einen groß genug, um die interessanten Phänomene (ohne triviale Lösung) zu umfassen, und zum anderen hinreichend klein, um nach kurzen Erläuterungen überschaubar zu bleiben. Abbildung 3 zeigt das Modul Online Tickets.

Oftmals wird das Modell eines Web Service nicht explizit neu entworfen, sondern aus einem bestehenden Geschäftsprozessmodell abgeleitet. Wir nehmen also an, dieses Modell ist aus einem ehemals umfangreicheren Geschäftsprozess ausgeschnitten und mit einem Interface versehen worden. Daraus resultiert die unerwartet komplexe Struktur. Zum Verständnis des Modells betrachten wir eine Strukturierung nach zwei Gesichtspunkten:

Die *Geschäftsstrategie* sieht die Unterteilung der Kunden vor. Nach der Anmeldung eines Kunden mit seinem Namen schaltet entweder die Transition Standard Kunde oder die Transition Premium Kunde. Diese Entscheidung liegt im Ermessen des Web Service. Die alternativen Zweige laufen mit der Transition Reiseunterlagen senden wieder zusammen.

Die *Organisation* des Online Ticket Service besteht aus drei Abteilungen (Auftragsannahme, Marketing und Buchhaltung) deren Aktivitäten weitgehend voneinander unabhängig sind. Daher hat jeder der beiden alternativen Bereiche drei nebenläufige Handlungsstränge.

Nachdem die Struktur des Moduls klar ist, stellt sich die Frage nach der Bedienbarkeit. Dazu betrachten wir folgende Umgebung: Ein Kunde sendet seinen Namen und gleichzeitig seine Konditionen, denn er wähnt sich als Premium Kunde. Aus der Struktur des Netzes geht hervor, dass beide Nachrichten unmittelbar nacheinander verarbeitet werden können, d. h. ohne weitere Kommunikation. Anschließend wartet der Kunde auf die Gutschrift. Im Modul schaltet jedoch nach der Anmeldung die Transition Standard Kunde, denn der Kunde hat lange nichts mehr bestellt und seinen Status verloren. Das Modul schickt die Werbung und wartet seinerseits auf die Reiseroute und die Zahlung. Beide Teile warten nun gegenseitig – ein typischer Deadlock. Eine für Menschen plausible Auflösung dieser Verklemmung wird an dieser Stelle nicht diskutiert, denn das gegebene Beispiel steht stellvertretend für vollautomatische Prozesse, denen kreative Strategien zur Fehlerbehandlung fehlen.

Nachweis der Bedienbarkeit

Es hat sich gezeigt, dass diese „naive“ Umgebung das gegebene Modul nicht bedient. Heißt das aber, dass das Modul nicht bedienbar ist oder wurde die Umgebung ungeschickt gewählt. Um die Bedienbarkeit eines Workflow-Moduls zu entscheiden, verwendet die Methode eine Datenstruktur, die dass *externe Verhalten* des Moduls, d. h. die Kommunikation mit der Umgebung explizit darstellt – den *Kommunikationsgraphen* des Moduls (*kurz:* K-Graphen). Abbildung 4 zeigt den K-Graphen des Online Ticket Service. Die unterschiedlichen Linien in der Darstellung werden später erläutert.

Der K-Graph besteht aus weißen und schwarzen Knoten und beschrifteten Kanten. Jeder weiße Knoten umfasst eine Menge von Zuständen des Moduls. Zu Beginn liegt im Modul Online Tickets nur eine Marke auf der Stelle p0, das heißt, das Modul befindet sich im Zustand [0] – dargestellt im Wurzelknoten des K-Graphen. Eine von einem weißen Knoten ausgehende Kante stellt eine *Eingabe* dar – eine (Multi-)Menge von Nachrichten *an* das Modul, die einerseits möglichst klein ist und andererseits ausreichend dafür ist, dass das Modul antworten kann. Im Ausgangszustand genügt der Name des Kunden ([n]). Ausgehend vom einem schwarzen Knoten sind alle daraufhin möglichen *Ausgaben vom* Modul als Kanten dargestellt. Als Antwort auf seinen Namen erhält der Kunde Werbung ([w]). Das Modul befindet sich nach diesem *Kommunikationsschritt* in einem von zwei *Folgezuständen* ([2,4,13] oder [5,7,16]). Die Konstruktion des Graphen endet, wenn das Moduls sich in einem *Endzustand* befindet.

Jeder Pfad vom Wurzelknoten zu einem Blattknoten im K-Graphen beschreibt eine Kommunikationssequenz zwischen dem Modul und einer potenziellen Umgebung. In einigen Blattknoten befinden sich Zustände, bei denen nicht alle Nachrichten aus den Kanälen entfernt wurden (z. B. [z,19]). Ein solcher Endzustände ist fehlerhaft und es gilt, die dorthin führenden Sequenzen zu vermeiden, d. h. die Umgebung muss ihre Kommunikation einschränken. In unserem Beispiel bleibt auf diese Weise nur der mit einer durchgehenden Linie dargestellte Teilgraph übrig. Diesen nennen wir den *Bediengraphen* des Moduls

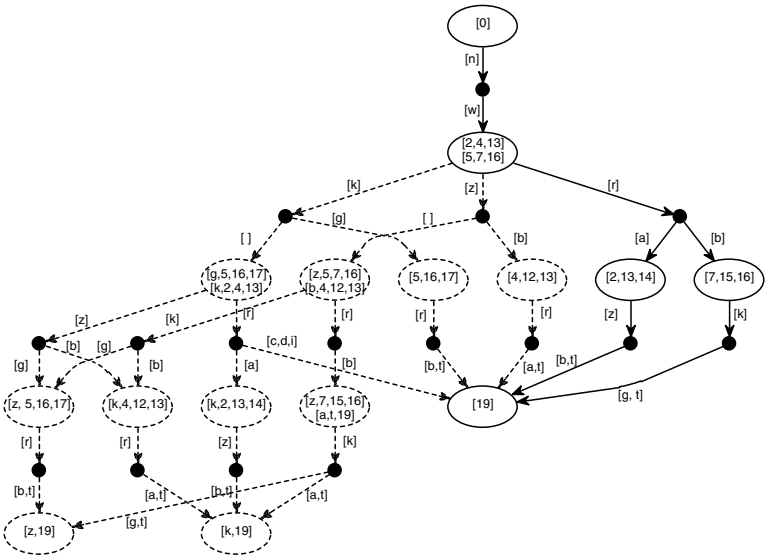


Abbildung 4: Kommunikationsgraph des Moduls Online Ticket

(*kurz:* B-Graph). Nur wenn es so einen Teilgraphen gibt, dann ist das Modul überhaupt bedienbar. Diese Aussage ist das zentrale Theorem der Arbeit.

Neben dem Nachweis der Bedienbarkeit an sich, ist der B-Graph die Bedienungsanleitung eines Moduls. Bezogen auf unser Beispiel sollte der Kunde zuerst seinen Namen ([n]) senden, die Werbung ([w]) empfangen und anschließend die Reiseroute ([r]) übermitteln. Ist er ein Premium Kunde, dann erhält er als Antwort eine Bestätigung ([b]). Anderenfalls erhält er die AGB's ([a]) des Online Ticket Service. Mit diesem Wissen kann er die weitere Kommunikation ohne Probleme fortsetzen.

4 Zusammenfassung

Web Services sind ein geeigneter Ansatz für verteilte Geschäftsprozesse. Petrinetze fokussieren auf die Konzepte des Anwendungsgebietes. Mit der entwickelten Methode ist es möglich, die *Bedienbarkeit* – eine essenzielle Qualitätseigenschaft von Web Services – effektiv und konstruktiv nachzuweisen. Dieser Artikel versteht sich als ein informaler Überblick der Methode. Alle notwendigen Begriffe sind in [Ma04] ausführlich hergeleitet und die Algorithmen präzise definiert sowie ihre Korrektheit formal bewiesen. Darüber hinaus finden sich dort Betrachtungen zur Laufzeitkomplexität und eine Klassifikation von Spezialfällen, die eine vereinfachte Analyse ermöglichen, darunter syntaktische Richtlinien, die Bedienbarkeit per Konstruktion garantierten. Die Handhabbarkeit der Methode ist durch die prototypische Implementierung der Analyse belegt [Ma03].

Neben der Analyse eines einzelnen Web Service stehen Beziehungen zwischen verschiedenen derartigen Komponenten im Blickpunkt: Als Voraussetzung für die Komposition von Web Services wird in dieser Arbeit die *semantische Kompatibilität* der Komponenten

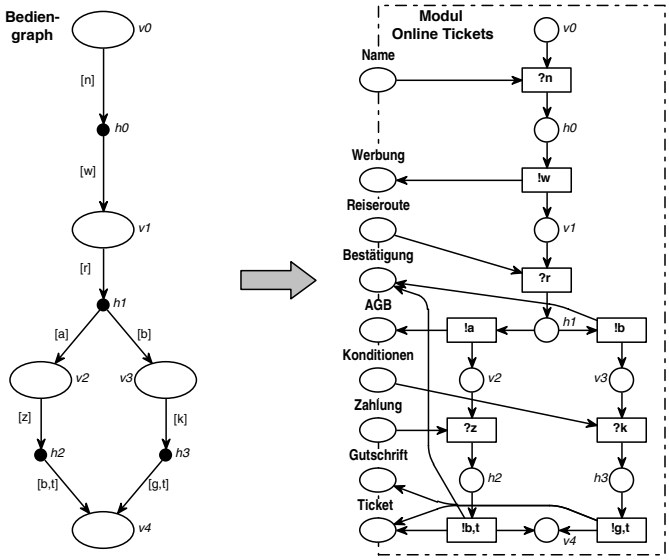


Abbildung 5: Vereinfachung des Moduls Online Ticket

gefordert. Zwei Workflow-Module sind semantisch kompatibel, wenn das komponierte System bedienbar ist. Wesentlich für den dynamischen Aufbau verteilter Geschäftsprozesse ist die *Austauschbarkeit* von Komponenten. Diese Eigenschaft wurde auf eine Simulationsbeziehung zwischen deren Kommunikationsgraphen zurückgeführt und ist somit ebenfalls effektiv entscheidbar.

Ein wichtiger Arbeitsschritt vor der Veröffentlichung eines Web Service ist die *Abstraktion* von internen Details. So lässt sich ein vereinfachtes Modell eines Web Service (, das sich nach außen hin genau so verhält wie das Original,) direkt aus dem Bediengraphen des Moduls ableiten. Abbildung 5 zeigt diese Transformation für das Beispiel des Online Ticket Service.

Weitere Forschungen zielen zum einen auf die Adaption der Methode auf eine konkrete Syntax – insbesondere auf die praxisrelevante Sprache BPEL4WS. Zum anderen ist die Verbesserung der Analysemethoden ein wichtiges Anliegen, um dem Problem der Zustandsraumexplosion durch geeignete Reduktionstechniken zu begegnen. Zu beiden Gebieten finden sich erste Ergebnisse in [MSW+04]

Literatur

[CKM02] Casati, G. A. F., Kuno, H., und Machiraju, V.: *Web Services*. Springer-Verlag. Berlin, Heidelberg, New York, Tokyo. December 2002.

[GV02] Girault, C. und Valk, R. (Hrsg.): *Petri Nets for System Engineering*. Springer-Verlag Berlin Heidelberg New York. January 2002.

[Ho02] Hostmann, B.: *Web Services for Business Intelligence – Hype and Reality.* Gartner Group Research, Inc. Whitepaper. June 2002.

[Ma03] Martens, A.: WOMBAT4WS– *Workflow Modeling and Business Analysis Toolkit for Web Services*. Manual. Humboldt-Universität zu Berlin. 2003. `http://www.informatik.hu-berlin.de/top/wombat`.

[Ma04] Martens, A.: *Verteilte Geschäftsprozesse – Modellierung und Verifikation mit Hilfe von Web Services*. Dissertation. WiKu-Verlag Stuttgart. 2004.

[MSW+04] Martens, A., Stahl, C., Weinberg, D., Fahland, D., und Heidinger, T.: *BPEL4WS – Semantik, Analyse und Visualisierung*. Informatik-Bericht 166. Humboldt-Universität zu Berlin. 2004.

[Re85] Reisig, W.: *Petri Nets*. Springer-Verlag. Berlin, Heidelberg, New York, Tokyo. Eatcs monographs on theoretical computer science. 1985.

[vdA98] van der Aalst, W. M. P.: The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*. 8(1):21–66. 1998.

Wissenschaftlicher Werdegang

Axel Martens hat an der Humboldt-Universität zu Berlin Informatik studiert und war im Anschluss als Softwarearchitekt für die SAP AG im Bereich Datenbankentwicklung tätig. Seit 1998 arbeitet er als wissenschaftlicher Mitarbeiter am Lehrstuhl für Theorie der Programmierung der Humboldt-Universität. Schwerpunkte seiner Forschung sind die Spezifikation, Modellierung und Analyse verteilter Systeme auf Basis von Petrinetzen. Er war unter an der DFG-Forschergruppe „Petrinetz-Technologie“ sowie verschiedenen industriellen Projekten beteiligt.

Im Juli 2003 hat Axel Martens mit der vorliegenden Arbeit promoviert und leitet seitdem die Arbeitsgruppe BPEL4WS – ein Forschungsprojekt zum Thema Web Services in Kooperation mit der IBM Deutschland GmbH. Derzeit arbeitet er im Rahmen eines einjährigen Forschungsaufenthalts am IBM T.J. Watson Research Center in Hawthorne (New York) an der Weiterentwicklung der Web-Service-Standards und der Integration seiner theoretischen Ergebnisse in die Produktstrategie von IBM.