Verbesserung der Modellbildung und Analyse verteilter Geschäftsprozesse durch Prozessumstrukturierung

Thomas S. Heinze[1], Wolfram Amme[1], Simon Moser[2]

[1] Friedrich-Schiller-Universität Jena
{T.Heinze,Wolfram.Amme}@uni-jena.de
[2] IBM Entwicklungslabor Böblingen
smoser@de.ibm.com

1 Einführung und Problemstellung

In den letzten Jahren konnte eine Entwicklung hin zu dienstorientierten und verteilten Software-Architekturen beobachtet werden. Diese Entwicklung findet auch innerhalb der informationstechnischen Modellierung und Realisierung von Geschäftsprozessen ihren Niederschlag. So lassen sich verteilte Geschäftsprozesse aus bestehenden, zumeist örtlich und organisatorisch unabhängigen Diensten zusammensetzen und derart neue Dienste erzeugen, die ihrerseits Grundlage für weitere Geschäftsprozesse bilden können. Die Web Services Business Process Execution Language (WS-BPEL) [1] ist die wohl derzeit vielversprechenste Spezifikationssprache für verteilte Geschäftsprozesse auf Grundlage der Web-Service-Technologie. Ein verteilter Geschäftsprozess der Sprache WS-BPEL wird als Web Service durch Komposition anderer Web Services, dass heißt durch Festlegen von deren Interaktion, implementiert. Die Interaktion der Web Services erfolgt dabei mittels Nachrichten, die über wohldefinierte Schnittstellen ausgetauscht werden. Eine wichtige Voraussetzung für eine fehlerfreie Komposition ist daher die Kompatibilität der an der Interaktion beteiligten Web Services.

Die Frage nach der Kompatibilität umfasst nicht nur die syntaktische Kompatibilität von Web Services, dass heißt die Übereinstimmung der Schnittstellen. Dies ist insbesondere der Fall, falls die Interaktion der Web Services ein für verteilte Geschäftsprozesse charakteristisches zustandsbehaftetes Kommunikationsprotokoll realisiert. Dann muss auch sichergestellt werden, dass es während der Interkation zu keinem unerwünschten Verhalten, beispielsweise zu Verklemmungen, kommt. In diesem Zusammenhang wird von der Verhaltenskompatibilität gesprochen. So werden zwei Web Services verhaltenskompatibel genannt, falls sie in einer möglichen Umgebung, in der sie miteinander interagieren, ordnungsgemäß terminieren können. Um Fehler während der Laufzeit von verteilten Geschäftsprozessen zu vermeiden, sollte die Verhaltenskompatibilität bereits beim Entwurf der Prozesse überprüft werden. Eine geeignete Methode zur Analyse von Eigenschaften wie der Verhaltenskompatibilität beruht auf der Modellbildung durch Petrinetze [4]. Herkömmliche Abbildungsvorschriften ignorieren jedoch aus Gründen der Analysierbarkeit des Petrinetzmodells die Datenaspekte

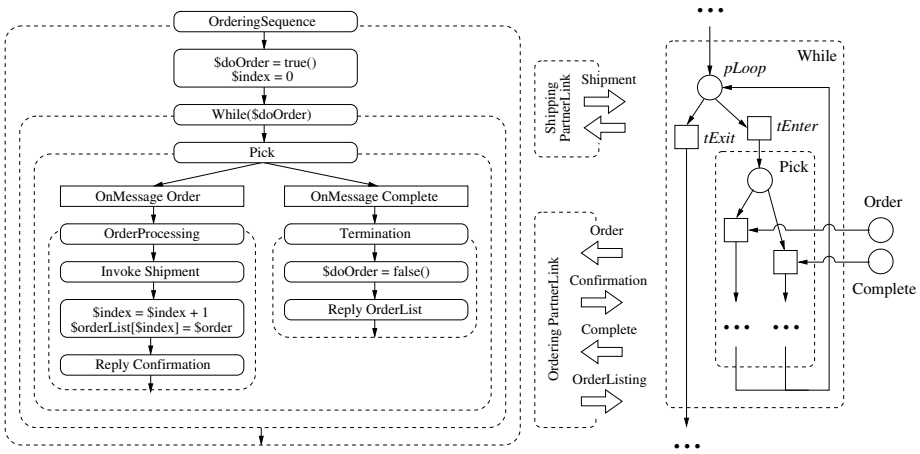

Abb. 1. `OrderingSequence` (links) und Teil des zugehörigen Petrinetzmodells (rechts)

und -abhängigkeiten der Prozesse. Als Folge dieser Abstraktion wird der bedingte Kontrollfluss durch nichtdeterministische Strukturen repräsentiert und derart fehlerhafte Analyseergebnisse, auch für einfache Prozesse, in Kauf genommen.

Ein Prozessfragment das nicht fehlerfrei analysiert werden kann, falls von Datenabhängigkeiten abstrahiert wird, ist in Abbildung 1 angegeben. Die dargestellte Aktivität `OrderingSequence` ist möglicherweise Bestandteil eines größeren Geschäftsprozesses zur Realisierung eines Online-Shops. Darin kann ein Kunde durch wiederholtes Senden der Nachricht `Order` mehrere Waren bestellen, und derart die Auftragsabwicklung für jede einzelne Bestellung einleiten. Nach Übermittlung aller Bestellungen kann er den Bestellvorgang schließlich durch Senden der Nachricht `Complete` abschließen. Zur Umsetzung dieses Protokolls enthält die Aktivität `OrderingSequence` eine Schleife, deren Ausführung durch die boolesche Variable `doOrder` geregelt wird: Ist der Wert der Variablen *true*, wird die Schleife ausgeführt, andernfalls nicht. Anfangs wird der Wert auf *true* gesetzt und so die Schleife ausgeführt. Die `Pick`-Aktivität innerhalb der Schleife aktiviert dann die Sequenz `OrderProcessing`, falls die Nachricht `Order` empfangen wurde. Wurde hingegen `Complete` übermittelt, wird stattdessen der Wert der Variablen `doOrder` auf *false* gesetzt und die Schleife in der Folge beendet.

Wird nun ein passendes Gegenstück zu diesem Prozessfragment betrachtet, dass lediglich eine Bestellung ausführt, so besteht die Interaktion aus den Nachrichten `Order`, `Confirmation`, `Complete` und `OrderList`. Offenbar sind die beiden Prozessfragmente dann verhaltenskompatibel, da es zu keiner Verklemmung kommen kann. Die in der Arbeit [4] beschriebene, petrinetzbasierte Kompatibilitätsanalyse kommt aber zum gegenteiligen Schluss: Die Schleife in `OrderingSequence` wird innerhalb des Petrinetzmodells auf die in Konflikt stehenden Transitionen *tEnter* und *tExit*, zur Modellierung von Schleifenein- und -austritt, abgebildet. Da der dadurch eingeführte Konflikt willkürlich zu lösen ist, kann die Schleife beliebig oft durchlaufen werden. Demnach können Situatio-

nen auftreten, in denen die Aktivität `OrderingSequence` weitere Bestellungen erwartet, obwohl die Nachricht `Complete` bereits empfangen wurde. Es kommt zu einer Verklemmung und die Kompatibilitätsanalyse infolgedessen zu dem Ergebnis, dass die zwei Prozessfragmente nicht kompatibel sind.

Zusammenfassend ist die Kompatibilitätsanalyse in [4] fehleranfällig. Das Weglassen der Datenabhängigkeiten von Schleifen- und Verzweigungsbedingungen führt zu einer zu starken Abstraktion innerhalb der zugrundeliegenden Petrinetzmodellierung. Um die Zahl solcher Analysefehler zu reduzieren, schlagen wir in [2] eine der eigentlichen Kompatibilitätsanalyse vorgelagerte Umstrukturierung von WS-BPEL-Prozessen vor. Diese Umstrukturierung soll im Folgenden anhand des eingeführten Prozessfragments `OrderingSequence` vorgestellt werden. Durch Anwendung unserer Umstrukturierungstechnik lassen sich bedingte Schleifen und Verzweigungen immer dann so transformieren, dass deren Daten- in Kontrollabhängigkeiten umgewandelt werden können, falls die zugehörigen Schleifen- oder Verzweigungsbedingungen zur Laufzeit nur auf Konstanten beliebigen Typs zugreifen. Als Resultat der Transformation entstehen semantisch äquivalente Prozesse, in denen die Schleifen- oder Verzweigungsbedingungen entfernt werden können. Auf diese Weise lässt sich die Anzahl nichtdeterministischer Strukturen innerhalb des Petrinetzmodells verringern und diese mögliche Ursache von fehlerhaften Ergebnissen der Kompatibilitätsanalyse einschränken.

2 Prozessrepräsentation

Zur präzisen Analyse von Geschäftsprozessen der Sprache WS-BPEL wird ein Repräsentationsformat benötigt, das in der Lage ist, sowohl den Kontrollfluss als auch die Datenaspekte und -abhängigkeiten der Prozesse wiederzugeben. Zu diesem Zweck wird eine Erweiterung von Workflow-Graphen [6], einem im Rahmen der Analyse von Geschäftsprozessen zum Einsatz kommenden Repräsentationsformat, genutzt. Da Workflow-Graphen nur den (unbedingten) Kontrollfluss der Prozesse modellieren, müssen sie für eine verlustfreie Prozessrepräsentation erweitert werden. Aufgrund ihrer Ähnlichkeit zu Kontrollflussgraphen können sie aber leicht mit der Concurrent Static Single Assignment Form (CSSA-Form) [3], einem aus dem Übersetzerbau stammenden Format, angereichert werden. Die auf diese Weise erweiterten Workflow-Graphen unterstützen damit auch die Analyse der Datenaspekte und -abhängigkeiten der repräsentierten Prozesse.

Der erweiterte Workflow-Graph für das Prozessfragment `OrderingSequence` ist in Abbildung 2 dargestellt. Aktivitäten sind darin durch Knoten wiedergegeben und Kanten verbinden diese entsprechend dem Kontrollfluss. Die elementaren Aktivitäten des Prozesses, wie beispielsweise `Reply Confirmation`, werden unter Verwendung einzelner Knoten abgebildet. Sequenzen von elementaren Aktivitäten sind dann durch mehrere sukzessive miteinander verbundende Knoten modelliert. Zur Repräsentation der strukturierten Aktivitäten `While` und `Pick` werden zusätzliche Knoten eingefügt, um die Aufspaltung (*Split*, *Pick*) und Vereinigung (*Header*, *Merge*) des Kontrollflusses abbilden zu können.

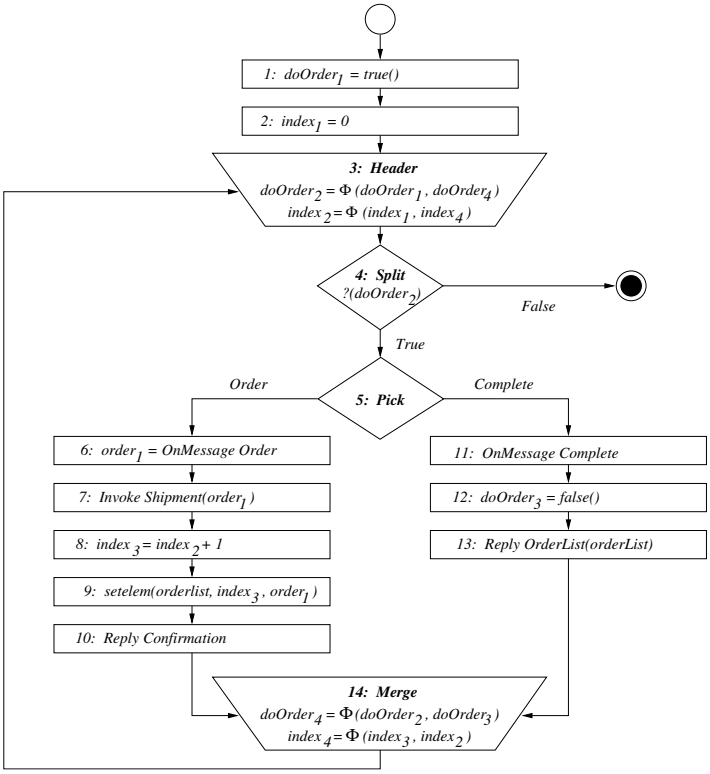

Abb. 2. Erweiterter Workflow-Graph

Wie bereits erwähnt, dient die CSSA-Form in erweiterten Workflow-Graphen der Wiedergabe von Prozessdaten und deren Manipulation. Anwendung und Vorteile dieses Repräsentationsformats für die Analyse von WS-BPEL-Prozessen wurden bereits in der Arbeit [5] beschrieben. Grundlegende Eigenschaft der CSSA-Form bildet die, statisch gesehen, einmalige Definition von Variablen. Zu diesem Zweck werden die Variablen eines Prozesses so umbenannt, dass jede Definition einen eigenen Namen erhält (beispielsweise $doOrder_1, \ldots, doOrder_4$ für Variable `doOrder` in `OrderingSequence`). Dadurch verhalten sich Variablen wie konstante Werte, insbesondere sind die Beziehungen zwischen Definition und Gebrauch einer Variablen explizit repräsentiert. Eine Ausnahme sind Schleifen, da aufgrund der statischen Betrachtungsweise Variablendefinitionen innerhalb von Schleifen als einmalige Definition gelten. Ein besonderes Vorgehen ist zudem notwendig, falls mehrere Definitionen einer Variablen auf verschiedenen Kontrollflusspfaden in einem Knoten zusammentreffen. In diesen Fällen werden Φ-Funktionen eingefügt, um die konkurrierenden Definitionen zusammenzufassen (vergleiche $doOrder_4 = \Phi(doOrder_2, doOrder_3)$ in *Merge*). Die Operanden der Φ-Funktionen bilden dabei gerade die Variablendefinitionen und der Funktionswert entspricht der Definition, die zur Laufzeit tatsächlich ausgeführt wurde.

3 Umstrukturierung

Aufbauend auf dieser Prozessrepräsentation lassen sich die Bedingungen von Schleifen und Verzweigungen analysieren. Die Bedingung der im Prozessfragment `OrderingSequence` enthaltenen Schleife entspricht genau der Variablen $doOrder_2$. Deren Wert wird durch eine Φ-Funktion im Knoten *Header* definiert. Diese führt die konkurrierenden Definitionen der Variablen vor Ausführung der Schleife ($doOrder_1$) und nach Ausführung eines Schleifendurchlaufs ($doOrder_4$) zusammen. Dabei ist die Definition nach Ausführung eines Durchlaufs ebenfalls durch eine Φ-Funktion angegeben, die die Werte auf den zwei möglichen Pfaden innerhalb der Schleife ($doOrder_2$, $doOrder_3$) zusammenfasst. Da alle diese Definitionen entweder einer Konstantenzuweisung oder einer Φ-Funktion entsprechen, hängt der Wert von $doOrder_2$ offenbar lediglich vom Pfad des Kontrollflusses während der Laufzeit ab: Wird die Schleife zum ersten Mal ausgeführt, wird $doOrder_2$ der Wert von $doOrder_1$, also *true*, zugewiesen und die Schleifenbedingung damit erfüllt. Dasselbe gilt für jeden weiteren Durchlauf der Schleife, solange bis die Zuweisung in Knoten *12* ausgeführt wird. Danach wird $doOrder_2$ der Wert von $doOrder_3$, also *false*, zugewiesen. In der Folge ist die Bedingung nicht mehr erfüllt und die Schleifenausführung wird abgebrochen.

Wir nennen Schleifenbedingungen dieser Art, in denen alle Variablen ausschließlich durch ineinander verschachtelte Φ-Funktionen und Konstantenzuweisungen definiert sind, statisch quasi-konstant. Da der Wert einer solchen Bedingung nur vom zur Laufzeit ausgeführten Kontrollflusspfad abhängig ist, können die Datenabhängigkeiten der Bedingung auch durch Kontrollabhängigkeiten repräsentiert werden. Zu diesem Zweck führen wir eine Art Schleifenabrollen durch, indem wir die Schleife durch mehrere Kopien des Schleifenrumpfs ersetzen. Jede Kopie, Schleifeninstanz genannt, entspricht dabei der Ausführung der Schleife für eine bestimmte Belegung der in der Schleifenbedingung vorkommenden Variablen. Auf diese Weise kann die Schleifenbedingung statisch ausgewertet und innerhalb des umstrukturierten Prozessfragments entfernt werden.

Vor Durchführung der eigentlichen Transformation einer Schleife mit statisch quasi-konstanter Schleifenbedingung wird diese in eine Normalform überführt. Die Normalform ist durch die Auftrennung aller Pfade des Kontrollflusses charakterisiert, auf denen in einem beliebigen Schleifendurchlauf verschiedene Werte für die Variablen der Schleifenbedingung definiert werden. Diese Überführung ist für die Schleife des Prozessfragments `OrderingSequence` in Abbildung 3 dargestellt. Die Variable der dort vorhandenen Bedingung ($doOrder_2$) kann, wie oben beschrieben, für einen beliebigen Schleifendurchlauf drei verschiedene Werte annehmen. Die Wahl des Wertes wird durch den zuvor ausgeführten Kontrollflusspfad festgelegt. Um nun die Pfade aufzutrennen, muss der Knoten *Merge* aufgelöst werden, da er zwei der drei möglichen Werte zusammenfasst ($doOrder_2$ und $doOrder_3$). Da der unmittelbare Nachfolger dieses Knotens jedoch gerade dem Kopf der Schleife *Header* entspricht, reicht es dazu aus, die Vorgängerknoten *10* und *13* direkt mit dem Kopf zu verbinden und die Operanden der darin enthaltenen Φ-Funktionen anzupassen (Operand $doOrder_4$ durch $doOrder_2$ und $doOrder_3$ ersetzen). Im Anschluss kann der Knoten *Merge* entfernt werden.

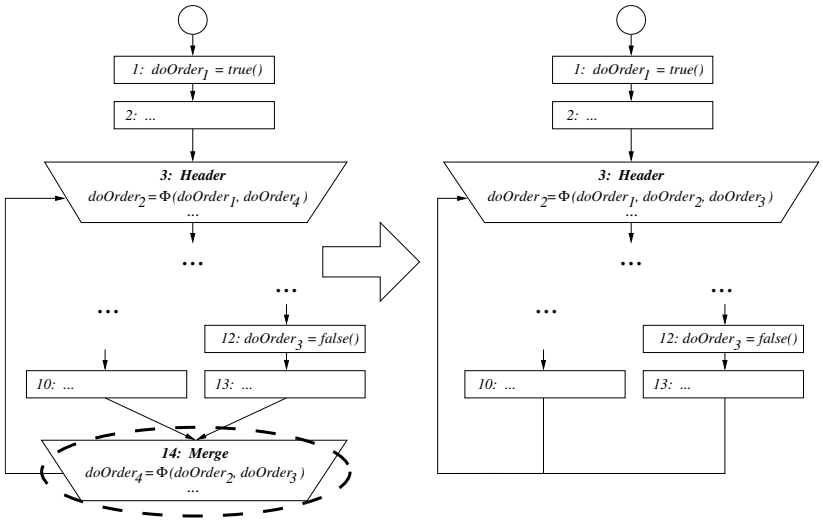

Abb. 3. Überführung der Schleife in Normalform

Im nachfolgenden Transformationsschritt werden dann die Datenabhängigkeiten der Schleife in Kontrollabhängigkeiten umgewandelt, indem die Schleife durch mehrere miteinander verbundene Schleifeninstanzen ersetzt wird. Zur Ableitung der Schleifeninstanzen kann die Normalform der Schleife als eine Art Blaupause benutzt werden. Zu diesem Zweck werden die Φ-Funktionen im Kopf der Schleife aufgelöst und die Überprüfung der Schleifenbedingung vor den Schleifenkopf positioniert. Abbildung 4 zeigt das auf diese Weise abgeleitete Muster für die Schleifeninstanzen des Fragments `OrderingSequence`. Wie zu sehen ist, befinden sich im Kopf der Schleife (*Header*) keine Φ-Funktionen mehr. Stattdessen wurde für jede eingehende Kante ein neuer Knoten (*15*, *16*, *17*) eingefügt, der Zuweisungen der der Kante zugeordneten Operanden an die ursprünglich im Schleifenkopf mittels Φ-Funktionen definierten Variablen enthält. Gleichzeitig wurden als Nachfolger dieser Knoten Kopien des Knotens *Split* mit der Schleifenbedingung eingefügt, die damit nun vor dem Kopf der Schleife überprüft wird (Knoten *18*, *19*, *20*). Da diese Kopien kontrollieren, ob die Schleife betreten wird oder nicht, werden sie im Folgenden auch als Wächterknoten bezeichnet.

Wie bereits erwähnt entspricht eine Schleifeninstanz der Ausführung der Schleife für eine bestimmte Belegung der in der Schleifenbedingung vorkommenden Variablen mit Werten. Dementsprechend lässt sich für eine Belegung dieser Variablen $(var_1, \ldots, var_n)$ mit Werten $(v_1, \ldots, v_n)$ die zugehörige Instanz erzeugen, indem jede Variable var_i innerhalb des Instanzmusters durch deren Belegung v_i ersetzt wird. Offensichtlich können die Bedingungen in den Wächtern einer Instanz statisch ausgewertet werden, da alle darin vorkommenden Variablen durch Konstanten definiert sind. Während der Transformation wird diese Eigenschaft ausgenutzt, indem die Wächterknoten sukzessive besucht und ausgewertet werden. Ist die Bedingung eines Wächters erfüllt, wird dieser durch eine

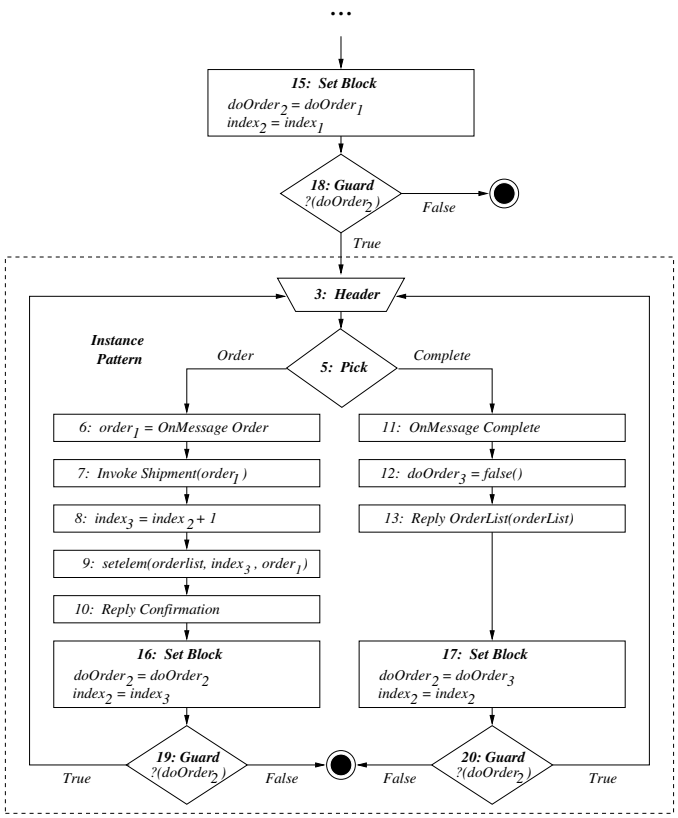

Abb. 4. Muster für Schleifeninstanzen

Kante zu einer Schleifeninstanz ersetzt, ansonsten durch eine Kante zum Austrittsknoten der Schleife. Im ersten Fall richtet sich die Auswahl der jeweiligen Instanz nach der im Vorgängerknoten des Wächters definierten Variablenbelegung. Ist noch keine Schleifeninstanz für eine Belegung vorhanden, wird eine entsprechende neue Instanz aus dem Instanzmuster generiert.

Das Ergebnis der Anwendung dieser Umstrukturierung auf das Prozessfragment `OrderingSequence` ist in Abbildung 5 angegeben. Die Transformation erfolgte in drei Schritten. Im ersten Schritt wurde der Wächter am Schleifeneintritt (Knoten *18*) ausgewertet und durch Erzeugen und Einfügen einer neuen Schleifeninstanz für die Belegung $doOrder_2 = true$ ersetzt. Diese Instanz enthielt wiederum selbst zwei Wächter, mit denen in den folgenden zwei Schritten fortgefahren wurde. Die Auswertung des Wächterknotens *19* ergab ebenfalls *true*. In der Folge wurde dieser Wächter durch eine Kante zu der im vorherigen Schritt bereits erzeugten Instanz $Instance_{doOrder_2=true}$ ersetzt. Da die Auswertung der Bedingung im Wächter *20* hingegen *false* ergab, wurde dieser durch eine Kante zum Austrittsknoten der Schleife ersetzt. Im Anschluss daran waren keine weiteren Wächterknoten mehr vorhanden, so das nur eine Instanz erzeugt wurde.

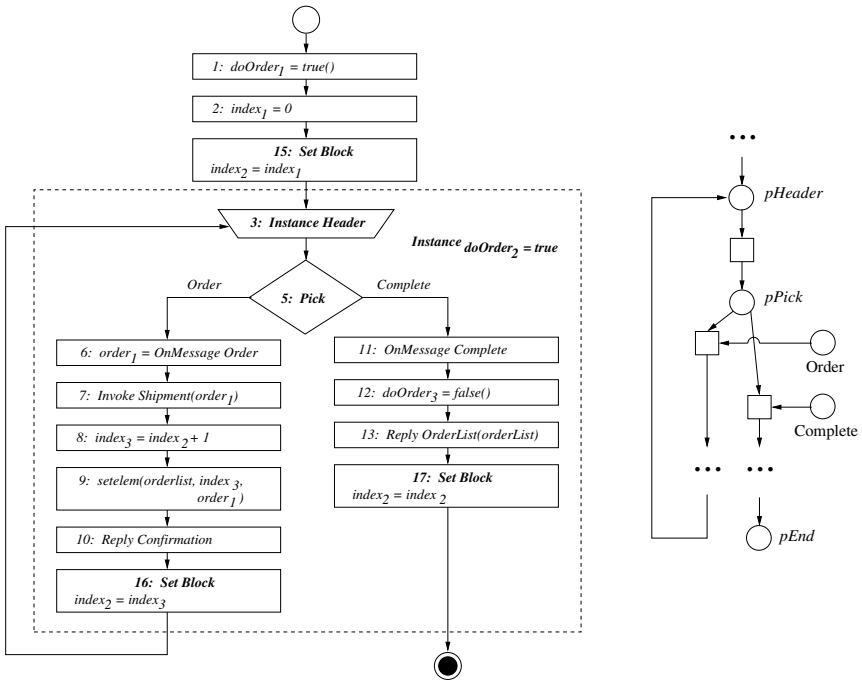

Abb. 5. Umstrukturierter Workflow-Graph (links) und Petrinetzmodell (rechts)

Da die Kompatibilitätsanalyse nach [4] ein petrinetzbasiertes Modell der zu analysierenden Prozesse nutzt, muss der erweiterte Workflow-Graph nach erfolgter Umstrukturierung wieder auf ein Petrinetz abgebildet werden. Zu diesem Zweck lässt sich die in [7] beschriebene, für reguläre Workflow-Graphen definierte Petrinetzsemantik anpassen und ausnutzen. Ein Ausschnitt des resultierenden Petrinetzes ist in Abbildung 5 dargestellt. Wie erwartet, ist darin der nichtdeterministische Konflikt des ursprünglichen Modells nicht mehr vorhanden, da die Schleifenbedingung im umstrukturierten Workflow-Graphen entfernt werden konnte. Dadurch kommt die Kompatibilitätsanalyse des Fragments `OrderingSequence` und des eingangs beschriebenen Gegenstücks nun zum korrekten Ergebnis, dass beide Aktivitäten verhaltenskompatibel sind.

4 Zusammenfassung und Ausblick

Der vorliegende Beitrag beschreibt die in [2] eingeführte Umstrukturierungstechnik für verteilte Geschäftsprozesse der Sprache WS-BPEL. Diese Technik ist in der Lage die Datenabhängigkeiten einer bestimmten Art von Schleifen und Verzweigungen in semantisch äquivalente Kontrollabhängigkeiten zu transformieren. Auf diese Weise können die Prozessmodelle bestehender Analysen präzisiert und so die Verfälschung von Analyseergebnissen eingeschränkt werden, wie sich am Beispiel einer petrinetzbasierten Kompatibilitätsanalyse zeigen lässt.

In darauf aufbauenden Arbeiten soll untersucht werden, inwiefern sich der Ansatz der Umstrukturierung auch zur Auflösung anderer Ausprägungen des bedingten Kontrollflusses anwenden lässt, neben den beschriebenen statisch quasikonstanten Bedingungen. Insbesondere sind wir an Verzweigungen und Schleifen interessiert, für die sich die möglichen Werte der in den Bedingungen vorkommenden Variablen nur teilweise oder überhaupt nicht einschränken lassen. Ein möglichen Ansatzpunkt dazu bieten abstraktere Notationen von Schleifeninstanzen. So sind Instanzen denkbar, in denen die Belegungen von Variablen mit Hilfe von Intervallen oder prädikatenlogischen Ausdrücken beschrieben werden.

Literatur

[1] Alves, Alexandre ; Arkin, Assaf ; Askary, Sid ; Barreto, Charlton ; Bloch, Ben ; Curbera, Francisco ; Ford, Mark ; Goland, Yaron ; Guízar, Alejandro ; Kartha, Neelakantan ; Liu, Canyang K. ; Khalaf, Rania ; König, Dieter ; Marin, Mike ; Mehta, Vinkesh ; Thatte, Satish ; van Rijn, Danny ; Yendluri, Prasad ; Yiu, Alex: *Web Services Business Process Execution Language Version 2.0*. Organization for the Advancement of Structured Information Standards, 2007

[2] Heinze, Thomas S. ; Amme, Wolfram ; Moser, Simon: A Restructuring Method for WS-BPEL Business Processes Based on Extended Workflow Graphs. In: Dayal, Umeshwar (Hrsg.) ; Eder, Johann (Hrsg.) ; Koehler, Jana (Hrsg.) ; Reijers, Hajo A. (Hrsg.): *Proceedings of the 7th International Conference on Business Process Management (BPM 2009), September 8-10, 2009, Ulm, Germany*, Springer-Verlag, 2009 (Lecture Notes in Computer Science 5701), S. 211–228

[3] Lee, Jaejin ; Midkiff, Samuel P. ; Padua, David A.: Concurrent Static Single Assignment Form and Constant Propagation for Explicitly Parallel Programs. In: Li, Zhiyuan (Hrsg.) ; Yew, Pen-Chung (Hrsg.) ; Huang, Chua-Huang (Hrsg.) ; Chatterjee, Siddharta (Hrsg.) ; Sadayappan, P. (Hrsg.) ; Sehr, David (Hrsg.): *Proceedings of the 10th International Workshop on Languages and Compilers for Parallel Computing (LCPC '97), August 7-9, 1997, Minneapolis, Minnesota, USA*, Springer-Verlag, 1998 (Lecture Notes in Computer Science 1366), S. 114–130

[4] Martens, Axel ; Moser, Simon ; Gerhardt, Achim ; Funk, Karoline: Analyzing Compatibility of BPEL Processes. In: *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), February 19-25, 2006, Guadeloupe, French Caribbean*, IEEE Computer Society Press, 2006, S. 147

[5] Moser, Simon ; Martens, Axel ; Görlach, Katharina ; Amme, Wolfram ; Godlinski, Artur: Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis. In: *Proceedings of the 2007 IEEE International Conference on Services Computing (SCC 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, IEEE Computer Society Press, 2007, S. 98–105

[6] Sadiq, Wasim ; Orlowska, Maria E.: Analyzing Process Models Using Graph Reduction Techniques. In: *Information Systems* 25 (2000), Nr. 2, S. 117–134

[7] van der Aalst, W. M. P. ; Hirnschall, A. ; Verbeek, H. M. W.: An Alternative Way to Analyze Workflow Graphs. In: Pidduck, A. B. (Hrsg.) ; Woo, C. (Hrsg.) ; Mylopoulos, J. (Hrsg.) ; Ozsu, M. T. (Hrsg.): *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE 2002), May 27-31, 2002, Toronto, Canada*, Springer-Verlag, 2002 (Lecture Notes in Computer Science 2348), S. 535–552