Umstrukturierung von WS-BPEL-Prozessen zur Verbesserung des Validierungsverhaltens

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

Innerhalb der letzten Jahre wurde eine Vielzahl von Methoden zur Analyse von verteilten Geschäftsprozessen der Sprache *Web Services Business Process Execution Language (WS-BPEL)* [2] entwickelt [3]. Mit Ausnahme einzelner Ansätze konzentrieren sich die meisten der Techniken auf die Analyse des Kontrollflusses und ignorieren die Datenabhängigkeiten der untersuchten Prozesse. Ein solches Vorgehen birgt aber die Gefahr der Verfälschung von Analyseergebnissen in sich. Insbesondere für die Analyse von Eigenschaften wie der *(Verhaltens-) Kompatibilität* [6] ist die Berücksichtigung der Datenabhängigkeiten von Bedeutung.

Ein Prozessfragment dass nicht fehlerfrei analysiert werden kann, falls Datenabhängigkeiten ignoriert werden, ist in Abbildung 1 dargestellt. Die abgebildete Aktivität `OrderingSequence` ist möglicherweise Bestandteil eines größeren Geschäftsprozesses zur Realisierung eines Online-Shop. Darin kann ein Kunde mehrere Bestellungen aufgeben (Nachricht `Order`), und so die Auftragsabwicklung (Aktivität `OrderProcessing`) zu jeder Bestellung einleiten. Nachdem der Kunde alle Bestellungen übertragen hat, kann er den Bestellvorgang beenden (Nachricht `Complete`). Zur Umsetzung enthält die Aktivität `OrderingSequence` eine Schleife (`While`), deren Ausführung durch die boolesche Variable `doOrder` gesteuert wird. Anfangs wird die Variable mit dem Wert true belegt und die Schleife daher durchlaufen. Die `Pick`-Aktivität innerhalb der Schleife führt dann entweder die Sequenz `OrderProcessing` aus, falls Nachricht `Order` empfangen wird, oder die Sequenz `Termination`, falls der Kunde die Nachricht `Complete` übermittelt. Im letzten Fall wird der Wert von `doOrder` auf false gesetzt und so die Schleife beendet. Der Schleifenabbruch wird demzufolge durch Empfang der Nachricht `Complete` ausgelöst, entsprechend nennen wir das verallgemeinernde Muster auch *nachrichtengesteuerter Schleifenabbruch*. Da im Sprachumfang von WS-BPEL derzeit kein Gegenstück zur break-Anweisung aus Sprachen wie Java enthalten ist [2], kann dieses Muster nur unter Verwendung einer Schleife mit einer booleschen Variablen als Schleifenbedingung realisiert werden.

Die in der Arbeit [6] angegebene und auf Petrinetzen basierende Kompatibilitätsanalyse führt für Prozesse, die Fragmente dieses Muster enthalten, zu fehlerhaften Ergebnissen. Dazu kann beispielsweise ein Partner zur Aktivität

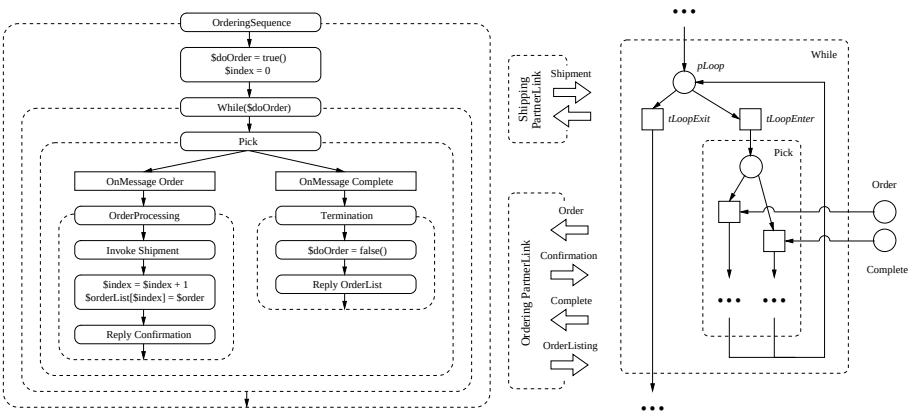

Abb. 1. `OrderingSequence` (links) und Teil des zugehörigen Petrinetzmodells (rechts)

`OrderingSequence` betrachtet werden, der genau eine Bestellung aufgibt. Die Kommunikation besteht dann aus den aufeinanderfolgenden Nachrichten `Order`, `Confirmation`, `Complete` und `OrderList`. Offensichtlich sind die beiden Fragmente kompatibel [6], da es zu keiner Verklemmung kommen kann. Die Kompatibilitätsanalyse kommt aber zum gegenteiligen Schluss. Um die Analysierbarkeit des darin verwendeten Petrinetzmodells zu gewährleisten, werden bedingte Schleifen und Verzweigungen durch Nichtdeterminismus modelliert. Die Schleife in `OrderingSequence` wird demnach auf die in Konflikt stehenden Transitionen *tLoopEnter* und *tLoopExit*, zur Repräsentation des Schleifenein- und -austritts, abgebildet (siehe auch Abbildung 1). Da der Konflikt willkürlich zu lösen ist, kann die Schleife beliebig oft durchlaufen werden. In der Folge ist möglich, dass die Aktivität `OrderingSequence` weitere Bestellungen erwartet, obwohl Nachricht `Complete` bereits empfangen wurde. Es kommt zu einer Verklemmung und die Analyse zum Ergebnis, dass die beiden Fragmente nicht kompatibel sind.

Zusammenfassend ist die Kompatibilitätsanalyse in [6] im Hinblick auf das beschriebene Muster nachrichtengesteuerter Schleifenabbruch fehleranfällig. Das Weglassen der Datenabhängigkeiten von bedingten Schleifen und Verzweigungen bedeutet eine zu starke Abstraktion innerhalb der verwendeten Petrinetzmodellierung. Um die Zahl der dadurch verursachten Analysefehler zu verringern, schlagen wir eine *Umstrukturierungsmethode* für Geschäftsprozesse der Sprache WS-BPEL vor. Diese stellen wir im Folgenden anhand des nun eingeführten Prozessfragments `OrderingSequence` vor. Die Methode erlaubt bedingte Schleifen immer dann so zu transformieren, dass deren Datenabhängigkeiten in Kontrollabhängigkeiten umgewandelt werden können, wenn deren Schleifenbedingungen zur Laufzeit nur auf Konstanten beliebigen Typs zugreifen. Das Resultat dieser Transformation ist ein semantisch äquivalenter Prozess, indem die Datenabhängigkeiten der Schleifen, dass heißt deren Bedingungen, entfernt werden können. Derart lässt sich die Anzahl von nichtdeterministischen Strukturen im Petrinetzmodell verringern und so diese mögliche Fehlerquelle einschränken.

2 Prozessrepräsentation

Um eine verlustfreie Repräsentation von Geschäftsprozessen der Sprache WS-BPEL zu ermöglichen, verwenden wir eine Erweiterung von *Workflow-Graphen*. Workflow-Graphen [8] werden häufig zur Analyse von Geschäftsprozessen genutzt, repräsentieren aber nur deren Kontrollfluss. Durch Anreicherung mit einem weiteren Repräsentationsformat, der *Concurrent Static Single Assignment Form (CSSA-Form)* [5, 7], lassen sich auch die Datenabhängigkeiten modellierter Prozesse wiedergeben. Wir nutzen daher eine Kombination beider Formate.

In Abbildung 2 ist der so erweiterte Workflow-Graph für das oben beschriebene Prozessfragment `OrderingSequence` dargestellt. Darin modellieren Knoten die Aktivitäten und Kanten verbinden die Knoten gemäß dem Kontrollfluss. Elementare Aktivitäten (beispielsweise `Reply Confirmation`) werden unter Verwendung eines einzelnen Knotens abgebildet. Sequenzen elementarer Aktivitäten sind dann durch mehrere sukzessive verbundener Knoten repräsentiert. Im Fall der Verzweigung `Pick` werden spezielle Knoten genutzt, um die Aufspaltung ($Pick$) und Vereinigung ($Merge$) des Kontrollflusses darstellen zu können. Dies gilt auch für die enthaltene Schleife `While`, für die der Knoten zum Aufspalten des Kontrollflusses ($Branch$) die Schleifenbedingung enthält und der Knoten zur Vereinigung ($Header$) des Kontrollflusses als Schleifenkopf bezeichnet wird.

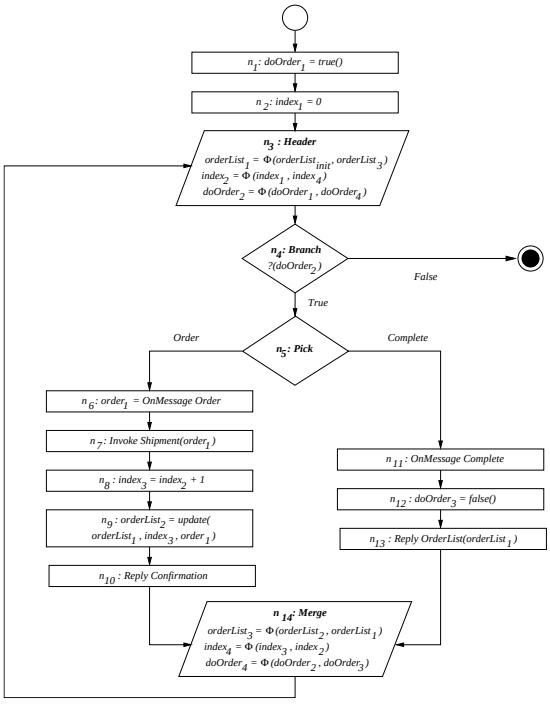

Abb. 2. Erweiterter Workflow-Graph

Grundlegende Eigenschaft der CSSA-Form ist, dass Variablen (statisch) nur einmal definiert werden dürfen.[1] Zu diesem Zweck werden die Variablen in `OrderingSequence` so umbenannt, dass jede Variablendefinition einen eigenen Namen besitzt (beispielsweise $doOrder_1, \ldots, doOrder_4$ für `doOrder`). Dadurch verhalten sich alle Variablen wie konstante Werte. Insbesondere sind Beziehungen zwischen Definition und Gebrauch einer Variablen nun explizit wiedergegeben. Ein besonderes Vorgehen ist notwendig, falls mehrere Definitionen einer Variablen auf verschiedenen Pfaden des Kontrollflusses in einem Knoten zusammentreffen (so in $Merge$ und $Header$). In diesem Fall werden Φ-Funktionen eingefügt, um die konkurrierenden Definitionen zusammenzufassen (beispielsweise $doOrder_4 = \Phi(doOrder_2, doOrder_3)$ in $Merge$). Die Operanden einer Φ-Funktion bilden 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 Verzweigungen und Schleifen analysieren. Die Bedingung der in `OrderingSequence` enthaltenen Schleife entspricht genau der Variablen $doOrder_2$. Deren Wert wird durch eine Φ-Funktion im Schleifenkopf $Header$ definiert. Diese führt die konkurrierenden Definitionen der Variablen vor Ausführung der Schleife ($doOrder_1$) und nach Ausführung eines Scheifendurchlaufs ($doOrder_4$) 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 zusammenfasst ($doOrder_2$, $doOrder_3$). Da alle Definitionen entweder einer Konstantenzuweisung oder einer Φ-Funktion entsprechen, hängt der Wert von $doOrder_2$ lediglich vom Pfad des Kontrollflusses zur Laufzeit ab: Wird die Schleife zum ersten Mal ausgeführt, wird $doOrder_2$ der Wert von $doOrder_1$, also true, zugewiesen und die Schleifenbedingung daher erfüllt. Dasselbe gilt für jeden weiteren Durchlauf, solange bis die Zuweisung in Knoten n_{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 Schleife wird abgebrochen.

Wir nennen Schleifenbedingungen dieser Art, in denen alle Variablen ausschließlich durch ineinander geschachtelte Φ-Funktionen und Konstantenzuweisungen definiert sind, *dynamisch konstant*. Da der Wert einer solchen Bedingung nur vom zur Laufzeit ausgeführten Kontrollflusspfad abhängig ist, können die Datenabhängigkeiten der Bedingung offenbar auch durch Kontrollabhängigkeiten repräsentiert werden. Durch eine geeignete Transformation der zugehörigen Schleife lassen sich die entsprechenden Kontrollabhängigkeiten erzeugen.[2] Auf diese Weise wird die Schleifenbedingung redundant und kann innerhalb des umstrukturierten Prozessfragments entfernt werden.

[1] Aufgrund der statischen Betrachtungsweise werden auch Variablendefinitionen innerhalb von Schleifen als einmalige Definitionen angesehen.

[2] Als eine Einschränkung der Umstrukturierungsmethode werden solche Φ-Funktionen ausgeschlossen, die innerhalb von Schleifenköpfen anderer Schleifen definiert werden.

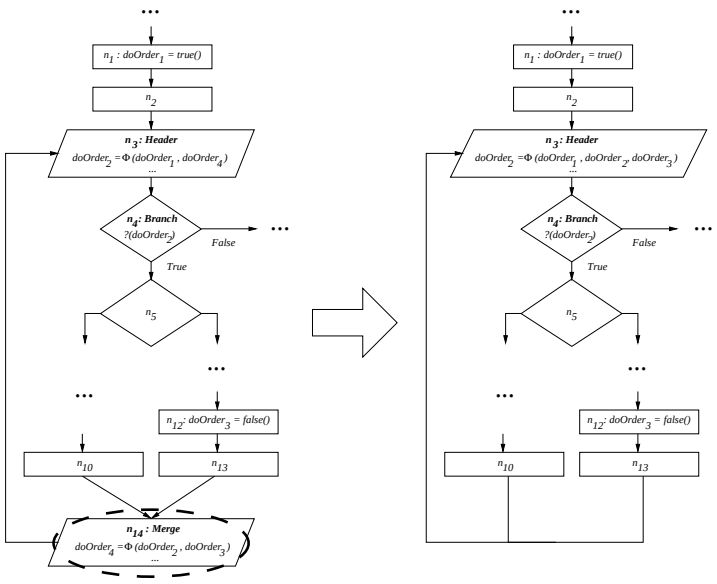

Abb. 3. Überführung der Schleife in Normalform

Vor der eigentlichen Transformation einer Schleife mit dynamisch konstanter Schleifenbedingung, wird diese in *Normalform* überführt. Die Normalform ist durch die Auftrennung aller Pfade des Kontrollflusses charakterisiert, auf denen in einem beliebigen Schleifendurchlauf unterschiedliche 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 dabei durch den zuvor ausgeführten Kontrollflusspfad bestimmt. Um nun die Pfade aufzutrennen, muss der Knoten $Merge$ aufgelöst werden, da er zwei der drei möglichen Werte zusammenführt ($doOrder_2$ und $doOrder_3$). Da der unmittelbare Nachfolger dieses Knotens gerade dem Schleifenkopf $Header$ entspricht, reicht es dazu aus, die Vorgängerknoten n_{10} und n_{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). Anschließend kann der Knoten $Merge$ entfernt werden.

Die Normalform wird dann als Blaupause im folgenden Transformationsschritt genutzt. In diesem Schritt werden mehrere Instanzen der Blaupause erzeugt und miteinander verbunden. Wir nennen den Schritt dementsprechend *Schleifeninstanziierung*. Jede Instanz repräsentiert eine mögliche Belegung der in der Schleifenbedingung genutzten Variablen mit konstanten Werten. Für die Schleife in `OrderingSequence` werden so zwei Instanzen erzeugt, wie in Abbildung 4 dargestellt. In der ersten Instanz ($Instance\,1$) wird die Variable $doOrder_2$ durch den Wert true ersetzt, in der zweiten ($Instance\,2$) durch den Wert false.

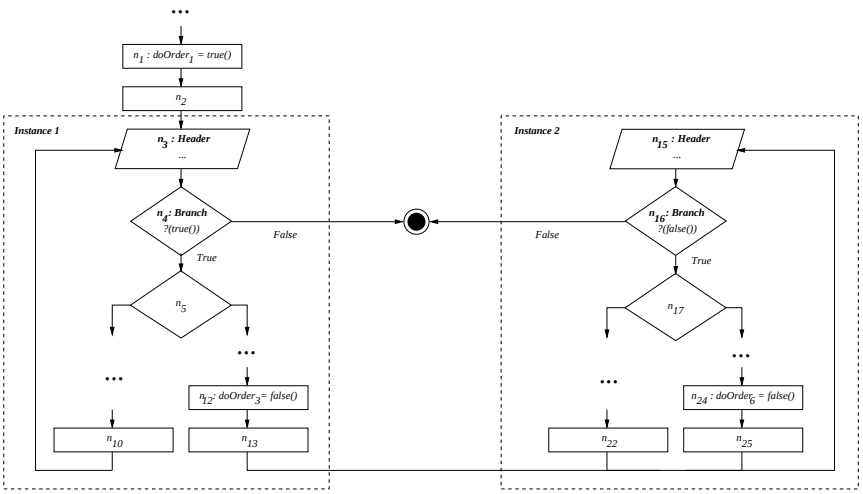

Abb. 4. Transformation unter Verwendung von Schleifeninstanzen

Dadurch ist die Schleifenbedingung in beiden Instanzen statisch auswertbar. In der Folge lassen sich die Bedingung und die unmöglichen Kontrollflusspfade in `OrderingSequence` entfernen (Kanten (n_4, End) und (n_{16}, n_{17})). Das Ergebnis dieser Umstrukturierung ist in Abbildung 5 angegeben. Offenbar ist darin der Abbruch der Schleifenausführung, nach Empfang der Nachricht `Complete`, explizit durch den Kontrollfluss wiedergegeben. Innerhalb des zugehörigen Petrinetzmodells [1] kann daher auf die Verwendung nichtdeterministischer Konflikte verzichtet, und so eine folgende Kompatibilitätsanalyse präzisiert werden.

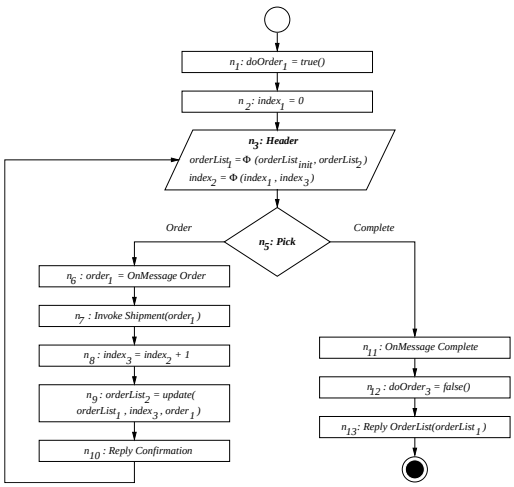

Abb. 5. Umstrukturiertes Prozessfragment

4 Zusammenfassung

Der vorliegende Beitrag beschreibt eine bereits vollständig entwickelte Umstrukturierungsmethode für verteilte Geschäftsprozesse der Sprache WS-BPEL. Die vorgestellte Methode ist in der Lage die Datenabhängigkeiten einer bestimmten Art von Schleifen in semantisch äquivalente Kontrollabhängigkeiten zu transformieren. Auf diese Weise können die Prozessmodelle bestehender Analysen präzisiert und so die Verfälschungen von Analyseergebnissen verringert werden.

Die vorgestellte Umstrukturierungsmethode kann auch auf Verzweigungsbedingungen angewendet werden und ist nicht auf Bedingungen mit einer einzelnen Variablen oder auf boolesche Variablen beschränkt. Eine detaillierte Darstellung der Methode, einschließlich der Beschreibung verwendeter Algorithmen und einem Korrektheitsbeweis, erfolgt in einem technischen Bericht [4].

Literatur

[1] Aalst, W. M. P. d. ; 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 (LNCS 2348), S. 535–552

[2] 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 ; Rijn, Danny van ; Yendluri, Prasad ; Yiu, Alex: *Web Services Business Process Execution Language Version 2.0*. 2007

[3] Breugel, Franck van ; Koshkina, Maria: *Models and Verification of BPEL*. 2006

[4] Heinze, Thomas S. ; Amme, Wolfram ; Moser, Simon: Resolving Conditional Branches in WS-BPEL Business Processes / Friedrich-Schiller-Universität Jena, Fakultät für Mathematik und Informatik. 2009. – noch nicht erschienen

[5] 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 (LNCS 1366), S. 114–130

[6] 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

[7] 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

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