EPK-Visualisierung von BPEL4WS Prozessdefinitionen

Jan Mendling
Abteilung für Wirtschaftsinformatik
Wirtschaftsuniversität Wien
jan.mendling@wu-wien.ac.at

Jörg Ziemann
Institut für Wirtschaftsinformatik
Universität des Saarlandes
ziemann@iwi.uni-sb.de

Zusammenfassung: Dieser Beitrag stellt eine Abbildung von BPEL auf EPKs vor. Eine solche Abbildung ist für die Abstimmung von Prozessmodellen mit Fachabteilungen von Bedeutung. Es wird gezeigt, wie sich BPEL-Aktivitäten auf entsprechende EPK-Elemente abbilden lassen. Zudem wird kurz ein Transformationsprogramm beschrieben, was die Abbildung implementiert.

1 Einleitung

Mit der zunehmenden Bedeutung von Web-Services steigt auch der Bedarf, neue Dienste aus bestehenden Web-Services zusammenzusetzen. Dieses Zusammensetzen, auch Web-Service-Komposition genannt, kann zur Implementierung von Geschäftsprozessen genutzt werden. Die Business Process Execution Language for Web Services (BPEL4WS bzw. WS-BPEL oder kurz BPEL) [1] ermöglicht die Definition solcher Web-Service-Kompositionen, die von einer dedizierten BPEL-Engine ausgeführt werden können.

Viele Workflow-Sprachen verwenden eine graphbasierte Darstellung, um die logischen Abhängigkeiten zwischen verschiedenen Aktivitäten zu spezifizieren. BPEL hingegen benutzt sogenannte strukturierte Aktivitäten, um den Kontrollfluss zu beschreiben. Zudem besitzt BPEL keine einheitliche graphische Notation, die das Verständnis von BPEL-Prozessdefinitionen erleichtern könnte. Um BPEL-Prozesse einfach zu kommunizieren, sind Abbildungen von BPEL auf andere Prozessmodellierungssprachen erforderlich, die leichter zu erfassen sind.

Als Zielsprache einer solchen Abbildung eignen sich Ereignis-gesteuerte Prozessketten (EPKs) [2] aus verschiedenen Gründen. Einerseits besitzen sie eine standardisierte graphische Notation, die das Verständnis erleichtert. Des Weiteren sind sie auch im betriebswirtschaftlichen Umfeld weit verbreitet. EPK-Modelle eignen sich daher, in BPEL implementierte Prozesse bspw. mit Fachabteilungen abzustimmen. Zudem existiert für EPKs ein standardisiertes Austauschformat in Form von EPML [3], das als Zielformat einer Transformation dienen kann.

2 Abbildung von BPEL auf EPKs

Mit BPEL können Kompositionen von Web-Services definiert werden, die ihrerseits als eigenständige Web-Services angesprochen werden können. BPEL benutzt zur Spezifikation von solchen Kompositionen hauptsächlich fünf Konzepte: *Variablen*, die Prozessdaten vorhalten und SOAP-Nachrichten speichern, *Partner-Links*, die den bilateralen Nachrichtenaustausch zwischen dem Prozess und entfernten Web-Services abstrakt beschreiben, *einfache Aktivitäten*, die Elementaroperationen eines BPEL-Prozesses darstellen, *strukturierte Aktivitäten* zur Spezifikation des Kontrollflusses zwischen einfachen Aktivitäten und Konzepte zur *Behandlung*[1] von Fehlern, Ereignissen und Kompensationen. Für die Abbildung von BPEL auf EPKs sind einfache und strukturierte Aktivitäten von besonderer Bedeutung, da sie den Kontrollfluss spezifizieren.

Das EPML-Zielformat beinhaltet die klassischen EPK-Elemente: *Funktionen* zur Modellierung von Aktivitäten, *Ereignisse* zur Darstellung von Vor- und Nachbedingungen von Funktionen, *Konnektoren* zur Beschreibung von u.a. Parallelverzweigungen und Entscheidungen, hierarchische Funktionen und Prozessschnittstellen für *Sub-Prozess-Aufrufe*. Diese können mit Kontrollflusskanten verbunden werden. Des Weiteren bietet EPML die Möglichkeit, *Organisationseinheiten*, *Anwendungen* und *Datenfelder* und deren Beziehungen zu anderen EPK-Elementen explizit mitzumodellieren.

BPEL unterscheidet sieben verschiedene einfache Aktivitäten. Grundsätzlich werden diese auf einen Block aus einer EPK Funktion und einem anschließenden EPK Ereignis abgebildet, die den Aktivitätstyp in ihrem Namen enthalten. **Invoke:** Diese Aktivität wird auf einen Funktions-Ereignis-Block abgebildet. Der Partner-Link wird auf eine Organisationseinheit gemappt, die für die Ausführung des Services verantwortlich ist. Ein- und Ausgabevariablen erzeugen Datenfelder, die mit der Funktion in Input- und Output-Beziehung stehen. **Receive:** Receive wird ähnlich wie Invoke abgebildet. Der Unterschied besteht darin, dass Receive nur mit einem Datenfeld verbunden ist, das die eingehende Nachricht aufnimmt. **Reply:** Gleichermaßen wie Receive wird die Reply-Aktivität nur mit dem Datenfeld verbunden, dass die ausgehende Nachricht vorhält. **Assign:** Diese Aktivität entspricht einen Funktions-Ereignis-Block, der mit einem Input- und einem Output-Datenfeld verbunden ist. **Throw:** Throw wird auf einen Funktions-Ereignis-Block abgebildet, wobei die Funktion den Fehler in ein Datenfeld schreibt. Nach einem solchen Fehler wird die normale Verarbeitung im aktuellen Scope abgebrochen und die Aktivitäten des Fault-Handlers abgear-

[1] Die Abbildungen der BPEL-Handler werden hier nicht dargestellt. Entsprechend abstrahiert das Invoke-Mapping von internen FaultHandlern.

beitet. **Wait:** Auch Wait entspricht einem Funktions-Ereignis-Block, der mit einem Datenfeld verbunden ist, das die Zeit repräsentiert. **Empty:** Diese Aktivität erzeugt im Gegensatz zu den anderen keinen Funktions-Ereignis-Block. Sie dient in BPEL zum Beispiel dazu, Fehler abzufangen und zu unterdrücken.

In BPEL gibt es fünf strukturierte Aktivitäten, die den Kontrollfluss zwischen einfachen oder geschachtelten strukturierten Aktivitäten beschreiben. **Sequence:** Mit Sequence wird modelliert, dass die als Kindelemente genannten Aktivitäten nacheinander ausgeführt werden. Deshalb erzeugt die Sequenz auch keine eigenen EPK-Elemente. **Switch:** Switch stellt eine Entscheidung dar und wird daher auf einen Block aus XOR-Split und XOR-Join abgebildet. Jedes Case- bzw. Otherwise-Element wird zu einem eigenen Zweig zwischen XOR-Split und -Join. **While:** Die While-Schleife wird in EPKs mit einem XOR-Join gefolgt von einem Funktions-Ereignis-Block zur Prüfung der Abbruchbedingung mit anschließendem XOR-Split modelliert. Von Split kann zu einem weiteren Schleifendurchlauf verzweigt werden, der in den XOR-Join mündet. **Pick:** Pick funkioniert ähnlich wie Switch, jedoch wird beim Pick auf alternative Ereignisse gewartet. **Flow:** Flow wird auf parallele Zweige abgebildet, die zwischen einem AND-Split und einem AND-Join eingeschlossen sind. Zudem kann jede Aktivität innerhalb eines Flows als Source und/oder Target von Kontrollkanten (BPEL Link) dienen. Abbildung 1 stellt diese Transformation dar. Für die Source-Aktivität kann in BPEL eine Transition Condition definiert sein. Wenn diese wahr ist, so wird einerseits mit einem AND-Split die folgende Aktivität des eigenen Zweiges aktiviert. Zudem synchronisiert die Target-Aktivität ihren eigenen Prozesszweig mit der Source über einen OR-Join. Die nicht-lokale Semantik des OR-Joins der EPK repräsentiert exakt die BPEL-Semantik, die mit Dead-Path-Elimination definiert ist.

3 Implementierung

Diese Abbildung von BPEL-Prozessdefinitionen auf EPK-Modelle wurde in XOTcl [4], der objektorientierten Erweiterung von Tcl, unter Nutzung des tDOM-Packets implementiert. Die programmiertechnische Herausforderung bestand darin, die block-orienterte Prozessbeschreibung von BPEL in die graph-basierte Darstellung von EPML zu transformieren. Dabei galt es zum einen die syntaktische Richtigkeit von EPKs sicherzustellen (Alternation von Funkionen und Ereignissen) und zum anderen eindeutige ID-Elemente in EPML zu generieren sowie diese korrekt zu referenzieren.

4 Zusammenfassung und Ausblick

In dieser Beitrag haben wir eine Abbildung von BPEL auf EPKs vorgestellt, die für die Abstimmung von

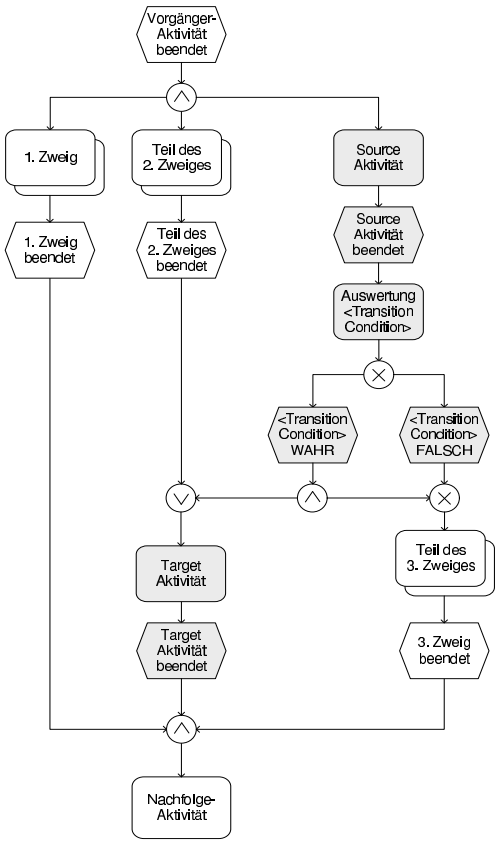

Abbildung 1: EPK-Visualisierung des Flow-Elements.

Prozessmodellen mit Fachabteilungen nützlich ist. Zudem wurde kurz ein Transformationsprogramm beschrieben, was die Abbildung implementiert. Damit ist eine einfache Visualisierung von BPEL-Prozessen als EPK möglich.

Literatur

[1] T. Andrews et al. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, 2003.

[2] G. Keller, M. Nüttgens und A. W. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik Saarbrücken, Saarbrücken, Germany, 1992.

[3] J. Mendling und M. Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). Technical Report JM-2005-03-10, Vienna University of Economics and Business Administration, Austria, 2005.

[4] G. Neumann und U. Zdun. XOTcl, an Object-Oriented Scripting Language. In *Proc. of Tcl2k: The 7th USENIX Tcl/Tk Conference, Austin, Texas, USA*, 2000.