Zur automatischen Ermittlung von Testszenarien aus EPK-Schemata

Oliver Skroch

Wirtschaftsinformatik und Systems Engineering
Universität Augsburg
Universitätsstr. 16, D-86159 Augsburg
oliver.skroch@wiwi.uni-augsburg.de

Abstract: Für funktionale Systemtests großer, komponenten- und serviceorientierter betrieblicher Anwendungssoftware sind durchgängige Testszenarien sinnvoll, die für die wichtigsten, geschäftskritischen betrieblichen Abläufe des Anwenders vorzubereiten und durchzuführen sind. Zur Unterstützung der Ermittlung solcher „higher order" Testfälle wird eine systematische Methode der Extraktion verzweigungsfreier Wege durch EPK-Schemata vorgeschlagen. Um die vorgestellte Methode praktikabel zu machen werden optionale Annahmen zur kombinatorischen Relaxation getroffen. Die Methode ist als Erweiterung der Open-Source Initiative „EPC-Tools" implementiert.

1 Einleitung und Motivation

Im komponenten- und serviceorientierten Entwicklungsparadigma betrieblicher Anwendungssysteme [SGM02; Tu03] kann man fragen, wie fremdbezogene Komponenten bzw. Dienste möglichst früh und effizient überprüft werden [Bo81; We98]. Man geht dabei für die Seite der Komponentenhersteller bzw. Dienstelieferanten davon aus, dass sie die Außensicht auf ihre Komponenten und Dienste spezifizieren, um sie am Markt anzubieten [Tu03]. Man kann weiter annehmen, dass ein Anwender solche Spezifikationen auf ihre Eignung für seine zu automatisierenden betrieblichen Abläufe überprüft, bevor er die entsprechenden Komponenten und Dienste einsetzt bzw. bevor er sie auch nur bezieht. Jedoch können im allgemeinen nicht alle möglichen Abläufe einer zu automatisierenden Domäne gegen alle Eigenschaften der sie unterstützenden Software getestet werden. Das dürfte auch, zumindest bei nicht sicherheitskritischen Systemen, in den allerwenigsten Fällen wirtschaftlich sinnvoll sein.

Damit geht es für den nachfragenden Anwender zunächst darum, die für ihn wichtigsten, geschäftskritischen Abläufe unabhängig von den angebotenen Komponenten bzw. Diensten systematisch zu identifizieren. Sind diese kritischen Abläufe gefunden, dann eignen sie sich gut als Grundlage zur weiteren Ausarbeitung der tatsächlichen Prüfschritte, dabei v.a. zur Definition verzweigungsfreier Szenarien für Positivtests mit unabhängigen Orakeln („sunshine paths" [Sk07, ST07]). Die Frage, die dieser Beitrag zu beantworten versucht ist damit, ob und wie aus einem betrieblichen Ablaufschema

verzweigungsfreie Wege als Grundlage für die Ausarbeitung von Testszenarien durch dieses Ablaufschema extrahiert werden können.

Der weitere Beitrag ist folgendermaßen aufgebaut: in Kapitel 2 werden weitere Arbeiten diskutiert, die mit dem hier vorgestellten Lösungsansatz verwandt sind oder für den Hintergrund und die verwendeten Begriffe eine Rolle spielen; in Kapitel 3 wird die Methode im Detail vorgestellt; Kapitel 4 stellt die dazu entwickelte Werkzeugerweiterung vor; Kapitel 5 schließt den Beitrag mit einer Zusammenfassung und einem Ausblick.

2 Hintergrund, Begriffe, und verwandte Arbeiten

Für die Modellierung von Geschäftsprozessen gibt es heute viele kommerzielle Werkzeuge („workflow management systems"), die verschiedenen methodischen Ansätzen und Notationsgrammatiken folgen (z.B. [AH02; Mu02]). Dieser Beitrag geht von der Ereignisgesteuerten Prozesskette (EPK) [KNS92] aus, weil sie eine besonders intuitive und speziell im deutschsprachigen Raum weit verbreitete, semiformale Notation für die Modellierung betriebliche Abläufe ist, die schon seit einiger Zeit ergiebig erforscht und auch erfolgreich in der Praxis eingesetzt wird (z.B. [Sc02], [DJ05]).

Szenarien können als (Teil-)Modelle betrachtet werden und sind wichtige Bausteine der Softwareentwicklung, sie helfen bei der Erfassung von funktionalen Anforderungen und beeinflussen Design, Implementierung und funktionalen Test [Br00; Su03; CR05]. Bei der funktionalen Prüfung von Software auf Anwenderseite (d.h. Black-Box Systemtest [Be95]) sind szenario-orientierte Vorgehensweisen gut bekannt und werden z.B. bei der Testfallermittlung auf Grundlage von UML Anwendungsfällen („use cases") eingesetzt [OA99; BL02; SZ04].

Prüfprozesse in der Softwareentwicklung werden allgemein etwa in [My79; Ri97; ISO01] diskutiert, für Test und Qualitätssicherung im komponentenorientierten Paradigma etwa in [We98; GTW03; Me03; Vi+03]. Für die Entwicklung – und dabei besonders für den funktionalen Systemtest – von flexiblen, komponenten- und serviceorientierten Anwendungssystemen werden wichtige Vorteile durch die Szenario-Orientierung diskutiert, cf. etwa [EDF96; SZ04].

Mit vollständig formalen Spezifikationen und „kleinen" betrachteten Domänen sind szenario-orientierte Prüfmethoden gut bekannt, z.B. in der technischen Informatik als Spuren („traces") im Rahmen des „model checking", vgl. etwa [CH03; Br+05] oder vertiefend [Ku+05]. Diese Ansätze lassen sich aber nur eingeschränkt auf semiformal und i.a. unvollständig beschriebene, funktional im Vergleich wesentlich größere Modelle betrieblicher Abläufe übertragen. Solche Modelle haben, wie z.B. in [DJ05] gezeigt wurde, eigene Besonderheiten, zumal es letztlich um „higher order" Prüfschritte [My79, S. 103ff] geht, die über rein syntaktische Kriterien hinaus reichen.

Um trotzdem klar abgegrenzte Szenarien als Ausgangspunkt für die Definition weiterer konkreter Prüfschritte in großen und semiformalen EPK-Modellen zu finden, erweist

sich die Idee der „Workflow Patterns“ [Aa+03] (Ablauf- oder Kontrollfluss-Muster) als nützlich. Betrachtet man das Schema der betrieblichen Abläufe analog zum Kontrollfluss eines deterministischen Programms, dann kann man den gerichteten Graph des Ablaufschemas mittels weniger grundlegender Ablaufmuster aus [Aa+03] auf enthaltene verzweigungsfreie Szenarien analysieren. Ein lineares Szenario hat (in Anlehnung an die Spuren der linearen temporalen Logik) von jedem Knoten maximal eine ausgehende Kante in einen Folgeknoten, und das komplexe betriebliche Ablaufschema wird über die Ablaufmuster an den „Verzweigungsstellen“ in einzelne solche linearen Durchläufe zerlegt.

In [Sc00] wird ein Ansatz vorgeschlagen, der eine ähnliche Linearisierungsidee, die dort an Konzepte aus der Netzplantechnik angelehnt wird, nicht erst für die Gewinnung von Testszenarien sondern schon für die Konstruktion der Ablaufmodelle selbst verfolgt. In [MNN05] wird eine (hier nicht erforderliche) Erweiterung der EPK zur Unterstützung aller „Workflow Patterns“ aus [Aa+03] diskutiert.

Für die automatische Ermittlung dieser Szenarien aus den als EPK modellierten Ablaufschemata ist eine vollständig formale Syntax und Semantik für EPK notwendig, wobei es wesentlich um die Formalisierung der logischen Konnektoren (der „Verzweigungsstellen“ im Schema) geht. Dieser Beitrag baut dabei auf den Vorschlägen [Aa99; NR02; GL06] auf, und zusätzlich auf [Ri99] für die Wohlgeformtheit von Split-Join-Paaren im Ablaufschema, sowie auf [Kin06] für die Minimierung der sog. nicht-lokalen EPK-Semantik.

3 Beschreibung der Methode

3.1 Formalisierung von Notation, Syntax und Semantik

Zur graphischen Darstellung der EPK gibt es als Notationssymbole einen gerichteten Kantentyp und fünf unterschiedliche Knotentypen (Abbildung 1). Kantentyp ist der *Kontrollfluss*. Die fünf Knotentypen sind *Ereignis*, *Funktion*, und logischer Konnektor vom konjunktiven Typ („a und b“, *AND*), disjunktiven Typ („entweder a oder b“, *XOR*), und adjunktiven Typ („a oder auch b“, *OR*).

Abbildung 1: EPK Notationssymbole

Ein syntaktisch korrektes EPK-Schema A sei ein Tupel $A = (E,F,C,T,S_0)$ mit den folgenden Eigenschaften:

– E ist eine nichtleere Menge von Ereignisknoten.
– F ist eine nichtleere Menge von Funktionsknoten.
– C ist eine Menge von logischen Konnektoren, bestehend aus paarweise disjunkten Teilmengen C_{AND}, C_{XOR}, C_{OR}.
– E, F, C sind paarweise disjunkt.

– Der Kontrollfluss $T : (N \times N)$ ist eine Übergangsfunktion mit $N = E \cup F \cup C$.
– S_0 ist eine Menge von Startereignissen.

Ein syntaktisch korrektes EPK-Schema sei mit den folgenden Eigenschaften auch sematisch korrekt:

1) $G = (N,T)$ ist ein gerichteter, zusammenhängender und endlicher Graph.
2) Zwischen zwei Knoten existiert maximal eine Kontrollflusskante.
3) Funktionsknoten haben genau eine eingehende und genau eine ausgehende Kontrollflusskante.
4) Ereignisknoten haben maximal eine eingehende und maximal eine ausgehende Kontrollflusskante.
5) Konnektoren haben entweder genau eine eingehende und mehrere ausgehende Kontrollflusskanten (und heißen dann Split), oder mehrere eingehende und genau eine ausgehende Kontrollflusskante (und heißen dann Join).
6) Ereignisknoten sind nur mit Funktionsknoten verbunden, möglicherweise über Konnektoren.
7) Funktionsknoten sind nur mit Ereignisknoten verbunden, möglicherweise über Konnektoren.
8) Es gibt keinen Kreis, der nur aus Konnektoren besteht.
9) Es gibt mindestens ein Startereignis (ohne eingehende Kontrollflusskante) und mindestens ein Endereignis (ohne ausgehende Kontrollflusskante).
10) Nach Ereignisknoten folgt kein *XOR*-Split und kein *OR*-Split im Kontrollfluss (vgl. [KNS92, S. 15]).
11) Jeder Knoten ist von einem Startereignis erreichbar.
12) Von jedem Knoten ist ein Endereignis erreichbar.

Zusätzlich seien semantisch wohlstrukturierte Split-Join-Paare gefordert, so dass der öffnende Split und der schließende Join eines Paares jeweils vom gleichen Typ sind (vgl. vertiefend [Ri99, S. 2]).

3.2 Behandlung der grundlegenden Ablaufmuster

Die musterbasierte Analyse der EPK-Schemata zur Ermittlung verzweigungsfreier Wege erfolgt anhand der drei fundamentalen Modellierungsprimitive Sequenz, Selektion und Iteration, die auch den „Workflow Patterns“ aus [Aa+03] zugrunde liegen.

Sequenz

Die Sequenz ist die fundamentale Idee des Ablaufs von diskreten Einzelschritten nach einander (Abbildung 2). Die gängige semantische Interpretation ist Implikation bzw. Kausalität.

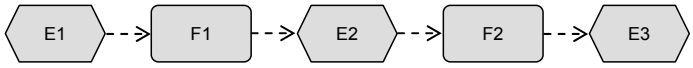

Abbildung 2: Lineare Sequenz

Lineare Sequenzen sind eben die verzweigungsfreie Wege, die das angestrebte Ergebnis der Linearisierung darstellen. Die Linearisierung eines EPK-Schema erzeugt alle möglichen solchen Wege vom Startereignis zum Endereignis. Ein EPK-Schema, das wie in Abbildung 2 nur aus einem verzweigungsfreien Weg besteht, ist bereits linear.

Selektion – Split-Konnektor

Split-Konnektoren besitzen genau eine eingehende und mehr als eine ausgehende Kante (vgl. 3.1, Nr. 9). Die Selektion mit Splits stellt die fundamentale Idee eines sich in der Zukunft verzweigenden Ablaufs dar. Abbildung 3 zeigt die zwei Muster, wobei der Split nach einem Ereignis (linke Seite der Abbildung 3) nur vom Typ *AND*, nicht aber vom Typ *OR* und auch nicht vom Typ *XOR* sein kann (vgl. 3.1, Nr. 10).

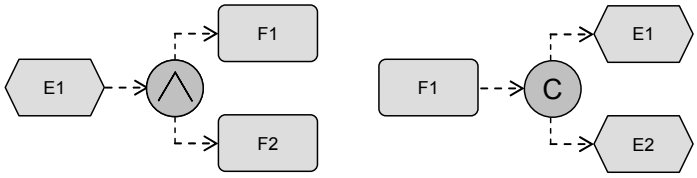

Abbildung 3: Selektion mit Split-Konnektor

Tabelle 1 zeigt die Ergebnisse der Linearisierung dieser Muster aus Abbildung 3, abhängig vom Konnektortyp. Die Muster werden in alle möglichen Abläufe expandiert.

Typ	**Linearisierung nach Ereignis**	**Linearisierung nach Funktion**
AND	(E1, F1, F2), (E1, F2, F1)	(F1, E1, E2), (F1, E2, E1)
XOR	nicht definiert	(F1, E1), (F1, E2)
OR	nicht definiert	(F1, E1), (F1, E2), (F1, E1, E2), (F1, E2, E1)

Tabelle 1: Linearisierung für Split-Konnektor

Gängige semantische Interpretationen für den *AND*-Split ist die Parallelbearbeitung, für den *XOR*-Split die Entscheidung. Der *OR*-Split ist nur schwierig intuitiv zu interpretieren. Er kann als Kombination aus *AND*- und *XOR*-Split behandelt werden, da die logische Semantik eine Kombination von *AND* und *XOR* darstellt.

Selektion – Join-Konnektor

Join-Konnektoren besitzen mehr als eine eingehende und genau eine ausgehende Kante (vgl. 3.1, Nr. 9). Die Selektion mit Joins stellt die fundamentale Idee sich vereinigender Abläufe dar. Daraus folgt, dass durch das jeweilige Schema in der Vergangenheit vor dem Konnektor mehr als ein Weg möglich ist. Abbildung 4 zeigt die zwei Muster.

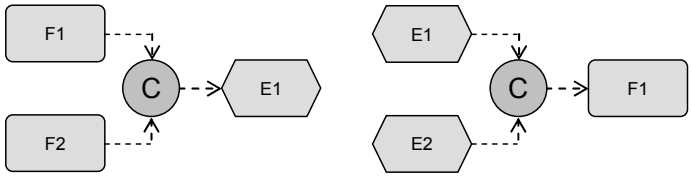

Abbildung 4: Selektion mit Join-Konnektor

Tabelle 2 zeigt die Ergebnisse der Linearisierung dieser Muster aus Abbildung 4, abhängig vom Konnektortyp. Die Muster werden in alle möglichen Abläufe expandiert.

Gängige semantische Interpretationen für den *AND*-Join ist die Synchronisierung, für den *XOR*-Join der Auslöser („trigger"). Der *OR*-Join ist ähnlich wie sein Split-Pendant nur schwierig intuitiv zu interpretieren, er kann aber ebenfalls als Kombination aus *AND*- und *XOR*-Join behandelt werden, da seine logische Semantik eine Kombination von *AND* und *XOR* darstellt.

Typ	**Linearisierung vor Ereignis**	**Linearisierung vor Funktion**
AND	(F1, F2, E1), (F2, F1, E1)	(E1, E2, F1), (E2, E1, F1)
XOR	(F1, E1), (F2, E1)	(E1, F1), (E2, F1)
OR	(F1, E1), (F2, E1), (F1, F2, E1), (F2, F1, E1)	(E1, F1), (E1, F2), (E1, E2, F1), (E2, E1, F1)

Tabelle 2: Linearisierung für Join-Konnektor

Es geht bei der Linearisierung der Join-Muster darum, den nach dem vereinigenden Join nächsten Schritt zu finden. Zwar ist, unabhängig von Typ des Join, der Folgeschritt nach dem Join immer eindeutig (vgl. 3.1, Nr. 5). Das Schema hat vor dem Join aber Verzweigungen, diese sind bis zum Join schon in die möglichen verzweigungsfreien Wege aufgespalten. Betrachtet man also einen einzelnen verzweigungsfreien Weg, dann wird jeder Join auf diesem Weg verzweigungsfrei erreicht. Die Behandlung der Join-Muster zur Linearisierung wäre soweit trivial. Es stellt sich aber die Frage, wie die miteinander zu kombinierenden Split-Join-Sequenzen bestimmt werden, da sie nicht unbedingt eindeutig sind.

Bestimmt werden muss zu jedem Join der entsprechend passende Split, und damit der genaue Nachbereich aller im Schema vorhandenen Splits. Die Forderung nach semantisch wohlstrukturierten Split-Join-Paaren (vgl. 3.1) verlangt, dass im Schema jedem Split ein Join vom gleichen Typ entspricht. Der Nachbereich beginnt mit dem Split-Konnektor selbst und wird entweder auf einem entsprechenden Join-Konnektor geschlossen oder endet mit einem Endereignis. Für einen Split ist der entsprechend passende Join: der unter allen Kindknoten (d.h. allen von Split aus erreichbaren Knoten), mittels Breitensuche (breadth first search), erste erreichbare Join vom gleichen Typ nach mindestens einem Ereignis- oder Funktionsknoten.

Die Linearisierung benötigt hier zur praktischen Handhabbarkeit zusätzlich noch sinnvolle Annahmen zur Einschränkung der zu prüfenden Kombinationen. Es kann für die Bildung von funktionalen Testszenarien von einer ausdrücklichen Betrachtung der unterschiedlichen Ausführungsreihenfolgen abgesehen werden, da angenommen werden kann, dass diese auch nicht modelliert sind. Zusätzlich können, in Anlehnung an die gängige Vorgehensweise des Testens an Randbereichen („boundary value"), bei *OR* mit $n > 2$ Alternativen nur die „Randfälle" $\binom{n}{1}$ und $\binom{n}{n}$ betrachtet werden.

Iteration

Die Iteration ist die fundamentale Idee eines sich gleichartig wiederholenden Ablaufs (Abbildung 5). Die gängige semantische Interpretation ist die Scheife.

Iterationen verwenden nur *XOR*-Konnektoren: die Joins der Iteration müssen offensichtlich disjunktiv sein, und für die entsprechenden Splits ist semantisch der gleiche Typ gefordert (vgl. 3.1).

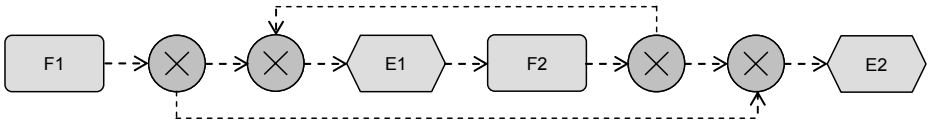

Abbildung 5: Iteration

Tabelle 3 zeigt die Ergebnisse der Linearisierung des Iterationsmusters aus Abbildung 5. Die Iteration wird in alle möglichen Abläufe expandiert.

Durchläufe	**Linearisierung vor Ereignis**
0	(F1, E2) – „oben geprüfte Schleife“
1	(F1, E1, F2, E2)
2	(F1, E1, F2, E1, F2, E2)
...	(F1, E1, F2, E1, F2, ..., E2)

Tabelle 3: Linearisierung der Iteration

Die Linearisierung der Iteration ist also einfach, benötigt aber zur praktischen Handhabbarkeit zusätzlich sinnvolle Annahmen zur Einschränkung der Anzahl der zu prüfenden Durchläufe. In Anlehnung z.B. an [Ri97] erfolgt die funktionale Prüfung mit keinem („oben geprüfte Schleife“), einem, oder $n > 1$ Durchläufen, wobei $n = 2$ gesetzt werden kann.

3.3 Komplexität und Relaxationsannahmen

Zur Bestimmung aller verzweigungsfreien Wege wird eine Erweiterung des BFS Standardalgorithmus („Breadth First Search“, Breitensuche) aus der Graphentheorie verwendet. Dabei ist BFS für die Auffindung aller Wege erweitert (Standard-BFS findet kürzeste Wege), und für die Behandlung der EPK-Konnektoren sowie der Iterationen angepasst.

Für die Zeitkomplexität des Algorithmus gilt $O(N+T)$, da alle möglichen Wege betrachtet werden, wofür jede Kante und jeder Knoten besucht wird und jeder Schritt eine konstante Zeit benötigt (mit Standard-BFS ist das der ungünstigste Fall, hier gilt es immer). Für die Platzkomplexität gilt ebenfalls $O(N+T)$, da alle bisher besuchten Knoten gespeichert werden bis die nächsttiefere Ebene des Graphen erreicht ist.

Die Methode ist damit für größere Probleme nur geeignet, wenn in den betrachteten EPK-Schemata die Anzahl der Knoten N (Ereignisse, Funktionen und Konnektoren) plus die Anzahl der Kanten T (Kontrollflüsse) mit zunehmender Tiefe des Schemas nicht

exponentiell wächst. Allgemein graphentheoretisch ist natürlich von exponentiellem Wachstum auszugehen. Wie aber aus den „No Free Lunch“ Theoremen [WM95; WM97] gefolgert werden kann, ist ein algorithmisches Such- oder Optimierungsverfahren möglichst gut an die untersuchte Problemklasse anzupassen, indem sinnvolle Annahmen zur Problemstellung getroffen werden.

Betrachtet man EPK-Schemata echter betrieblicher Abläufe in der Praxis, sind Splits (einschließlich Iterationen) oft nicht tief verschachtelt, sondern werden häufig ohne viele weitere Verschachtelungen durch entsprechende Joins geschlossen bzw. treffen auf Endereignisse. Stellt man die rein praktische Überlegung an, dass die hier betrachteten betriebliche Abläufe letztlich von Menschen verstanden, geplant, (zumindest teilweise) umgesetzt, verwaltet und verantwortet werden müssen, so ist diese Eigenart nachvollziehbar, und es ist daher auch anzunehmen, dass reale betriebliche Abläufe nicht beliebig komplex werden. Aus dieser Überlegung kann man auf eine nicht exponentielle Zunahme von $N+T$ bei grossen betrieblichen EPK-Schemata ausgehen. Weiterhin kann auch durch die mehrfach vorgeschlagene hierarchische Modellierung betrieblicher Abläufe (z.B. [KM94; Sc00]), bzw. durch die nachträgliche hierarchische Interpretation der Abläufe (z.B. mittels „blocking“ [ST07]), dem Problem der tief verschachtelten Modellteile und damit des exponentiell wachsenden Prüfaufwands begegnet werden. Unter diesen Annahmen für reale EPK-Schemata wäre die Standard-BFS als Grundlage der vorgeschlagenen Methode praktikabel.

Zusätzlich muss die Komplexität der vorgeschlagenen Methode noch hinsichtlich ihrer BFS Erweiterungen, der Behandlung von Iterationen und (nicht-disjunktiven) Konnektoren, eingeschränkt werden – soweit das ihrem letztlichen Ziel, der Erzeugung von Testszenarien für funktionale Systemtests auf Anwenderseite, nicht schädlich ist. Die folgende Analyse hinsichtlich der szenario-orientierten Prüfungsmöglichkeiten in EPK-Schemata führt zu solchen begründeten Relaxationsannahmen.

Sei $n > 1$ die Anzahl der ausgehenden Kontrollflusskanten eines Split, dann vermehrt (nur) ein Split die Anzahl der möglichen verzweigungsfreien Abläufe auf einem Weg im Schema. Bei einem Join wird zunächst der entsprechende Split bestimmt (wobei u.a. gefordert ist, dass Split und Join vom selben Konnektortyp sind), dann werden alle linearen Szenarien für das Split-Join-Paar gebildet. Deren Anzahl hängt vom Konnektortyp ab, wie Tabellen 4 und 5 zeigen.

Split-Join-Typ	**Anzahl der Abläufe im Split-Join-Paar**	
	vollständig	**relaxiert**
AND	$\#L(n, AND) = n!$	$\#LR(n, AND) = 1$
XOR	$\#L(n, XOR) = n$	$\#LR = \#L$

Tabelle 4: Anzahl der Szenarien in Split-Join-Paaren *AND* und *XOR*

Die linke Spalte in Tabelle 4 nennt den Typ des Split-Join-Paares, die linke Seite der rechten Spalte die Anzahl der Abläufe bei vollständiger linearer Expansion, und die rechte Seite der rechten Spalte die Anzahl der Abläufe bei relaxierter linearer Expansion. Für *AND* bedeutet die Relaxation die Nichtberücksichtigung der unterschiedlichen

Ausführungsreihenfolgen (Permutationen). Für *XOR* erscheint eine Relaxation für die spätere Bildung von Testszenarien nicht sinnvoll.

Für *OR* wird zunächst festgestellt, dass es gleichbedeutend mittels *XOR* gefolgt von *AND* modelliert werden kann. Damit ergibt sich eine Abhängigkeit der *OR* Relaxation von der Behandlung des *AND*. Die Anzahl der Szenarien ist in Tabelle 5 dargestellt.

Bei vollständig expandiertem *AND* ergeben sich mit ebenso vollständig expandiertem *OR* alle Variationen der hypergeometrischen Verteilung; mit relaxiertem *OR* werden in der vorgeschlagenen Methode nur die beiden „Randfälle“ $\binom{n}{1}$ und $\binom{n}{n}$ behandelt. Bei relaxiertem *AND* ergeben sich mit vollständig expandiertem *OR* $\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{n}=2^n-1$ Szenarien, und mit ebenfalls relaxiertem *OR* entsprechend der vorgeschlagenen Methode nur $\binom{n}{1}+\binom{n}{n}=n+1$ Szenarien.

Split-Join OR	**Anzahl der Abläufe im Split-Join-Paar**	
	***OR* vollständig**	***OR* relaxiert**
AND vollständig	$\#L(n, OR)=\sum_{k=1}^{n} k!\binom{n}{k}$	$\#LR(n, OR)=n!+1$
AND relaxiert	$\#L(n, OR)=2^n-1$	$\#LR(n, OR)=n+1$

Tabelle 5: Anzahl der Szenarien in Split-Join-Paaren *OR*

Für Iterationen wird davon ausgegangen, dass die funktionale Prüfung mit keinem, einem oder $n > 1$ Durchläufen erfolgt, wobei $n = 2$ gesetzt werden kann (in Anlehnung z.B. an [Ri97]). Die Iteration wird also bei „oben geprüfter Schleife“ nicht berücksichtigt und expandiert ansonsten in zwei verzweigungsfreie Szenarien.

Bei gleichzeitiger Relaxation von *AND* und *OR* und bei der Relaxation von Iterationen scheint also, unter den getroffenen Annahmen zur Ermittlung von Testszenarien, die Komplexität der vorgeschlagenen Methode für reale EPK-Schemata praktikabel.

4 Erweiterung der „EPC Tools“

Die hier vorgeschlagene Methode ist als Erweiterung der Open-Source Initiative „EPC Tools“ [CK05] auf der Eclipse Java IDE implementiert. Der Algorithmus liest in seiner ersten Phase eine EPK ein und konvertiert sie in eine Datenstruktur für Graphen; es wird dabei das EPK Austauschformat EPML [MN06] („EPC Markup Language“) unterstützt. In seiner zweiten Phase durchläuft der Algorithmus diese Datenstruktur und expandiert alle Verzweigungen und Iterationen entsprechend der beschriebenen Methode in einzelne verzweigungsfreie Wege.

Die Implementierung ist in der Lage, zwischen gewähltem Start- und Endknoten korrekte EPKs zu linearisieren. Die implementierten Relaxations-Parameter für die Behandlung von Schleifen, *AND* und *OR* Konnektoren erlauben es, das Verhalten des Algorithmus einzustellen. Die folgenden drei Parameter sind implementiert:

Iterativer Relaxations-Parameter: über diesen Parameter kann die Anzahl der zu testenden Iterationen in den Zyklen des EPK-Schemas eingestellt werden. Sinnvolle Werte sind 0 (nur bei „oben geprüften Schleifen“), 1 und 2.

Konjunktiver Relaxations-Parameter: ist dieser Parameter gesetzt, so werden Split-Join-Paare vom Typ *AND* zu einem einzigen Szenario, mit einer beliebigen Ablaufreihenfolge (Permutation) der parallel zu bearbeitenden Abläufe, expandiert; andernfalls werden $n!$ Szenarien expandiert, jeweils ein Szenario für jede mögliche Reihenfolge.

Adjunktiver Relaxations-Parameter: ist dieser Parameter gesetzt, so werden für $n > 2$ nur wie beschrieben die beiden „Randsituationen“ expandiert, andernfalls werden alle 2^n-1 möglichen Kombinationen expandiert. Dabei hängt die Behandlung der konjunktiven Teilkombinationen jeweils vom konjunktiven Relaxations-Parameter ab, es wird also je nachdem entweder nur ein Szenario mit beliebiger Reihenfolge gebildet, oder faktoriell alle kombinatorisch möglichen Szenarien.

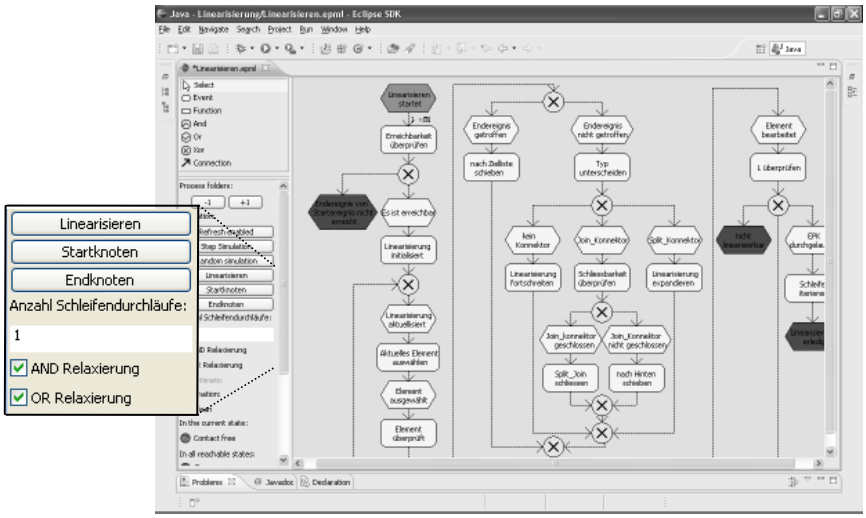

Abbildung 6: Erweiterung der Open-Source Initiative „EPC-Tools“ [CK05]

Die Implementierung ist weder instrumentiert noch auf Laufzeit oder Speicherbedarf optimiert. Unter Verwendung der Relaxations-Parameter wurden mit dieser Implementierung – neben kleineren funktionsprüfenden Beispielen – verschiedene EPK-Schemata aus betrieblichen Referenzmodellen auf einem Standard-PC ohne wahrnehmbare Laufzeit linearisiert.

Da Start- und Endknoten eines Linearisierungslaufs im Schema frei gewählt werden können, können auch verzweigungsfreie Szenarien unterschiedlicher Auflösung in

unterschiedlichen Teilen des Schemas durch entsprechende Anpassung der Parameter-Einstellungen für den jeweiligen Lauf erzeugt werden. Beispielsweise kann ein kleiner Teilprozess, in dem die Ausführungsreihenfolge der parallel modellierten Schritte als kritisch für den späteren Test angesehen wird, mit einem Linearisierungslauf unter eingeschalteter *AND* Kombinatorik analysiert werden.

5 Zusammenfassung und Ausblick

In diesem Beitrag wurde eine Methode vorgeschlagen, mit der unter bestimmten Annahmen zur praktischen Handhabbarkeit alle verzweigungsfreien Wege durch ein korrektes EPK-Schema betrieblicher Abläufe extrahiert werden können. Zur Methode wurde ein entsprechendes Werkzeug als Eclipse-Plugin Erweiterung der Open-Source Initiative „EPC-Tools“ implementiert.

Diese verzweigungsfreien Wege können nach bestimmten Kriterien priorisiert werden um so die wichtigsten, geschäftskritischen Abläufe zu identifizieren. Diese können dann – mit unabhängigen Orakeln versehen – als Grundlage für lineare Positivtest-Szenarien („sunshine paths“) zum funktionalen Black-Box Systemtest im komponenten- und serviceorientierten Entwicklungsparadigma verwendet werden.

Vorteile ergeben sich erstens durch die Möglichkeit, alle zu testenden Abläufe zu kennen und sie damit bewusst priorisieren zu können, sowie zweitens durch die Möglichkeit, Testorakel aufzustellen, die vom Entwicklungsprozess auf Herstellerseite unabhängig sind. Dadurch wird die Definition aussagefähiger Testmetriken auch bei „higher order“ Tests auf Domänenebene möglich.

Literaturverzeichnis

[Aa99] van der Aalst, W. M. P.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology 41 (1999) 639-650

[Aa+03] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., Barros, A. P.: Workflow Patterns. Distributed and Parallel Databases 14 (2003) 5-51

[AH02] van der Aalst, W. M. P., van Hee, K. M.: Workflow Management: Models, Methods, and Systems. The MIT Press, Cambridge, Ma, USA (2002)

[Be95] Beizer, B.: Black-Box Testing: Techniques for Functional Testing of Software and Systems. John Wiley & Sons, New York, NY, USA (1995)

[BL02] Briand, L., Labiche, Y.: A UML-Based Approach to System Testing. Journal of Software and Systems Modeling 1 (2002) 10-42

[Bo81] Boehm, B. W.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ, USA (1981)

[Br00] de Bruin, H.: Scenario-Based Analysis of Component Compositions. In: Generative and Component-Based Software Engineering: Second International Symposium (GCSE2000): Revised Papers. 2000/10/09-12, Erfurt (2000) 129-146

[Br+05] Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems: Advanced Lectures. Springer, Berlin (2005)

[CH03] Choi, Y., Heimdahl, M.: Model Checking Software Requirement Specifications using Domain Reduction Abstraction. In: 18th IEEE International Conference on Automated Software Engineering (ASE 2003). 2003/10/06-10, Montreal, Canada (2003) 314-317

[CK05] Cuntz, N., Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation. In: van der Aalst, W., Benatallah, B., Casati, F., Curbera, F. (Hrsg.): Business Process Management. LNCS 3649, Springer, Berlin (2005) 398-493

[CR05] Carroll, J. M., Rosson, M. B.: Cases as Minimalist Information. In: 38th Hawaii International Conference on System Sciences (HICSS2005) Track 1. 2005/01/03-06, Big Island, Hi, USA (2005) 40b–

[DJ05] van Dongen, B. F., Jansen-Vullers, M. H.: Verifcation of SAP Reference Models. In: van der Aalst, W. M. P., Benatallah, B., Casati, F., Curbera, F. (Hrsg.): Business Process Management. LNCS 3649, Springer, Berlin (2005) 464-469

[EDF96] Ecklund, E., Delcambre, L., Freiling, M.: Change cases: use cases that identify future requirements. In: 11th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA96). 1996/10/06-10, San Jose, Ca, USA (1996) 342-358

[GL06] Gruhn, V., Laue, R.: Validierung syntaktischer und anderer EPK-Eigenschaften mit PROLOG. In: 5. Workshop Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK2006). 2006/11/30-2006/12/01, Wien, Österreich (2006) 69-84

[GTW03] Gao, J. Z., Tsao, H.-S. J., Wu, Y.: Testing and quality assurance for component-based software. Artech House, Boston, Ma, USA (2003)

[ISO01] International Organization for Standardization: International Standard ISO/IEC 9126-1: Software engineering - Product quality - Part 1: Quality model. ISO/IEC, Genf, Schweiz (2001)

[Kin06] Kindler, E.: On the semantics of EPCs: A framework for resolving the vicious circle. Data & Knowledge Engineering 56 (2006) 23-40

[KM94] Keller, G., Meinhardt, S.: SAP R/3 Analyzer: Optimierung von Geschäftsprozessen auf Basis des R/3-Referenzmodells. SAP AG, Walldorf (1994)

[KNS92] Keller, G., Nüttgens, M., Scheer, A.-W.: Sematische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89. Universität des Saarlandes, Saarbrücken (1992) 2-30

[Ku+05] Kugler, H., Harel, D., Pnueli, A., Lu, Y., Bontemps, Y.: Temporal Logic for Scenario-Based Specifications. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS05). 2005/04/04-08, Edinburgh, UK (2005) 445-460

[Me03] Meyer, B.: The Grand Challenge of Trusted Components. In: 25th International Conference on Software Engineering (ICSE2003). 2003/05/03-10, Portland, Or, USA (2003) 660-667

[MN06] Mendling, J., Nüttgens, M.: EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC). Information Systems and e-Business Management 4 (2006) 245-263

[MNN05] Mendling, J., Neumann, G., Nüttgens, M.: Yet Another Event-Driven Process Chain. In: van der Aalst, W. M. P., Benatallah, B., Casati, F., Curbera, F. (Hrsg.): Business Process Management. LNCS 3649, Springer, Berlin (2005) 428-433

[Mu02] zur Muehlen, M.: Workflow-based Process Controlling: Foundation, Design and Application of Workflow-driven Process Information Systems. Logos, Berlin (2002)

[My79] Myers, G. J.: The Art of Software Testing. John Wiley & Sons, New York, NY, USA (1979)

[NR02] Nüttgens, M., Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen (Promise2002). 2002/10/09-11, Potsdam (2002) 64-77

[OA99] Offutt, A. J., Abdurazik, A.: Generating Tests from UML Specifications. In: The Unified Modeling Language: Beyond the Standard: Second International Conference (UML99). 1999/10/28-30, Fort Collins, Co, USA (1999) 416-429

[Ri97] Riedemann, E. H.: Testmethoden für sequentielle und nebenläufige Software-Systeme. Teubner, Stuttgart (1997)

[Ri99] Rittgen, P.: Modified EPCs and Their Formal Semantics. In: Frank, U., Hampe, F. (Hrsg.): Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 19. Universität Koblenz-Landau, Koblenz (1999) 1-11

[Sc00] Schimm, G.: Generic Linear Business Process Modeling. In: Liddle, S. W., Mayr, H. C., Thalheim, B. (Hrsg.): Conceptual Modeling for E-Business and the Web. LNCS 1921, Springer, Berlin (2000) 31-39

[Sc02] Scheer, A.-W.: ARIS - vom Geschäftsprozess zum Anwendungssystem. Springer, Berlin (2002)

[SGM02] Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. 2nd Edition, Addison-Wesley, London, UK (2002)

[Sk07] Skroch, O.: Validation of Component-based Software with a Customer Centric Domain Level Approach. In: 14th Annual IEEE Conference and Workshop on the Engineering of Computer Based Systems (ECBS07), Doctoral Symposium. 2007/03/26-29, Tucson, Az, USA (2007) 459-466.

[ST07] Skroch, O., Turowski, K.: Reducing Domain Level Scenarios to Test Component-based Software. Journal of Software 2 (2007) 64-73.

[Su03] Sutcliffe, A. G.: Scenario-Based Requirements Engineering. In: 11th IEEE International Conference on Requirements Engineering (RE2003). 2003/09/08-12, Monterey, Ca, USA (2003) 320-329

[SZ04] Strembeck, M., Zdun, U.: Scenario-based Component Testing Using Embedded Metadata. In: First International Workshop on Software Quality (SOQUA2004) and Workshop Testing Component-Based Systems (TECOS2004). 2004/09/27-30, Erfurt (2004) 31-45

[Tu03] Turowski, K.: Fachkomponenten: Komponentenbasierte betriebliche Anwendungs-systeme. Shaker, Aachen (2003)

[Vi+03] Vincenzi, A. M. R., Maldonado, J. C., Delamaro, M. E., Spoto, E. S., Wong, W. E.: Component-Based Software: An Overview of Testing. In: Cechich, A., Piattini, M., Vallecillo, A. (Hrsg.): Component-Based Software Quality: Methods and Techniques. LNCS 2693, Springer, Berlin (2003) 99-127

[We98] Weyuker, E. J.: Testing Component-Based Software: A Cautionary Tale. IEEE Software 15 (1998) 54-59

[WM95] Wolpert, D. H.. Macready, W. G.: No Free Lunch Theorems for Search. Working Papers of the Santa Fe Institute # 95-02-010, The Santa Fe Institute, Santa Fe, NM, USA (1995)

[WM97] Wolpert, D. H.. Macready, W. G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation 1 (1997) 67-82