EPK-Validierung zur Modellierungszeit in der bflow* Toolbox

Volker Gruhn[1], Ralf Laue[1], Heiko Kern[2] und Stefan Kühne[2]

[1] Angewandte Telematik / e-Business, Universität Leipzig
Klostergasse 3, 04109 Leipzig
{gruhn, laue}@ebus.informatik.uni-leipzig.de

[2] Betriebliche Informationssysteme, Universität Leipzig
Johannisgasse 26, 04103 Leipzig
{kern, kuehne}@informatik.uni-leipzig.de

Abstract: Dieser Beitrag stellt den Prototyp eines EPK-Modellierungswerkzeugs vor, der Verfahren zur Suche in Graphen nutzt, um Fehler in EPK-Modellen zu identifizieren. Dieses Werkzeug hat gegenüber bekannten Ansätzen zwei Vorzüge: Zum einen ist es nicht notwendig, den (oft sehr großen) Zustandsraum aller möglichen Abläufe in einem Modell zu berechnen. Zum Zweiten kann unser Ansatz auch auf noch nicht vollständig fertiggestellte Modelle angewendet werden. Der Modellierer wird sofort zur Modellierungszeit über mögliche Probleme sowie deren Ursachen informiert und erhält unmittelbare Vorschläge zur Beseitigung der Probleme.

1 Einleitung

Verfahren zur Überprüfung der Korrektheit von EPKs und anderen Geschäftsprozessmodellen sind seit langer Zeit Gegenstand der Forschung. Wynn et al. [WVvE] schreiben in einem kürzlich veröffentlichten Artikel, dass „die Verifizierung von Prozessen so weit ausgereift ist, dass sie in der Praxis eingesetzt werden kann." Dem ist zwar zuzustimmen, es ist jedoch auch festzustellen, dass gängige Validierungstechniken den Geschäftsprozessmodellierer noch nicht auf die bestmögliche Weise unterstützen. Der Grund für diese Aussage liegt darin, dass formale Validierungsmethoden in der Regel erst angewendet werden, nachdem ein Modell fertiggestellt wurde. So können verschiedene Ansätze, die eine EPK zum Zwecke der Validierung in ein Petrinetz übertragen, nicht auf noch nicht komplett fertiggestellte Modelle angewendet werden.

In diesem Beitrag stellen wir einen Validierungsansatz vor, der dem Modellierer bereits während der Modellierung Rückmeldungen zu möglichen Problemen im Modell gibt. Wir beschreiben eine prototypische Implementierung dieses Ansatzes im Open-Source-Modellierungswerkzeug *bflow* Toolbox*[1]. Mit dieser können nicht nur typische Kontrollflussfehler (wie Deadlocks oder Lifelocks) gefunden werden, sondern auch Probleme, die

[1] http://www.bflow.org/

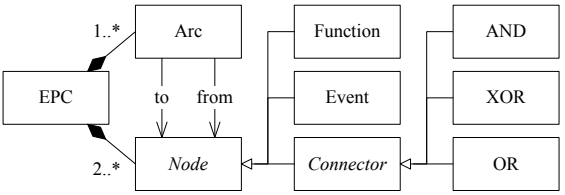

Abbildung 1: EPK-Metamodell

man unter der Bezeichnung „schlechter Modellierungsstil“ zusammenfassen könnte. Fehler werden im Modell durch direktes Markieren der betroffenen Modellelemente angezeigt. Dem Modellierer werden Möglichkeiten aufgezeigt, die gefundenen Fehler zu korrigieren. Mit Hilfe einer erweiterbaren Regelsprache können eigene Validierungsregeln (etwa für organisationsweite Stilregeln) hinzugefügt werden.

In Anlehnung an die aus der modernen Softwareentwicklung bekannten Begriffe Continuous Compilation und Continuous Testing bezeichnen wir unseren Ansatz als *Continuous Validation*. Damit soll hervorgehoben werden, dass eine Überprüfung des Modells ständig während des Modellierungsprozesses im Hintergrund abläuft und der Modellierer eine unmittelbare Rückmeldung über gefundene Probleme erhält.

2 Begriffe und Definitionen

2.1 Ereignisgesteuerte Prozessketten

Wir verwenden in diesem Beitrag ereignisgesteuerte Prozessketten mit den üblichen in der Literatur eingeführten Syntaxregeln [NR02]. Mögliche syntaktische Variationsmöglichkeiten (wie das Erlauben von aufeinanderfolgenden Funktionen ohne ein Ereignis dazwischen) sind ohne Einfluss auf die Grundideen unseres Ansatzes. Dank der Möglichkeit, die Regelbasis selbst anzupassen, können sie je nach Bedarf berücksichtigt werden.

Wir verwenden zur Beschreibung von EPKs das in Abb. 1 gezeigte Metamodell. EPKs sind nach diesem Metamodell endliche gerichtete zusammenhängende Graphen, bestehend aus Knoten *(Node)* und Kanten *(Arc)*. Knoten unterteilen sich in Funktionen *(Function)*, Ereignisse *(Event)* und die verschiedenen Arten von Konnektoren *(Connector)*.

Die Begriffe *Zustand* einer EPK (worunter wir eine Markierung von Kontrollflusspfeilen verstehen), *Start- und Endzustand*, *Folgezustand* und *Zustandsübergangsrelation* verwenden wir wie in [NR02, Kin04, Men07] eingeführt.

2.2 Kontrollflussfehler

In [van99], definiert van der Aalst den Begriff der *Soundness* für EPK-Modelle. Dieses Korrektheitskriterium für Geschäftsprozessmodelle erfordert drei Eigenschaften:

1. In jedem Zustand einer EPK, der von einem Startzustand aus erreicht werden kann, muss es möglich sein, einen Endzustand zu erreichen (also einen Zustand, für den es in der die jeweilige Semantik beschreibenden Zustandsübergangsrelation keine Folgezustände gibt). *(option to complete)*
2. Wenn ein Zustand keinen Folgezustand hat, dürfen in diesem Zustand nur Endereignisse markiert sein. *(proper completion)*
3. Es gibt in der EPK kein Modellelement, für die es keinen Ablauf der EPK von einem Start- zu einem Endzustand gibt, in dem dieses Element markiert wird. *(no needless elements)*

Ist die Soundness-Eigenschaft verletzt, bedeutet dies, dass die EPK fehlerhaft ist. Ein typisches Beispiel ist eine Deadlock-Situation, in der ein XOR-Split mehrere Kontrollflusspfade eröffnet, die später von einem AND-Join zusammengeführt werden.

3 Ansätze zur Validierung des EPK-Kontrollflusses

Das erste Problem, das bei der Validierung von EPK-Modellen zu überwinden ist, besteht darin, dass die Modellierungssprache EPK ursprünglich eingeführt wurde, ohne formell die Semantik eines Modells zu definieren [KNS92]. Daher besteht der erste Schritt gängiger Validierungsansätze darin, ein EPK-Modell in einen Formalismus mit wohldefinierter Semantik zu transformieren. Genaugenommen wird erst durch diese Transformation die Bedeutung des Modells definiert. Verschiedene Autoren zeigen, dass Petrinetze ein geeigneter Formalismus sind, um die Semantik von EPK-Modellen geeignet zu beschreiben [van99, vvV05, Men07][2].

Nachdem ein EPK-Modell in ein Petrinetz überführt wurde, steht das komplette Arsenal an Petrinetz-Validierungswerkzeugen zur Verfügung, um Eigenschaften des Modells zu überprüfen. Wie verschiedene Autoren zeigten [LSW97, vJVVv07, Wyn06, Men07], sind diese Werkzeuge geeignet, um Fehler in EPKs aufzudecken.

Petrinetz-basierte Ansätze weisen aber häufig zwei Nachteile auf: Zum einen ist das Ergebnis einer Validierung oft nur die Tatsache, dass ein Fehler (z. B. Deadlock) vorliegt, ohne die Modellelemente benennen zu können, die für diesen Fehler „verantwortlich“ sind.

Selbst in den Fällen, in denen das Validierungswerkzeug das im Petrinetz erkannte Problem wieder in die Problemdomäne der analysierten EPK „zurückübersetzt“, können Informationen zu Fehlern verloren gehen.

[2] Die Liste der zitierten Arbeiten ist bei Weitem nicht vollständig. Eine ausführliche Übersicht findet sich in [vJVVv07] und [Men07].

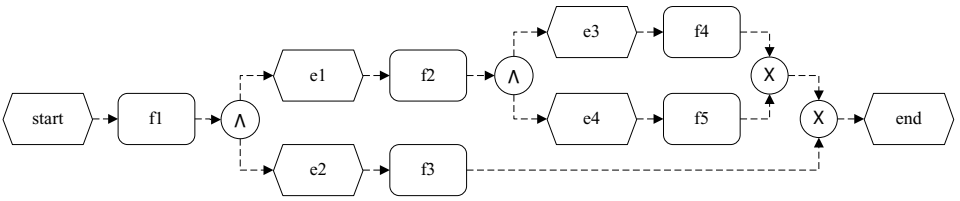

Abbildung 2: Zwei verschachtelte Kontrollflussblöcke mit Modellierungsfehler durch die Paarung AND-Split/XOR-Join

Dies soll in Abb. 2 illustriert werden. In diesem Modell ergeben sich aus einer unsachgemäßen Kombination eines AND-Splits mit einem XOR-Join offensichtlich zwei Fehler – einer im inneren AND-XOR-Kontrollblock und ein weiterer im äußeren AND-XOR-Kontrollblock.

Nehmen wir an (wie dies in der Literatur zur Semantik von EPK-Modellen in der Regel der Fall ist), dass ein XOR-Join „blockiert“, wenn an mehr als einem seiner Eingänge ein Kontrollfluss ankommt. Dann leitet im gezeigten Modell der innere XOR-Join auf Grund dieser Blockade nie einen Kontrollfluss an seinen ausgehenden Pfeil weiter. Daraus folgt aber, dass am äußeren XOR-Join (in der Abbildung rechts) immer nur am unteren eingehenden Pfeil ein Kontrollfluss ankommt. Ein Analysewerkzeug, das den gesamten Zustandsraum aller möglichen Abläufe im Modell untersucht, wird daher für den äußeren AND-Split/XOR-Join-Block nie einen Fehler feststellen.[3]

Ein zweiter Nachteil von Petrinetz-basierten Validierungsverfahren ist, dass sie häufig komplett modellierte EPKs als Eingabe voraussetzen. Eine während des Modellierungsprozesses im Entstehen begriffene EPK, die aus mehreren noch nicht miteinander verbundenen Teilen besteht, kann in diesem Falle nicht untersucht werden.

Eine weitere in der Literatur beschriebene Methode zur Suche nach Kontrollflussfehlern in EPKs und anderen Modellierungssprachen sind Reduktionsverfahren (erstmals beschrieben in [SO00]). Die Grundidee dieser Reduktionsverfahren besteht darin, dass in einem Modell sukzessive wohlstrukturierte Teile (wie beispielsweise eine Kombination aus öffnendem AND-Split und schließendem AND-Join ohne weitere Konnektoren dazwischen) entfernt werden. Es kann gezeigt werden, dass ein Entfernen dieser wohlstrukturierten Teile die Soundness-Eigenschaft des Modells nicht verändert [Wyn06]. Ist es also möglich, durch wiederholtes Anwenden von Reduktionsregeln das ursprüngliche Modell zu einem einzigen Knoten zu reduzieren, ist nachgewiesen, dass dieses Modell die Soundness-Eigenschaft erfüllt. Ist eine solche Reduktion jedoch nicht möglich, kann die Frage, ob das Modell sound ist, nicht entschieden werden. Somit liefern Reduktionsverfahren auf die Frage „Gibt es Kontrollfluss-Fehler im Modell?“ entweder die Antwort „Ja“ oder aber die Antwort „Das kann nicht entschieden werden“. Wünschenswert wäre jedoch eine Antwort, die entweder „Nein“ oder „Ja, und der erkannte Fehler lässt sich wie folgt beheben...“ heißt.

[3] Etwas anderes gilt für Ansätze wie [van98], wo strukturelle Eigenschaften des Petrinetzes untersucht werden, die direkte Rückschlüsse auf Fehler in der EPK ermöglichen.

Einen Schritt in diese Richtung geht Mendling [Men07]. In dieser Arbeit werden nicht nur Reduktionsregeln für korrekte Modellteile betrachtet, sondern auch Modellelemente entfernt, die typische Fehlersituationen aufweisen. Diese Fehler werden dann gemeldet und eindeutige Hinweise auf die Modellelemente, die zum Fehler führen, gegeben. An Hand des SAP-Referenzmodells zeigte Mendling, dass sich auf diese Weise für die meisten EPKs entweder Soundness feststellen lässt oder sich Fehler lokalisieren lassen. In den Fällen, wo sich mit Hilfe von Reduktionsregeln kein Fehler im Modell finden lässt, jedoch das Modell auch nicht auf einen einzelnen Knoten reduziert werden konnte, verwendete Mendling die oben diskutierten Petrinetz-basierten Verfahren, um zumindest zu einer Ja/Nein-Entscheidung zur Soundness-Eigenschaft zu gelangen.

Die in der Arbeit von Mendling [Men07] beschriebenen Reduktionsregeln für typische Fehlersituationen betrachteten wir als einen Ausgangspunkt zum Aufstellen unserer Validierungsregeln.

Unser Beitrag wurde ebenso von dem in [vDMvdA06] veröffentlichten Ansatz beeinflusst. Dieser überträgt bekannte Begriffe und Eigenschaften von Petrinetzen (etwa den Begriff der *Handles* in einem Petrinetz [ES89, van98])) in die Anwendungsdomäne der EPKs. Zwischen EPK-Modellelementen werden Beziehungen der Art „nachdem Modellelement X durchlaufen wurde, muss immer ein Modellelement aus der Menge $\{Y, Z\}$ durchlaufen werden“ betrachtet. Diese Beziehungen werden in [vDMvdA06] *causal footprints* genannt. Sie dienen als Grundlage zur Erkennung von Fehlermustern (z. B. Deadlocks oder Lifelocks). Mit Hilfe der causal footprints ist es möglich, direkt die Modellelemente zu identifizieren, die für einen Fehler verantwortlich sind. Ebenso lässt sich das Verfahren auch auf noch unvollständige Modelle anwenden. Allerdings ist der in [vDMvdA06] beschriebene Ansatz noch nicht für den Einsatz in der Praxis geeignet. Der erste Grund hierfür, dass keine OR-Konnektoren betrachtet werden, wiegt weniger schwer, da eine Erweiterung des Ansatzes von [vDMvdA06] um OR-Konnektoren ohne größere Schwierigkeiten möglich ist. Eine weit größere Einschränkung ist, dass die Berechnung aller causal footprints deutlich zu zeit- und speicherintensiv ist, da die Zahl der zu berücksichtigenden Beziehungen schnell sehr groß wird.

Ein weiterer heuristischer Ansatz ist in [VVL07] vorgestellt. Hier werden Workflow-Graphen in Single-Entry/Single-Exit-Regionen (also Teilmodelle, die nur einen eingehenden und einen ausgehenden Kontrollflusspfeil haben) unterteilt, was eine effiziente Untersuchung der Soundness-Eigenschaft für diese Teilmodelle ermöglicht.

Dem in diesem Beitrag verwendeten Ansatz am ähnlichsten ist das von Awad und Puhlmann [AP08] beschriebene Verfahren, das eine Abfragesprache namens BPMN-Q dazu nutzt, Fehlermuster in BPMN-Modellen zu finden. Es umfasst jedoch noch deutlich weniger Fehlermuster; insbesondere werden Modelle, die einen OR-Konnektor enthalten, nicht betrachtet.

4 Validierung zur Modellierungszeit

4.1 Validierungsansatz

In modernen Entwicklungsumgebungen für Programmiersprachen, wie beispielsweise der Eclipse-Umgebung, ist es heutzutage eine Standardfunktion, bereits während der Eingabe des Programms, die Syntax auf ihre Richtigkeit zu prüfen und entsprechende Fehler direkt anzuzeigen. Diese Form der Unterstützung wollen wir mit unserem Werkzeug auf die Geschäftsprozessmodellierung übertragen. Die Modellierung wird hierbei im Hintergrund ständig überprüft und dem Modellierer gegebenenfalls sofortige Rückmeldungen über Schwächen und Inkonsistenzen in möglicherweise auch unvollständigen Modellen angezeigt. Dadurch wird der typische Modellierungsprozess mit sequentieller Modellierung, Validierung und evolutionärer Weiterentwicklung soweit wie möglich verkürzt zu einem Modellierungsprozess mit integrierter Validierungsunterstützung. Darüber hinaus wird in dem hier vorgestellten Validierungsansatz nicht nur eine Syntaxüberprüfung vorgenommen, sondern es werden auch semantische und pragmatische Aspekte berücksichtigt.

Die zu entwickelte Validierungsunterstützung soll auf drei Prinzipien basieren.

1. Die Validierungsregeln sollen modular ausdrückbar und leicht lesbar sein, um Anpassbarkeit und Erweiterbarkeit zu gewährleisten. Hierfür eignen sich deklarative Validierungsregeln, weil diese leicht zu beschreiben sind und neue Regeln hinzugefügt werden können, ohne die bereits existierenden zu beeinflussen.

2. Der Validierungsansatz soll direkt auf den Eingabemodellen arbeiten, um möglichst den Nachteil von nicht-lokalisierten Fehlernachrichten zu vermeiden.

3. Die Validierungslösung soll nahtlos in ein Modellierungswerkzeug integriert werden können. Fehlerursachen und mögliche Verbesserungsvorschlage sollen direkt im Modell angezeigt werden.

Die Umsetzung dieser drei Prinzipien im Kontext der Geschäftsprozessmodellierung resultiert in prozessspezifischen Validierungsregeln und dazugehörigen Hilfsfunktionen, die von diesen Regeln verwendet werden. Da sich ereignisgesteuerte Prozessketten und andere Geschäftsprozessmodellierungssprachen durch Graphen darstellen lassen, stellen diese Hilfsfunktionen Graph-Berechnungsfunktionen, wie „Gibt es einen Pfad zwischen zwei Knoten?“ oder „Geht jeder Pfad zwischen zwei Knoten durch ein anderen bestimmten Knoten?“ bereit. Weitere grundlegende Funktionen, die typischerweise in diesen Validierungsregeln verwendet werden, beziehen sich auf Modellelemente, wie „Ist dieses Element ein Startelement?“ oder „Ersetze ein Element durch ein anderes Element“, und mengenorientierte Operationen, wie „Selektiere alle Startereignisse einer EPK“ oder „Berechne die Schnittmenge aus allen Nachfolgern eines Elements und den Vorgängern eines anderen Elements“.

4.2 Implementierung

Der vorgestellte Validierungsansatz ist als eine Erweiterung des Open-Source-EPK-Modellierungswerkzeugs bflow* Toolbox implementiert. Die bflow* Toolbox verwendet als Basistechnologien das Eclipse Modeling Framework (EMF)[4] und das Graphical Modeling Framework (GMF)[5]. Für die Implementierung der angestrebten Validierungsunterstützung wird eine deklarative Validierungssprache und eine Modell-zu-Modell-Transformationssprache benötigt, um Anfragen auf Modelle auszudrücken und mögliche Fehler in Form einer Modell-zu-Modell-Transformation zu korrigieren. Im technischen Raum EMF stellen die Werkzeuge openArchitectureWare (oAW)[6] und Epsilon[7] solche Funktionalitäten zur Verfügung. Da oAW neben seinen Modellverarbeitungsmöglichkeiten auch eine Integration in GMF bietet, wurde oAW als Implementierungstechnologie gewählt.

Zur Beschreibung von Validierungsregeln wird die von oAW zur Verfügung gestellte Sprache *Check* verwendet. Ein einfache Check-Regel, die überprüft, ob ein Join gleichzeitig ein Split ist, ist in Listing 1 als Beispiel dargestellt. Eine Check-Regel beginnt mit der Definition eines Kontexts, der durch ein Element aus dem Metamodell (konkret einer EClass) festgelegt wird. Die Anwendung der Regel wird somit auf das Metamodellelement bzw. dessen Instanzen eingeschränkt. Diese Einschränkung kann durch eine optionale Bedingung in Form einer If-Anweisung verstärkt werden. Das darauf folgende Schlüsselwort ERROR signalisiert, dass es sich bei der verletzten Regel, um einen Fehler handelt. Eine weitere Fehlerkategorie, die eine Warnung an den Modellierer darstellt, wird durch WARNING eingeleitet. Nach diesen Schlüsselwörtern kann eine Nachricht spezifiziert werden, die dann in der Modellierungsumgebung eingeblendet wird. Nach dieser Nachricht muss hinter einem Doppelpunkt ein Ausdruck angegeben werden, der ein Boolean-Wert zurückliefern muss. Dieser Boolean-Ausdruck beschreibt die Zusicherung, die für valide Modelle gelten muss.

Listing 1: Beispiel einer Check-Regel

```
// Connectors are either splits or joins
context epc::Connector
    if this.isJoin()
    ERROR "Connector is a split and \
    a join as well. ..." :
    !this.isSplit();
```

Listing 2: Beispiel einer XTend-Funktion

```
// Is a connector a join−connector
Boolean isJoin(Element e) :
    epc::Connector.isInstance(e)
    && e.incomingArcs().size > 1;
```

Der Restriktions- und Zusicherungsausdruck in Listing 1 bezieht sich auf zwei separate in XTend definierte Funktionen. XTend ist eine weitere in oAW enthaltene funktionale Programmiersprache, um Modell-zu-Modell-Transformationen zu implementieren. Die Definition der Funktion *isJoin()*, die in Listing 1 verwendet wird, ist in Listing 2 gezeigt.

Die definierten Check-Regeln arbeiten auf EMF-Modellen. Um mögliche Fehlernachrichten, die durch die Verletzung bestimmter Regeln erzeugt wurden, im Editor anzuzeigen,

[4] http://www.eclipse.org/modeling/emf/
[5] http://www.eclipse.org/modeling/gmf/
[6] http://www.openarchitectureware.org/
[7] http://www.eclipse.org/gmt/epsilon/

wird der von oAW bereitgestellte GMF-Adapter verwendet. Dadurch ist es möglich, Modellelemente, die an der Verletzung einer Regel beteiligt sind, zu markieren und entsprechende Fehlernachrichten mit Verbesserungsvorschlägen anzuzeigen. Auf diese Weise erhält der Modellierer eine lokalisierte Rückmeldung.

Die verwendeten Technologien sind unabhängig von der EPK-Modellierung in der bflow* Toolbox und können auch zur Validierung von anderen Modelltypen eingesetzt werden. Lediglich die Validierungsregeln sind abhängig vom Modelltyp. Weiterhin sind die zugrunde liegenden Prinzipien des Validierungsansatzes nicht auf die beschriebenen Technologien beschränkt. Sie können auch in anderen EPK-Modellierungswerkzeugen, wie der ARIS Design Platform von IDS Scheer[8] mit ihrer Makrosprache angewendet werden.

5 Typische Modellierungsfehler

5.1 Syntax-Fehler

Sehr typische Fehler in EPKs sind syntaktische Fehler. Die Überprüfung auf syntaktische Korrektheit bereitet wenig Schwierigkeiten. Da einige syntaktische Einschränkungen, wie das Verbot von Zyklen zwischen Konnektoren, nicht im EPK-EMF-Metamodell festgelegt werden können, erfordert dies zusätzliche Regeln [NR02, GL06]. Mit dem vorgestellten Ansatz lässt sich dies durch die Definition einfacher Regeln leicht realisieren.

Abb. 3(a) zeigt einen Syntax-Fehler, bei dem ein Konnektor gleichzeitig ein Join und Split ist. Dieser Fehler wird durch die Check-Regel in Listing 1 erkannt. Ein weiterer Syntax-Fehler ist, dass ein Ereignis oder eine Funktion mehr als eine ein- oder ausgehende Kante hat (siehe Abb. 3(b)). Solche Fehler sind nicht ungewöhnlich: Sie traten beispielsweise in 14 von 604 Modellen des SAP-Referenzmodells auf.

(a) (b)

Abbildung 3: Konnektor mit mehr als einem eingehenden und mehr als einem ausgehenden Pfeil (a), Funktion mit mehr als einem eingehenden Pfeil (b)

5.2 Konfligierende Konnektoren

Deadlocks und Synchronisierungs-Fehler in einer EPK resultieren daraus, dass der Typ eines Split-Konnektors nicht dem Typ des zugehörigen Join-Konnektors entspricht. Wird

[8] http://www.ids-scheer.com

beispielsweise der Kontrollfluss durch ein XOR-Split in zwei alternative Pfade aufgeteilt und später durch ein AND-Join wieder zusammengeführt, führt dies zu einem Deadlock, da der AND-Join auf die Fertigstellung beider Kontrollflusspfade wartet.

Zu beachten ist, dass der oben verwendete Begriff des „zugehörigen Join-Konnektors“ für nicht wohlstrukturierte Modelle zunächst einer Definition bedarf. Zu diesem Zwecke führen wir die Relation *match* wie folgt ein:

Ein Split s und ein Join j werden als zusammengehörig betrachtet (symbolisiert durch die Relation $match(s, j)$) genau dann, wenn es zwei gerichtete Pfade von s nach j gibt, deren einzige gemeinsame Elemente s und j sind.

Falls es keinen „Einsprung in“ oder „Aussprung aus“ dem mit einem XOR-Split s beginnenden und mit einem AND-Join j endenden Kontrollflussblock gibt, ist das Modell in jedem Falle fehlerhaft, und zwar unabhängig von weiteren auf dem Pfad von s nach j liegenden Modellelementen. Dabei definieren wir die Begriffe Einsprung und Aussprung wie folgt:

Sei s ein Split und j ein Join, für den $match(s, j)$ zutrifft. Der Kontrollflussblock zwischen s und j weist keine Ein- und Aussprünge auf ($seseMatch(s, j)$ für *Single-Entry-Single-Exit*), gdw. die folgenden Bedingungen zutreffen:

1. Jeder Pfad von s zu einem Endereignis enthält j.
2. Jeder Pfad von einem Startereignis zu j enthält s.
3. Jeder Pfad von s zu s enthält j.
4. Jeder Pfad von j zu j enthält s.

Diese Definitionen ermöglichen das Finden von Kontrollflussfehlern, wo der Typ des Splits sich vom Typ des zugehörigen Joins unterscheidet. Von den in [Men07] gefunden 178 Fehlern fallen 44 in diese Kategorie. Abb. 4 zeigt einen solchen offensichtlichen Fehler, der aus dem Nicht-Übereinstimmen zwischen dem Typ des XOR-Splits (links) und dem Typ des AND-Joins resultiert. Die beiden XOR-Konnektoren im Inneren zeigen, dass die Regeln in Listing 3 auch Fehler in nicht wohlstrukturierten Modellen finden können.

Listing 3: Kontrollflussproblem: XOR-Split beginnt einen Block und AND-Join schließt ihn

```
// XOR−AND−Mismatch
context iepc::Connector if (this.isAndJoin())
    ERROR "Mismatched XOR−split ..."
    this.allPredessesors().notExists(p| p.isXorSplit() && p.seseMatch(this));
```

Bei einer Kombination aus XOR-Split und OR-Join liegen die Dinge anders: Der OR-Join wartet auf alle eingehenden Kontrollflüsse. Wird am davorliegenden XOR-Split nur einer der ausgehenden Pfade aktiviert, heißt das, dass der OR-Join auf den Abschluss dieses einen Pfades wartet und die Ausführung dann problemlos fortsetzt. Da auf diese Weise kein tatsächlicher Fehler entsteht, wird diese Situation bei den meisten Validierungsansätzen (eine Ausnahme ist [Wyn06]) nicht beachtet. Unser Werkzeug gibt dem Modellierer den Hinweis, dass der OR-Join zu einem XOR-Join geändert werden sollte, was die Lesbarkeit des Modells erhöhen kann.

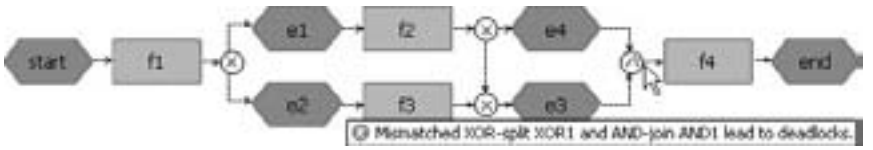

Abbildung 4: Synchronisationsproblem am AND-Join

Listing 4: Check-Regel für ein durch einen (X)OR-Split hervorgerufenes Synchronisationsproblem am AND-Join

```
context epc::Connector if this.isAndJoin() // An AND−Join J
    ERROR "AND−split might not get control" :
    // has no S with S is connected to J and
    this.allPredessesors().notExists(S|
        // S is an XOR−split or an OR−split
        (S.isXorSplit() || S.isOrSplit())
        // and S has a successor SSucc not conntected to J
        && S.successors().exists(SSucc|
            SSucc != this && SSucc.isNotConnectedTo(this))
        // and J has a predecessor JPred not connected from S
        && this.precessors().exists(JPred|
            JPred != S && S.isNotConnectedTo(JPred)) );
```

5.3 Synchronisation-Probleme in AND-Joins

Abb. 5 zeigt einen verbreiteten Fehlertyp, der in [Men07] am häufigsten gefunden wurde (102 von 178 entdeckten Fehlern). Wenn der obere am XOR-Split ausgehende Pfad durchlaufen wird, entsteht ein Deadlock am AND-Join.[9]

Listing 4 zeigt die entsprechende Check-Regel, um dieses Problem zu erkennen. Bei Ausführung der Regel wird eine Fehlernachricht erzeugt, die in Abb. 5 dargestellt ist.

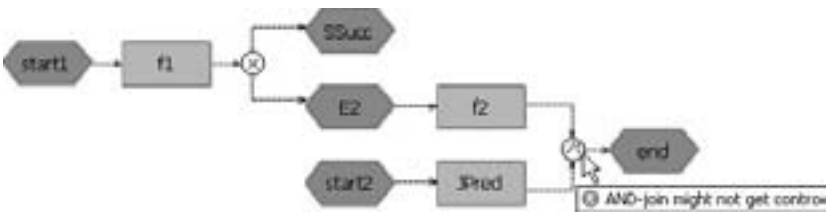

Abbildung 5: Synchronisationsproblem am AND-Join

[9] In der Praxis würden wir nicht alle Modelle, die dieses Muster enthalten, als fehlerhaft betrachten, insbesondere wenn der in Listing 4 als JPred bezeichnete Knoten ein Startereignis ist. Diese Diskussion führt jedoch über die Zielstellung dieses Beitrags hinaus.

5.4 Unternehmensweite Stilregeln

Die Regelmenge in unserem Werkzeug ist leicht erweiterbar. Es ist beispielsweise denkbar, dass ein Unternehmen EPKs zu ausführbaren BPEL-Modellen über eine Modelltransformation überführen möchte, welche wohlstrukturierte Modelle verlangt. In diesem Fall können unstrukturierte EPKs durch unternehmensweite Stilregeln verboten werden.

Solche Anpassungen können in unserem Werkzeug leicht durch die Erweiterung der bestehenden Regelmenge realisiert werden. Ein Beispiel für ein solche mögliche Regel ist in Abb. 6 dargestellt. Hier soll nur nach Eintreten des Ereignisses e1 die Funktion f2 ausgeführt werden. Dies entspricht einer einfachen If-Anweisung ohne Else-Zweig. Da durch die direkte Kontrollflusskante zwischen den Konnektoren kein Ereignis modelliert ist, könnte dieser Pfad missverständlich auch so interpretiert werden, dass er in jedem Falle durchlaufen werden kann. Um diese Fehlinterpretation zu vermeiden, sollte auch in diesem Zweig ein Ereignis notiert werden. Mit einer einfachen Regel kann dieses Fehlermuster gefunden und das Einfügen der Negation von e1 im oberen Zweig empfohlen werden.

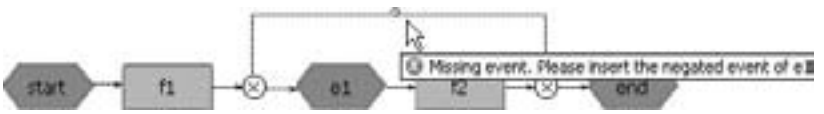

Abbildung 6: Fehlendes Ereignis zwischen XOR-Split und XOR-Join (mit Verbesserungsvorschlag)

6 Validierung

Material für eine erste Validierung unserer Korrektheitsregeln lieferte der Anhang von Mendlings Promotionsschrift [Men07]. Er enthält das Ergebnis der Soundness-Überprüfung von 604 EPK-Modellen des SAP-Referenzmodells. Durch die Verwendung von Reduktionsregeln wurden (wie in Abschnitt 3 beschrieben) 178 Fehlermuster in 90 EPK-Modellen gefunden. Unser Werkzeug fand 176 dieser Fehlermuster (sowie zahlreiche weitere, die durch die Anwendung von Reduktionsregeln nicht entdeckt wurden). Die zwei von unserem Werkzeug nicht gefundenen Fehler beziehen sich auf Schleifen, in denen der Einstieg in die Schleife mit einem OR-Join statt eines XOR-Joins modelliert war. Nach der in [Men07] verwendeten Semantikdefinition sind diese Modelle nicht sound. Unserer Auffassung nach handelt es sich bei diesen Fällen jedoch eher um eine Eigenwilligkeit der Semantikdefinition in [Men07] als um einen tatsächlichen Fehler im Modell. Andere Semantiken (wie die in [Kin04] beschriebene Fixpunktsemantik) sehen solche Modelle nicht als fehlerhaft an.

In weiteren 57 Modellen im Anhang von [Men07] konnten die verwendeten Reduktionsregeln weder einen Fehler finden noch das Modell zu einem einzigen Modellelement reduzieren. Für diese Modelle prüfte Mendling die Soundness-Eigenschaft mit Hilfe eines Petrinetz-Analysewerkzeuges. Für 36 der Modelle wurde auf diese Weise gezeigt, dass sie

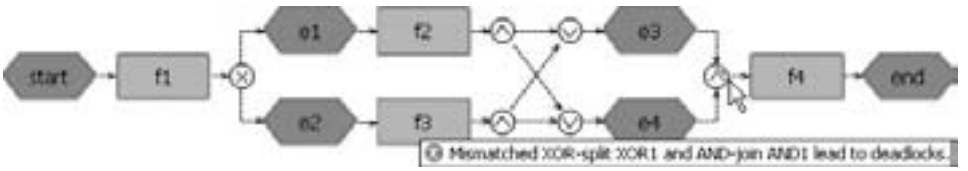

Abbildung 7: Diese EPK ist sound, unser Werkzeug meldet jedoch ein Problem.

nicht sound sind, und unser Werkzeug fand einen oder mehrere Fehler in *allen* diesen 36 Modellen. Die verbleibenden 21 EPKs erwiesen sich als sound. Unser Werkzeug lieferte nur für eine dieser EPKs eine Meldung über einen vermeintlichen Fehler.

Für jede der in [Men07] beschriebenen Fehlerklassen konnten wir eine Regel aufstellen und implementieren, die Fehler dieser Art findet. Ebenso konnten wir feststellen, dass alle im Beitrag von Koehler und Vanhatalo [KV07] beschriebenen Fehlermuster, gewonnen „aus hunderten reellen Prozessmodellen, die mit verschiedenen Werkzeugen erstellt wurden“ [KV07] durch unsere Regeln abgedeckt werden.

7 Zusammenfassung und weitere Forschungsziele

Die im vorigen Abschnitt durchgeführte Validierung zeigte, dass wir mit wenigen implementierten Regeln bereits in der Lage sind, nahezu alle Kontrollflussfehler in EPK-Modellen zu finden. Wir möchten aber an dieser Stelle die heuristische Natur unseres Ansatzes betonen. Die in den Editor integrierten Überprüfungen sollen vor allem ein schnelles Resultat liefern. Sie sollen nicht die in Abschnitt 3 beschriebenen exakten Validierungsverfahren ersetzen: Es kann nicht mit Sicherheit davon ausgegangen werden, dass ein Modell, für das unser Ansatz keine Fehler meldet, sound ist. Ebenso gelingt es, Beispiele zu konstruieren, in denen unsere Regeln fälschlicherweise einen Fehler melden. Ein Beispiel ist in Abb. 7 gezeigt. Obwohl das Modell sound ist, wird durch unsere Regeln ein Problem am AND-Join angezeigt. Ebenso wird nicht gemeldet, dass die OR-Joins durch XOR-Joins ersetzt werden sollten. Wir haben im Sinne einer schnellen Rechenzeit bewusst darauf verzichtet, solche eher exotischen Fälle bei der Formulierung unserer Regeln zu beachten.[10]

Eine Validierung unseres Werkzeug-Prototyps mit einer größeren Menge von EPK-Modellen aus dem SAP-Referenzmodell zeigte, dass bereits eine überschaubare Zahl von Validierungsregeln den überaus größten Teil von Fehlern in EPKs finden kann. Da unser Ansatz ohne Erzeugung des gesamten Zustandsraumes für mögliche Abläufe einer EPK auskommt, können diese Prüfungen sehr schnell erfolgen und während des Modellierens ständig im Hintergrund ablaufen. Die Tatsache, dass auf diese Art und Weise die Validierung zur Modellierungszeit stattfinden kann und somit der Modellierer noch vor Fertigstellung des Modells über mögliche Probleme und Verbesserungsmöglichkeiten in-

[10] Im Übrigen sind wir der Auffassung, dass der Modellierer eines Modells in Abb. 7 durchaus gut darin beraten ist, sein Modell zu überdenken, da es schwer verständlich sein dürfte. Insofern kann eine Problemmeldung durch unser Werkzeug hier sogar gute Dienste leisten.

formiert wird, sehen wir als Hauptvorteil unseres Werkzeuges. Wir nahmen uns hierbei das UML-Werkzeug ARGO/UML [RR00] zum Vorbild, dass UML-Modellierer durch ständige Modellüberprüfungen im Hintergrund mit Hinweisen auf Verbesserungsmöglichkeiten unterstützt. Bei der Darstellung möglicher Probleme im Arbeitsbereich des Editors legten wir Wert darauf, dass das Signalisieren erkannter Fehlermuster so geschieht, dass die Probleme klar erkennbar dargestellt werden, ohne die Aufmerksamkeit des Modellierers von der Tätigkeit des Modellierens abzulenken [McF02].

Eine Richtung für unsere zukünftige Forschung wird es sein, sich auch mit Modellen auseinanderzusetzen, die zwar sound (und somit von rein technischem Standpunkt aus betrachtet korrekt) sind, für die es jedoch Modellierungsalternativen gibt, die einfacher verständlich sind [GL07].

Abschließend sei bemerkt, dass sich die Grundgedanken unseres Ansatzes auch auf andere Geschäftsprozessmodellierungssprachen wie BPMN übertragen lassen.

Literatur

[AP08] Ahmed Awad und Frank Puhlmann. Structural Detection of Deadlocks in Business Process Models. In Witold Abramowicz und Dieter Fensel, Hrsg., *BIS*, Jgg. 7 of *Lecture Notes in Business Information Processing*, Seiten 239–250. Springer, 2008.

[ES89] Javier Esparza und Manuel Silva. Circuits, handles, bridges and nets. In *Applications and Theory of Petri Nets*, Seiten 210–242, 1989.

[GL06] Volker Gruhn und Ralf Laue. Validierung syntaktischer und anderer EPK-Eigenschaften mit PROLOG. In Markus Nüttgens, Frank J. Rump und Jan Mendling, Hrsg., *EPK 2006, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 5. Workshop der Gesellschaft für Informatik e.V. (GI)*, Seiten 69–84, 2006.

[GL07] Volker Gruhn und Ralf Laue. Good and Bad Excuses for Unstructured Business Process Models. In *Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007)*, 2007.

[Kin04] Ekkart Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In *Business Process Management*, Seiten 82–97, 2004.

[KNS92] G. Keller, M. Nüttgens und A.W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, (89), 1992.

[KV07] Jana Koehler und Jussi Vanhatalo. Process anti-patterns: How to avoid the common traps of business process modeling, Part 1 - Modelling control flow. *IBM WebSphere Developer Technical Journal*, (10.4.), 2007.

[LSW97] Peter Langner, Christoph Schneider und Joachim Wehler. Relating Event-driven Process Chains to Boolean Petri Nets. *Report*, (9707), Dezember 1997.

[McF02] Daniel C. McFarlane. Comparison of Four Primary Methods for Coordinating the Interruption of People in Human-Computer Interaction. *Human-Computer Interaction*, 17:63–139, 2002.

[Men07] Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. Dissertation, Vienna University of Economics and Business Administration, 2007.

[NR02] Markus Nüttgens und Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, Seiten 64–77, 2002.

[RR00] Jason E. Robbins und David F. Redmiles. Cognitive support, UML adherence, and XMI interchange in Argo/UML. *Information & Software Technology*, 42(2):79–89, 2000.

[SO00] Wasim Sadiq und Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, June 2000.

[van98] Wil M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[van99] Wil M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.

[vDMvdA06] B.F. van Dongen, J. Mendling und W.M.P. van der Aalst. Structural Patterns for Soundness of Business Process Models. *EDOC*, Seiten 116–128, 2006.

[vJVVv07] Boudewijn F. van Dongen, M.H. Jansen-Vullers, H. M. W. Verbeek und Wil M.P. van der Aalst. Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. *Comput. Ind.*, 58(6):578–601, 2007.

[VVL07] Jussi Vanhatalo, Hagen Völzer und Frank Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. *Service-Oriented Computing – ICSOC 2007*, Seiten 43–55, jan 2007.

[vvV05] Boudewijn F. van Dongen, Wil M.P. van der Aalst und H. M. W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In *CAiSE*, Seiten 372–386, 2005.

[WVvE] Moe Thandar Wynn, H. M. W. Verbeek, Wil M.P. van der Aalst und David Edmond. Business Process Verification - Finally a Reality! *Business Process Management Journal*, to appear.

[Wyn06] Moe Thandar Wynn. *Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins*. Dissertation, Queensland University of Technology, Brisbane, 2006.