

**Markus Nüttgens, Frank J. Rump, Jan Mendling  
(Hrsg.)**

## **EPK 2006**

### **Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten**

5. Workshop der Gesellschaft für Informatik e.V. (GI)  
und Treffen ihres Arbeitskreises „Geschäftsprozessmanagement  
mit Ereignisgesteuerten Prozessketten (WI-EPK)“

30. November - 01. Dezember 2006 in Wien

Proceedings

## **Veranstalter**

veranstaltet vom GI-Arbeitskreis "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" der GI-Fachgruppe WI-MobIS (FB-WI) in Kooperation mit der GI-Fachgruppe EMISA (FB-DBIS) und der GI-Fachgruppe Petrinetze (FB-GInf).

Prof. Dr. Markus Nüttgens (Sprecher)  
Universität Hamburg  
Email: markus.nuettgens@wiso.uni-hamburg.de

Prof. Dr. Frank J. Rump (stellv. Sprecher)  
FH Oldenburg/Ostfriesland/Wilhelmshaven  
Email: rump@informatik-emden.de

## **Sponsor**

Qualysoft GmbH (Wien)

EPK 2006 / Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Hrsg.:  
Markus Nüttgens, Frank J. Rump., Jan Mendling – Wien 2006

© Gesellschaft für Informatik, Bonn 2006

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

## Vorwort

Ereignisgesteuerte Prozessketten (EPK) haben sich in der Praxis als Beschreibungsmittel für betriebliche Abläufe etabliert. Mit dem Aufbau der Arbeitsgruppe "Formalisierung und Analyse Ereignisgesteuerter Prozessketten (EPK)" im Jahre 1997 wurde ein erster Schritt unternommen, einen organisatorischen Rahmen für Interessenten und Autoren wesentlicher Forschungsarbeiten zu schaffen und regelmäßige Arbeitstreffen durchzuführen (Organisatoren: Markus Nüttgens, Andreas Oberweis, Frank J. Rump). Im Jahr 2002 wurden die Arbeiten der "informellen" Arbeitsgruppe in den GI-Arbeitskreis "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" der GI-Fachgruppe WI-MobIS (FB-WI) überführt und inhaltlich erweitert. Der 1. Workshop „EPK 2002“ fand im November 2002 in Trier statt, der 2. Workshop „EPK 2003“ im Oktober 2003 in Bamberg im Vorfeld der 11. Fachtagung „MobIS 2003“, der 3. Workshop „EPK 2004“ in Luxemburg im Rahmen der GI-Fachtagung „EMISA 2004“, der 4. Workshop „EPK 2005“ in Hamburg statt.

Der Arbeitskreis soll Praktikern und Wissenschaftlern als Forum zur Kontaktaufnahme, zur Diskussion und zum Informationsaustausch dienen. Die Aktivitäten des Arbeitskreises werden unter der Internetadresse <http://www.epk-community.de> dokumentiert (aktuell: 250 Mitglieder).

Der vorliegende Tagungsband enthält 11 vom Programmkomitee ausgewählte und auf dem Workshop präsentierte Beiträge. Jeder Beitrag wurde innerhalb der Kategorien Fachbeiträge und Diskussionsbeiträge zweifach begutachtet.

Die Beiträge decken ein breites Spektrum zur Spezifikation und Anwendung Ereignisgesteuerter Prozessketten (EPK) ab:

- EPK Konzepte
- EPKs und Werkzeuge
- EPKs und andere Formalismen
- Qualitätsaspekte von EPKs

Der Tagungsband wurde ausschließlich in digitaler Form publiziert und ist im Internet frei verfügbar (CEUR Workshop Proceedings).

Wir danken der Wirtschaftsuniversität Wien für die Bereitstellung der Räumlichkeiten und der Qualysoft GmbH (Wien) für die finanzielle Unterstützung und den Autorinnen und Autoren und den Mitgliedern des Programmkomitees für die Beiträge zur Realisierung des Workshops.

Hamburg, Emden und Wien im November 2006

Markus Nüttgens  
Frank J. Rump  
Jan Mendling

## **Programmkomitee**

Thomas Allweyer, Fachhochschule Kaiserslautern  
Jörg Becker, Universität Münster  
Jörg Desel, Katholische Universität Eichstätt  
Andreas Gadatsch, Fachhochschule Bonn-Rhein-Sieg  
Ekkart Kindler, Universität Paderborn  
Peter Loos, Institut für Wirtschaftsinformatik / DFKI Saarbrücken  
Jan Mendling, Wirtschaftsuniversität Wien  
Markus Nüttgens, Universität Hamburg (Vorsitz)  
Andreas Oberweis, Technische Universität Karlsruhe  
Michael Rebstock, Fachhochschule Darmstadt  
Michael Rosemann, Queensland University of Technology  
Frank J. Rump, Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven  
Oliver Thomas, Institut für Wirtschaftsinformatik / DFKI Saarbrücken

## **Organisation**

Jan Mendling, Wirtschaftsuniversität Wien

## Inhaltsverzeichnis

### *Fachbeiträge*

<b>O. Thomas, M. Fellmann</b> Semantische Integration von Ontologien und Ereignisgesteuerten Prozessketten .....	7
<b>J. Mendling, W. v. d. Aalst</b> Towards EPC Semantics based on State and Context.....	25
<b>O. Thomas, T. Dollmann</b> Fuzzy-EPK-Modelle: Attributierung und Regelintegration .....	49
<b>V. Gruhn, R. Laue</b> Validierung syntaktischer und anderer EPK-Eigenschaften mit PROLOG .....	69
<b>O. Kopp, T. Unger, F. Leymann</b> Nautilus Event-driven Process Chains: Syntax, Semantics, and their mapping to BPEL .....	85
<b>P. Barborka, L. Helm, G. Köldorfer, J. Mendling, G. Neumann, B. v. Dongen, E. Verbeek, W. v.d. Aalst</b> Integration of EPC-related Tools with ProM .....	105
<b>C. Simon, J. Freiheit, S. Olbrich</b> Using BPEL processes defined by Event-driven Process Chains .....	121

### *Diskussionsbeiträge*

<b>D. Lübke</b> Transformation of Use Cases to EPC Models.....	137
<b>S. Denne</b> Verifying Properties of (Timed) Event Driven Process Chains by Transformation to Hybrid Automata .....	157
<b>H. Störrle</b> A Comparison of (e)EPCs and UML 2 Activity Diagrams.....	177
<b>V. Gruhn, R. Laue, F. Meyer</b> Berechnung von Komplexitätsmetriken für ereignisgesteuerte Prozessketten.....	189



# Semantische Integration von Ontologien und Ereignisgesteuerten Prozessketten

Oliver Thomas, Michael Fellmann

Institut für Wirtschaftsinformatik (IW<sub>i</sub>)  
im Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI)  
Universität des Saarlandes  
Stuhlsatzenhausweg 3, Geb. D3 2, D-66123 Saarbrücken  
{oliver.thomas|michael.fellmann}@iwi.dfki.de

**Abstract.** Dieser Beitrag beschreibt eine Erweiterung der Ereignisgesteuerten Prozesskette (EPK), mit deren Hilfe die in natürlicher Sprache formulierte Semantik der Bezeichner von EPK-Modellelementen durch formale Konzepte einer Ontologie repräsentiert werden kann. Hierfür wird ein mehrschichtiger Ansatz entwickelt, der eine Ontologieebene, eine Metadatenebene sowie eine Modellebene umfasst. Durch den vorgestellten Ansatz, der am Beispiel der Web Ontology Language (OWL) illustriert wird, können die Suche und Navigation in EPK-Modelldatenbanken verbessert, eine fortgeschrittene semantische Validierung von EPK-Modellen ermöglicht sowie die Ausführbarkeit der Prozessmodelle erleichtert werden.

## 1 Semantik in EPK-Modellen

Sowohl die alltagssprachliche als auch die wissenschaftliche Verwendung des Begriffs „Semantik“ sind nicht einheitlich. In den verschiedenen Lebens- und Wissenschaftsbereichen haben sich unterschiedliche Semantikdefinitionen entwickelt. Dies führt insbesondere in der Wissenschaft bei der Verwendung des Semantikbegriffs zu Kommunikations- und Verständnisproblemen. Während im alltagssprachlichen Sinne mit dem Begriff der Semantik häufig auf die Bedeutung oder den Inhalt eines Wortes oder Satzes hingewiesen wird, stützt sich das wissenschaftliche Verständnis von Semantik häufig auf die Begriffsbildung der Sprachwissenschaft (Linguistik) [z. B. Dürr06]. Innerhalb dieser Disziplin bezeichnet die Semantik (auch: Bedeutungslehre) dasjenige sprachwissenschaftliche Teilgebiet, das sich mit dem Sinn und der Bedeutung von Sprache beziehungsweise sprachlichen Zeichen befasst, mit anderen Worten: die Lehre von den Bedeutungen und von der Beziehung der Zeichen zum gemeinten Gegenstand.

Überträgt man dieses Verständnis im Allgemeinen auf Modellierungssprachen sowie im Speziellen auf die Prozessbeschreibungssprache Ereignisgesteuerte Prozesskette (EPK) und auf die mit Hilfe der Sprache explizierten EPK-Modelle, so kann unter der Semantik eines Prozessmodells die Beziehung zwischen den Elementen des Modells (Zeichen) und einem existierenden oder neu zu schaffenden betrieblichen Geschäftsprozess (Gegenstandsbereich) verstanden werden. Diese Auffassung von Semantik entspricht im Wesentlichen derjenigen, die von Keller, Nüttgens und Scheer bereits im Ursprungspapier

„Semantische Prozeßmodellierung auf der Grundlage ‚Ereignisgesteuerter Prozessketten (EPK)‘“ verwendet wurde [KeNS92]. Das Adjektiv „semantisch“ gebrauchten die Autoren insbesondere, um auf die Bedeutung der Modellierung von Abläufen aus einer fachlichen, d. h. nicht-technischen, betriebswirtschaftlichen Perspektive hinzuweisen.

Bei genauerem Hinsehen zeigt sich allerdings, dass zwei verschiedene Arten der Semantik für EPK-Modelle unterschieden werden müssen: eine über das Metamodell der EPK bzw. mit logisch-mathematischen Methoden definierte formale Semantik der EPK-Sprache (im Sinne der exakten Bedeutung dieser künstlichen Sprache) und eine über die Bezeichner der EPK-Modellelemente definierte Semantik, die an die natürliche Sprache gebunden ist.

Die formale Semantik der EPK wird zum einen durch die festgelegte Bedeutung ihrer Sprachkonstrukte, wie z. B. Ereignisse, Funktionen und Konnektoren bestimmt, zum anderen mit Hilfe einer abstrakten Syntax, die unabhängig von der EPK-Notation vorgibt, welche Beziehungen deren Sprachkonstrukte eingehen können. Diese Beziehungen können entweder zwischen den Sprachkonstrukten der EPK selbst in Form eines Kontrollflusses bestehen oder zwischen Sprachkonstrukten der EPK und denen anderer Modellierungssprachen der ARIS-Methode (z. B. Organigramm, Fachbegriffsmodell oder Leistungsbaum). Aus der Sicht der formalen Semantik sind individuelle Modellelemente austauschbare Platzhalter, eine Bedeutung resultiert ausschließlich aus der Art der Verbindung der Elemente, mithin also im systemtheoretischen Sinne aus der strukturalen Form des Modells [Wies59, S. 12]. Untersuchungen zur Semantik der EPK haben sich bislang hauptsächlich auf diese formale Semantik konzentriert [LaSW98; Aals99; NüRu02; Kind06; RoAa06].

Gleichwohl die Diskussion der formalen Semantik zum Verständnis und zur Anwendung der EPK aus modellierungssprachlicher Sicht notwendig erscheint, bleibt sie unvollständig in Bezug auf eine exakte Spezifikation der Semantik individueller Modellelemente. Dies rührt daher, dass ein wesentlicher Teil der Semantik eines individuellen Modellelementes auch an dessen Bezeichner gebunden ist, der mit Hilfe der natürlichen Sprache vom Modellkonstrukteur formuliert wird. Dieser Umstand wurde bisher nur wenig beachtet, obwohl die Wichtigkeit einer exakten Spezifikation der Semantik auf Modellelementebene bereits im Ursprungspapier von 1992 betont wurde. Hierin argumentieren Keller, Nüttgens und Scheer unter anderem, dass „im Rahmen der betrieblichen Informationsmodellierung [...] die eindeutige Definition des durch die Syntax repräsentierten semantischen Inhalts eines Informationsobjekts eine besondere Rolle [spielt]“ [KeNS92, S. 8]. Diese Forderung sollte allerdings nicht nur für Informationsobjekte gelten, sondern auch auf weitere Konstrukte der EPK wie Funktionen und Ereignisse übertragen werden (so kann z. B. eine Funktion „Auftrag prüfen“ unterschiedliche Aufgaben beinhalten, je nachdem, ob es sich um die Bearbeitung von Entwicklungsaufträgen, Bestellaufträgen oder Fertigungsaufträgen handelt).

Insgesamt ist somit ein wesentlicher Teil der Semantik eines EPK-Modells an die natürliche Sprache gebunden, die mit ihren Mehrdeutigkeiten ein hohes Maß an Interpretationsspielräumen zulässt. Solange ein Modell nur von einem Individuum erstellt und gelesen wird, ist dies weniger problematisch. Werden jedoch Modelle verschiedener Model-

lierer zusammengeführt, durchsucht und übersetzt, oder soll die in den Modellen enthaltene Semantik automatisch validiert und zur Konfiguration eines Informationssystems herangezogen werden, ist eine klar definierte Semantik eines jeden Modellelementes erforderlich.

Diese Problemstellung kann durch eine Verknüpfung der Elemente eines EPK-Modells mit Konzepten aus einer Ontologie gelöst werden. Der vorliegende Beitrag beschreibt die dazu erforderlichen Schritte und Werkzeuge. Er ist im weiteren Verlauf wie folgt strukturiert: Zunächst werden in Abschnitt 2 Ontologien und Ontologiekonstruktionen für das semantische Geschäftsprozessmanagement skizziert. Nach diesen Einführungen wird in Abschnitt 3 detailliert die semantische Erweiterung von EPK-Modellen beschrieben. Abschnitt 4 behandelt die Potenziale einer IT-Unterstützung der semantischen Geschäftsprozessmodellierung. Der Beitrag schließt mit der Analyse verwandter Arbeiten in Abschnitt 5 und der Diskussion der Ergebnisse in Abschnitt 6.

## 2 Ontologien für das semantische Geschäftsprozessmanagement

### 2.1 Prozessmodellierung im Kontext des semiotischen Dreiecks

Die aus der Verwendung der natürlichen Sprache zur Bezeichnung von EPK-Modellelementen resultierenden Probleme lassen sich mit Hilfe des Semiotischen Dreiecks verdeutlichen. In seiner Grundform wurde dies bereits 1923 veröffentlicht [OgRi23]. Die zentrale Idee des semiotischen Dreiecks ist eine Trennung von Ding, Begriff und Symbol. Begriffe fungieren gleichsam als die „Griffe“, die Dinge (Objekte) begreifbar machen. Sie werden von Symbolen (Bezeichnungen, Benennungen) aktiviert. Somit stehen die Symbole schließlich für die Dinge. Diesen Zusammenhang veranschaulicht Abbildung 1.

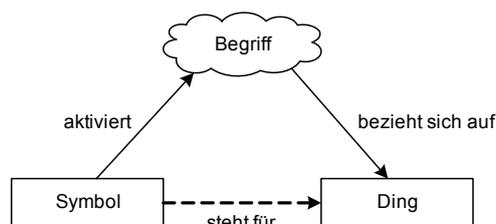


Abbildung 1: Semiotisches Dreieck [Stef02]

Bezogen auf die Modellierung bedeutet dies, dass ein Modellkonstrukteur zunächst durch die Interpretation von Sinneseindrücken oder durch Überlegung zu Begriffen gelangt, die Bestandteile seines internen, mentalen Modells sind. Wird dieses Modell schließlich expliziert, so versieht ein Modellkonstrukteur seine Begriffe mit Benennungen. Beide Vorgänge sind mit Problemen verbunden – die Begriffsbildung ist abhängig vom Vorwissen (Weltbild) und der Vorstellungskraft des Modellkonstrukteurs, die Be-

nennung der Begriffe von dessen Wortschatz und der zur Verfügung stehenden natürlichen Sprache. Solange ein Modell nur von einem Individuum erstellt und gelesen wird, ist dies weniger problematisch. Werden jedoch Modelle verschiedener Modellierer zusammengeführt, durchsucht und übersetzt, oder soll die in den Modellen enthaltene Semantik automatisch validiert und zur Konfiguration eines Informationssystems herangezogen werden, ist eine terminologische und konzeptuelle Vereinheitlichung dringend erforderlich. Dazu muss ein von allen beteiligten Akteuren geteiltes Verständnis der relevanten Begriffe einer Domäne gefunden werden.

## 2.2 Ontologien und Ontologiesprachen

Im Bereich der Künstlichen Intelligenz und des Semantic Web wird eine solche Vereinheitlichung von Begriffen und Konzepten über Ontologien seit Jahren erforscht. Eine Ontologie ist nach Gruber [Grub95] „a formal, explicit specification of a shared conceptualisation“, frei übersetzt also das formal explizierte Verständnis der von einer Gruppe von Individuen geteilten Auffassung über bestimmte Sachverhalte einer Domäne. Mit OWL (Web Ontology Language) steht eine durch das W3C (World Wide Web Consortium) normierte Beschreibungssprache für Ontologien zur Verfügung. Der Grundgedanke des Semantic Web, die Bedeutung der im Web verfügbaren Ressourcen durch das Hinzufügen von Metadaten für Menschen und Maschinen besser interpretierbar zu machen [BeHL01], wird im Rahmen dieser Arbeit auf das Gebiet der Geschäftsprozessmodellierung übertragen. Dies geschieht durch die Zuordnung von Konzepten aus formalen Ontologien zu Modellelementen, womit die in den Modellelementbezeichnern enthaltene, mit Hilfe der natürlichen Sprache formulierte Semantik explizit spezifiziert werden kann. Der Ansatz ist damit komplementär zur Betrachtung der Semantik auf Metaebene: Die in einem Prozessmodell enthaltene, formale Semantik des Kontrollflusses wird um eine formale Spezifikation der im Prozess involvierten Entitäten ergänzt.

Zur expliziten und formalen Repräsentation einer Ontologie existieren prinzipiell verschiedene Sprachen, wie z. B. CML, Conceptual Representation, CycL, KIF, Loom, OIL und OWL. Die Web Ontology Language (OWL) [SmWM04] ist ein Standard des World Wide Web Consortiums (W3C), der aus der Verschmelzung von DARPA und OIL hervorgegangen ist. Aufgrund der wachsenden Akzeptanz und damit verbunden der Unterstützung der Ontologiesprache durch Softwarebibliotheken und -werkzeuge wird im Rahmen dieses Beitrags OWL als Sprache zur Repräsentation von Ontologien verwendet. OWL steht in drei Varianten zur Verfügung: OWL Lite, OWL-DL und OWL Full. Für die Zwecke des semantischen Geschäftsprozessmanagements werden hier Ontologien der Stufen „Lite“ und „DL“ verwendet, da somit im Gegensatz zu OWL Full die Berechenbarkeit („computational completeness“) erhalten bleibt.

Über die reine Ontologiesprache hinaus existieren bereits fertige OWL-Ontologien, die für das semantische Geschäftsprozessmanagement herangezogen werden können. Bezogen auf ein gesamtes Unternehmen besteht mit BMO (Business Management Ontology) bereits ein Ansatz [Jenz03]. Zur Klassifikation von Produkten und Dienstleistungen existiert mit eclassOWL eine Portierung des eCI@ss-Standards nach OWL Lite [Hepp05]. Es können weiter Ontologien herangezogen werden, die (noch) nicht in OWL vorliegen,

wie beispielsweise die „Enterprise Ontology“ [UKMZ98] und TOVE (TOronto Virtual Enterprise) [Fox92]. Neben der Verwendung von Ontologien als Ganzem können auch Teilbereiche bestehender Ontologien genutzt werden. So besitzt die Top-level Ontologie CYC (abgeleitet vom englischen enCYClopedia) u. a. einen Bereich „Business & Commerce“, der für Geschäftsprozesse relevante Konzepte enthält [MCWD06].

Sind ein oder mehrere Ontologien gefunden, die dazu geeignet sind, die Semantik eines Geschäftsprozesses zu beschreiben, so können diese durch einen Prozess der Ontologiekonstruktion derart miteinander verschmolzen werden, dass sie schließlich als einheitliche Ontologie zur Verfügung stehen. Zur *Integration* verschiedener Ontologien stellt OWL vordefinierte Sprachkonstrukte bereit, wie etwa `owl:imports` oder zur Beschreibung von Äquivalenzen oder Verschiedenheiten `owl:equivalentClass`, `owl:equivalentProperty`, `owl:sameAs`, `owl:differentFrom`, sowie `owl:allDifferent` (für weiterführende Informationen sei auf den „OWL Guide“ des W3C verwiesen [SmWM04]). Die (teil-) automatisierte Zusammenführung von Ontologien ist darüber hinaus ein etablierter Forschungsgegenstand [MeIG00; SSFI04; StGH04] (vgl. auch ein Portal zu diesem Thema unter der URL <http://www.ontologymatching.org>).

### 2.3 Grundgerüst einer Ontologie zur Annotation von Prozessmodellen

In den folgenden Ausführungen wird ein einfaches Beispiel für eine Ontologie gezeigt und anhand einer grafischen Darstellung veranschaulicht, die Abbildung 2 zeigt. Beschriftungen werden mit den durch `rdfs:label`-Elemente gegebenen Bezeichnern gezeigt, die Konzepten einer Ontologie für die Anzeige in Editoren hinzugefügt werden können. Die XML-Repräsentationen hingegen enthalten die innerhalb der Ontologie vergebenen Namen. Als Pfeil dargestellte Eigenschaften bezeichnen Objekteigenschaften (ObjectProperties) in OWL, die Instanzen von Klassen zueinander in Beziehung setzen. Vererbungsbeziehungen beziehen sich auf das in OWL nutzbare Sprachkonstrukt `rdfs:subclassof`, das durch RDF Schema definiert wird.

Die Struktur der im Rahmen der weiteren Arbeit verwendeten Beispielontologie wird in Abbildung 2 verdeutlicht (aus Darstellungsgründen werden nicht alle in der Ontologie enthaltenen Klassen grafisch repräsentiert). Die Ontologie enthält exemplarisch Klassen für Organisationseinheiten, Aufgaben, Ereignisse, Dienste und Regeln. Diese Klassen können beliebige Spezialisierungen erfahren, beispielhaft wurden die Klassen Ereignis und Dienst weiter spezialisiert. Neben Klassen enthält die Beispielontologie Instanzen, die Individuen oder konkrete Vertreter einer Klasse repräsentieren. Die Properties `istTeilvon` und `nutzt` sind transitiv definiert, sodass bei Anfragen an die Ontologie mit Hilfe von Anfragesprachen zusätzliche Fakten geschlossen werden können (vgl. dazu auch Abschnitt 3.3). Im weiteren Verlauf wird diese Beispielontologie dazu benutzt, die modellelementspezifische Semantik von Elementen eines EPK-Modells zu spezifizieren.

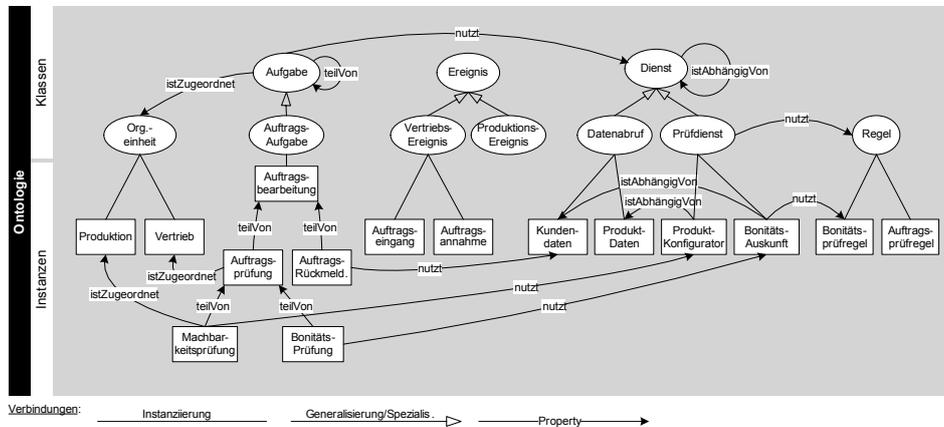


Abbildung 2: Grundgerüst einer Unternehmensontologie

### 3 Semantische Erweiterung der EPK

Bevor der Beitrag auf eine semantische Erweiterung der EPK eingeht, wird zunächst in Abschnitt 3.1 das Erweiterungsprinzip erläutert. Der darauf folgende Abschnitt 3.2 expliziert die zur Schaffung der Erweiterung benötigten Grundlagen, Abschnitt 3.3 beschreibt die Erweiterung der EPK in Form der Verknüpfung von Modellelementen mit Ontologieinstanzen. Abschnitt 3.4 zeigt dessen Operationalisierung auf Basis der EPK- und Ontologierepräsentationsformate EPML/XML bzw. RDF/XML auf.

#### 3.1 Erweiterungsprinzip

Zur semantischen Erweiterung von EPK-Modellen wurde ein mehrstufiger Ansatz entwickelt, der drei separate Ebenen umfasst (vgl. Abbildung 3). Die oberste Ebene „Ontologie“ enthält eine Ontologie, die alle relevanten Konzepte eines Unternehmenskontextes und deren Bezüge untereinander als OWL-Klassen, Instanzen und Eigenschaften (Properties) enthält. Die im Rahmen dieses Beitrags beispielhaft verwendete Ontologie wurde bereits eingeführt. In der darunter liegenden Ebene „Metadaten“ werden Instanzen der Ontologie dazu verwendet, die individuelle Semantik von EPK-Modellelementen zu spezifizieren. Auf der untersten Ebene befindet sich das EPK-Modell, dessen Modellelemente mit Hilfe der Ontologie der obersten Schicht beschrieben werden sollen. Gestrichelte Linien zwischen den Ebenen deuten die Beziehungen der Elemente der verschiedenen Ebenen an. EPK-Modellelemente werden durch diese Zuordnungsbeziehungen Elementen der Metadatenebene zugeordnet und diese wiederum den Konzepten der Ontologie.

Um die Verknüpfung von EPK-Modellelementen mit Ontologieinstanzen über eine Metadatenebene zu realisieren, müssen die Elemente der EPK zunächst innerhalb der Ontologie sichtbar bzw. referenzierbar sein. Wenn dies der Fall ist, können weitere Aussa-

gen – in Abbildung 3 als gestrichelte Linien zwischen der Metadatenebene und der Ontologieebene dargestellt – über die zugrunde liegen Modellelemente getroffen werden. Somit ist zunächst eine Repräsentation der EPK in der Ontologie vonnöten, die Gegenstand des nächsten Abschnitts ist.

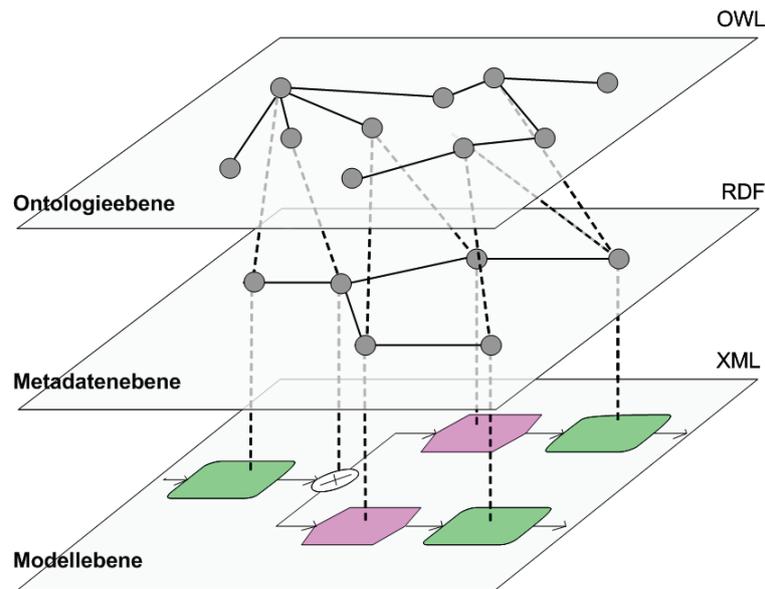


Abbildung 3: Ebenenmodell zur semantischen Geschäftsprozessbeschreibung

### 3.2 Repräsentation der EPK in der Ontologie

Bezüglich der Repräsentation der EPK in der Ontologie lässt sich eine Repräsentation der EPK-Sprachkonstrukte und eine Repräsentation von EPK-Modellelementen unterscheiden.

Die Repräsentation der EPK-Sprachkonstrukte geschieht durch gleichnamige Klassen in der Ontologie, wie beispielsweise Ereignis, Funktion oder Konnektor. Diese können als Spezialisierung einer allgemeinen Klasse Modellelement angelegt werden. Der in EPK-Modellen auftretende Kontrollfluss kann als Property fluss definiert werden. Zur Abbildung des Kontrollflusses ist es ausreichend, diesen einmalig mit der Domäne Modellelement zu spezifizieren. Kontrollflussrestriktionen, wie beispielsweise die Tatsache, dass EPK-Ereignisse oder Funktionen jeweils nur maximal eine aus- und eingehende Kontrollflusskante besitzen dürfen, könnten in der Ontologie ebenfalls angelegt werden. Allerdings sind diese Restriktionen bereits Gegenstand des Metamodells der EPK und daher aus der Perspektive der hier verfolgten semantischen Erweiterung der EPK nicht relevant – es wird vielmehr davon ausgegangen, dass bereits ein syntaktisch korrektes EPK-Modell vorliegt.

Modellelemente sind weiter Bestandteil eines Modells, dies wird durch ein entsprechendes Property gehörtZu mit der Domäne Modellelement und dem Wertebereich Modell berücksichtigt. Da Funktionen in EPK-Modellen mit Funktionshinterlegungen detailliert werden können, wird ein Property hatDetailModell mit der Domäne Funktion und dem Wertebereich Modell definiert. Insgesamt wird somit die EPK-Sprache grundlegend in der Ontologie repräsentiert. Abbildung 4 zeigt im oberen Bereich diese Repräsentation der EPK-Sprachkonstrukte in der Ontologie.

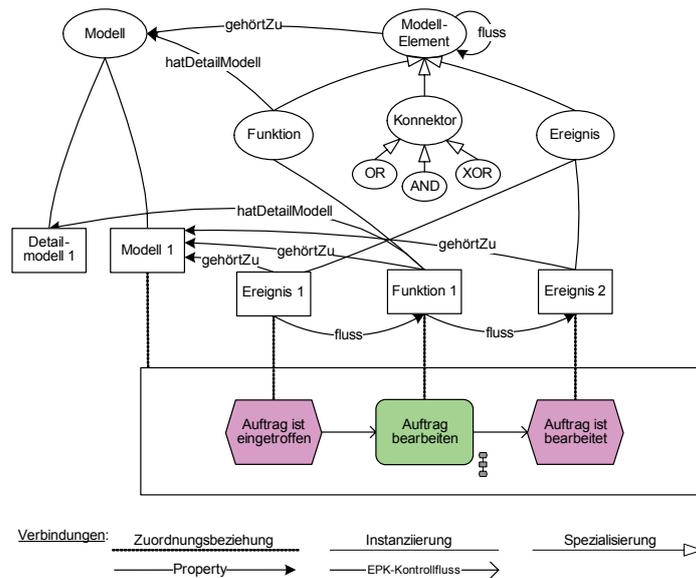


Abbildung 4: Repräsentation der EPK in der Ontologie

Die Repräsentation von EPK-Modellelementen in der Ontologie erfolgt durch die Instanziierung der zuvor definierten Sprachkonstruktclassen. Abbildung 4 verdeutlicht diesen Zusammenhang anhand eines EPK-Beispiels. Das EPK-Ereignis Auftrag ist eingetroffen wird als Ontologieinstanz Ereignis 1 der Ontologiekategorie Ereignis angelegt. Die somit erzeugte Instanz fungiert nun als Stellvertreter für das betreffende Modellelement des EPK-Modells in der Ontologie. Sie wird später dazu verwendet, eine Verknüpfung zu einer weiteren Ontologieinstanz zu etablieren, welche die Semantik des Modellelementes spezifiziert. Aus der Perspektive der Ontologie besitzen diese Stellvertreterinstanzen zunächst noch keine weitere, über die Bedeutung des jeweiligen EPK-Sprachkonstrukts hinausgehende Semantik.

Auch der Kontrollfluss der EPK wird in die Ontologierepräsentation übernommen. Dies ist zur späteren Suche in Modellen relevant und ermöglicht Anfragen wie „Welche Ereignisse lösen Funktionen aus, die einer bestimmten Organisationseinheit zugeordnet sind?“. Im Beispiel der Abbildung 4 wird der im EPK-Modell vorhandene Kontrollfluss als Property fluss zwischen den entsprechenden Ontologieinstanzen ebenfalls abgebildet. Das gesamte EPK-Beispiel der Abbildung 4 wird durch eine Ontologieinstanz Modell 1



der Definition des Ontologiebegriffs von einer Gruppe von Individuen geteilt wird. Somit besitzt ein EPK-Modellelement nun eine explizit spezifizierte Bedeutung, die nicht mehr an den Modellelementbezeichner gebunden ist.

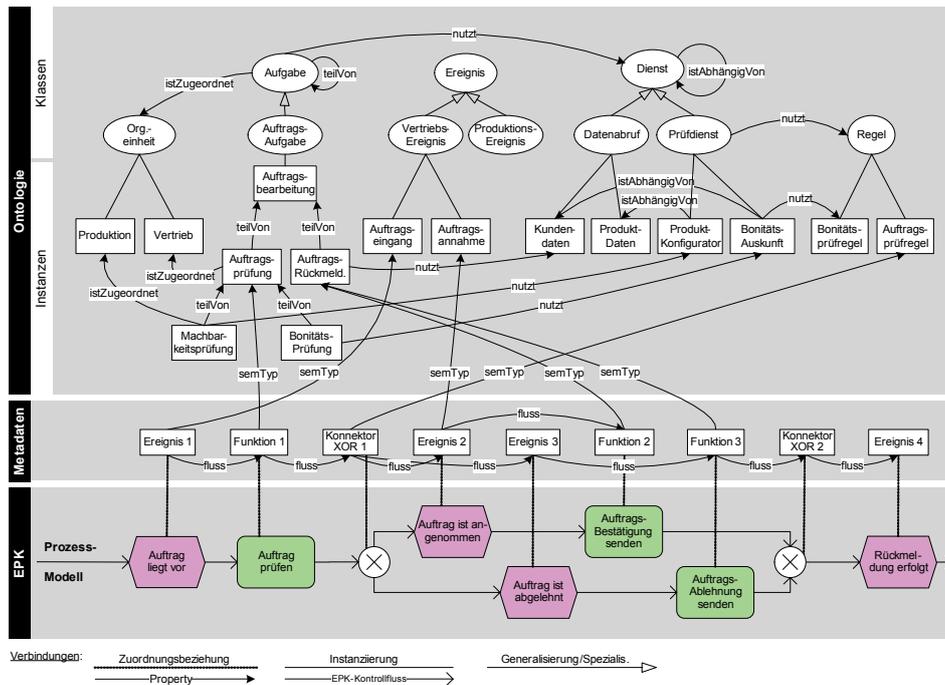


Abbildung 6: Semantisch annotierter Auftragsbearbeitungsprozess

Zum anderen wird durch die Verknüpfung der Kontext eines Modellelementes genauer spezifiziert. Dies geschieht indirekt über Relationen, die zwischen der ein EPK-Modellelement repräsentierenden Ontologieinstanz und weiteren Instanzen der Ontologie bestehen. Prinzipiell ist eine solche Spezifikation von Beziehungen zu weiteren Elementen wie Organisationseinheiten oder Ressourcen bereits mit den Konstrukten der ARIS-erweiterten EPK möglich. Im Unterschied zur ARIS-EPK und den weiteren Sprachen der ARIS-Methode können zum einen jedoch die zwischen diesen weiteren Elementen bestehenden Beziehungen mit einer umfangreicheren formalen Semantik definiert werden. Dies geschieht mit Hilfe der von OWL bereitgestellten Möglichkeiten wie die der Definition von Hierarchien oder transitiven, inversen oder symmetrische Eigenschaften. Zum anderen stehen diese in der Ontologie enthaltenen Beziehungen bei Anfragen an ein Modell ausgehend vom semantisch annotierten EPK-Modellelement zur Verfügung.

Im Beispiel der Abbildung 6 kann durch die transitive Definition des Properties `teilVon` durch maschinelle Inferenz geschlossen werden, dass die `Machbarkeitsprüfung` ein Teil der `Auftragsbearbeitung` ist. Wird weiter ein Property `hatTeilAufgabe` in der Ontologie definiert und dieses als invers in Bezug auf `teilVon` definiert, so kann auch in umgekehrter

Richtung geschlossen werden, dass die Auftragsbearbeitung aus den Teilaufgaben Auftragsprüfung, Machbarkeitsprüfung, Bonitätsprüfung und Auftragsrückmeldung besteht. In gleicher Weise kann bei einer transitiven Definition des Properties genutzt werden, dass die Bonitätsprüfung eine Bonitätsprüfregel benutzt.

Ein Beispiel für eine verkettete Anfrage ausgehend von einem EPK-Modellelement ist im Beispiel der Abbildung 6 die Frage, von welchen Diensten die Funktion Auftragsbearbeitung außer den unmittelbar spezifizierten Diensten abhängt. Zunächst werden dazu die Aufgabe Auftragsbearbeitung und deren Teilaufgaben auf benutzte Dienste untersucht. Anschließend wird für alle diese Dienste untersucht, ob diese ein Property ist. Abhängig davon besitzen und ggf. der entsprechende Dienste als Ergebnis der Anfrage zurückgegeben.

Ein Beispiel für eine symmetrische Eigenschaft wäre ein Property `hatGeschäftsPartner` zwischen Organisationseinheiten. Wird für eine Organisationseinheit A eine Organisationseinheit B als Geschäftspartner definiert, so kann auch gefolgert werden, dass Organisationseinheit B einen Geschäftspartner Organisationseinheit A besitzt.

Insgesamt können sehr weit reichende Anfragen an die semantisch annotierten Geschäftsprozessmodelle gerichtet werden. Obwohl sich Anfragesprachen für Ontologien noch weiterhin in der Entwicklung befinden, existiert mit SPARQL [PrSe05] bereits eine vom W3C vorgeschlagene Anfragesprache, mit Hilfe derer die oben beschriebenen Anfragen durchgeführt werden können.

### 3.4 RDF-Repräsentation der semantischen EPK

In technischer Hinsicht wird die Verknüpfung von EPK-Modellelementen durch das Einfügen von Attributen in die XML-Repräsentation eines EPK-Modells realisiert. Diese Attribute identifizieren die jeweils zu einem Modellelement zugehörige Ontologieinstanz, die das betreffende Element semantisch spezifiziert. Abbildung 7 veranschaulicht dies sowohl grafisch als auch mit Hilfe der entsprechenden XML-Vokabulare EPML/XML (Event Driven Process Markup Language) für die EPK-Repräsentation, RDF/XML (Resource Description Framework) für eine semantische Repräsentation der EPK – im Rahmen dieses Beitrags auch als sEPK bezeichnet – und OWL/XML zur Repräsentation der Ontologieklassen und -instanzen.

Wie in Abbildung 7 ebenfalls zu erkennen ist, geschieht eine Verknüpfung von EPK-Modellelement und Ontologieinstanz in Übereinstimmung zu Abbildung 3 über eine Zwischenstufe in Form von Metadaten. Diese enthalten in Form eines `semanticType`-Attributs (vgl. auch Abbildung 6) die mit dem EPK-Modellelement korrespondierende Ontologieinstanz. Diese wird auch in der EPML-Repräsentation als Attribut hinterlegt. Weiter werden die natürlichsprachigen Bezeichner der EPK-Modellelemente als Bezeichner in den Metadaten im Feld `rdfs:label` verwendet. Beide Zuordnungsbeziehungen sind grafisch auf der rechten Seite der Abbildung 7 mit Hilfe einer gestrichelten Linie dargestellt.

Aus einer konzeptionellen Sicht ist der Sprachumfang von RDF für die Metadaten ausreichend, da keine Sprachkonstrukte aus OWL verwendet werden. Aus einer technischen

Sicht ergibt sich dennoch OWL-DL als Sprachumfang, da die zur Annotation verwendeten Ontologieinstanzen beispielsweise bei Anfragen in die Metadaten importiert werden müssen.

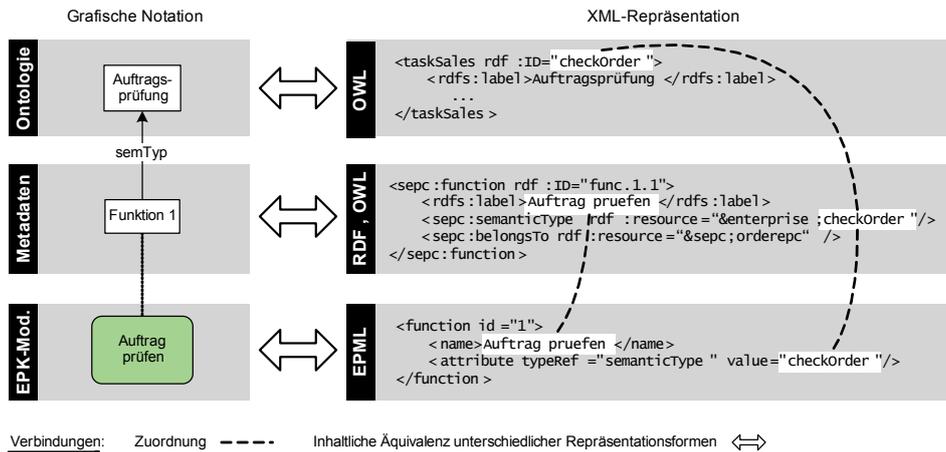


Abbildung 7: Verknüpfung von EPK-Modellen mit Ontologien auf Repräsentationsbasis

Nachdem eine Verknüpfung von EPK-Modell und Ontologieinstanzen wie in Abbildung 7 gezeigt realisiert ist, kann auf Basis der Repräsentationsformate eine komplette Transformation der EPK in eine sEPK-Repräsentation erfolgen. Diese besteht aus der XML-Repräsentation der Metadaten, die beispielhaft bereits in der mittleren Ebene der Abbildung 7 gezeigt wurde. Die Transformation ist in Abbildung 8 dargestellt.

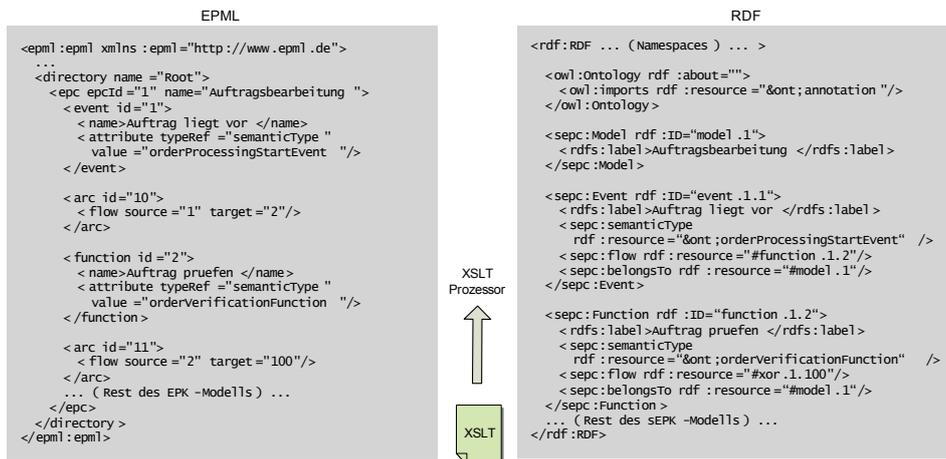


Abbildung 8: Transformation von EPML nach RDF

## 4 IT-Unterstützung der semantischen Geschäftsprozessmodellierung

Eine IT-Unterstützung des hier vorgeschlagenen Ansatzes ist grundlegend in den Bereichen der *Annotation* von EPK-Modellelementen mit Instanzen der Ontologie, der *Transformation* von EPK-Modellen in sEPK-Modelle und der *Speicherung und Anfrage* von sEPK-Modellen notwendig.

Die *Annotation* betreffend müssen Werkzeuge zur Verfügung stehen, die innerhalb der EPK-Modellierungsumgebung die semantische Annotation von Modellelementen durch einen Auswahlmechanismus für Ontologieinstanzen unterstützen. Dies kann durch geeignete Browsing-Verfahren und Visualisierungstechniken erfolgen, wobei die gesamte Ontologie oder eine vereinfachte Version als Browsing-Struktur dienen kann. Viel versprechend erscheint hier eine Lösung auf Basis der Entwicklungsumgebung Eclipse, da bereits Eclipse-basierte Werkzeuge sowohl zur EPK-Modellierung in Form der „EPC Tools“ (<http://www.wcs.uni-paderborn.de/cs/kindler/research/EPCTools>) als auch zur Ontologiemodellierung durch das „Integrated Ontology Development Toolkit“ von IBM existieren (<http://www.alphaworks.ibm.com/tech/semanticstk>).

Des Weiteren kann die Entwicklung einer zur Annotation geeigneten Ontologie unterstützt werden, indem aus bereits vorhandenen Prozessmodellen, die sich den Konstrukten der ARIS-EPK bedienen, zur Annotation geeignete Instanzen in der Ontologie angelegt werden. Im Beispiel der Abbildung 9 wird aus einer EPK-Funktion, die mit einer Organisationseinheit „Vertrieb“ verbunden ist und als Ressource „Kundendaten“ benötigt, eine korrespondierende Ontologieinstanz „Auftrag prüfen“ angelegt. Die Auswahl geeigneter Properties „benötigt“ und „istZugeordnet“ erfolgt automatisiert aufgrund der im Metamodell definierten Semantik der EPK-Sprachkonstrukte.

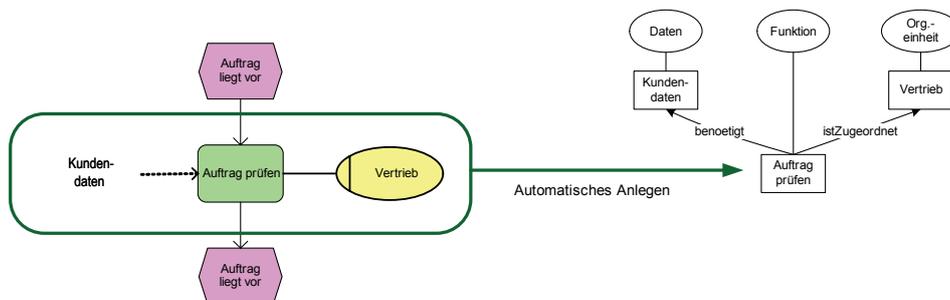


Abbildung 9: Automatisches Anlegen einer Ontologieinstanz

Zur *Transformation* wurde bereits ein XSLT-basierter Ansatz am Institut für Wirtschaftsinformatik in Saarbrücken entwickelt, mit dem eine zuvor semantisch annotierte EPK in eine sEPK überführt werden kann.

Die *Speicherung und Anfrage* wurde ebenfalls bereits prototypisch mit Hilfe des RDF-Rahmenwerks Jena und Tomcat als Server entwickelt. Anfragen an sEPK-Modelle sind

mit der RDF-Anfragesprache SPARQL möglich, wobei die in Jena enthaltene Inferenzmaschine benutzt wird. Weitere geplante Entwicklungen betreffen einen Query-Builder für die interaktive Zusammenstellung von SPARQL-Anfragen sowie ein sEPK-Repository. Dieses soll für die Anfrage, Transformation und Speicherung von Modellen über web-services-basierte Schnittstellen verfügen sowie über Schnittstellen zu etablierten Modellierungswerkzeugen.

## 5 Verwandte Arbeiten

Die EPK ist Gegenstand aktueller Forschungsbemühungen. Die vorliegenden Arbeiten widmen sich einerseits der Konstruktion von Prozessmodellen, andererseits werden modellierungssprachliche und -methodische Aspekte der Konstruktion von EPK-Modellen untersucht. Hierbei werden vor allem die Möglichkeiten einer formalen Spezifikation der EPK-Syntax und -Semantik geprüft [Aals99; NüRu02; Kind06; RoAa06]. Diese ist auch im Hinblick auf eine Transformation der EPK in ausführbare BPEL-Prozessdefinitionen relevant. Zur Lektüre entsprechender Arbeiten wird ergänzend auf die umfangreiche Literaturliste der EPK-Community verwiesen (vgl. URL <http://epk.et-inf.fho-emden.de/literatur.php>).

Verwandte Arbeiten existieren ebenso hinsichtlich des semantischen Geschäftsprozessmanagements [Jenz03]. Bezogen auf das Geschäftsprozessmanagement im Zusammenspiel mit (Semantic) Web Services existiert mit eine Beschreibung der Potenziale einer Kombination semantischer Prozessbeschreibungen mit semantischen Web Services insbesondere unter Berücksichtigung der fachlichen Perspektive des Prozessmanagements [HLDW05]. Allerdings behandelt dieser Ansatz keine konkreten fachlichen Beschreibungssprachen.

Die IT-Unterstützung betreffend existiert mit Sementalk bereits ein gut ausgebauter Ansatz für die Verknüpfung von EPK-Modellen mit Ontologien auf der Basis des grafischen Modellierungs- und Zeichenwerkzeugs Microsoft VISIO [FiWe04]. Die Semantik der EPK-Modellelemente wird dabei objektorientiert als Zustände und Aktionen von Objekten aufgefasst, womit der Ansatz im Gegensatz zu diesem Beitrag eine objektorientierte und eine prozessorientierte Abstraktion des unternehmerischen Geschehens erfordert.

## 6 Diskussion der Ergebnisse und Ausblick

Die genaue semantische Spezifikation von Modellelementen wurde zwar bereits bei der Einführung der EPK gefordert [KeNS92], aber deren Durchführung jedoch in der Folge nicht aufgezeigt. Die erneute Betonung der Semantik von EPK-Modellen geschieht nun vor dem Hintergrund der Ergebnisse des Semantic Web, das Sprachen, Techniken und Werkzeuge zur Spezifikation und Verarbeitung von Semantik zur Verfügung stellt. Mit dessen Hilfe ist die Semantik betrieblicher Abläufe exakter spezifizierbar und damit auch einer maschinellen Verarbeitung zugänglich.

Die Vorteile der Transformation von Prozessmodellen in semantische Prozessmodelle unter Verwendung von OWL sind im Wesentlichen:

- Anfragen an Prozessmodelle können auf semantischer Ebene erfolgen. Durch die Verwendung von Inferenzmechanismen können zum Anfragezeitpunkt neue Fakten geschlossen werden, die nicht explizit in den Prozessmodellen gespeichert worden sind. Über einfache Vererbungsbeziehungen hinaus ergeben sich durch die von OWL bereitgestellten Möglichkeiten wie transitiven, symmetrischen oder inversen Eigenschaften zusätzliche Möglichkeiten der Schlussfolgerung und damit des Retrievals.
- Durch die Annotation von Ontologieinstanzen wird das Verständnis der Geschäftsprozesse erhöht, da Ontologien die von Individuen geteilten Konzeptionalisierungen und Terminologie enthalten. Durch die Möglichkeit, eine vorhandene Ontologie beliebig um weitere Aussagen zu erweitern, beispielsweise die relevanten gesetzlichen Grundlagen in Bezug auf die Abwicklung einer betrieblichen Aufgabe, eignen sich semantisch annotierte Prozessketten als Ausgangspunkt eines prozessorientierten Wissensmanagements.
- Der Aufwand zur „Internationalisierung“ von Prozessmodellen wird reduziert, da Übersetzungsarbeiten der Bezeichner individueller Modellelemente aufgrund von in der Ontologie mehrsprachig hinterlegbaren Bezeichnungen der Ontologieinstanzen erleichtert oder teilautomatisiert werden können.
- Die Ausführbarkeit von Prozessen kann erleichtert werden, da die Ontologie um technische Details zur Ausführung erweitert werden kann und somit aus einem sEPK-Modell beispielsweise eine BPEL-Repräsentation generiert werden kann. Die semantische Beschreibung von Prozessen aus einer fachlichen Perspektive ergänzt und komplettiert die aktuellen Bestrebungen der semantischen Beschreibung technischer Prozesse, wie etwa WSMO [ACDF05], WSDL-S [AFMN05], OWL-S [MBLP04], KDSWS [HoKe04], METEOR-S [ASSV04] und IRS-II [MDCG03]. Abbildungen fachlicher Sachverhalte auf technische erfolgen durch die zentrale Speicherung von Geschäftsprozesselementen als Ontologieinstanzen redundanzfrei.

Weiterer Forschungsbedarf existiert hinsichtlich der Integration semantisch heterogener Ontologien in eine zur Annotation geeigneten Ontologie, hinsichtlich der Abbildung von Dynamik, d.h. Änderungen an der Ontologie, sowie der Verbindung des Ansatzes zu Semantic Web Services oder Web Services Repositories.

## Literaturverzeichnis

- [Aals99] van der Aalst, W. M. P.: Formalization and verification of event-driven process chains. In: Information and Software Technology 41 (1999), Nr. 10, S. 639–650
- [ACDF05] Arroyo, S.; Cimpian, E.; Domingue, J.; Feier, C.; Fensel, D.; König-Ries, B.; Lausen, H.; Polleres, A.; Stollberg, M.: Web Service Modeling Ontology Primer : W3C Member Submission 3 June 2005. Innsbruck : DERI, 2005

- [AFMN05] Akkiraju, R.; Farrell, J.; Miller, J.; Nagarajan, M.; Schmidt, M.-T.; Sheth, A.; Verma, K.: Web Service Semantics – WSDL-S : W3C Member Submission 7 November 2005. University of Georgia Research Foundation, Inc., 2005
- [ASSV04] Abhijit, P.; Swapana, O.; Sheth, A.; Verma, K.: METEOR-S Web Service Annotation Framework. In: The Thirteenth International World Wide Web Conference (WWW 2004), May 17–22, 2004, New York, USA. ACM, 2004, S. 553–562
- [BeHL01] Berners-Lee, T.; Hendler, J.; Lassila, O.: The Semantic Web. Scientific American, 2001. – URL <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&sc=I100322> [Zugriffsdatum 23.11.2005]
- [Dürr06] Dürr, M.: Deskriptive Linguistik: Grundlagen und Methoden. Göttingen : Vandenhoeck & Ruprecht, 2006
- [FiWe04] Fillies, C.; Weichhardt, F.: On Ontology-based Event-driven Process Chains. URL <http://www.semtalk.com/pub/semtalkep.pdf> [Zugriffsdatum 25.11.2005]
- [Fox92] Fox, M. S.: The TOVE Project: A Common-sense Model of the Enterprise. In: Belli, F.; Radermacher, F. J. (Hrsg.): Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, 5th International Conference, IEA/AIE 92, Paderborn, June 9–12, 1992, Proceedings. London : Springer, 1992, S. 25–34
- [Grub95] Gruber, T. R.: Toward principles for the design of ontologies used for knowledge sharing? In: International Journal of Human-Computer Studies 43 (1995), Nr. 5–6, S. 907–928
- [Hepp05] Hepp, M.: eClassOWL: A Fully-Fledged Products and Services Ontology in OWL. In: Poster Proceedings of the 4th International Semantic Web Conference (ISWC2005), November 7–11, 2005. Galway, Ireland, 2005
- [HLDW05] Hepp, M.; Leymann, F.; Domingue, J.; Wahler, A.; Fensel, D.: Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In: Proceedings of the IEEE ICEBE 2005, October 18–20, Beijing, China. Beijing, China, 2005, S. 535–540
- [HoKe04] Howard, R.; Kerschberg, L.: A Knowledge-based Framework for Dynamic Semantic Web Services Brokering and Management. In: Galindo, F.; Takizawa, M.; Traummüller, R. (Hrsg.): Database and expert systems applications : 15th International Conference, DEXA 2004, Zaragoza, Spain, August 30-September 3, 2004 : proceedings. Berlin : Springer, 2004, S. 174–178
- [Jenz03] Jenz, D. E.: Strategic White Paper: Ontology-Based Business Process Management ; The Vision Statement. Erlensee : Jenz & Partner GmbH, 2003
- [KeNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Nr. 89, Saarbrücken : Universität des Saarlandes, 1992
- [Kind06] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In: Data & Knowledge Engineering 56 (2006), Nr. 1, S. 23–40
- [LaSW98] Langner, P.; Schneider, C.; Wehler, J.: Petri Net Based Certification of Event driven Process Chains. In: Desel, J.; Silva, M. (Hrsg.): Application and theory of Petri nets 1998 : 19th international conference ; proceedings. Berlin [u. a.] : Springer, 1998, S. 286–305
- [MBLP04] Martin, D.; Burstein, M.; Lassila, O.; Paolucci, M.; Payne, T.; McIlraith, S. A.: Describing Web Services using OWL-S and WSDL. Arlington, VA : BBN Rosslyn office, 2004

- [MCWD06] Matuszek, C.; Cabral, J.; Witbrock, M.; DeOliveira, J.: An Introduction to the Syntax and Content of Cyc. In: Baral, C. (Hrsg.): Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering : Papers from the 2006 AAAI Spring Symposium. Menlo Park, CA : AAAI Press, 2006, S. 44–49. – Technical Report SS–06–05]
- [MDCG03] Motta, E.; Domingue, J.; Cabral, L.; Gaspari, M.: IRS–II: A Framework and Infrastructure for Semantic Web Services. In: Fensel, D.; Sycara, K.; Mylopoulos, J. (Hrsg.): The SemanticWeb – ISWC 2003 : Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20–23, 2003 : Proceedings. Berlin : Springer, 2003, S. 306–318
- [MeIG00] Mena, E.; Illarramendi, A.; Goni, A.: Automatic Ontology Construction for a Multiagent-Based Software Gathering Service. In: Klusch, M.; Kerschberg, L. (Hrsg.): Cooperative Information Agents IV – The Future of Information Agents in Cyberspace: 4th International Workshop, CIA 2000, Boston, MA, USA, July 7–9, 2000. Proceedings. Berlin : Springer, 2000, S. 232–243
- [NüRu02] Nüttgens, M.; Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Desel, J.; Weske, M. (Hrsg.): Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen (Promise ‘2002), Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 9.–11. Oktober 2002. Bonn : Köllen, 2002, S. 64–77
- [OgRi23] Ogden, C. K.; Richards, I. A.: The meaning of meaning : A study of the influence of language upon thought and of the science of symbolism. 1. Aufl. London : Kegan Paul, Trench, Trubner, 1923
- [PrSe05] Prud’hommeaux, E.; Seaborne, A. (Hrsg.): SPARQL Query Language for RDF : W3C Working Draft 23 November 2005. W3C, 2005. – URL <http://www.w3.org/TR/rdf-sparql-query/> [Zugriffsdatum 02.12.2005]
- [RoAa06] Rosemann, M.; van der Aalst, W. M. P.: A configurable reference modelling language. In: Information Systems (2006). – In Press, Corrected Proof
- [SmWM04] Smith, M. K.; Welty, C.; McGuinness, D. L. (Hrsg.): OWL Web Ontology Language Guide : W3C Recommendation 10 February 2004. W3C, 2004. – URL <http://www.w3.org/TR/owl-guide/> [Zugriffsdatum 29.01.2006]
- [SSFI04] Sugiura, N.; Shigeta, Y.; Fukuta, N.; Izumi, N.; Yamaguchi, T.: Towards On-the-Fly Ontology Construction – Focusing on Ontology Quality Improvement. In: Bussler, C. et al. (Hrsg.): The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece, May 10–12, 2004. Proceedings. Berlin : Springer, 2004, S. 1–15
- [Stef02] Steffen, S.: Wissensmanagement mit Ontologien und Metadaten. In: Informatik-Spektrum 25 (2002), Nr. 3, S. 194–209
- [StGH04] Stephens, L. M.; Gangam, A. K.; Huhns, M. N.: Constructing Consensus Ontologies for the Semantic Web: A Conceptual Approach. In: World Wide Web 7 (2004), Nr. 4, S. 421–442
- [UKMZ98] Uschold, M.; King, M.; Moralee, S.; Zorgios, Y.: The Enterprise Ontology. In: The Knowledge Engineering Review 13 (1998), Nr. 1, S. 31–89
- [Wies59] Wieser, W.: Organismen, Strukturen, Maschinen : Zu einer Lehre vom Organismus. Frankfurt am Main [u. a.] : Fischer Bücherei, 1959



# Towards EPC Semantics based on State and Context

Jan Mendling                      Wil van der Aalst  
WU Vienna                        TU Eindhoven  
jan.mendling@wu-wien.ac.at    w.m.p.v.d.aalst@tue.nl

**Abstract:** The semantics of the OR-join have been discussed for some time, in the context of EPCs, but also in the context of other business process modeling languages like YAWL. In this paper, we show that the existing solutions are not satisfactory from the intuition of the modeler. Furthermore, we present a novel approach towards the definition of EPC semantics based on state and context. The approach uses two types of annotations for arcs. Like in some of the other approaches, arcs are annotated with positive and negative tokens. Moreover, each arc has a context status denoting whether a positive token may still arrive. Using a four-phase approach tokens and statuses are propagated thus yielding a new kind of semantics which overcomes some of the well-known problems related to OR-joins in EPCs.

## 1 Introduction

The Event-driven Process Chain (EPC) is a business process modeling language for the representation of temporal and logical dependencies of activities in a business process (see [KNS92]). EPCs offer *function type* elements to capture the activities of a process and *event type* elements describing pre- and post-conditions of functions. Some EPC definitions also include *process interface type* elements. A process interface is a syntax element that links two consecutive EPCs: at the end of the first EPC, a process interface points to the second EPC, and at the beginning of the second, there is a process interface representing the preceding EPC. Furthermore, there are three kinds of *connector types* (i.e. AND, OR, and XOR) for the definition of complex routing rules. Connectors have either multiple incoming and one outgoing arc (join connectors) or one incoming and multiple outgoing arcs (split connectors). As a syntax rule, functions and events have to alternate, either directly or indirectly when they are linked via one or more connectors. Moreover, OR- and XOR-splits after events are not allowed, since events cannot make decisions. Control flow arcs are used to link elements.

The informal (or intended) semantics of an EPC can be described as follows. The AND-split activates all subsequent branches in a concurrent fashion. The XOR-split represents a choice between one of alternative branches. The OR-split triggers one, two or up to all of multiple branches based on conditions. In both cases of the XOR- and OR-split, the activation conditions are given in events subsequent to the connector. Accordingly, event-function-splits are forbidden with XOR and OR to avoid the situation where the activation conditions do not become clear in the model. The AND-join waits for all in-

coming branches to complete, then it propagates control to the subsequent EPC element. The XOR-join merges alternative branches. The OR-join synchronizes all active incoming branches, i.e., it needs to know whether the incoming branches may receive tokens in the future. This feature is called non-locality since the state of all (transitive) predecessor nodes has to be considered.

Several formal approaches were presented for the definition of EPC semantics. A particular problem of these semantics is that refining a function of an EPC with a structured OR-block can yield unexpected behavior. We will illustrate this problem by the help of an example. Against this background, we present a concept for the definition of EPC semantics based on state and context. The remainder of the paper is structured as follows. Section 2 provides a definition of EPC syntax. Section 3 discusses problems of existing EPC formalizations and an approach to formalize EPCs based on state and context. Section 4 concludes the paper and gives an outlook on future research.

## 2 EPC Syntax

There is not only one, but there are several approaches towards the formalization of EPC syntax. A reason for that is that the original EPC paper introduces them only in an informal way (see [KNS92]). The subsequent syntax definition of flat EPCs essentially follows the presentation in [NR02] and [MN03]. If it is clear from the context that a flat EPC is discussed, the term EPC will be used instead for brevity. Please note that an initial marking as proposed in [Rum99, NR02] is not included in the syntax definition, but discussed in the context of soundness in Section 3.6.

**Definition 1** (Flat EPC). A flat  $EPC = (E, F, P, C, l, A)$  consists of four pairwise disjoint and finite sets  $E, F, C, P$ , a mapping  $l : C \rightarrow \{and, or, xor\}$ , and a binary relation  $A \subseteq (E \cup F \cup P \cup C) \times (E \cup F \cup P \cup C)$  such that

- An element of  $E$  is called *event*.  $E \neq \emptyset$ .
- An element of  $F$  is called *function*.  $F \neq \emptyset$ .
- An element of  $P$  is called *process interface*.
- An element of  $C$  is called *connector*.
- The mapping  $l$  specifies the type of a connector  $c \in C$  as *and*, *or*, or *xor*.
- $A$  defines the control flow as a coherent, directed graph. An element of  $A$  is called an *arc*. An element of the union  $N = E \cup F \cup P \cup C$  is called a *node*.

In order to allow for a more concise characterization of EPCs, notations are introduced for preset and postset nodes, incoming and outgoing arcs, paths, transitive closure, corona, and several subsets.

**Definition 2** (Preset and Postset of Nodes). Let  $N$  be a set of *nodes* and  $A \subseteq N \times N$  a binary relation over  $N$  defining the arcs. For each *node*  $n \in N$ , we define its *preset*  $\bullet n = \{x \in N \mid (x, n) \in A\}$ , and its *postset*  $n \bullet = \{x \in N \mid (n, x) \in A\}$ .

**Definition 3** (Incoming and Outgoing Arcs). Let  $N$  be a set of *nodes* and  $A \subseteq N \times N$  a binary relation over  $N$  defining the arcs. For each *node*  $n \in N$ , we define the set

of incoming arcs  $n_{in} = \{(x, n) | x \in N \wedge (x, n) \in A\}$ , and the set of outgoing arcs  $n_{out} = \{(n, y) | y \in N \wedge (n, y) \in A\}$ .

**Definition 4** (Paths, Connector Chains, and Transitive Closure). Let  $a, b \in N$  be two nodes of an EPC. A *path*  $a \xrightarrow{c} b$  refers to a sequence of nodes  $n_1, \dots, n_k \in N$  with  $a = n_1$  and  $b = n_k$  such that for all  $i \in 1, \dots, k$  holds:  $(n_1, n_2), \dots, (n_i, n_{i+1}), \dots, (n_{k-1}, n_k) \in A$ . This includes the empty path of length zero, i.e., for any node  $a : a \xrightarrow{c} a$ . If  $a, b \in N$  and  $n_2, \dots, n_{k-1} \in C$ , the path  $a \xrightarrow{c} b$  is called *connector chain*. This includes the empty connector chain, i.e.,  $a \xrightarrow{c} b$  if  $(a, b) \in A$ .  $a, b$ . The *transitive closure*  $A^*$  contains  $(n_1, n_2)$  if there is a non-empty path from  $n_1$  to  $n_2$ , i.e., there is a non-empty set of arcs of  $A$  leading from  $n_1$  to  $n_2$ . For each node  $n \in N$ , we define its *transitive preset*  $*n = \{x \in N | (x, n) \in A^*\}$ , and its *transitive postset*  $n* = \{x \in N | (n, x) \in A^*\}$ .

**Definition 5** (Upper Corona, Lower Corona). For a node  $n \in N$  of an EPC, its upper corona is defined as  $*n = \{v \in (E \cup F \cup P) | v \xrightarrow{c} n\}$ . It includes those non-connector nodes of the transitive preset that reach  $n$  via a connector chain. In analogy, its lower corona is defined as  $n* = \{w \in (E \cup F \cup P) | n \xrightarrow{c} w\}$ .

**Definition 6** (Subsets). For an EPC, we define the following subsets of its nodes and arcs:

- $E_s = \{e \in E \mid |\bullet e| = 0\}$  being the set of start-events,
- $E_{int} = \{e \in E \mid |\bullet e| = 1 \wedge |e\bullet| = 1\}$  being the set of intermediate-events, and
- $E_e = \{e \in E \mid |e\bullet| = 0\}$  being the set of end-events.
- $P_s = \{p \in P \mid |\bullet p| = 0\}$  being the set of start-process-interfaces and
- $P_e = \{p \in P \mid |p\bullet| = 0\}$  being the set of end-process-interfaces.
- $J = \{c \in C \mid |\bullet c| > 1 \text{ and } |c\bullet| = 1\}$  as the set of join- and
- $S = \{c \in C \mid |\bullet c| = 1 \text{ and } |c\bullet| > 1\}$  as the set of split-connectors.
- $C_{and} = \{c \in C \mid (c, and) \in l\}$  being the set of and-connectors,
- $C_{xor} = \{c \in C \mid (c, xor) \in l\}$  being the set of xor-connectors, and
- $C_{or} = \{c \in C \mid (c, or) \in l\}$  being the set of or-connectors.
- $J_{and} = \{c \in J \mid (c, and) \in l\}$  being the set of and-join-connectors,
- $J_{xor} = \{c \in J \mid (c, xor) \in l\}$  being the set of xor-join-connectors,
- $J_{or} = \{c \in J \mid (c, or) \in l\}$  being the set of or-join-connectors,
- $S_{and} = \{c \in S \mid (c, and) \in l\}$  being the set of and-split-connectors,
- $S_{xor} = \{c \in S \mid (c, xor) \in l\}$  being the set of xor-split-connectors, and
- $S_{or} = \{c \in S \mid (c, or) \in l\}$  being the set of or-split-connectors.
- $C_{EF} = \{c \in C \mid *c \xrightarrow{c} E \wedge c* \xrightarrow{c} (F \cup P)\}$  as the set of event-function-connectors (ef-connectors) and
- $C_{FE} = \{c \in C \mid *c \xrightarrow{c} (F \cup P) \wedge c* \xrightarrow{c} E\}$  as the set of function-event-connectors (fe-connectors).
- $A_{EF} \subseteq (E \cup C_{EF}) \times (F \cup P \cup C_{EF})$  as the set of event-function-arcs and
- $A_{FE} \subseteq (F \cup P \cup C_{FE}) \times (E \cup C_{FE})$  as the set of function-event-arcs.
- $A_s \subseteq \{(x, y) \in A \mid x \in E_s\}$  as the set of start-arcs,
- $A_{int} \subseteq \{(x, y) \in A \mid x \notin E_s \wedge y \notin E_e\}$  as the set of intermediate-arcs, and
- $A_e \subseteq \{(x, y) \in A \mid y \in E_e\}$  as the set of end-arcs.

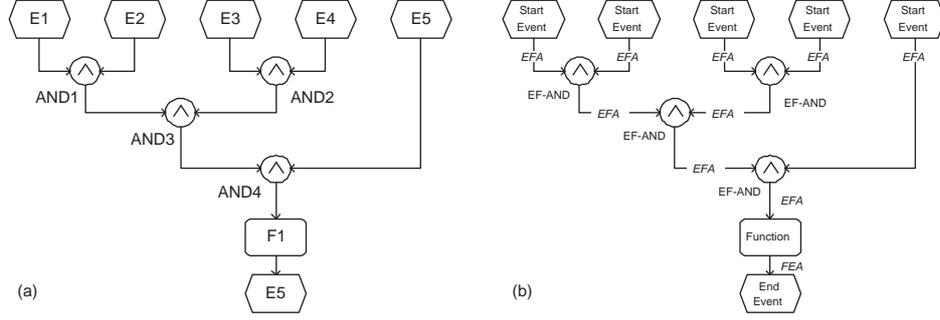


Figure 1: An EPC (a) with labelled nodes and (b) its nodes related to the subsets of Definition 6.

Figure 1 illustrates the different subsets of an EPC. Consider for example the connector  $AND3$ . It is an event-function-connector since its upper corona, i.e. those non-connector nodes from which there is a connector chain to  $AND3$ , contains only events and its lower corona contains functions only, in this case exactly one function. Furthermore, the arc from  $AND1$  to  $AND3$  is an event-function-arc since it connects two event-function-connectors. Note that arcs from events to event-function-connectors and arcs from event-function-connectors to functions are event-function-arcs, too.

In the remainder we assume an EPC to satisfy the following requirements.

**Definition 7** (Syntactically Correct EPC). An  $EPC = (E, F, P, C, l, A)$  is called syntactically correct, if it fulfills the requirements:

1.  $EPC$  is a simple, directed, coherent, and antisymmetric graph.
2. There are no connector cycles, i.e.  $\forall a, b \in C : \text{if } a \neq b \text{ and } a \xrightarrow{c} b, \text{ then } \nexists b \xrightarrow{c} a$ .
3.  $|E| \geq 2$ . There are at least two events in an EPC.
4.  $|F| \geq 1$ . There is at least one function in an EPC.
5. Events have at most one incoming and one outgoing arc.  
 $\forall e \in E : |\bullet e| \leq 1 \wedge |e \bullet| \leq 1$ .
6. Functions have exactly one incoming and one outgoing arcs.  
 $\forall f \in F : |\bullet f| = 1 \wedge |f \bullet| = 1$ .
7. Process interfaces have one incoming or one outgoing arcs.  
 $\forall p \in P : (|\bullet p| = 1 \wedge |p \bullet| = 0) \vee (|\bullet p| = 0 \wedge |p \bullet| = 1)$ .
8. Connectors have one incoming and multiple outgoing arcs or multiple incoming and one outgoing arc.  $\forall c \in C : (|\bullet c| = 1 \wedge |c \bullet| > 1) \vee (|\bullet c| > 1 \wedge |c \bullet| = 1)$ .
9. Events must have function, process interface, and fe-connector nodes in the preset, and function, process interface and ef-connector nodes in the postset.  
 $\forall e \in E : \bullet e \subseteq (F \cup P \cup C_{FE}) \wedge e \bullet \subseteq (F \cup P \cup C_{EF})$ .
10. Functions must have events and ef-connectors in the preset and events and fe-connectors in the postset.  
 $\forall f \in F : \bullet f \subseteq (E \cup C_{EF}) \wedge f \bullet \subseteq (E \cup C_{FE})$ .
11. Process interfaces are connected to events only.  
 $\forall p \in P : \bullet p \subseteq E \wedge p \bullet \subseteq E$ .

12. Connectors must have either functions, process interfaces, and fe-connectors in the preset and functions, process interfaces, and ef-connectors in the postset or events and ef-connectors in the preset and events and fe-connectors in the postset.  

$$\forall c \in C : (\bullet c \subseteq (F \cup P \cup C_{FE})) \wedge c \bullet \subseteq (F \cup P \cup C_{EF}) \vee$$

$$(\bullet c \subseteq (E \cup C_{EF}) \wedge c \bullet \subseteq (E \cup C_{FE})).$$
13. Arcs either connect events and ef-connectors with functions, process interfaces, and ef-connectors or functions, process interfaces, and fe-connectors with events and fe-connectors.  

$$\forall a \in A : (a \in (E \cup C_{EF}) \times (F \cup P \cup C_{EF})) \vee (a \in (F \cup P \cup C_{FE}) \times (E \cup C_{FE})).$$

### 3 EPC Semantics

In addition to related work on the syntax of EPCs, there are several contributions towards the formalization of EPC semantics. This section first illustrates the semantical problems related to the OR join using an illustrative set of examples, then it gives a historical overview of semantical definitions described in literature, and provides a formalization for EPCs that is used throughout this thesis.

#### 3.1 Informal Semantics as a Starting Point

Before discussing EPC formalization problems, we need to establish an informal understanding of state representation and state changes of an EPCs. Although we provide a formal definition not before Section 3.4, the informal declaration of state concepts helps to discuss formalization issues in this section. The *state*, or *marking*, of an EPC is defined by assigning a number of *tokens* (or process folders) to its arcs.<sup>1</sup> The formal semantics of an EPC define which state changes are possible for a given marking. These state changes are formalized by a *transition relation*. A node is called *enabled* if there are enough tokens on its incoming arcs that it can fire, i.e., a state change defined by a transition can be applied. This process is also called *firing*. A firing of a node  $n$  consumes tokens from its input arcs  $n_{in}$  and produces tokens at its output arcs  $n_{out}$ . The formalization of whether an OR-join is enabled is a non-trivial issue since not only the incoming arcs have to be considered. The sequence  $\tau = n_1 n_2 \dots n_m$  is called a *firing sequence* if it is possible to execute a sequence of steps, i.e., after firing  $n_1$  it is possible to fire  $n_2$ , etc. Through a sequence of firings the EPC moves from one *reachable* state to the next. The *reachability graph* of an EPC represents how states can be reached from each other. A state that is not a final state, but from which no other state can be reached is called a *deadlock*. The notion of a final state will be formally defined in Section 3.6.

<sup>1</sup>This state representation based on arcs reflects the formalization of *Kindler* [Kin06] and can be related to arcs between tasks in YAWL that are interpreted as implicit conditions [AH05]. Other approaches assign tokens to the nodes of an EPC, e.g., [Rum99].

### 3.2 EPC Formalization Problems

We have briefly stated that the OR-join synchronizes all active incoming branches. This bears a non-trivial problem: if there is a token on one incoming arc, does the OR-join have to wait or not? Following the informal semantics of EPCs, it is only allowed to fire if it is not possible that a token might arrive at the other incoming arcs (see [NR02]). In the following subsection, we will show what the formal implications of these intended semantics are. Before that, we introduce some example EPCs. The discussion of them raises some questions that are not answered yet. Instead, we revisit them afterwards to illustrate the characteristics of the different formalization approaches.

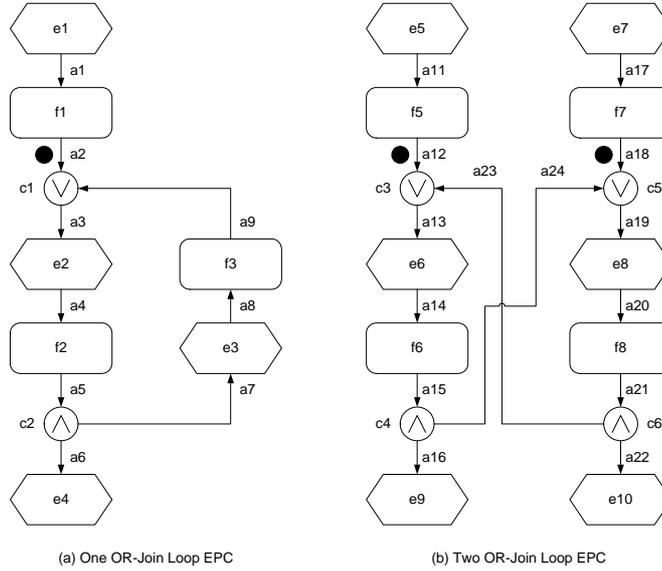


Figure 2: EPCs (a) with one OR-join and (b) with two OR-joins on the loop

Figure 2(a) shows an EPC with an OR-join on a loop. There is a token on arc  $a_2$  from function  $f_1$  to the OR-join  $c_1$ . The question is whether  $c_1$  can fire. If it could fire, then it would be possible that a token may arrive at arc  $a_9$  from  $f_3$  to the join. This would imply that it should wait and not fire. On the other hand, if it must wait, it is not possible that a token might arrive at  $a_9$ . Figure 2(b) depicts an EPC with two OR-joins,  $c_3$  and  $c_5$ , on a loop which are both enabled (cf. [ADK02]). Here, the question is whether both can fire or none of them. Since the situation is symmetric it seems not reasonable that only one should be allowed to fire.

The situation might be even more complicated as Figure 3 illustrates (cf. [Kin06]). This EPC includes a loop with three OR-joins:  $c_1, c_3$ , and  $c_5$ . All of them are enabled. Following the informal semantics the first OR-join  $c_1$  is allowed to fire if it is not possible for a token to arrive on arc  $a_{21}$  from the AND-split  $c_6$ . To put it differently, if  $c_1$  is allowed to fire, it is possible for a token to arrive on arc  $a_7$  that leads to the OR-join  $c_3$ . Furthermore,

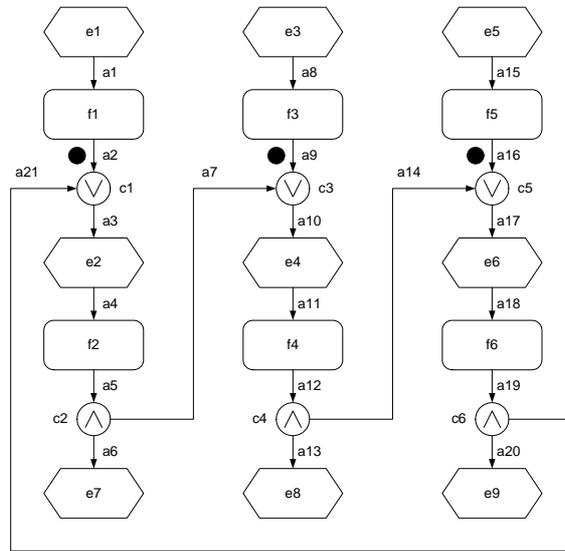


Figure 3: EPCs with three OR-joins on the loop

the OR-join  $c5$  could eventually fire. Finally, the first OR-join  $c1$  would have to wait for that token before firing. In short, if  $c1$  could fire, it would have to wait. One can show that this holds also the other way around: if it could not fire, it would not have to wait. Furthermore, this observation also holds for the two other OR-joins. In the subsequent section, we will discuss whether this problem can be resolved.

Refinement is another issue related to OR-joins. Figure 4 shows two versions of an EPC process model. In Figure 4(a) there is a token on  $a7$ . The subsequent OR-join  $c2$  must wait for this token and synchronize it with the second token on  $a5$  before firing. In Figure 4(b) the sequence  $e3$ - $a7$ - $f3$  is refined with a block of two branches between an OR-split  $c3a$  and an OR-join  $c3b$ . The OR-join  $c2$  is enabled and should wait for the token on  $a7f$ . The question here is whether such a refinement might change the behavior of an OR-join. Figure 4 is just one simple example. The answer to this question may be less obvious if the refinement is introduced in a loop that already contains an OR-join. Figure 5 shows a respective case of an OR-join  $c1$  on a loop that is refined with an OR-Block  $c3a$ - $c3b$ . One would expect that the EPC of Figure 4(a) exhibits the same behavior as the one in (b). In the following subsection, we will discuss these questions from the perspective of different formalization approaches.

### 3.3 Approaches to EPC Semantics Formalization

The transformation to Petri nets plays an important role in early formalizations of EPC semantics. In *Chen and Scheer* [CS94] the authors define a mapping to colored Petri nets

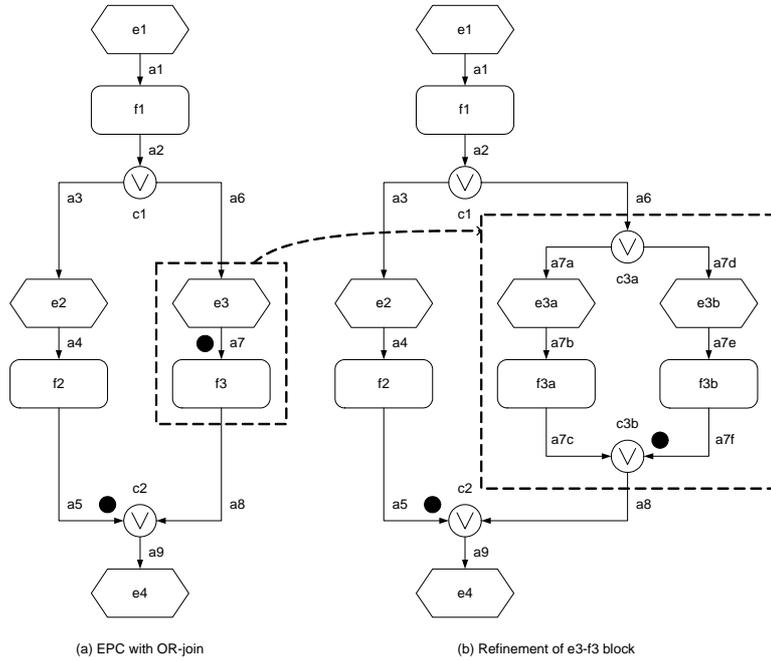


Figure 4: EPC refined with an OR-Block

and address the non-local synchronization behavior of OR-joins. This formalization builds on the assumption that an OR-split always matches a corresponding OR-join. The colored token that is propagated from the OR-split to the corresponding OR-join signals which combination of branches is enabled. Furthermore, the authors describe the state space of some example EPCs by giving reachability graphs. Yet, this first Petri net semantics for EPCs has mainly two weaknesses. First, a formal algorithm to calculate the state space is missing. Second, the approach is restricted to EPCs with matching OR-split and -join pairs. Therefore, this approach does not provide semantics for the EPCs shown in figures 2 and 3. Even though the approach is not formalized in all its details, it should be able to handle the refined EPC of Figure 4(b) and the inner OR-join  $c3b$  in Figure 4(b).

The transformation approach by *Langner, Schneider, and Wehler* [LSW98] maps EPCs to Boolean nets in order to define formal semantics. Boolean nets are a variant of colored Petri nets whose token colors are 0 (negative token) and 1 (positive token). Connectors propagate both negative and positive tokens according to its logical type. This mechanism is able to capture the non-local synchronization semantics of the OR-join similar to dead-path elimination in workflow systems (see [LA94]). The XOR-join only fires if there is one positive token on incoming branches and a negative token on all other incoming branches. Otherwise it blocks. A drawback of this semantics definition is that the EPC syntax has to be restricted: arbitrary structures are not allowed. If there is a loop it must have an XOR-join as entry point and an XOR-split as exit point. This pair of connectors in a cyclic structure is mapped to one place in the resulting Boolean net. As a consequence,

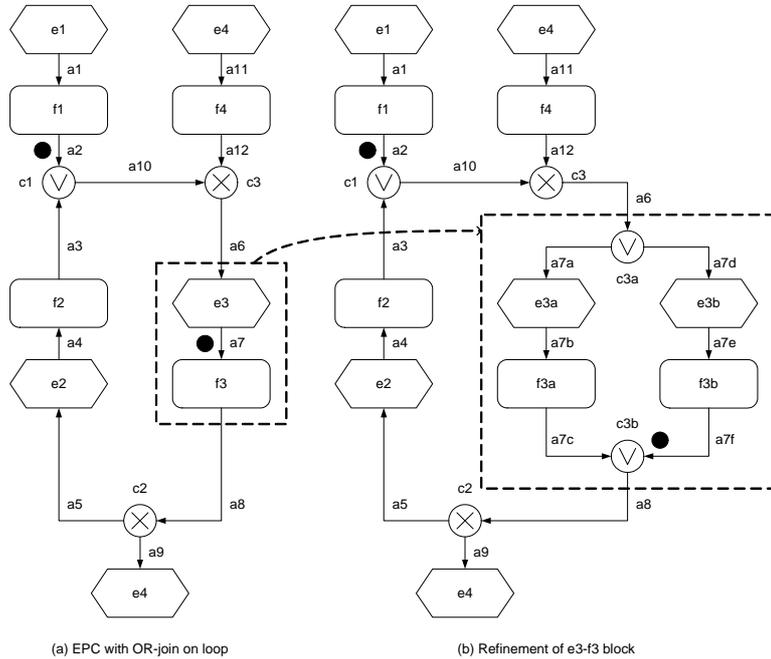


Figure 5: Cyclic EPC refined with an OR-Block

this approach does not provide semantics for the EPCs in Figures 2 and 3. Yet, it is able to deal with any kind of OR-join that is not an entry or an exit to a loop. Accordingly, the Boolean nets define the expected semantics of the refined EPC of Figure 4(b) and of the inner OR-Block introduced as a refinement in Figure 4(b).

*Van der Aalst* [Aal99] presents an approach to derive Petri nets from EPCs, but he does not give a mapping for OR-connectors because of the semantical problems illustrated in Section 3.2. This mapping provides clear semantics for XOR- and AND-connectors as well as for the OR-split, but since the OR-join is not formalized the approach does not provide semantics for the EPCs of Figures 2 to 5. *Dehnert* presents an extension of this approach by mapping the OR-join to a Petri net block [DA04]. Since the resulting Petri net block may or may not necessarily synchronize multiple tokens at runtime (i.e., a non-deterministic choice), its state space is larger than the actual state space with synchronization. Based on the so-called relaxed soundness criterion it is possible to check whether a join should synchronize (cf. [DA04]).

In [Rit00] *Rittgen* discusses the OR-join. He proposes to distinguish three types of OR-joins already on the syntactic level: every-time, first-come, and wait-for-all. The every-time OR-join basically reflects XOR-join behavior; the first-come OR-join passes the first incoming token and blocks afterwards; and the wait-for-all OR-join depends on a matching split similar to the approach of *Chen and Scheer*. This proposal could provide a semantics for the example EPCs of Figures 2 to 5. If we assume an every-time semantics, all OR-joins of the example EPCs could fire. While the loops would not block in this case, there

would be no synchronization at all which contradicts the intended OR-join semantics. If the OR-joins behave according to the first-come semantics, all OR-joins could fire. Yet, there would also be no synchronization and the loops could be run only once. If the OR-joins had wait-for-all semantics, we would have the same problems as before with the loops. Altogether, the proposal by Rittgen does not provide a satisfactory solution to the formalization problem.

*Nüttgens and Rump* [NR02] define a transition relation for EPCs that addresses also the non-local semantics of the OR-join, yet with a problem: the transition relation for the OR-join refers to itself under negation. *Van der Aalst, Desel, and Kindler* show, that a fixed point for this transition relation does not always exist [ADK02]. They present an example to prove the opposite: an EPC with two OR-joins on a circle that wait for each other as depicted in Figure 2(b). This vicious circle is the starting point for the work of *Kindler* towards a sound mathematical framework for the definition of non-local semantics for EPCs. In a series of papers [Kin06], *Kindler* elaborates on this problem in detail. The technical problem is that for the OR-join transition relation  $R$  depends upon  $R$  itself in negation. Instead of defining one transition relation, he considers a pair of transition relations  $(P, Q)$  on the state space  $\Sigma$  of an EPC and a monotonously decreasing function  $R : 2^{\Sigma \times N \times \Sigma} \rightarrow 2^{\Sigma \times N \times \Sigma}$ . Then, a function  $\varphi((P, Q)) = (R(Q), R(P))$  has a least fixed point and a greatest fixed point.  $P$  is called pessimistic transition relation and  $Q$  optimistic transition relation. An EPC is called *clean*, if  $P = Q$ . For most EPCs, this is the case. Some EPCs such as the vicious circle EPC are *unclean* since the pessimistic and the optimistic semantics do not coincide. Moreover, *Cuntz* provides an example of a clean EPC which is refined with another clean EPC and becomes unclean [Cun04, p.45]. *Kindler* also shows that there are even acyclic EPCs that are unclean (see [Kin06, p.38]). Furthermore, *Cuntz and Kindler* present optimizations for an efficient calculation of the state space of an EPC and a respective prototype implementation called EPC Tools [CK05]. EPC Tools also offers a precise answer to the questions about the example EPCs of Figures 2 to 5.

- Figure 2(a): For the EPC with one OR-join on a loop, there is a fixed point and the connector is allowed to fire.
- Figure 2(b): The EPC with two OR-joins on a loop is not clean. Therefore, it is not clear whether the optimistic or the pessimistic semantics should be considered.
- Figure 3: The EPC with three OR-joins is also not clean, i.e., the pessimistic deviates from the optimistic semantics.
- Figure 4(a): The OR-join  $c2$  must wait for the second token on  $a7$ .
- Figure 4(b): The OR-join  $c2$  must wait for the second token on  $a7f$ .
- Figure 5(a): The OR-join  $c1$  must wait for the second token on  $a7$ .
- Figure 5(b): The OR-join  $c1$  is allowed to fire, the second OR-join  $c2$  in the OR-block must wait.

Even though the approach by *Kindler* provides semantics for a large subclass of EPCs, i.e. clean EPCs, there are some cases like the EPCs of Figure 2(b) and 3 that do not have semantics. The theorem by *Kindler* proves that it is not possible to give these EPCs semantics as long as the transition relation is defined with a self-reference under negation. Furthermore, such a semantics definition may imply some unexpected results, e.g. if an

EPC such as that of Figure 5(a) behaves differently than its refinement as given in Figure 5(b).

*Van der Aalst and Ter Hofstede* defined a workflow language called YAWL [AH05] which also offers an OR-join with non-local semantics. As *Mendling, Moser, and Neumann* propose a transformation semantics for EPCs based on YAWL [MMN06], we discuss in the following paragraph how the OR-join behavior is formalized in YAWL. In [AH05], the authors propose a definition of the transition relation  $R(P)$  with a reference to a second transition relation  $P$  that ignores all OR-joins. A similar semantics that is calculated on history-logs of the process is proposed by *Van Hee, Oanea, Serebrenik, Sidorova, and Voorhoeve* in [HOS<sup>+</sup>06]. The consequence of this definition can be illustrated using the example EPCs.

- Figure 2(a): The single OR-join on the loop can fire.
- Figure 2(b): The two OR-joins on the loop can fire.
- Figure 3: The three OR-joins on the loop can fire.
- Figure 4(a): The OR-join  $c2$  must wait for the second token between  $e3$  and  $f3$ .
- Figure 4(b): Both OR-joins can fire.
- Figure 5(a): The OR-join  $c1$  must wait for the second token between  $e3$  and  $f3$ .
- Figure 5(b): Both OR-joins can fire.

*Kindler* criticizes that each choice for defining  $P$  “appears to be arbitrary or ad hoc in some way” [Kin06] and uses the pair  $(P, Q)$  instead. The example EPCs illustrate that the original YAWL semantics provide for a limited degree of synchronization. Consider for example the vicious circle EPC with three OR-joins: all are allowed to fire, but if one does so, the subsequent OR-join has to wait. Furthermore, the refined EPCs exhibit different behavior than their unrefined counterparts since OR-joins are ignored, i.e., they are considered to be not able to fire.

*Wynn, Edmond, van der Aalst, and ter Hofstede* illustrate that these OR-join semantics in YAWL exhibit some non-intuitive behavior when OR-join depend upon each other [WEAH05]. Therefore, they present a novel approach based on a mapping to Reset nets. If an OR-join can fire (i.e.  $R(P)$ ) is decided depending on (a) a corresponding Reset net (i.e.  $P$ ) that treats all OR-joins as XOR-joins<sup>2</sup> and (b) a predicate called *superM* that hinders firing if an OR-join is on a directed path from another enabled OR-join. In particular, the Reset net is evaluated using backward search techniques that grant coverability to be decidable (see e.g. [FS01]). A respective verification approach for YAWL nets is presented in [WAHE06]. Using these semantics, the example EPCs behave as follows:

- Figure 2(a): The single OR-join on the loop can fire since *superM* evaluates to false and hence no more tokens can arrive at  $c_1$ .
- Figure 2(b): The two OR-joins are not enabled since *superM* evaluates to true because if the respectively other OR-join is replaced by an XOR-join an additional token may arrive.

---

<sup>2</sup>In fact, [WEAH05] proposes two alternative treatments for the “other OR-joins” when evaluating an OR-joins: treat them either as XOR-joins (optimistic) or as AND-joins (pessimistic). However, the authors select the optimistic variant because the XOR-join treatment of other OR-joins matches more closely the informal semantics of the OR-join.

- Figure 3: The three OR-joins are not enabled because if one OR-join assumes the other two to be XOR-joins then this OR-join has to wait.
- Figure 4(a): The OR-join  $c2$  must wait for the second token on  $a7$ .
- Figure 4(b): The OR-join  $c2$  must wait for the second token on  $a7f$ .
- Figure 5(a): The OR-join  $c1$  must wait for the token on  $a7$ .
- Figure 5(b): The OR-join  $c1$  must wait because if  $c3b$  is assumed to be an XOR-join a token may arrive via  $a3$ . The OR-join  $c3b$  must also wait because if  $c1$  is an XOR-join another token may move to  $a7c$ .

The novel approach based on Reset nets provides interesting semantics but in some cases also leads to deadlocks.

OR-join semantics	Limitations
[CS94]	OR-join must match OR-split
[LSW98]	Joins as loop entry undefined
[Rit00] every-time	missing synchronization
[Rit00] first-come	OR-join can block
[Rit00] wait-for-all	OR-join as loop entry undefined
[Kin06]	EPC can be unclean
[AH05]	limited synchronization
[WAHE06]	OR-join may block

Table 1: Overview of EPC semantics and their limitations

Table 1 summarizes existing work on the formalization of the OR-join. Several early approaches define syntactical restrictions such as OR-splits to match corresponding OR-joins or models to be acyclic (see [CS94, LSW98, Rit00]). Newer approaches impose little or even no restrictions (see [Kin06, AH05, WAHE06]), but exhibit unexpected behavior for OR-block refinements on loops with further OR-joins on it. The solution based on Reset nets seems to be most promising from the intuition of its behavior. Yet, it requires extensive calculation effort since it depends upon backward search to decide coverability (Note that reachability is undecidable for reset nets illustrating the computational complexity of the OR-join in the presence of advanced routing constructs). In the following subsection, we propose a novel approach that addresses the refinement problems of the Reset nets semantics and that provides a more efficient solution since all OR-join decisions can be taken with local knowledge.

### 3.4 A Novel Approach towards EPC Semantics

In this subsection, we introduce a novel formalization of the EPC semantics. The principal idea of these semantics lends some concepts from *Langner, Schneider, and Wehler* [LSW98] and adapts the idea of Boolean nets with true and false tokens in an appropriate manner. Furthermore, we utilize the notations of *Kindler* [Kin06] whenever possible and modify them where needed. The transition relation that we will formalize afterwards depends on the state and the context of an EPC. The *state* of an EPC is basically an assign-

ment of positive and negative tokens to the arcs. Positive tokens signal which functions have to be carried out in the process, negative tokens indicate which functions are to be ignored. The transition rules of AND- and OR-connectors are adopted from the Boolean nets formalization which facilitates synchronization of OR-joins in structured blocks. In order to allow for a more flexible utilization of XOR-connectors in cyclic structure, we modify and extend the approach of Boolean nets in three ways:

1. XOR-splits produce positive tokens on their output arcs, but no negative tokens. XOR-joins fire each time there is a positive token on an incoming arc. This mechanism provides the expected behavior in both structured XOR-loops and structured XOR-blocks.
2. In order to signal OR-joins that it is not possible to have a positive token on an incoming branch, we define the *context* of an EPC. The context assigns a status of *wait* or *dead* to each arc of an EPC. A wait context indicates that it is still possible that a positive token might arrive; a dead context status means that no positive token can arrive anymore. For example, XOR-splits produce a dead context on those output branches that are not taken and a wait context on the output branch that receives a positive token. A dead context at an input arc is then used by an OR-join to determine whether it has to synchronize with further positive tokens or not.
3. The propagation of context status and state tokens is arranged in a four phase cycle: (a) dead context, (b) wait context, (c) negative token, and (d) positive token propagation.
  - (a) In this phase, all *dead context* information is propagated in the EPC until no new dead context can be derived.
  - (b) Then, all *wait context* information is propagated until no new wait context can be derived. It is necessary to have two phases (i.e., first the dead context propagation and then the wait context propagation) in order to avoid infinite cycles of context changes (details below).
  - (c) After that, all *negative tokens* are propagated until no negative token can be propagated anymore. This phase can neither run into an endless loop (details below).
  - (d) Finally, one of the enabled nodes is selected and propagates *positive tokens* leading to a new iteration of the four phase cycle.

In the following subsection, we first give an example to illustrate the behavior of the EPC semantics before defining state, context, and the transition relation. Then, we define the initialization of an EPC. After that, we present the transition relations for the two phases of dead context and wait context propagation. Then, we discuss the termination of these phases and why a separation in dead context and wait context propagation is necessary. Subsequently, we define the transition relation for the phase of negative token propagation and its termination. Finally, the positive token propagation is presented.

**Revisiting the cyclic EPC refined with an OR-block** Figure 6 revisits the example of the cyclic EPC refined with an OR-block that we introduced as Figure 5 in Section 3.2.

In Figure 6(a) there are two positive tokens on the arcs  $a2$  and  $a12$ . The context status is indicated by a letter next to the arc:  $w$  for wait and  $d$  for dead. In (a) all arcs are in a wait context status which implies that the OR-join  $c1$  is not allowed to fire, but has to synchronize with positive and negative tokens that might arrive on arc  $a3$ . The XOR-join is allowed to fire without considering the second arc  $a10$ . In (b) the OR-split  $c3a$  has fired (following the execution of  $c3$ ) and happened to produce a positive token on  $a7a$  and a negative token on  $a7d$ . Accordingly, the context of  $a7d$  is changed to dead. This dead context is propagated down to arc  $a7f$ . The rest of the context remains unchanged. The state shown in (b) is followed by (c) where the positive and the negative tokens are synchronized at the connector  $c3b$  and one positive token is produced on the output arc  $a8$ . Please note that the OR-join  $c3b$  does not synchronize with the other OR-join  $c1$  that is also on the loop. In the Kindler and the Reset nets semantics,  $c3b$  would have to wait for the token from  $a2$ . Here, the wait context propagation is blocked by the negative token. In (d) the XOR-split  $c2$  produces a positive token on  $a9$  and a dead context on  $a5$ . This dead context is propagated to  $a3$  in the dead context propagation phase, but not further. In the wait context propagation phase, this dead context is changed to wait which is propagated from  $c2$ . As a consequence, the OR-join  $c1$  is not enabled. This example permits two observations. First, the context propagation blocks OR-joins that are entry points to a loop in a wait position since the self-reference is not resolved. Second, the XOR-split produces a dead context, but not a negative token. The disadvantage of producing negative tokens would be that the EPC was flooded with negative tokens if an XOR-split was used as an exit of a loop. These tokens would give downstream joins the wrong information about the state of the loop since it would be still live. An OR-join could then synchronize with a negative token while a positive token was still in the loop. In contrast to that, the XOR-split as a loop exit produces a dead context. Since there is a positive token in the loop, it is overwritten in the wait context propagation phase. Downstream OR-joins then have the correct information that there are still tokens to wait for.

**Definition of State and Context** We define both state and context as an assignment to the arcs. The EPC transition relation defines which state and/or context changes are allowed for a given state/context. We will first illustrate this before defining state and context formally. As we define the EPC semantics as a transition system based on state and context, we refer to it as a state-context-system.

**Definition 8** (State and Context). For an  $EPC = (E, F, C, l, A)$  the mapping  $\sigma : A \rightarrow \{-1, 0, +1\}$  is called a state (or marking) of an  $EPC$ . The positive token captures the state as it is observed from outside the process. It is represented by a black circle. The negative token depicted by a white circle with a minus on it has a similar semantics as the negative token in the Boolean nets formalization. Arcs with no state tokens on them have no circle depicted. Furthermore, the mapping  $\kappa : A \rightarrow \{wait, dead\}$  is called a context of an  $EPC$ . A wait context is represented by a  $w$  and a dead context by a  $d$  next to the arc.

**Initial and Final State** The initial state is the starting point for applying an iteration of the four phase cycle. In [Rum99] the initial state of an EPC is specified as a marking that assigns tokens to one, some, or all start events. While such a definition contains enough

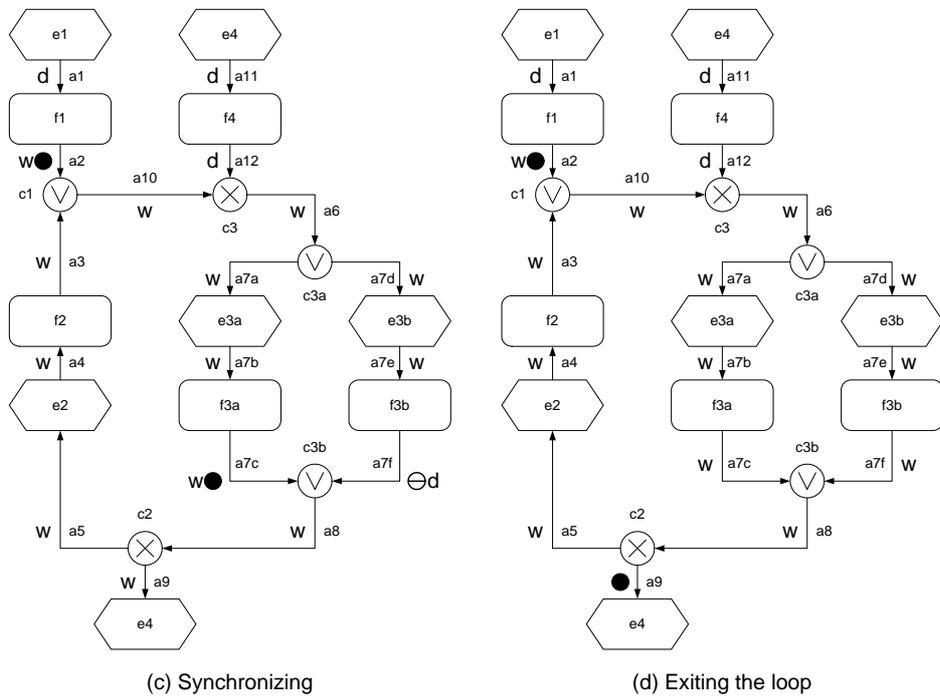
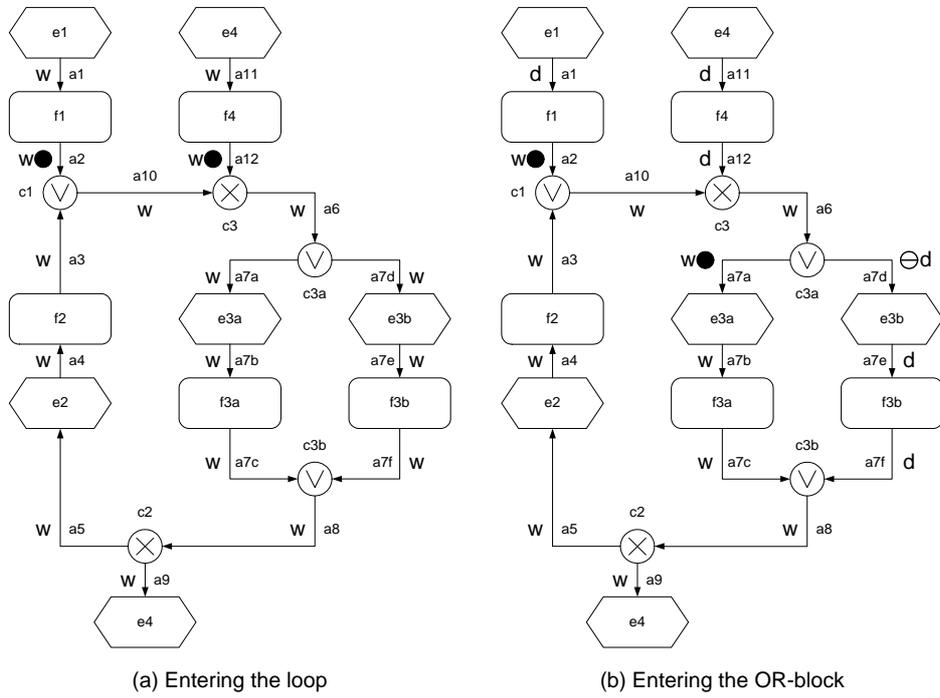


Figure 6: Example of EPC state changes

information for verification purposes, e.g. by the bundling of start and end events with OR-connectors as proposed in [MMN06], it does not provide executable semantics according to the original definition of EPCs. As pointed out in [Rit00], it is not possible to equate the triggering of a single start event with the instantiation of a new process. This is because EPC start events do not only capture alternative instantiation, but also external events that influence the execution of a running EPC (cf. [CS94]). In our approach, we assume that state and context is known for each of the start arcs. A respective formalization of initial and final state is given in Definitions 11 and 12. In the following, we describe the transition relations of each node  $n \in E \cup F \cup C$  in the phases of dead context, wait context, negative and positive token propagation.

**Phase 1: Transition Relation for Dead Context Propagation** The transition relation for dead context propagation defines rules for deriving a dead context if one or more input arcs of a node have a dead context status. Figure 7 gives an illustration of the transition relation. *Please note that the figure does not depict the fact that the rules for dead context propagation can only be applied if the respective output arc does not hold a positive or a negative token.* Concrete tokens override context information, e.g., an arc with a positive token will always have a wait context. Rules (a) and (b) indicate that if an input arc of a function or an event is dead, then also the output arc has to have a dead context status. Rule (c) represents that each split-connector propagates a dead context to its output arcs. These transition relations formalize the observation that if an input arc cannot be reached anymore, also its output arcs cannot be reached anymore (unless they already hold positive or negative tokens). The AND-join requires only one dead context status at its input arcs to replicate it at its output arc, see (d). XOR- and OR-joins propagate a dead context if all input arcs are dead, see (e) and (f). It is important to note that a dead context is propagated until it reaches a node where one of the output arcs holds a token or an (X)OR-join where one of the inputs has a wait context.

**Phase 2: Transition Relation for Wait Context Propagation** The transition relation for wait context propagation defines rules for deriving a wait context if one or more input arcs of a node have a wait context status. Figure 8 gives an illustration of the transition relation. *All transitions can only be applied if the respective output arc does not hold a positive or a negative token.* Rules (a) and (b) show that if an input arc of a function or an event has a wait context, then also the output arc has to have a wait context status. Rule (c) represents that each split-connector propagates a wait context to its output arcs. The AND-join requires all inputs to have a wait context status in order to reproduce it at its output arc, see (d). XOR- and OR-joins propagate a wait context if one of their input arcs has a wait context, see (e) and (f). Similar to the dead context propagation, the wait context is propagated until it reaches a node where one of the output arcs holds a token or an AND-join where one of the inputs has a dead context.

**Observations on Context Propagation** The transition relations of context propagation permit the following observations:

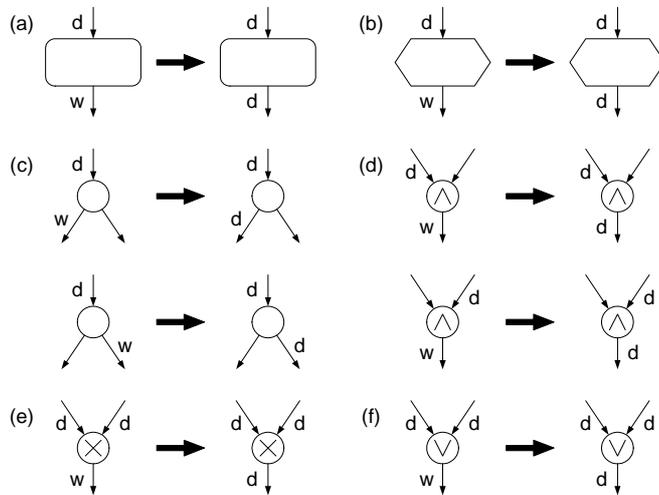


Figure 7: Transition relation for dead context propagation

- *Context changes terminate:* It is intuitive that context propagation cannot run into an infinite loop. It is easy to verify that the first two phases indeed stop. The propagation of dead context stops because the number of arcs is finite, i.e., the number of arcs is an upper bound for the number of times the rules in Figure 7 can be applied. A similar argument applies to the propagation of wait context. As a consequence, the context change phase will always terminate and enable the consideration of new state changes in the subsequent phase.
- *State tokens block context propagation:* The transition relations for context propagation require that the output arcs to be changed do not hold any state token, i.e., arcs with a positive token always have a wait context and arcs with a negative token always have a dead context.
- *Context propagating elements:* Functions, events, and split nodes reproduce the context that they receive at their input arcs.
- *OR- and XOR-joins:* Both these connectors produce a *wait* context if at least one of the input arcs has a *wait* context. A *dead* context is produced if all inputs are *dead*.
- *AND-joins:* AND-joins produce *wait* context status only if all inputs are *wait*. Otherwise, the output context is set to *dead*.

Figure 9 illustrates the need to perform context propagation in two separate phases and not together in one phase. If there are context changes (a) at  $i1$  to and  $i2$  the current context enables the firing of the transition rules for both connectors producing a *dead* context status in  $a1$  and a *wait* context status in  $a3$ . This leads to a new context in (b) with an additional *dead* context status in  $a2$  and a new *wait* context status in  $a4$ . Since both arcs from outside the loop to the connectors are marked in such a way that incoming context changes on the other arc is simply propagated, there is a new context in (c) with a *wait* status in  $a1$  and a *dead* context status in  $a3$ . Note that this new context can be propagated

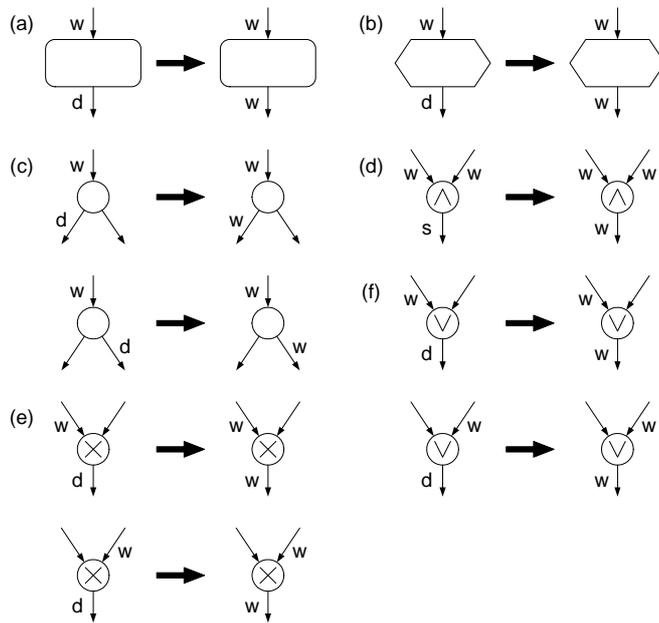


Figure 8: Transition relation for wait context propagation

and this way the initial situation is reached. This can be repeated again and again. Without a sequence of two phases the transitions may continue infinitely and the result may be non-defined.

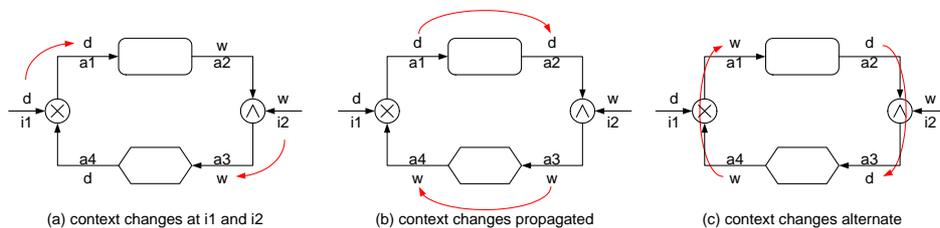


Figure 9: Situation of unstable context changes without two phases

**Phase 3: Transition Relation for Negative Token Propagation** Negative tokens can result from branches that are not executed after OR-joins or start events. The transition relation for negative token propagation includes four firing rules that consume and produce negative tokens. Furthermore, the output arcs are set to a dead context. Figure 10 gives an illustration of the transition relation. *All transitions can only be applied if all input arcs hold negative tokens and if there is no positive token on the output arc.*

The propagation of negative tokens for an *EPC* terminates. First, we have to note that the

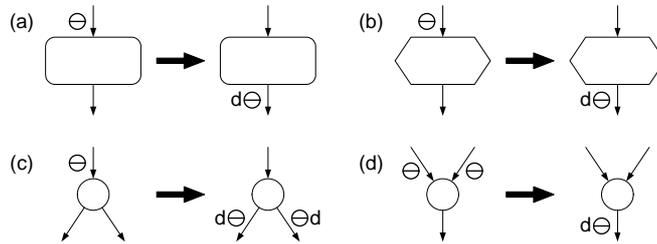


Figure 10: Transition Relation for Negative Token Propagation

number of negative and positive tokens on an arc is limited to one. It is a prerequisite for an infinite propagation that there is a cyclic structure in the process in which the negative token runs into an infinite loop. To this loop there must be an entry point, i.e. a join connector that has an input arc from outside the cycle. According to the transition relation of negative tokens, this join can only produce a negative token on its output arc if it has negative tokens on all inputs. Since there is one arc from an acyclic structure, there is a finite number  $a$  of arcs from one or multiple start arcs to this arc limiting the number of negative tokens that can arrive to the number of arcs leading to the join. Therefore, the negative token in the cyclic structure cannot pass the join more than  $a$  times.

**Phase 4: Transition Relation for Positive Token Propagation** The transition relation for positive token propagation specifies firing rules that consume negative and positive tokens from the input arcs of a node to produce positive tokens on its output arcs. Figure 11 gives an respective illustration. Rules (a) and (b) show that functions and events consume positive tokens from the input arc and propagate them to the output arc. Furthermore, and this holds for all rules, consuming a positive token from an arc implies setting this arc to a dead context status. Rules (c) and (d) illustrate that AND-splits consume one positive token and produce one on each output arc while AND-joins synchronize positive tokens on all input arcs to produce one on the output arc. Rule (e) depict the fact that XOR-splits forward positive tokens to one of their output arcs. In contrast to the Boolean net formalization, they do not produce negative tokens, but a dead context on the output arcs that do not receive the token. Correspondingly, XOR-joins (f) propagate each incoming positive token to the output arc, no matter what the context or the state of the other input arcs is. The OR-split (g) produces positive tokens on those output arcs that have to be executed and negative tokens on those that are ignored. Note that the OR-join is the only construct that may introduce negative tokens (apart from start events without an initial token). Rule (h) shows that on OR-join can only fire either if it has full information about the state of its input arcs, i.e., each input has a positive or a negative token, or all arcs that do not hold a token are in a dead context. Finally, each output arc that receives a negative token is set to a dead context and each that gets a positive token is set to a wait context.

This semantics definition based on state and context results in the following behavior for the examples of Section 3.

- Figure 2(a): The single OR-join on the loop produces a wait context at  $a_9$ . There-

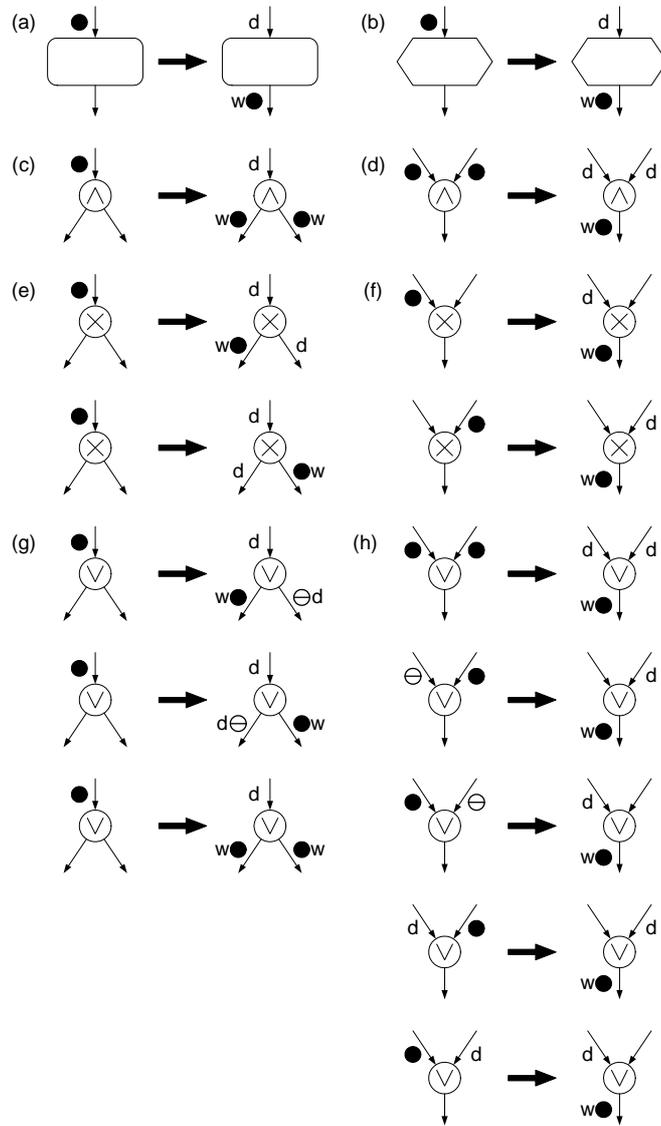


Figure 11: Transition Relation for Positive Token Propagation

fore, it is blocked.

- Figure 2(b): The two OR-joins produce a wait context at  $a23$  and  $a24$ . Therefore, they are both blocked.
- Figure 3: The three OR-joins are blocked due to a wait context at  $a7$ ,  $a14$ , and  $a21$ .
- Figure 4(a): The OR-join  $c2$  must wait for the second token on  $a7$ .
- Figure 4(b): The OR-join  $c2$  must wait for the second token on  $a7f$ .
- Figure 5(a): The OR-join  $c1$  must wait for the token on  $a7$ .
- Figure 5(b): The OR-join  $c1$  must wait for the token on  $a7$ . The OR-split  $c3a$  produces a negative token on  $a7c$  such that  $c3b$  can fire.

### 3.5 Transition Relation of EPCs

We define the transition relation of EPCs based on state and context mappings  $\sigma$  and  $\kappa$ . In contrast to Petri nets we distinguish the terms marking and state: the term marking refers to state and context collectively.

**Definition 9** (Marking of an EPC). For an *EPC* the mapping  $m : A \rightarrow \{-1, 0, +1\} \times \{wait, dead\}$  is called a marking. The set of all markings  $M$  of an EPC is called marking space with  $M \subseteq A \times \{-1, 0, +1\} \times \{wait, dead\}$ . The projection of a given marking  $m$  to a subset of arcs  $S \subseteq A$  is referred to as  $m_S$ . The marking  $m_a$  of an arc  $a$  can be written as  $m_a = (\kappa(a) + \sigma(a)) \cdot a$ .

The marking of the arcs  $a_1, a_2, a_3 \in A$  can then be written as e.g.  $(d - 1) \cdot a_1 + w \cdot a_2 + (w + 1) \cdot a_3$ . Furthermore, we define the transition relation, the initial marking, and the final marking of an EPC.

**Definition 10** (Transition Relation of an EPC). For an *EPC* the transition relation is a triple  $R = M \rightarrow N \times M$  where  $M$  refers to the marking space of the EPC. A single transition  $(m, n, m') \in R$  represents a marking change of an EPC. Furthermore, we define the following notations:

- $m_1 \xrightarrow{n} m_2$ : node  $n$  is enabled in marking  $m_1$  and its firing results in  $m_2$ .
- $m_1 \rightarrow m_2$ : there exists a node  $n$  such that  $m_1 \xrightarrow{n} m_2$ .
- $m_1 \xrightarrow{\tau} m_q$ : the firing sequence  $\tau = n_1 n_2 \dots n_q$  produces from marking  $m_1$  the new state  $m_q$  with  $m_1 \xrightarrow{n_1} m_2, m_2 \xrightarrow{n_2} \dots \xrightarrow{n_q} m_q$ .
- $m_1 \xrightarrow{*} m_q$ : there exists a sequence  $\tau$  such that  $m_1 \xrightarrow{\tau} m_q$ . In this case  $m_q$  is called *reachable* from  $m_1$ .

**Definition 11** (Initial Marking of an EPC). For an *EPC* an initial marking  $i \in M$  is defined as a state and context mapping that fulfills the following constraints<sup>3</sup>:

- $\exists a_s \in A_s : \sigma(a_s) = +1,$

<sup>3</sup>Note that the state is given in terms of arcs. Intuitively, one can think of start event holding positive or negative tokens. However, the corresponding arc will formally represent this token.

- $\forall a_s \in A_s: \sigma(a_s) \in \{-1, +1\}$ ,
- $\forall a_s \in A_s: \kappa(a_s) = \textit{wait}$  if  $\sigma(a_s) = +1$  and  $\kappa(a_s) = \textit{dead}$  if  $\sigma(a_s) = -1$ , and
- $\forall a \in A_{int} \cup A_e: \kappa(a) = \textit{wait}$  and  $\sigma(a) = 0$ .

The set of all initial markings is referred to as  $I \subseteq M$ .

**Definition 12** (Final Marking of an EPC). For an *EPC* and a final marking  $o \in M$  is defined as a marking that fulfills the following constraints:

- $\exists a_e \in A_e: \sigma(a_e) = +1$  and
- $\forall a \in A_{int} \cup A_s: \sigma(a) \leq 0$ .

The set of all final markings is referred to as  $O \subseteq M$ .

### 3.6 Soundness of EPCs

Soundness is an important correctness criterion for business process models first introduced in [Aal97]. The original soundness property is defined for a Workflow net, a Petri net with one source and one sink, and requires that (i) for every state reachable from the source, there exists a firing sequence to the sink (option to complete); (ii) the state with a token in the sink is the only state reachable from the initial state with at least one token in it (proper completion); and (iii) there are no dead transitions [Aal97]. For EPCs, this definition cannot be used directly since EPCs may have multiple start and end events. Based on the definitions of the initial and final state of an EPC, we define soundness of an EPC analogously to soundness of Workflow nets [Aal97].

**Definition 13** (Soundness of an EPC). An *EPC* is sound if there is a set of initial markings  $I$  such that:

- For each start-arc  $a_s$  there exists an initial marking  $i \in I$  where the arc (and hence the corresponding start event) holds a positive token. Formally:  
 $\forall a_s \in A_s: \exists i \in I: \sigma(a_s) = +1$
- For every marking  $m$  reachable from an initial state  $i \in I$ , there exists a firing sequence leading from marking  $m$  to a final marking  $o \in O$ . Formally:  
 $\forall i \in I: \forall m \in M (i \xrightarrow{*} m) \Rightarrow \exists o \in O (m \xrightarrow{*} o)$
- The final markings  $o \in O$  are the only markings reachable from a marking  $i \in I$  such that there is no node that can fire. Formally:  
 $\forall m \in M: \nexists m' (m \rightarrow m') \Rightarrow m \in O$

Given this definition, the EPCs of Figures 2 and Figure 3 are not sound since the OR-joins block each other. Both EPCs of Figure 4 are sound. Finally, both EPCs of Figure 5 are not sound because if the token at  $a7$  or  $a7f$ , resp., exits the loop, the OR-join  $c1$  is blocked.

## 4 Summary

In this paper, we revisited existing work on formalization of EPCs and OR-joins in general. We found that one of the disadvantages of these solutions is that introducing an OR-block in the process might yield unexpected behavior. Against this background, we presented a concept for the definition of EPC semantics based on state and context. In future research, we aim to provide formal semantics for the ideas presented in this paper. We also aim to support it with efficient verification techniques and apply it to a range of real-life EPCs to see if these semantics match the intuition of the modeler.

## References

- [Aal97] W. M. P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *LNCS*, pages 407–426, 1997.
- [Aal99] W. M. P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
- [ADK02] W. M. P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In M. Nüttgens and F. J. Rump, editor, *Proc. of the 1st GI-Workshop EPK 2002*, pages 71–79, 2002.
- [AH05] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
- [CK05] N. Cuntz and E. Kindler. On the semantics of epcs: Efficient calculation and simulation. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management, 3rd Int. Conference BPM 2005*, volume 3649 of *LNCS*, pages 398–403, 2005.
- [CS94] R. Chen and A. W. Scheer. Modellierung von Prozessketten mittels Petri-Netz-Theorie. Heft 107, Institut für Wirtschaftsinformatik, Saarbrücken, Germany, 1994.
- [Cun04] Nicolas Cuntz. Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Master’s thesis, Univ. of Paderborn, June 2004. (in German).
- [DA04] Juliane Dehnert and Wil M. P. van der Aalst. Bridging The Gap Between Business Models And Workflow Specifications. *Int. J. Cooperative Inf. Syst.*, 13(3):289–332, 2004.
- [FS01] A. Finkel and Ph. Schnoebelen. Well-structured Transition Systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, April 2001.

- [HOS<sup>+</sup>06] K. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Workflow model compositions perserving relaxed soundness. In S. Dustdar, J. . Fiadeiro, and A. Sheth, editors, *Business Process Management, 4th Int. Conference, BPM 2006*, volume 4102 of *LNCS*, pages 225–240, 2006.
- [Kin06] Ekkart Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data Knowl. Eng.*, 56(1):23–40, 2006.
- [KNS92] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage “Ereignisgesteuerter Prozessketten (EPK)”. Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany, 1992.
- [LA94] Frank Leymann and Wolfgang Altenhuber. Managing business processes an an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.
- [LSW98] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel and M. Silva, editor, *Application and Theory of Petri Nets*, volume 1420 of *LNCS*, pages 286–305, 1998.
- [MMN06] Jan Mendling, Michael Moser, and Gustaf Neumann. Transformation of yEPC Business Process Models to YAWL. In *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, volume 2, pages 1262–1267, Dijon, France, 2006. ACM.
- [MN03] J. Mendling and M. Nüttgens. EPC Modelling based on Implicit Arc Types. In M. Godlevsky and S. W. Liddle and H. C. Mayr, editor, *Proc. of the 2nd International Conference on Information Systems Technology and its Applications (ISTA)*, volume 30 of *LNI*, pages 131–142, 2003.
- [NR02] M. Nüttgens and F. J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel and M. Weske, editor, *Proceedings of Promise 2002, Potsdam, Germany*, volume 21 of *LNI*, pages 64–77, 2002.
- [Rit00] P. Rittgen. Paving the Road to Business Process Automation. In *Proc. of the Europ. Conf. on Information Systems (ECIS)*, pages 313–319, 2000.
- [Rum99] F. J. Rump. *Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten - Formalisierung, Analyse und Ausführung von EPKs*. Teubner Verlag, 1999.
- [WAHE06] M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Reset Nets and Reachability Analysis. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *Proceedings of BPM 2006*, volume 4102 of *LNCS*, pages 389–394, Vienna, Austria, 2006.
- [WEAH05] M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536, pages 423–443, 2005.

# **Fuzzy-EPK-Modelle: Attributierung und Regelintegration**

Oliver Thomas, Thorsten Dollmann

Institut für Wirtschaftsinformatik (IWi)  
im Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI),  
Universität des Saarlandes  
Stuhlsatzenhausweg 3, Geb. D3 2, 66123 Saarbrücken  
{oliver.thomas|thorsten.dollmann}@iwi.dfki.de

**Abstract:** Im Geschäftsprozessmanagement sind Entscheidungssituationen häufig durch Unschärfe geprägt. Dies bedeutet, dass auch die jeweiligen Entscheidungsprämissen nicht in Form mathematischer Modelle oder numerischer Werte vorliegen, sondern in Form unscharfer Bedingungen, wie z. B. „geringe Durchlaufzeit“ oder „hohe Qualität“. Der vorliegende Beitrag zeigt, wie unscharfe Bedingungen und vage formulierte Zielvorstellungen mit Hilfe der Fuzzy-Set-Theorie in Geschäftsprozessmodellen berücksichtigt werden können. Diese Fuzzy-Erweiterung der Prozessmodellierung erfolgt am Beispiel der Ereignisgesteuerten Prozesskette.

## **1 Unschärfe im Geschäftsprozessmanagement**

Die Ziele gegenwärtiger Business-Engineering-Projekte liegen in der Gestaltung der Geschäftsprozesse sowie in der Analyse der Anforderungen an deren IT-Unterstützung unter Berücksichtigung von Unternehmensstrategien [ÖsWi03]. Die Prozessgestaltung muss dabei einem umfassenden Ansatz folgen, der sowohl die Planung und Kontrolle als auch die Steuerung, d. h. das Management, der betrieblichen Abläufe umfasst [BeKR05]. Zur Unterstützung eines systematischen Vorgehens bei der Prozessgestaltung hat sich die Modellierung als hilfreich erwiesen. Modellierungssprachen wie die Ereignisgesteuerte Prozesskette (EPK) [KeNS92] dienen als operationalisierter Ansatz zur Modellkonstruktion. Softwarewerkzeuge zur Geschäftsprozessmodellierung [BISi06] können den Business Engineer durch Systemkomponenten zur Erhebung, Erstellung, Analyse und Simulation von Geschäftsprozessmodellen unterstützen.

Zur Erfassung und Verbesserung von Geschäftsprozessen, deren Generalisierung in Referenzmodellen sowie zur unternehmensspezifischen Anpassung im Customizing sind zahlreiche Konzepte erarbeitet worden, die situationsspezifische Problemstellungen betrachten. Viele Ansätze legen einen Schwerpunkt auf die nutzerfreundliche und intuitive Verwendbarkeit der Methoden, indem diese an menschliche Denkweisen angenähert werden. Dabei werden jedoch für notwendige Entscheidungen die exakte Quantifizierung und Formalisierung der Entscheidungsregeln verlangt. Bei Geschäfts-

prozessen liegen jedoch vielfach nur unsichere, unpräzise und vage Informationen über die häufig nicht technisch determinierten Abläufe vor [Völk98; Fort02; Hüß03]. Ebenso ist das der Prozessgestaltung zugrunde liegende Zielsystem in der Regel durch ungenaue Formulierungen und implizite Interdependenzen geprägt. Dies demonstriert beispielsweise die Aussage „die Durchlaufzeit von Aufträgen mit ‚sehr hoher‘ Priorität soll unter Beibehaltung einer ‚hohen‘ Bearbeitungsqualität ‚wesentlich‘ gesenkt werden, indem die Bearbeitungsintensität ‚angemessen‘ reduziert wird“. In diesem Beispiel können weder die konkrete Ausprägung der beiden genannten Ziele bzgl. Durchlaufzeit und Bearbeitungsqualität noch die abgeleitete Maßnahme ohne Informationsverlust quantifiziert und damit operationalisiert werden. Informations-, insbesondere Referenzmodelle, sowie Methoden zu deren unternehmensspezifischer Adaption berücksichtigen diese Formen der Unschärfe nach wie vor unzureichend.

Diesem Umstand soll in diesem Beitrag durch die Erweiterung der Prozessmodellierung zur Berücksichtigung und Verarbeitung von Unschärfe mit Hilfe der Fuzzy-Set-Theorie begegnet werden. Diese unscharfe Erweiterung wird am Beispiel der EPK nachvollzogen.

Zunächst wird im folgenden Abschnitt 2 der Begriff „Unschärfe“ präzisiert sowie die Berücksichtigung unscharfer Daten mit Hilfe der Fuzzy-Set-Theorie motiviert. Anschließend wird in Abschnitt 3 die EPK als Modellierungssprache eingeführt, formal definiert sowie um die zur Fuzzifizierung notwendigen Sprachkonstrukte erweitert. Die Einführung der Fuzzy-EPK, die auf einer Attributierung der EPK-Sprachkonstrukte aufbaut, erfolgt in Abschnitt 4. In Abschnitt 5 wird ein Anwendungsszenario des entwickelten Konzepts aufgezeigt. Der Beitrag schließt mit der Analyse verwandter Arbeiten in Abschnitt 6 und der Diskussion der Ergebnisse in Abschnitt 7.

## **2 Von scharfen zu unscharfen Mengen**

In der Literatur existiert keine einheitliche Definition des Unschärfebegriffs – es scheint so, als müsse das Verständnis dieses Begriffs selbst unscharf sein. Unschärfe wird meist durch eine Abgrenzung gegenüber deterministischen, stochastischen und unsicheren Informationszuständen definiert [Reh98, S. 39]. Als Unschärfe wird in diesem Beitrag die Unsicherheit hinsichtlich von Daten und ihrer Interdependenzen verstanden. Im betriebswirtschaftlichen Kontext lassen sich verschiedene Auslöser von Unschärfe identifizieren [Tiet99, S. 45 ff.] (vgl. Abb. 1).

Zunächst entsteht Unschärfe bei der Erfassung der Realität durch die Komplexität des Umweltsystems und die Wahrnehmungsgrenzen des Menschen. Die resultierende informationale Unschärfe, bedingt durch die menschliche Sprache und das menschliche Denken, wird auf ein Überangebot an Informationen zurückgeführt [ZALW93, S. 5]. Sie tritt dann auf, wenn Begriffe verwendet werden, die aufgrund ihres hohen Abstraktionsniveaus Unschärfe enthalten (z.B. „Kreditwürdigkeit“). So enthalten beispielsweise wissensintensive Prozesse kurzlebige Informationen aus einer Vielzahl von Quellen, sodass zu einem festen Zeitpunkt nur ein Teil des Gesamtprozesses erfasst werden kann, der je-

doch während der Erfassung anderer Teilaspekte bereits veraltet. Zur Beschreibung solcher komplexer Begriffe sind viele verschiedene Merkmale zu berücksichtigen. Unschärfe entsteht, da der Mensch oft nicht in der Lage ist, sämtliche relevanten Informationen zu verarbeiten und möglicherweise die einzelnen Informationen selbst schon unscharf sind. Die beschreibenden Eigenschaften des Begriffs werden gemäß der menschlichen Informationsverarbeitung über sprachliche Termini aggregiert.

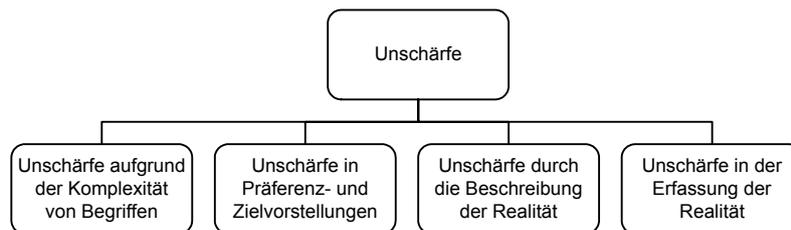


Abb. 1: Unschärfeaspekte

Ebenso existiert eine Unschärfe in den Präferenz- und Zielvorstellungen des Menschen. Menschliche Präferenzordnungen sind in vielen Situationen nicht exakt bestimmbar, sodass es zu einer mit der informationalen Unschärfe verwandten Vagheit des Zielsystems kommt. So impliziert z. B. das Ziel „wesentliche Verminderung der Durchlaufzeit“ zwar Maßnahmen, jedoch lassen sich häufig wegen der nicht explizierten Höhe der angestrebten Änderung und der unklaren Wertungsinterdependenzen mit anderen Zielen keine exakten Handlungen ableiten.

Die Beschreibung der Realität mit natürlicher Sprache erzeugt die intrinsische (auch: verbale oder linguistische) Unschärfe. Sowohl die Bildung eines sprachlichen Modells als auch die Kontextsensitivität von sprachlichen Aussagen tragen zur Entstehung dieser Unschärfe bei. Hiermit ist auch die Ungenauigkeit in sprachlichen Vergleichen eng verbunden. Die Aussage „der Objektwert ist viel höher als x“ ist ein Beispiel hierfür. Hierbei liegt die Ursache der Unschärfe nicht in der Sprache, sondern ist vielmehr darin zu sehen, dass das menschliche Wahrnehmungsvermögen der Realität beschränkt und subjektiv ist [Tiet99, S. 47]. Durch die Sprache werden also die subjektiven Vorstellungen des Beschreibenden über den Sachverhalt ausgedrückt und es besteht keine einheitliche Definition der beschreibenden Begriffe.

Die Unschärfe in der Erfassung der Realität resultiert daraus, dass Daten und Beziehungen zwischen Größen nicht genau erfasst werden können oder sollen. Die Verwendung ungenauer Daten kann jedoch vorteilhaft sein, wenn geeignete Messmethoden fehlen, der Realweltausschnitt von hoher Dynamik geprägt ist oder nicht exakt ermittelbare Abhängigkeiten bestehen. Beim für den Menschen üblichen größenordnungsmäßigen Erfassen der Realität wird in der Regel auf verbale Beschreibungen zurückgegriffen, womit zusätzlich ein Zusammenhang zur beschriebenen intrinsischen Unschärfe vorliegt.

Die Fuzzy-Set-Theorie versucht die Trennung zwischen einer modell- und verfahrenstechnisch notwendigen Präzision einerseits sowie einer empirisch wünschenswerten Berücksichtigung qualitativer Informationen andererseits zu überwinden und einen Anteil an fehlender Präzision sowie Vagheit und Unsicherheit bei Modellierungsprozessen zu tolerieren.

Die Fuzzy-Set-Theorie als heutiges Teilgebiet des Soft Computing hat sich Mitte der 1960er-Jahre entwickelt [Zade65]. Kernpunkt der Fuzzy-Theorie ist es, Zustände (von Objekten) nicht ausschließlich mit „wahr“ oder „falsch“ zu bewerten, sondern Zwischenstufen zuzulassen. Der ursprünglichen Idee von Zadeh folgend, wird die klassische Mengenlehre, d. h. die Theorie der scharfen Mengen, durch die Beschreibungen und Verknüpfungen unscharfer Mengen (Fuzzy-Mengen) erweitert. Für jedes Element  $\omega$  einer vorgegebenen (scharfen) Grundmenge  $\Omega$  wird der Grad der Zugehörigkeit zu einer Teilmenge  $A \in \Omega$  durch einen Wert  $\mu_A(\omega)$  einer Abbildung  $\mu_A: \Omega \rightarrow [0;1]$  ausgedrückt. Man wählt diese Zugehörigkeitsgrade aus dem Intervall  $[0;1]$  und gibt folgende Interpretation: Je größer der Zugehörigkeitsgrad eines Elements bzgl. einer (unscharfen) Menge ist, desto mehr gehört das Element zu dieser Menge.  $\mu_A$  wird Zugehörigkeitsfunktion der unscharfen Menge (Fuzzy-Menge)  $\{(\omega, \mu_A(\omega)) \mid \omega \in \Omega\}$  genannt.

Mit Fuzzy-Mengen lassen sich linguistische Variablen [Zade73] formulieren, die natürlichsprachliche Ausdrücke – so genannte linguistische Terme – als Werte annehmen. Abb. 2 zeigt die linguistische Variable „Auftragswert“. Sie weist die Terme „gering“, „mittel“ und „hoch“ auf. Die Zugehörigkeiten eines Objektwerts zu diesen unscharfen Mengen sind durch die Zugehörigkeitsfunktionen  $\mu_{\text{gering}}$ ,  $\mu_{\text{mittel}}$  und  $\mu_{\text{hoch}}$  ausgedrückt. Der Objektwert 70.000 € gehört z. B. zu 0.5 sowohl zur Fuzzy-Menge „mittel“ als auch zur Fuzzy-Menge „hoch“. Diese Abbildung scharfer Werte auf unscharfe Mengen heißt Fuzzifizierung. In einem scharfen Kontext wäre es nur möglich, z. B. einen Objektwert ab 70.000 € als „hohen“ Auftragswert zu charakterisieren, während 69.999 € bereits als „mittel“ gelten würde.

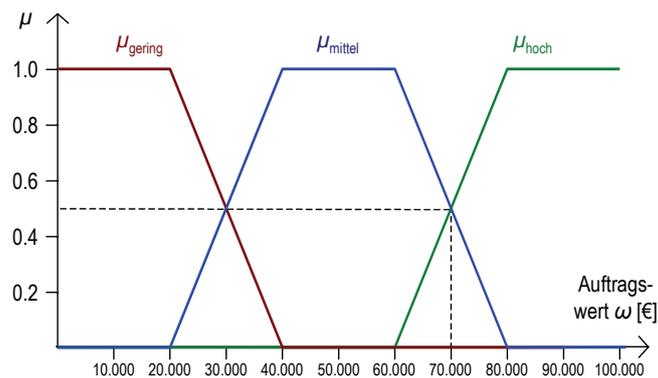


Abb. 2: Linguistische Variable „Auftragswert“

Ein Fuzzy-System besitzt eine festgelegte Menge von Ein- und Ausgangsvariablen, deren jeweilige Terme durch Fuzzy-Regeln, bestehend aus Prämissen- und Konklusionsteil, z. B. in der Form „WENN Kundeneinschätzung = mittel UND Auftragsvolumen = sehr hoch DANN Kundenauftragsbewertung = hoch“, miteinander verknüpft sind. Die Wertebereiche der (linguistischen) Variablen sind durch Fuzzy-Mengen partitioniert, die zur Repräsentation der linguistischen Terme dienen. Formal lässt sich damit eine Fuzzy-Regel als Tupel  $(\mu^{(1)}, \dots, \mu^{(n)}, \nu)$  darstellen. Dabei sind  $\mu^{(1)}, \dots, \mu^{(n)}$  Fuzzy-Mengen über dem Wertebereich der Eingangsvariablen und  $\nu$  eine Fuzzy-Menge über dem Wertebereich der Ausgangsvariablen. Durch Inferenzverfahren werden die Eingangs- und Ausgangsvariablen einander zugeordnet. Ist  $X = X_1 \times \dots \times X_n$  der Eingaberaum und  $Y$  der Ausgaberaum, so lässt sich ein Fuzzy-System  $FS$  formal als Abbildung  $FS: X \rightarrow Y$  darstellen [BKNK03, S. 162f].

Die Fuzzy-Regelbasis bestimmt die Struktur des Fuzzy-Systems. Basierend auf einem Vektor von Eingangsgrößen  $\vec{x} = (x_1, \dots, x_n) \in X$  berechnet sich der (scharfe) Ausgangswert eines typischen Fuzzy-Systems  $y = FS(\vec{x})$  in mehreren Schritten. Zunächst wird der Erfüllungsgrad jeder einzelnen Regel bestimmt, indem für jeden Messwert der Zugehörigkeitsgrad zur korrespondierenden Fuzzy-Menge ermittelt wird. Die entsprechenden Zugehörigkeitsgrade müssen dann mit einem geeigneten Fuzzy-Operator konjunktiv verknüpft werden. Aus jeder einzelnen Regel resultieren so Fuzzy-Mengen, die zur Bestimmung der Gesamtausgabe des Fuzzy-Systems noch geeignet disjunktiv verknüpft werden müssen. Für eine ausführbare Aktion, z. B. „Priorität festlegen“, wird ein scharfer Wert der Ausgangsvariablen benötigt. Ein Defuzzifizierungsschritt liefert aus der Ausgabe-Fuzzy-Menge diesen scharfen Wert  $y \in Y$ .

Ist die Ausgangsvariable keine kontinuierliche Größe, sondern eine kategorielle Variable, die einen von endlich vielen diskreten Werten (Klassen) annehmen kann, so spricht man von einem Klassifikationsproblem. Eine regelbasierte Klassifikation lässt sich mit einem Fuzzy-System modellieren, indem jede Klasse als spezielle Fuzzy-Menge aufgefasst wird und im Defuzzifizierungsschritt die Klasse mit dem größten Zugehörigkeitsgrad als Ausgangswert des Fuzzy-Systems gewählt wird.

### **3 Prozessmodellierung mit der EPK**

#### **3.1 Grundlegende Sprachkonstrukte der EPK**

Seit der Etablierung des Prozessdenkens für die Organisation von Unternehmen und für die Gestaltung von Informationssystemen wird eine Vielzahl von Modellierungssprachen zur Beschreibung von Geschäftsprozessen eingesetzt [DuAH05]. Zur Konstruktion von Geschäftsprozessmodellen auf fachlicher Ebene hat sich aufgrund ihrer Anwendungsorientierung und umfassenden Werkzeugunterstützung insbesondere im deutschsprachigen Raum die EPK etabliert. Sie wurde am Institut für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Saarbrücken, in Zusammenarbeit mit der SAP AG entwickelt [KeNS92].

In graphentheoretischer Terminologie ist ein EPK-Modell ein gerichteter und zusammenhängender Graph, dessen Knoten Ereignisse, Funktionen und Verknüpfungsoperatoren sind. Ereignisse sind die passiven Elemente der EPK. Sie beschreiben das Eintreten eines Zustands und werden durch Sechsecke dargestellt. Funktionen, die durch an den Ecken abgerundete Rechtecke repräsentiert werden, sind die aktiven Elemente der EPK. Der Funktionsbegriff wird in der EPK mit dem der Aufgabe gleichgesetzt. Im Gegensatz zu einer Funktion, die ein zeitverbrauchendes Geschehen ist, ist ein Ereignis auf einen Zeitpunkt bezogen. Während zur Bezeichnung der Funktionen in der Literatur [z. B. HoKS92, S. 5] vorgeschlagen wird, das jeweilige Objekt der Bearbeitung und ein Verb im Infinitiv zur Kennzeichnung der zu verrichtenden Tätigkeit zu verwenden (z. B. „Kundenauftrag definieren“, vgl. Abb. 3), wird für Ereignisse empfohlen, das Objekt, das eine Zustandsänderung erfährt, mit einem Verb im Partizip Perfekt zu verbinden, das die Art der Änderung beschreibt (z. B. „Kundenauftrag (ist) definiert“, vgl. Abb. 3).

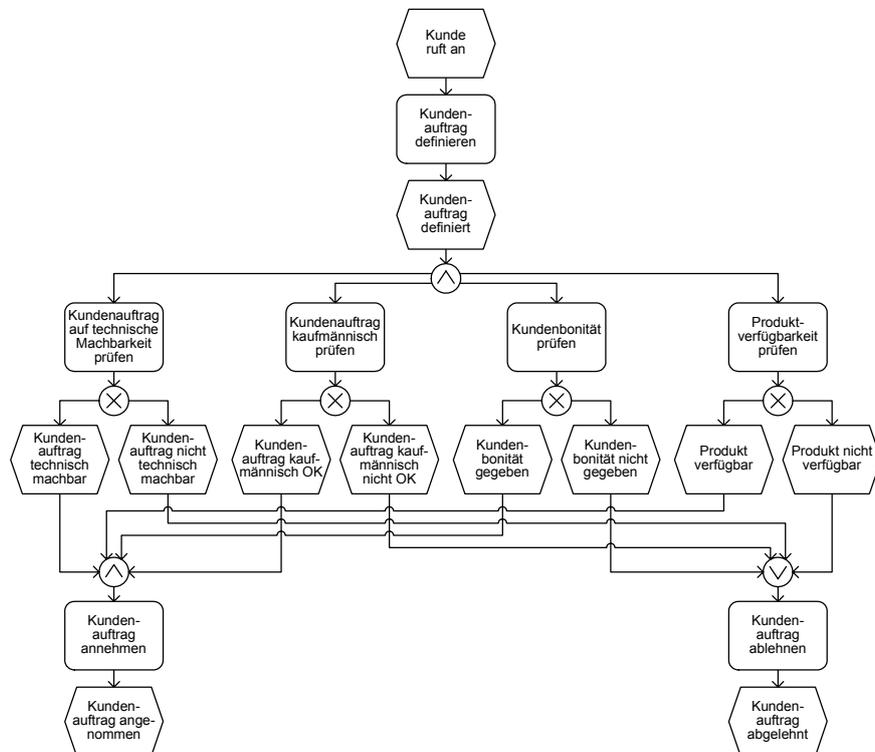


Abb. 3: EPK-Modell der Kundenauftragsbearbeitung

Ereignisse lösen Funktionen aus und sind deren Ergebnis. Diese beiden Beziehungen zwischen Funktionen und Ereignissen werden durch Kontrollflusskanten, die durch Pfeile repräsentiert werden, dargestellt. Um auszudrücken, dass eine Funktion durch ein oder mehrere Ereignisse gestartet werden bzw. eine Funktion ein oder mehrere Ereignisse als Ergebnis erzeugen kann, werden Verknüpfungsoperatoren (Konnektoren) eingeführt.

Dabei wird in Anlehnung an die Terminologie der Aussagenlogik zwischen konjunktiven „ $\otimes$ “, adjunktiven „ $\odot$ “ und disjunktiven Verknüpfungen „ $\otimes$ “ unterschieden (vgl. Abb. 3). Die entsprechenden Konnektoren werden vereinfacht als AND-, OR- bzw. XOR-Operatoren bezeichnet.

Mit diesen Informationen ergibt sich für das in Abb. 3 dargestellte Prozessmodell die folgende Interpretation: Das Modell beschreibt den Ablauf zur Definition und Durchführung von Prüffunktionen für einen Kundenauftrag. Die Entscheidung über die Annahme oder die Ablehnung des Kundenauftrags wird durch die parallele Ausführung verschiedener Teilfunktionen getroffen. Der Kundenauftrag wird auf technische Machbarkeit und aus kaufmännischer Sicht geprüft, ferner werden die Kundenbonität und die Verfügbarkeit des Produkts ermittelt. Negativergebnisse, wie z.B. „Kundenauftrag technisch nicht machbar“ oder „Kundenbonität nicht gegeben“, führen zur Ablehnung des Kundenauftrags durch die Funktion „Kundenauftrag ablehnen“.

### 3.2 Formalisierung der EPK

Die von Keller, Nüttgens und Scheer [KeNS92] eingeführte Notation der Ereignisgesteuerten Prozesskette wurde zunächst als eine nicht vollständig formalisierte Notation entwickelt und ohne eine feste formale Semantik benutzt. Zur Dokumentation von Prozessen und zur Verwendung der Modelle als Diskussionsgrundlage ist dies ausreichend. Für eine Konsistenzprüfung oder eine automatisierte Verarbeitung von EPK-Modellen, z. B. in Werkzeugen zur Simulation oder Verifikation, ist jedoch eine formale Definition der Syntax und Semantik der Modelle erforderlich. Im akademischen Umfeld werden verschiedene Ansätze zur formalen Syntax- und Semantikdefinition der EPK vorgeschlagen und diskutiert [Aals99; NüRu02; AaDK02; Kind04; Kind06; RoAa06]. Im Folgenden stellen wir eine formale Definition der grundlegenden EPK-Syntax in Anlehnung an [RoAa06] vor, um darauf aufbauend eine syntaktisch präzise Definition einer unscharfen Erweiterung aufzeigen zu können.

In formaler Schreibweise ist ein EPK-Modell ein 4-Tupel

$$EPC = (E, F, C, A).$$

Dabei ist  $E$  eine endliche (nichtleere) Menge von Ereignissen (events),  $F$  eine endliche (nichtleere) Menge von Funktionen (functions),  $C = C_{AND} \cup C_{OR} \cup C_{XOR}$  eine endliche Menge logischer Konnektoren (connectors), wobei  $C_{AND}$ ,  $C_{OR}$  und  $C_{XOR}$  paarweise disjunkte Teilmengen von  $C$  sind, und

$$A \subseteq (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C)$$

eine Menge von Kanten. Die Relation  $A$  spezifiziert die Menge der gerichteten Kontrollflusskanten (arcs), welche Funktionen, Ereignisse und Konnektoren zueinander in Verbindung setzt.  $V = E \cup F \cup C$  wird die Menge aller Knoten des EPK-Modells genannt.

In Wissenschaft und Unternehmenspraxis haben sich einige Regeln etabliert, die der Konstruktion syntaktisch korrekter EPK-Modelle dienen [KeTe99, S. 172–174; NüRu02, S. 68–70]. Unter Zuhilfenahme dieser Regeln kann die Konsistenz eines EPK-Modells im Sinne einer Widerspruchsfreiheit und Stimmigkeit überprüft werden. Wir sprechen im Folgenden weiter von EPK-Modellen und beziehen uns dabei immer auf die Menge der gemäß den bekannten Regeln syntaktisch korrekten EPK-Modelle.

### 3.3 ARIS-Erweiterung der EPK

Aus der Ableitung der EPK als zentrale Modellierungssprache der Architektur integrierter Informationssysteme (ARIS) [Sche01; Sche02] resultieren erweiterte Aussagen, die auf dem ARIS-Sichtenkonzept aufbauen. Diese werden durch Annotation von zusätzlichen Sprachkonstrukten an EPK-Funktionen getroffen [ScTA05]. So werden u.a. Sprachkonstrukte vorgeschlagen, die Umfelddaten, Nachrichten, menschliche Arbeitsleistung, maschinelle Ressourcen und Computer-Hardware, Anwendungssoftware, Leistungen in Form von Sach-, Dienst- und Informationsdienstleistungen, Finanzmittel, Organisationseinheiten oder Unternehmensziele repräsentieren (vgl. Abb. 4).

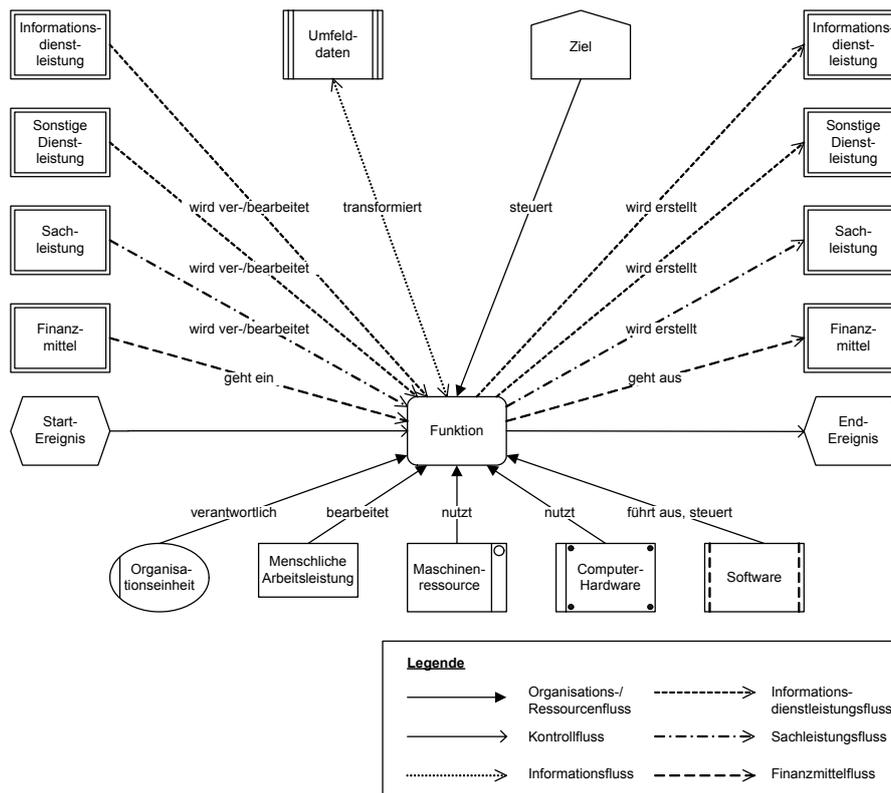


Abb. 4: Erweiterung der EPK um ARIS-Sprachkonstrukte [Sche02, S. 31]

Die Verbindung der Konstrukte, die nur mit Funktionen der EPK erfolgen kann, wird über Kanten hergestellt, die neben dem bereits eingeführten Kontrollfluss in Organisations-/Ressourcen-, Informations-, Informationsdienstleistungs- und Sachleistungs- sowie Finanzmittelfluss unterschieden werden [Sche02, S. 31].

Exemplarisch werden aus den in Abb. 4 vorgestellten ARIS-Sprachelementen die Konstrukte der Organisations-, Daten- und Leistungssicht als zusätzliche Artefakte mit den entsprechenden Verbindungen dieser Konstrukte mit den Funktionen der EPK über Kanten des Organisations-, Daten- bzw. Leistungsflusses in die formale Repräsentation des EPK-Modells hinzugefügt und in einem nächsten Schritt um Attribute angereichert. Diese Erweiterung wird anschließend für die Vorstellung der beispielhaften Verarbeitung von Unschärfe in Geschäftsprozesse herangezogen.

Hierzu wird ein um ARIS-Sprachkonstrukte erweitertes EPK-Modell als ein Tupel

$$EPC_{ARIS} = (E, F, C, A, O, D, L, R)$$

eingeführt. Dabei ist  $(E, F, C, A)$  ein EPK-Modell mit der Menge der Kontrollflussknoten  $V = E \cup F \cup C$  und der Menge der Kontrollflusskanten  $A$ . Die Knotenmengen, welche die Artefakte der Organisations-, Daten bzw. Leistungssicht repräsentieren, sind  $O$  für die Menge der Organisationseinheiten (organizational units),  $D$  für die Menge der Datenobjekte (data objects) und  $L$  für die Menge der Leistungen (outputs). Für die Mengen  $O, D$  und  $L$  wird gefordert, dass sie paarweise disjunkt sind. Die Menge  $R$  enthält Mengen von Relationen, die den Funktionen die unterschiedlichen Artefakte zuordnen. Sie wird definiert als Menge  $R = R^{OF} \cup R^{DF} \cup R^{FD} \cup R^{LF} \cup R^{FL}$ , wobei

- $R^{OF} = \{R_1^{OF}, \dots, R_{n_{OF}}^{OF}\}$ , mit  $R_i^{OF}$  ( $1 \leq i \leq n_{OF}$ ),  $n_{OF} \in \mathbb{N}$ , Relationen auf  $O \times F$ ,
- $R^{DF} = \{R_1^{DF}, \dots, R_{n_{DF}}^{DF}\}$ , mit  $R_i^{DF}$  ( $1 \leq i \leq n_{DF}$ ),  $n_{DF} \in \mathbb{N}$ , Relationen auf  $D \times F$ ,
- $R^{FD} = \{R_1^{FD}, \dots, R_{n_{FD}}^{FD}\}$ , mit  $R_i^{FD}$  ( $1 \leq i \leq n_{FD}$ ),  $n_{FD} \in \mathbb{N}$ , Relationen auf  $F \times D$ ,
- $R^{LF} = \{R_1^{LF}, \dots, R_{n_{LF}}^{LF}\}$ , mit  $R_i^{LF}$  ( $1 \leq i \leq n_{LF}$ ),  $n_{LF} \in \mathbb{N}$ , Relationen auf  $L \times F$  und
- $R^{FL} = \{R_1^{FL}, \dots, R_{n_{FL}}^{FL}\}$ , mit  $R_i^{FL}$  ( $1 \leq i \leq n_{FL}$ ),  $n_{FL} \in \mathbb{N}$ , Relationen auf  $F \times L$  sind.

Die einzelnen Relationen aus den Mengen  $R^{OF}, R^{DF}, R^{FD}, R^{LF}$  und  $R^{FL}$  tragen dabei verschiedene Bedeutungen und bestimmen den Beziehungstyp zwischen den Elementen aus  $O \times F, \dots, F \times L$ . Eine Auswahl praxisnaher Beziehungstypen ist in Tab. 1 aufgelistet.

Ein um ARIS-Sprachkonstrukte erweitertes EPK-Modell

$$EPC_{ARIS} = (E, F, C, A, O, D, L, R)$$

ist genau dann syntaktisch korrekt, wenn  $(E, F, C, A)$  ein syntaktisch korrektes EPK-Modell ist und zusätzlich jedes Artefakt mit mindestens einem Knoten des EPK-Graphen  $(V, A)$  verbunden ist, wobei wir hier nur anmodellierte Artefakte an Funktionen zulassen. Wir fordern also, dass der durch die ARIS-Erweiterung aufgespannte Graph

$G = (V_A, A \cup R)$  mit der Knotenmenge  $V_A = E \cup F \cup C \cup O \cup D \cup L$  und der Kantenmenge  $A \cup R$  zusammenhängend sein soll.

Tab. 1: Beziehungstypen zwischen Funktionen und ARIS-Sprachkonstrukten

Quellobjekttyp	Zielobjekttyp	Mögliche Beziehungstypen
Organisationseinheit	Funktion	führt aus, entscheidet über, ist verantwortlich für, stimmt zu, wirkt mit bei, muss informiert werden, muss informieren über Ergebnis von
Datenobjekt	Funktion	ist Input für, wird genehmigt von, wird geprüft von
Funktion	Datenobjekt	ändert, hat Output, erzeugt
Leistung	Funktion	ist Input für, wird verbraucht von, wird verwendet von
Funktion	Leistung	hat Output, produziert

## 4 Fuzzy-Ereignisgesteuerte Prozesskette

### 4.1 Erweiterung der EPK um Attribute

Die Objekttypen in EPK-Modellen (z.B. die einzelnen Datenobjekte aus  $D$  oder Organisationseinheiten aus  $O$ ), aufgefasst als Objektmengen einzelner Objekte, man spricht auch von *Instanzen* des jeweiligen Typs<sup>1</sup>, zeichnen sich durch bestimmte Merkmale aus. Diese Merkmale werden zur Beschreibung der einzelnen Objekte sowie zu deren interner Repräsentation, bspw. bei der Speicherung in relationalen Datenbanken, herangezogen und als *Attribute* bezeichnet. Während beschreibende Attribute fachliche Eigenschaften darstellen, dienen die so genannten *Schlüsselattribute* zur eindeutigen Identifikation eines Objekts. Ein Kunde kann bspw. über seinen Namen, seine Adresse und sein Geburtsdatum identifiziert werden, während sein Umsatz oder die Kundeneinschätzung anwendungsrelevante Eigenschaften darstellen. Im Folgenden sollen nur fachlich relevante Attribute herangezogen und im Unschärfekonzept berücksichtigt werden.

Jedes Attribut hat einen Wertebereich, der die Menge der möglichen Attributwerte festlegt. Beispielsweise kann der Wertebereich des Attributs „Auftragssumme“ eines Datenobjekttyps „Auftrag“ als Menge natürlicher Zahlen festgelegt werden. Ebenso kann die Wertemenge für das Attribut „Name“ des Objekttyps „Kunde“ auf die Menge von Zeichenketten festgelegt werden, die aus alphabetischen Zeichen besteht.

<sup>1</sup> Bisher haben wir auf die Unterscheidung zwischen der Typ- und Instanzebene bei Prozessmodellen verzichtet. An dieser Stelle werden wir sprachlich exakter und sprechen von Objekttypen in EPK-Modellen und ihren Instanzen. Der Funktionstyp „Kundenbonität prüfen“ als Element der Menge  $F$  kann bspw. zur Laufzeit des Modells beliebig viele Instanzen generieren. Auf Basis möglicher Merkmalsausprägungen der Instanzen soll eine im Prozessmodell auf Typebene zu modellierende Entscheidungsunterstützung methodisch aufgebaut werden. Hierbei werden die Merkmalsausprägungen selbst wesentlich.

Auf die Darstellung von Attributen wird in konzeptionellen EPK-Modellen aus Gründen der Übersichtlichkeit und Komplexität in den meisten Fällen zunächst verzichtet. Allerdings erfordert die nachfolgend dargelegte Unschärfeerweiterung an dieser Stelle einer Präzisierung des konzeptionellen Modells und daher eine explizite Modellierung entscheidungsrelevanter Attribute bei der Prozessmodellierung.

Es seien  $S$  eine Menge von Objekten der Diskurswelt,  $Dom(A_i)$  ( $i=1, \dots, n$ ),  $n \in \mathbb{N}$ , Mengen von Werten und  $A_i$  ( $i=1, \dots, n$ ) wohldefinierte Abbildungen der Form

$$A_i : S \rightarrow Dom(A_i) \quad (i=1, \dots, n).$$

Dann heißt  $\{A_1, \dots, A_n\}$  eine Menge von Attributen (attributes) auf den Objekten der Menge  $S$  oder kurz auf  $S$ . Die Mengen  $Dom(A_i)$  bezeichnet man als die Wertebereiche (domain) der Attribute  $A_i$  und die Elemente  $A_i(s) \in Dom(A_i)$  werden die Attribute der Objekte  $s$  genannt. Gilt  $Dom(A_i) = \{0, 1\}$ , so wird  $A_i$  binäres (binary) Attribut auf  $S$  genannt. Unter den Voraussetzungen dieser Definition gibt es somit eine (interne) Darstellung der Objekte als Tupel  $(A_1(s), \dots, A_n(s))$  von Attribut-Werten, d.h. als Elemente der Menge

$$Dom(A_1) \times \dots \times Dom(A_n) = \prod_{i=1}^n Dom(A_i).$$

In einem ARIS-EPK-Modell

$$EPC_{ARIS} = (E, F, C, A, O, D, L, R)$$

notieren wir die folgenden Attribute:

- $A_1^e, \dots, A_{n_e}^e$  ( $n_e \in \mathbb{N}$ ) sind die  $n_e$  Attribute auf dem Ereignis  $e \in E$ ,
- $A_1^f, \dots, A_{n_f}^f$  ( $n_f \in \mathbb{N}$ ) sind die  $n_f$  Attribute auf der Funktion  $f \in F$ ,
- $A_1^o, \dots, A_{n_o}^o$  ( $n_o \in \mathbb{N}$ ) sind die  $n_o$  Attribute auf der Organisationseinheit  $o \in O$ ,
- $A_1^d, \dots, A_{n_d}^d$  ( $n_d \in \mathbb{N}$ ) sind die  $n_d$  Attribute auf dem Datenobjekt  $d \in D$ ,
- $A_1^l, \dots, A_{n_l}^l$  ( $n_l \in \mathbb{N}$ ) sind die  $n_l$  Attribute auf der Leistung  $l \in L$ .

Umgangssprachlich gesprochen werden damit in einem fachkonzeptionellen, auf Typ-Ebene modellierten EPK-Modell jedem Knotenelement im EPK-Graph eigene Attribute zugeordnet. Verdeutlicht wird dies beispielsweise durch die Tatsache, dass ein Datenobjekt(-typ) „Kundenauftrag“ ein Attribut „Auftragssumme“ besitzt, wohingegen dieses Attribut kein Merkmal eines Datenobjektes „Artikel“ darstellt.

Wir definieren ein um Attribute erweitertes ARIS-EPK-Modell als ein Tupel

$$EPC_{ARIS,attr} = (E, F, C, A, O, D, L, R, M).$$

Dabei sind den einzelnen Ereignissen aus  $E$ , Funktionen aus  $F$ , Organisationseinheiten aus  $O$ , Datenobjekten aus  $D$  und Leistungen aus  $L$  Attribute zugeordnet. Auf die Zu-

ordnung von Attributen für die Menge der Kontrollflusskanten  $A$  und den Relationen aus  $R$  wird an dieser Stelle verzichtet, da diese beschreibenden Attribute nicht zur Fuzzifizierung herangezogen werden. Die aufgezählten Attribute von Elementen aus  $E, F, O, D$  und  $L$  werden in der Menge  $M$  zusammengefasst.

Jedes Objekt weist dabei eigene identifizierende und anwendungsrelevante Attribute mit eigenen Wertemengen auf. Es sollen nur solche Attribute modelliert werden, die im jeweiligen Kontext relevant werden. Veränderungen der Attribute der Artefakte werden in der Folge nur berücksichtigt, soweit dies aus dem EPK-Modell heraus ersichtlich ist.

## 4.2 Fuzzy-Erweiterung der EPK

Wir definieren ein Fuzzy-EPK-Modell

$$FEPC = (E, F, C, A, O, D, L, R, M, FC)$$

als ein um Attribute angereichertes ARIS-EPK-Modell

$$EPC_{ARIS,attr} = (E, F, C, A, O, D, L, R, M)$$

mit den folgenden Eigenschaften:

- $M$  ist die Menge der unscharfen Attribute des Fuzzy-EPK-Modells  $FEPC$ . Die Bezeichnung „unscharfes Attribut“ bezieht sich hierbei auf zwei Aspekte. Erstens wird angenommen, dass die Wertebereiche der Attribute nicht notwendigerweise scharfe Mengen sind, sondern aus Fuzzy-Mengen bestehen können. Zweitens können die Attribute als linguistische Variablen interpretiert werden. Dies impliziert, dass der Name der linguistischen Variable der Bezeichnung des Attributs entspricht und der Wertebereich des Attributs zugleich die Grundmenge der linguistischen Variablen ist.
- $O$ ,  $D$  bzw.  $L$  sind Mengen von Organisationseinheiten, Datenobjekten bzw. Leistungen, die unscharfe Organisationseinheiten, unscharfe Datenobjekte bzw. unscharfe Leistungen enthalten. Eine unscharfe Organisationseinheit, ein unscharfes Datenobjekt bzw. eine unscharfe Leistung ist hierbei eine Organisationseinheit, ein Datenobjekt bzw. eine Leistung, welche unscharfe Attribute besitzt.
- $FC$  ist eine Menge von Fuzzy-Systemen (vgl. Abschnitt 2). Die möglichen Input- und Outputgrößen werden durch die Funktion restringiert, der ein solches System zugeordnet wird.
- $F$  ist die Menge der unscharfen Funktionen des EPK-Modells. Eine unscharfe Funktion zeichnet sich dabei entweder durch ein oder mehrere unscharfe Attribute aus oder durch die Zuordnung eines Fuzzy-Systems  $FS \in FC$  zur Entscheidungsunterstützung auf der Basis unscharf formulierter Regeln bei der Ausführung. Dabei müssen alle Organisationseinheiten, Datenobjekte bzw. Leistungen des EPK-Modells, deren Attribute Input- und Outputgrößen des zugeordneten Fuzzy-Systems darstellen,

über eine Kante mit dieser unscharfen Funktion verbunden sein. Wird das Fuzzy-System direkt als Klassifikator zur Entscheidung über die weitere Verzweigung des Kontrollflusses eingesetzt, dürfen nur die nachfolgenden Ereignisse dieser Funktion im Konklusionsteil der Regeln vorkommen.

- Die Menge  $R$  enthält Mengen von unscharfen Relationen<sup>2</sup> zwischen Kontrollflussobjekten und den unterschiedlichen Artefakten:  $R = \{R^{OF}, R^{DF}, R^{FD}, R^{LF}, R^{FL}\}$ , mit
  - $R^{OF} = \{R_1^{OF}, \dots, R_{n_{OF}}^{OF}\}$ , wobei  $R_i^{OF}$  ( $1 \leq i \leq n_{OF}$ ),  $n_{OF} \in \mathbb{N}$ , unscharfe Relationen auf  $O \times F$  sind.
  - $R^{DF} = \{R_1^{DF}, \dots, R_{n_{DF}}^{DF}\}$ , wobei  $R_i^{DF}$  ( $1 \leq i \leq n_{DF}$ ),  $n_{DF} \in \mathbb{N}$ , unscharfe Relationen auf  $D \times F$  sind.
  - $R^{FD} = \{R_1^{FD}, \dots, R_{n_{FD}}^{FD}\}$ , wobei  $R_i^{FD}$  ( $1 \leq i \leq n_{FD}$ ),  $n_{FD} \in \mathbb{N}$ , unscharfe Relationen auf  $F \times D$  sind.
  - $R^{LF} = \{R_1^{LF}, \dots, R_{n_{LF}}^{LF}\}$ , wobei  $R_i^{LF}$  ( $1 \leq i \leq n_{LF}$ ),  $n_{LF} \in \mathbb{N}$ , unscharfe Relationen auf  $L \times F$  sind.
  - $R^{FL} = \{R_1^{FL}, \dots, R_{n_{FL}}^{FL}\}$ , wobei  $R_i^{FL}$  ( $1 \leq i \leq n_{FL}$ ),  $n_{FL} \in \mathbb{N}$ , unscharfe Relationen auf  $F \times L$  sind.

Die im scharfen Modell vorkommenden Relationen können somit im Sinne des Erweiterungsprinzips von Zadeh als Spezialfall des unscharfen Falls betrachtet werden.

Die Fuzzy-Erweiterung der Ereignisgesteuerten Prozesskette wird im nachfolgenden Abschnitt durch ein Beispielszenario erläutert.

## 5 Anwendungsszenario „Fuzzy-Customizing“

Die Konstruktion von Prozessmodellen ist aus Gründen ihrer möglichen Wiederverwendung vielfach mit dem Anspruch verbunden, von unternehmensspezifischen Eigenschaften zu abstrahieren. Sie werden daher in unternehmensspezifische Prozessmodelle und Referenzprozessmodelle unterschieden. Der Begriff „unternehmensspezifisch“ charakterisiert hierbei den individuellen Charakter des entsprechenden Modells. Im Gegensatz dazu stellt ein Referenzmodell für die Entwicklung spezifischer Modelle einen Bezugspunkt dar, da es eine Klasse von Anwendungsfällen repräsentiert [Broc03; Thom06]. Prominente Beispiele sind im wissenschaftlichen Umfeld das Referenzmodell für industrielle Geschäftsprozesse (Y-CIM-Modell) von Scheer [Sche97] sowie das der Unternehmenspraxis entstammende SAP R/3-Referenzmodell [KeTe99].

---

<sup>2</sup> Eine unscharfe Relation (Fuzzy-Relation)  $R$  über Grundmengen  $\Omega_1, \Omega_2$  ist eine Fuzzy-Menge des kartesischen Produkts  $\Omega_1 \times \Omega_2$ , die über eine Zugehörigkeitsfunktion  $\mu_R : \Omega_1 \times \Omega_2 \rightarrow [0, 1]$  charakterisiert wird. Dabei ist jedem Element  $(\omega_1, \omega_2)$  als 2-stelliges Tupel in  $R$  ein Zugehörigkeitsgrad  $\mu_R(\omega_1, \omega_2) \in [0, 1]$  zugeordnet. Der Zugehörigkeitsgrad wird als Stärke der Fuzzy-Relation  $R$  zwischen den Elementen des Tupels interpretiert.

Abb. 3 stellt einen Ausschnitt eines Referenzprozesses zur Kundenauftragsabwicklung in Form einer EPK dar. Ein Schwachpunkt des modellierten Prozesses, der in diesem Beitrag bislang nicht diskutiert wurde, ist erkennbar: Jedes der Negativergebnisse führt zur unmittelbaren Ablehnung des Kundenauftrags – unabhängig von den Prüfergebnissen der anderen Funktionen. Dies steht im Widerspruch zur Unternehmenspraxis, in der solche absoluten Ausschlusskriterien nur selten scharf eingehalten werden. Vielmehr werden durch menschliche Entscheidungsträger Kompensationsmechanismen angewendet, die eine Überschreitung von Grenzwerten in einem Bereich durch bessere Werte in anderen Bereichen ausgleichen. Hierbei sind die Regeln für die Wirkungszusammenhänge nicht dokumentiert, sondern beruhen auf Erfahrungswissen der Entscheidungsträger. Es handelt sich zudem meist um einfache Regeln, die nur größenordnungsmäßige Verknüpfungen herstellen und sich an Zielsystemen mit vagen Interdependenzen orientieren.

Im vorliegenden Fall könnte etwa die Entscheidung, ob das Produkt verfügbar ist, nicht nur mit einem scharfen „Ja“ oder „Nein“ beantwortet werden, sondern auch durch zusätzlichen Beschaffungsaufwand von Verhältnismäßigkeitsüberlegungen geprägt sein, sodass das Produkt z. B. aus einem anderen Lager angefordert wird, wenn alle anderen Prüfungen positiv ausfallen. Eine entsprechende Entscheidung orientiert sich an einem Trade-off zwischen den Zielen der Vermeidung von Zusatzkosten und der Ausrichtung an Kundenbedürfnissen. Hieraus ergibt sich neben dem Problemfeld der Erschließung impliziten Wissens die Herausforderung der Abbildung von Unschärfe in Referenzmodellen und Vorgehensmodellen zu deren Anpassung.

Abb. 5 zeigt die unscharfe Erweiterung des Referenzprozesses der Kundenauftragsabwicklung – eingebettet in die grafische Benutzeroberfläche eines Fuzzy-Modellierungswerkzeugs. Der Prozess ist im Hauptfenster in Form einer Fuzzy-EPK dargestellt. Die unscharfen Konstrukte der EPK sind durch graue Schattierung gekennzeichnet.

Nach der Definition des Kundenauftrags wird unverändert dessen Annahme geprüft. Die Prüfungen der einzelnen Funktionen des „scharfen“ Prozesses werden jedoch um Prüfungen zum Auftragsvolumen und zur Kundeneinschätzung erweitert. Die Funktionen sind dabei nicht als „untergeordnete“ Aktivitäten der Kundenauftragsprüfung, sondern als unscharfe Objektattribute der entsprechenden Datenobjekt- und Leistungstypen in Form linguistischer Variablen modelliert (vgl. Abb. 5, Fenster „Attribute“). Im Attribut-Explorer ist beispielsweise das Objektattribut „Auftragsvolumen“ des Datenobjekttyps „Kundenauftrag“ aktiviert. Es weist als linguistische Variable die Terme „sehr niedrig“, „niedrig“, „mittel“, „hoch“ und „sehr hoch“ auf (vgl. auch Abb. 2).

Im rechten Teil des Attributfensters kann der Benutzer über einen Variableneditor die Zugehörigkeitsfunktionen der linguistischen Terme verändern, z. B. durch „Ziehen“ der durch kleine Quadrate dargestellten „Eckpunkte“ der Funktionen. Ein Variablenassistent unterstützt den Benutzer durch eine automatisierte Variablendefinition. Ein Regleditor (vgl. gleichnamiges Fenster in Abb. 5) zeigt die der Funktion hinterlegten Regeln an.

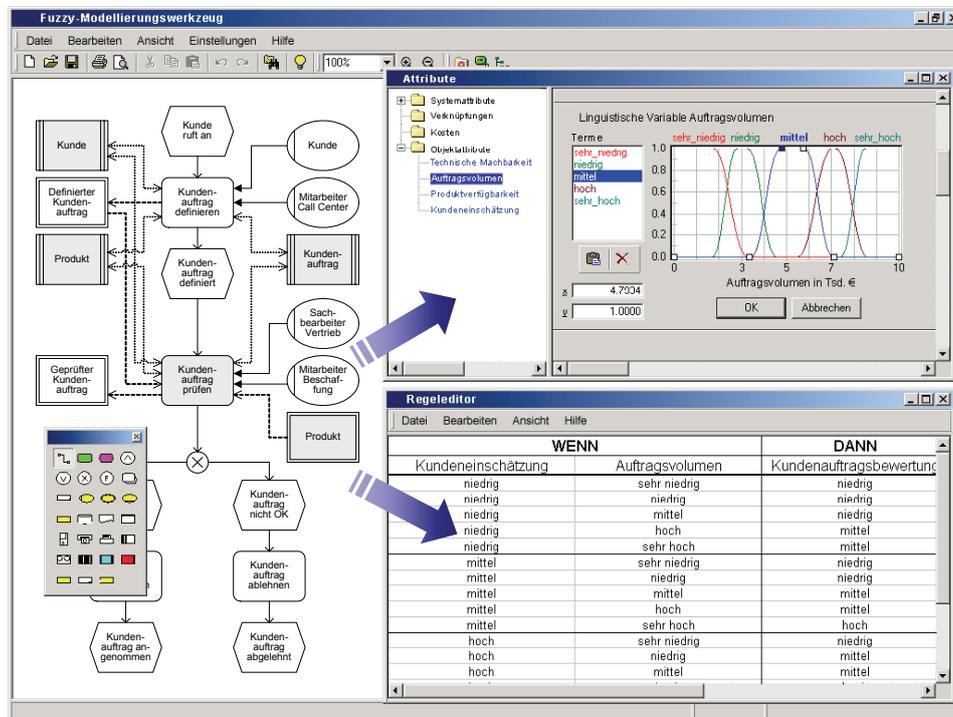


Abb. 5: Benutzeroberfläche des Fuzzy-Modellierungswerkzeugs

Im Beispiel ist der Ausschnitt einer Regelmenge mit den Eingangsvariablen „Kundeneinschätzung“ und „Auftragsvolumen“ sowie der Ausgangsvariablen „Kundenauftragsbewertung“ gegeben. Der Benutzer erzeugt die Regelmengen in der Tabelle z.B. durch eine automatisierte Übernahme vollständiger Regelmengen aus einem mit „Konstistenzchecks“ ausgerüsteten Regelassistenten (Schnittstelle zum Fuzzy-System).

Der Referenzprozess besteht – entsprechend der in Abschnitt 4 vorgestellten Formalisierung der Fuzzy-EPK – in seiner Erweiterung aus zwei Ebenen. Die Modellierungsebene (vgl. Abb. 5, links) zeigt nach wie vor das Prozessmodell, im dargestellten Fall ein Fuzzy-EPK-Modell. In dieser Ebene ist die semi-formale Modellierung auf die zum Verstehen der Geschäftslogik durch den Endanwender notwendigen Inhalte begrenzt. In einer weiteren Ebene (vgl. Abb. 5, rechts) sind die entscheidungsunterstützenden Regeln hinterlegt, welche im Ergebnis die Annahme oder Ablehnung des Kundenauftrags bewirken. Diese Ebene greift auf Erkenntnisse der Fuzzy-Set-Theorie zurück, um die Eigenschaften abwägender Entscheidungen abzubilden.

Die Adaption eines solchen Prozesses wird nun auf das in den Entscheidungsregeln hinterlegte fachliche Wissen beschränkt und lässt die Ablauflogik des Prozesses unberührt. Durch die Berücksichtigung unscharfer Bedingungen und vage formulierter Zielvorstellungen mit Hilfe von Ansätzen der Fuzzy-Set-Theorie kann der Anwender,

der über das fachliche Wissen verfügt, durch intuitive und einfache linguistische Bewertungen selbst die Adaption des Referenzprozesses vornehmen.

Dies hat ebenfalls zur Folge, dass ein bereits adaptierter Prozess prinzipiell als Referenzprozess aufgefasst werden kann – die Ablauflogik des Prozesses bleibt bei seiner Adaption unverändert, lediglich die Art der Entscheidungsfindung muss angepasst werden.

Gleichwohl ist darauf hinzuweisen, dass mit dem Anwendungsszenario „Fuzzy-Customizing“ keine Evaluierung der unscharf erweiterten EPK verbunden ist. Hierzu wurde am Institut für Wirtschaftsinformatik im DFKI, Saarbrücken, bereits eine werkzeuggestützte Simulation durchgeführt [AdTL06]. Der vorliegende Anwendungsfall dient vielmehr dazu, aufzuzeigen, dass aus der EPK-Spracherweiterung neue Anforderungen für die fachliche Built-Time-Modellierung resultieren. Beim Design der Prozessmodelle sind Entscheidungen darüber zu treffen, welche Situationen, die bisher in der scharfen Ablauflogik des Prozessmodells selbst abgebildet werden mussten, nun mithilfe von Regeln in der Entscheidungslogik beschrieben werden können. Somit ändert sich die Vorgehensweise beim Design der fachlichen Modelle und das Konstruktionsergebnis, wie dies am Referenzprozess der Kundenauftragsabwicklung verdeutlicht wurde.

## 6 Verwandte Arbeiten

Es existieren nur wenige Ansätze, die Unschärfeaspekte in die Informations- bzw. Prozessmodellierung mit Hilfe der Fuzzy-Set-Theorie integrieren.

Die Fuzzy-Erweiterung des Entity-Relationship-Modells (ERM) wurde von Zvieli, Chen [ZvCh86] beschrieben. Hierbei können Entitytypen, Beziehungstypen und Attributmenge Fuzzy-Werte annehmen. Die Berücksichtigung dieser fuzzifizierten Datenstrukturen führt konsequent zur Verarbeitung der unscharfen Daten in den entsprechenden betrieblichen Geschäftsprozessen.

Fuzzy-Theorie-basierte Erweiterungen objektorientierter Modellierungsmethoden für Geschäftsprozesse sind bei Benedicenti et al. [BSVV98] und Cox [Cox99; Cox02] zu finden. Ein auf der Fuzzy-Set-Theorie basierender objektorientierter Ansatz zur Simulation von Geschäftsprozessen wird durch Völkner, Werners [Völk98; VöWe02] vorgestellt.

Zur Beschreibung dynamischer Aspekte betrieblicher Informationssysteme werden u. a. Petri-Netze eingesetzt. Das zweiwertige Verhalten von Stellen und Transitionen eines Petri-Netzes ist bei der Abbildung wissensintensiver und schwach strukturierter Prozesse jedoch von Nachteil. Um das Systemverhalten auch bei unscharfen Prozessbedingungen oder unvollständigen, vagen Informationen darstellen zu können, wurden Petri-Netze durch Fuzzy-Konzepte erweitert. Das Fuzzy-Petri-Netz [Lipp82] entsteht durch die Projektion mehrerer scharfer Petri-Netze, bei der die Strukturinformationen als unscharfe Mengen abgebildet werden.

Becker, Rehfeldt, Turowski [BeRT96; Reh98] zeigen am Beispiel der industriellen Auftragsabwicklung die Berücksichtigung unscharfer Daten in der Geschäftsprozessmodellierung mit Ereignisgesteuerten Prozessketten exemplarisch auf. Als wesentliche, mit Unschärfe in Form von Unsicherheit behaftete exogene Eingangsdaten werden vage Vertriebsinformationen betrachtet, die in vorläufige Kundenaufträge umgewandelt werden. Diese „unscharfe Ergänzung“ der Prozesse wird durch schattierte Objekte visualisiert. Aus methodischer Sicht müssen unscharfe und scharfe Modellobjekte bei der fachkonzeptionellen Darstellung eines Geschäftsprozesses jedoch nicht unterschieden werden. Vielmehr sind auch die das Verhalten beschreibenden Regeln sowie bekannte Parameter (z.B. Partitionierungen) im Sinne eines umfassenden Wissensmanagements unabhängig von einem Implementierungsmodell bereits beim Design der Prozesse auf der fachlichen Ebene zu erfassen.

Thomas und Adam [ThHA02; AdTh05; AdTL06] untersuchen mit weiteren Co-Autoren, wie unscharfe Daten zum Design wissensintensiver und schwach strukturierter Geschäftsprozesse und ihrer Implementierung in Anwendungssystemen genutzt werden können. Die von den Autoren entwickelte Idee wurde in diesem Beitrag erweitert und formalisiert.

## **7 Zusammenfassung und zukünftige Forschungsfragen**

In dem vorliegenden Beitrag wurde ein Ansatz zur Integration von Unschärfeaspekten in das Geschäftsprozessmanagement entwickelt. Die Integration wurde in zweierlei Hinsicht beispielhaft vollzogen. Erstens erfolgte die Berücksichtigung unscharfer Daten mit Hilfe der Fuzzy-Set-Theorie als Teilgebiet des Soft Computing. Zweitens wurde sie am Beispiel einer etablierten Modellierungssprache für Geschäftsprozesse, der Ereignisgesteuerten Prozesskette, durchgeführt. Das Konzept entspricht im übertragenen Sinne einer „Ebenenerweiterung“ der Beschreibungssprache: Während die Geschäftsprozessmodelle auf die zum Verstehen der Geschäftslogik durch den Endanwender notwendigen Inhalte begrenzt sind, wird das Fachwissen zur Entscheidungsunterstützung einzelnen Modellelementen hinterlegt.

Insbesondere in den beschriebenen Anwendungen zeigte sich, dass durch die Modellierung vagen Wissens mit Fuzzy Logic im Geschäftsprozessmanagement viele Situationen exakter als bisher beschrieben werden können. Daher eignen sich auf Fuzzy-Logik aufbauende regelbasierte Systeme in hohem Maße zur Steuerung von Prozessen. Da die Regelbasis auf Wenn-Dann-Regeln basiert, kann ihr funktionales Verhalten relativ leicht nachvollzogen und vorhandenes Wissen relativ einfach integriert werden. Bei Modifikation des zu steuernden Prozesses können alte Regeln direkt übernommen oder müssen nur geringfügig modifiziert werden. Dies erleichtert die ständige Verbesserung der Prozessdefinitionen im Sinne eines Continuous Process Improvement [Robs91; Sche96].

Eine zukünftige Herausforderung für ihre Forschungstätigkeiten sehen die Autoren vor allem in der Beantwortung der Frage, ob im Fuzzy-Geschäftsprozessmanagement die Erstellung adäquater linguistischer Variablen und Regelbasen wirtschaftlich sinnvoll erfol-

gen kann. Als problematisch erweist sich in der Praxis insbesondere das Aufstellen der Regelbasis. Jedes ungewünschte Fehlverhalten muss vom Entwickler analysiert und entsprechend „von Hand“ korrigiert werden. Durch die Optimierung regelbasierter Fuzzy-Systeme mittels neuronaler Netze können Fuzzy-Mengen angepasst und die Regelbasis erlernt bzw. korrigiert werden. Die Fähigkeit von künstlichen neuronalen Netzen zum Aufdecken von Geschäftslogik in Prozessen („Process Mining“) sowie zur Verbesserung von Geschäftsprozessen durch Lernen werden aktuell diskutiert [AdTL06].

**Danksagung.** Das in diesem Beitrag präsentierte Anwendungsszenario resultiert aus dem Forschungsprojekt „Referenzmodell-gestütztes Customizing unter Berücksichtigung unscharfer Daten“, Kennwort: Fuzzy-Customizing, Teilprojekt der Forschungskohorte „Betriebliche Referenz-Informationsmodellierung – Designtechniken und domänenbezogene Anwendung“ (BRID<sup>2</sup>), gefördert von der Deutschen Forschungsgemeinschaft (Förderkennzeichen: SCHE 185/25–1). Die Autoren möchten den beiden anonymen Gutachtern danken, deren äußerst wertvollen Hinweise zur Verbesserung einer früheren Version dieses Artikels beigetragen haben.

## Literaturverzeichnis

- [AdTh05] Adam, O.; Thomas, O.: A Fuzzy Based Approach to the Improvement of Business Processes. In: Castellanos, M.; Weijters, T. (Hrsg.): *BPI'05 : Workshop on Business Process Intelligence ; Nancy, France, September 5, 2005*. Nancy, 2005, S. 25–35
- [AdTL06] Adam, O.; Thomas, O.; Loos, P.: Soft Business Process Intelligence – Verbesserung von Geschäftsprozessen mit Neuro-Fuzzy-Methoden. In: Lehner, F.; Nösekabel, H.; Kleinschmidt, P. (Hrsg.): *Multikonferenz Wirtschaftsinformatik 2006 : Band 2*. Berlin : GITO, 2006, S. 57–69
- [BeKR05] Becker, J.; Kugeler, M.; Rosemann, M. (Hrsg.): *Prozessmanagement : Ein Leitfaden zur prozessorientierten Organisationsgestaltung ; mit 41 Tabellen*. 5. Aufl. Berlin [u.a.] : Springer, 2005
- [BeRT96] Becker, J.; Rehfeldt, M.; Turowski, K.: Auftragsabwicklung mit unscharfen Daten in der Industrie. In: Biethahn, J. et al. (Hrsg.): *Betriebliche Anwendungen von Fuzzy-Technologien : Tagungsband zum 2. Göttinger Symposium Softcomputing am 29. Feb. 1996 an der Universität Göttingen*. Univ. Göttingen, Inst. f. Wirtschaftsinf., Abt. I, 1996, S. 51–61
- [BSVV98] Benedicenti, L.; Succi, G.; Vernazza, T.; Valerio, A.: Object Oriented Process Modeling with Fuzzy Logic. In: Carroll, J. et al. (Hrsg.): *Applied computing 1998 : Proceedings of the 1998 ACM Symposium on Applied Computing ; Atlanta Marriott Marquis Hotel, Atlanta, Georgia, February 27-March 1, 1998*. Danvers, MA : ACM Press, 1998, S. 267–271
- [BISi06] Blechar, M. J.; Sinur, J.: *Magic Quadrant for Business Process Analysis Tools, 2006*. Stamford, CT : Gartner Research, 2006. – Gartner RAS Core Research Note G00137850, 27 February 2006 R1713 06052006
- [BKNK03] Borgelt, C.; Kruse, R.; Nauck, D.; Klawonn, F.: *Neuro-Fuzzy-Systeme : Von den Grundlagen künstlicher Neuronaler Netze zur Kopplung mit Fuzzy-Systemen*. 3. Aufl. Wiesbaden : Vieweg, 2003
- [Cox99] Cox, E.: Striving for Imprecision: Fuzzy Knowledge Bases for Business Process Modeling. In: *PC AI 13* (1999), Nr. 4

- [Cox02] Cox, E.: Knowledge-Based Business Process Modeling: Complex Systems Design Through A Fusion of Computational Intelligence And Object-Oriented Models. In: *PC AI 16* (2002), Nr. 2, S. 15–23
- [DuAH05] Dumas, M.; van der Aalst, W. M. P.; ter Hofstede, A. H. M. (Hrsg.): *Process-aware Information Systems : Bridging People and Software through Process Technology*. Hoboken, New Jersey : Wiley, 2005
- [Fort02] Forte, M.: *Unschärfen in Geschäftsprozessen*. Berlin : Weißensee, 2002
- [HoKS92] Hoffmann, W.; Kirsch, J.; Scheer, A.-W.: Modellierung mit Ereignisgesteuerten Prozeßketten : Methodenhandbuch. In: Scheer, A.-W. (Hrsg.): *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Nr. 101, Saarbrücken : Universität des Saarlandes, 1992
- [Hüss03] Hüßelmann, C.: *Fuzzy-Geschäftsprozessmanagement*. Lohmar [u. a.] : Eul, 2003
- [KeNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". In: Scheer, A.-W. (Hrsg.): *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Nr. 89, Saarbrücken : Universität des Saarlandes, 1992
- [KeTe99] Keller, G.; Teufel, T.: *SAP R/3 prozeßorientiert anwenden : Iteratives Prozess-Prototyping mit Ereignisgesteuerten Prozessketten und Knowledge Maps*. 3. Aufl. Bonn [u. a.] : Addison-Wesley, 1999
- [Kind04] Kindler, E.: On the Semantics of EPCs : A Framework for Resolving the Vicious Circle. In: Desel, J.; Pernici, B.; Weske, M. (Hrsg.): *Business Process Management : Second International Conference (BPM 2004), Potsdam, Germany, June 17–18, 2004 ; Proceedings*. Springer, 2004, S. 82–97. – Zugl.: Technical Report, Reihe Informatik tr-ri-03–243, Institut für Informatik, Universität Paderborn, August 2003
- [Kind06] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In: *Data & Knowledge Engineering* 56 (2006), Nr. 1, S. 23–40
- [Lipp82] Lipp, H.-P.: Anwendung eines Fuzzy Petri Netzes zur Beschreibung von Koordinationssteuerungen in komplexen Produktionssystemen. In: *Wissenschaftliche Zeitschrift der Technischen Universität Karl-Marx-Stadt* 24 (1982), Nr. 5, S. 633–639
- [NüRu02] Nüttgens, M.; Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Desel, J.; Weske, M. (Hrsg.): *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen (Promise '2002) : Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 9.–11. Oktober 2002*. Bonn : Köllen, 2002, S. 64–77
- [ÖsWi03] Österle, H.; Winter, R.: Business Engineering. In: Österle, H.; Winter, R. (Hrsg.): *Business Engineering : Auf dem Weg zum Unternehmen des Informationszeitalters*. 2. Aufl. Berlin [u. a.] : Springer, 2003, S. 3–19
- [Reht98] Rehfeldt, M.: *Koordination der Auftragsabwicklung : Verwendung von unscharfen Informationen*. Wiesbaden : DUV [u. a.], 1998
- [Robs91] Robson, G. D.: *Continuous process improvement : simplifying work flow systems*. New York, NY : Free Press [u. a.], 1991
- [RoAa06] Rosemann, M.; van der Aalst, W. M. P.: A configurable reference modelling language. In: *Information Systems* (2006). – In Press, Corrected Proof
- [Sche96] Scheer, A.-W.: ARIS-House of Business Engineering : Von der Geschäftsprozeßmodellierung zur Workflow-gesteuerten Anwendung ; vom Business Process Reengineering zum Continuous Process Improvement. In: Scheer, A.-W. (Hrsg.): *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Nr. 133, Saarbrücken : Universität des Saarlandes, 1996
- [Sche97] Scheer, A.-W.: *Wirtschaftsinformatik : Referenzmodelle für industrielle Geschäftsprozesse*. 7. Aufl. Berlin [u. a.] : Springer, 1997
- [Sche01] Scheer, A.-W.: *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. 4. Aufl. Berlin [u. a.] : Springer, 2001
- [Sche02] Scheer, A.-W.: *ARIS – Vom Geschäftsprozess zum Anwendungssystem*. 4. Aufl. Berlin [u. a.] : Springer, 2002

- [ScTA05] Scheer, A.-W.; Thomas, O.; Adam, O.: Process Modeling Using Event-driven Process Chains. In: Dumas, M.; van der Aalst, W. M. P.; ter Hofstede, A. H. M. (Hrsg.): *Process-aware Information Systems : Bridging People and Software through Process Technology*. Hoboken, New Jersey : Wiley, 2005, S. 119–145
- [Thom06] Thomas, O.: Understanding the Term Reference Model in Information Systems Research: History, Literature Analysis and Explanation. In: Bussler, C.; Haller, A. (Hrsg.): *Business Process Management Workshops : BPM 2005 International Workshops, BPI, BPD, ENEL, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005 ; Revised Selected Papers*. Berlin : Springer, 2006, S. 484–496
- [ThHA02] Thomas, O.; Hüsselmann, C.; Adam, O.: Fuzzy-Ereignisgesteuerte Prozessketten : Geschäftsprozessmodellierung unter Berücksichtigung unscharfer Daten. In: Nüttgens, M.; Rump, F. J. (Hrsg.): *EPK 2002 : Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten ; Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)", 21.–22. November in Trier, Proceedings*. Bonn : GI, 2002, S. 7–16
- [Tiet99] Tietze, M.: *Einsatzmöglichkeiten der Fuzzy-Set-Theorie zur Modellierung von Unschärfe in Unternehmensplanspielen*. Göttingen : Unitext, 1999
- [Aals99] van der Aalst, W. M. P.: Formalization and verification of event-driven process chains. In: *Information and Software Technology* 41 (1999), Nr. 10, S. 639–650
- [AaDK02] van der Aalst, W. M. P.; Desel, J.; Kindler, E.: On the semantics of EPCs: A vicious circle. In: Nüttgens, M.; Rump, F. J. (Hrsg.): *EPK 2002 : Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten ; Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)", 21.–22. November in Trier, Proceedings*. Bonn : GI, 2002, S. 71–79
- [Völk98] Völkner, P.: *Modellbasierte Planung von Geschäftsprozeßabläufen : Entwicklung eines Entscheidungsunterstützungssystems auf Grundlage objektorientierter Simulation*. Wiesbaden : Gabler, 1998
- [VöWe02] Völkner, P.; Werners, B.: A simulation-based decision support system for business process planning. In: *Fuzzy Sets and Systems* 125 (2002), Nr. 3, S. 275–288
- [Broc03] vom Brocke, J.: *Referenzmodellierung : Gestaltung und Verteilung von Konstruktionsprozessen*. Berlin : Logos, 2003
- [Zade65] Zadeh, L. A.: Fuzzy sets. In: *Information and Control* 8 (1965), Nr. 3, S. 338–353
- [Zade73] Zadeh, L. A.: Outline of a new approach to the analysis of complex systems and decision processes. In: *IEEE Transactions on Systems, Man and Cybernetics* 3 (1973), Nr. 1, S. 24–44
- [ZALW93] Zimmermann, H.-J. et al. (Hrsg.): *Fuzzy-Technologien : Prinzipien, Werkzeuge, Potentiale*. Düsseldorf : VDI-Verl., 1993
- [ZvCh86] Zvieli, A.; Chen, P. P.-S.: Entity-Relationship Modeling and Fuzzy Databases. In: *International Conference on Data Engineering, February 5–7, 1986, Bonaventure Hotel, Los Angeles, California, USA*. Washington, DC : IEEE Computer Society Press, 1986, S. 320–327

# Validierung syntaktischer und anderer EPK-Eigenschaften mit PROLOG

Volker Gruhn, Ralf Laue  
{gruhn, laue}@ebus.informatik.uni-leipzig.de  
Lehrstuhl für Angewandte Telematik und E-Business\*  
Universität Leipzig, Fakultät für Informatik

**Abstract:** Die XML-basierte Austauschsprache EPML[MN04] für Ereignisgesteuerte Prozessketten wurde entwickelt, um eine Möglichkeit des Datenaustausches zwischen Modellierungswerkzeugen zu schaffen. Wird ein EPML-Modell aus einer fremden Quelle von einem Modellierungswerkzeug importiert, sollte zunächst geprüft werden, ob die importierte XML-Datei ein syntaktisch korrektes EPK-Modell darstellt. Hierzu müssen neben den syntaktischen Forderungen, die sich aus dem XML-Schema der Sprache EPML ergeben, weitere Eigenschaften getestet werden. Diese Eigenschaften (beispielsweise die Forderung, dass sich Ereignisse und Funktionen im EPK-Kontrollfluss abwechseln) sind in [Rum99], [Kel99] und [NR02] formalisiert. Mendling und Nüttgens zeigten in [MN03b], wie der überwiegende Teil der in [NR02] formalisierten Anforderungen mit Hilfe der Sprache Schematron validiert werden kann. Neben der Tatsache, dass zwei der Anforderungen aus [NR02] auf diese Weise nicht überprüft werden können, hat der Ansatz aus [MN03b] einen weiteren Nachteil: Er setzt voraus, dass das EPML-Modell zusätzliche Attribute zum Typ der Modellelemente enthält. Dies schafft jedoch Redundanz im EPML-Modell und vergrößert unnötig Größe und Komplexität der EPML-Austauschdateien. In unserem Beitrag zeigen wir, wie alle in [Rum99], [Kel99] und [NR02] beschriebenen Eigenschaften recht einfach mit Hilfe der Sprache PROLOG überprüft werden können, ohne die erwähnten zusätzlichen Attribute zu benötigen. Damit wird eine 100%-ige Überprüfung der Eigenschaften bei gleichzeitiger Reduzierung der Komplexität des Austauschformats erreicht. Wir geben ferner Beispiele für weitere Eigenschaften an, die mit unserem Ansatz effizient überprüft werden können.

## 1 Einführung

Ereignisgesteuerte Prozessketten (EPK) sind eine verbreitete Sprache zur Modellierung von Geschäftsprozessen. Obwohl diese Sprache von verschiedenen Werkzeugen unterstützt wird, gestaltet sich der Austausch von Modellen zwischen den Werkzeugen oft schwierig, da die Modelle in proprietären Dateiformaten gespeichert werden.

Diese Tatsache veranlasste Mendling und Nüttgens, EPML als XML-basierte Austauschsprache für EPK-Modelle vorzuschlagen[MN04]. Dies ermöglicht den Austausch von Mo-

---

\*Der Lehrstuhl für Angewandte Telematik und E-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG

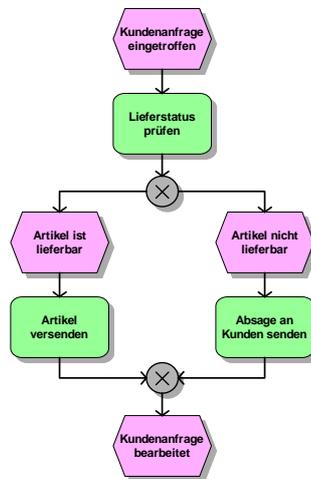


Abbildung 1: Eine Beispiel-EPK

dellen zwischen verschiedenen Modellierungs-, Simulations-, Monitoring- und anderen Werkzeugen.

Bevor ein Werkzeug eine EPML-Datei aus einer fremden Quelle importiert, sollte es sicherstellen, dass es sich bei der zu importierenden Datei tatsächlich um ein syntaktisch korrektes EPK-Modell handelt.

Unser Beitrag befasst sich mit dieser Prüfung von EPML-Modellen auf syntaktische Korrektheit.

Nachdem im Abschnitt 2 kurz die Sprache EPML skizziert wird, werden im Abschnitt 3 die Anforderungen, die an ein korrektes EPK-Modell zu stellen sind, besprochen. In Abschnitt 4 wird die in [MN03b] veröffentlichte Lösung für die Überprüfung dieser Regeln diskutiert. Im Abschnitt 5 präsentieren wir einen alternativen Ansatz und diskutieren in 6 dessen Vorteile gegenüber bisherigen Ansätzen. Schließlich zeigen wir in Abschnitt 7, dass das von uns vorgeschlagene Verfahren auch zum Testen anderer interessanter Eigenschaften genutzt werden kann.

## 2 Die EPC Markup Language (EPML)

Im Folgenden wird eine kurze skizzenhafte Einführung in die Sprache EPML gegeben. Diese ist bei weitem nicht vollständig, reicht aber zum Verständnis der in diesem Beitrag beschriebenen Verfahren aus. Für eine vollständige Einführung in EPML verweisen wir auf [MN04] sowie auf die Website [www.epml.de](http://www.epml.de).

Das unten gezeigte EPML-Fragment beschreibt den in Abb. 1 dargestellte EPK. Jeder Funktion der EPK entspricht ein function-Element, jedem Ereignis ein event-Element und

jedem Konnektor ein AND-, OR- oder XOR-Element. Jedes der genannten Elemente hat ein Attribut namens id, durch das es eindeutig identifiziert ist. Der Kontrollfluss wird durch arc-Elemente codiert. In unserem Beispiel bezeichnet das arc-Element mit dem Attribut id=12 den Kontrollfluss-Pfeil zwischen dem Element mit id=1 und dem Element mit id=2, also zwischen dem Ereignis „Kundenanfrage eingetroffen“ und der Funktion „Lieferstatus prüfen“.

```
<?xml version="1.0" encoding="UTF-8"?>
<epml:epml xmlns:epml="http://www.epml.de"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="epml_1_draft.xsd">
  <epc EpcId="1" Name="EPC">
    <event id="1">
      <name>Kundenanfrage eingetroffen</name>
    </event>
    <function id="2">
      <name>Lieferstatus prüfen</name>
    </function>
    <xor id="4">
      <name/>
    </xor>
    <event id="5">
      <name>Artikel ist lieferbar</name>
    </event>
    <event id="6">
      <name>Artikel nicht lieferbar</name>
    </event>
    <function id="7">
      <name>Artikel versenden</name>
    </function>
    <function id="8">
      <name>Absage an Kunden senden</name>
    </function>
    <xor id="9">
      <name/>
    </xor>
    <event id="10">
      <name>Kundenanfrage bearbeitet</name>
    </event>
    <arc id="11">
      <flow source="1" target="2"/>
    </arc>
    ... sowie je ein weiteres <arc>-Element für jeden Pfeil im Modell
  </epc>
</epml:epml>
```

### 3 Syntaktische Anforderungen an EPKs

Eine ausführliche formale Diskussion der Syntax von EPKs findet sich in [Rum99], [Kel99] und [NR02]. Im Unterschied zu diesen Arbeiten betrachten wir hier zur Vereinfachung nur EPKs ohne Prozesswegweiser und hierarchische Funktionen.

Der Grund hierfür liegt darin, dass wir beabsichtigen, die in Abschnitt 5 gezeigten Syntaxprüfungen in das Werkzeug EPCTools[Cun04] zu integrieren, das Prozesswegweiser und hierarchische Funktionen nicht darstellen kann. Grundsätzlich sollten aber mit dem vorgestellten Ansatz auch die syntaktischen Eigenschaften von EPKs mit Prozesswegweisern und hierarchischen Funktionen problemlos überprüft werden können.

Ausgehend von [Rum99], [Kel99] und [NR02] nennen wir im Folgenden die Eigenschaften, die ein Graph  $G$  erfüllen muss, damit  $G$  eine syntaktisch korrekte EPK ist. Dabei bezeichnen wir mit  $K$  die Knotenmenge und mit  $E \subseteq K \times K$  die Kantenmenge von  $G$ . Die Knotenmenge  $V$  ist die Vereinigung der folgenden fünf paarweise disjunkten Mengen:

- $F$  (Menge der Funktionen)
- $E$  (Menge der Ereignisse)
- $V_{xor}$  (Menge der XOR-Verknüpfungsoperatoren)
- $V_{or}$  (Menge der OR-Verknüpfungsoperatoren)
- $V_{and}$  (Menge der AND-Verknüpfungsoperatoren)

Für eine syntaktisch korrekte EPK müssen die folgenden Eigenschaften erfüllt sein:

1. Der Graph  $G$  ist gerichtet.
2. Der Graph  $G$  ist zusammenhängend.
3. Der Graph  $G$  ist endlich.
4. Enthält der Graph  $G$  eine Kante von  $a$  nach  $b$ , so gibt es in  $G$  keine Kante von  $b$  nach  $a$  und keine weitere Kante von  $a$  nach  $b$ .
5. Die Mengen  $E$  und  $F$  sind nicht leer, d.h. es gibt mindestens ein Ereignis und mindestens eine Funktion.
6. Funktionen besitzen genau eine eingehende und genau eine ausgehende Kante.
7. Ereignisse besitzen genau eine eingehende oder genau eine ausgehende Kante. (Besitzt ein Ereignis genau eine eingehende und keine ausgehende Kante, nennen wir es Endereignis. Besitzt ein Ereignis keine eingehende und genau eine ausgehende Kante, nennen wir es Starterereignis.)
8. Verknüpfungsoperatoren (also Knoten, die zur Menge  $V_{xor} \cup V_{or} \cup V_{and}$  gehören) haben entweder genau eine eingehende und mehr als eine ausgehende Kante (dann heißen sie Split) oder mehr als eine eingehende und genau eine ausgehende Kante (dann heißen sie Join).
9. Der Graph  $G$  enthält keinen gerichteten Zyklus, der nur aus Verknüpfungsoperatoren (also Knoten, die zur Menge  $V_{xor} \cup V_{or} \cup V_{and}$  gehören) besteht.

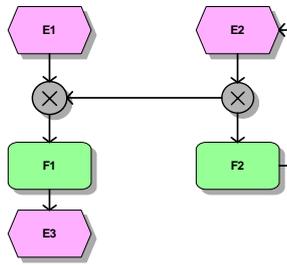


Abbildung 2: EPK, für die nur Eigenschaft 14 verletzt ist

10. Auf ein Ereignis folgen im Graphen immer  $n$  Verknüpfungsoperatoren ( $n=0,1,\dots$ ), direkt gefolgt von einer Funktion.
11. Auf eine Funktion folgen im Graphen immer  $n$  Verknüpfungsoperatoren ( $n=0,1,\dots$ ), direkt gefolgt von einem Ereignis.
12. Auf ein Ereignis folgen im Graphen nie  $n$  AND-Konnektoren ( $n=0,1,\dots$ ), direkt gefolgt von einem Split aus der Menge  $V_{xor} \cup V_{or}$ .
13. Es gibt mindestens ein Start- und mindestens ein Endereignis.
14. Für jeden Knoten im Graphen gibt es einen gerichteten Pfad von einem Startereignis zu diesem Knoten.
15. Für jeden Knoten im Graphen gibt es einen gerichteten Pfad von diesem Knoten zu einem Endereignis.

Die Eigenschaften 1 bis 15 sind im Wesentlichen [Rum99], [Kel99] und [NR02] entnommen. Eigenschaft 3 wird in keiner der angegebenen Arbeiten ausdrücklich genannt, aber stillschweigend vorausgesetzt. Die Eigenschaften 14 und 15 werden in [Kel99], nicht aber in [Rum99] und [NR02] erwähnt. Da diese Eigenschaft keineswegs, wie man vielleicht vermuten könnte, aus den anderen Eigenschaften folgt (siehe Gegenbeispiel in Abb. 2), ist es aber sinnvoll, diese Forderung zu stellen. Auf die in [MN03b] genannte Forderung, dass der Graph keine reflexive Kante enthält (also eine Kante, die einen Knoten mit sich selbst verbindet) verzichten wir, da sich diese Forderung als Folgerung aus den Forderungen 9, 10 und 11 ergibt.

## 4 Bekannte Lösungsansätze

Mendling und Nüttgens untersuchten in [MN03b], wie gut sich XML-Schemasprachen dazu eignen, die im vorigen Abschnitt genannten Syntaxforderungen zu validieren. Es wurde gezeigt, dass mit XML Schema [Wor01a, Wor01b] sowie Relax NG[CM01] nur die einfachsten Eigenschaften (Eigenschaften 1, 3, 4 und 5) validieren lassen. Daher sind diese Schemasprachen nicht geeignet, um die syntaktische Korrektheit von EPML-Dateien

zu überprüfen. Wesentlich bessere Resultate wurden in [MN03b] bei Nutzung der Sprache Schematron[Jel02] erzielt. Schematron gestattet es, erwünschte Eigenschaften eines XML-Dokuments unter Nutzung von XPath[Wor99] zu beschreiben. Diese Abfragesprache erlaubt komplexe Operationen, z.B. logische Verknüpfungen. Daher kann, wie in [MN03b] gezeigt, Schematron benutzt werden, um den größten Teil der Forderungen für syntaktisch korrekte EPML-Dateien zu validieren. Allerdings können auch durch den in [MN03b] präsentierten Ansatz nicht alle Eigenschaften validiert werden: Die Eigenschaften 2, 9, 14 und 15 können nicht mit Schematron geprüft werden.

Um Schematron wie in [MN03b] vorgeschlagen zur Syntaxvalidierung nutzen zu können, hat man allerdings einen gewissen Preis zu zahlen: Zu den Elementen der EPML-Datei müssen zusätzliche Attribute notiert werden, die Aussagen über die Knoten der EPK treffen. Beispielsweise wird das Endereignis der in Abb. 1 dargestellten EPK statt

```
<event id="10">
  <name>Kundenanfrage bearbeitet</name>
</event>
<arc id="11">
  <flow source="1" target="2"/>
</arc>
```

jetzt notiert:

```
<event id="10" type="EventEnd">
  <name>Kundenanfrage bearbeitet</name>
</event>
<arc id="11" type="EventFunctionArc">
  <flow source="1" target="2"/>
</arc>
```

Damit wird angegeben, dass es sich beim Ereignis „Kundenanfrage bearbeitet“ um ein Endereignis handelt und die gerichtete Kante zwischen dem Startereignis (mit der id=1) und der Funktion „Lieferstatus prüfen“ (mit der id=2) den Typ „Ereignis-Funktion-Kante“ hat[MN03a]. Die zusätzlichen Attribute sind im XML Schema der Sprache EPML seit der Anfangsversion 1.0 vorhanden. Ihre Nutzung hat allerdings zwei Nachteile: Zum einen werden die EPML-Dokumente größer und komplexer. Zum zweiten sind die Informationen, die aus diesen Attributen gewonnen werden können, ausnahmslos *redundant*. Die kurze Version der EPML-Datei (ohne zusätzliche Attribute) enthält nämlich schon alle Informationen, um beispielsweise festzustellen, dass es sich beim Ereignis „Kundenanfrage bearbeitet“ um ein Endereignis handelt. Da eine Austauschsprache möglichst einfach und ohne redundante Informationen sein sollte, halten wir einen Verzicht auf die zusätzlichen Attribute für wünschenswert. Im kommenden Absatz werden wir ein Validierungsverfahren vorstellen, das ohne die zusätzlichen „type“-Attribute arbeitet.

Neben den bereits genannten Lösungsansätzen, die bereits in [MN03b] auf ihre Eignung zur Validierung von EPML-Dateien bewertet wurden, ist die Constraint Language in XML (CLiXML)[DJ06] betrachtenswert. Diese Sprache wurde mit dem Ziel entwickelt, komplexere Validierungen von XML-Dateien durchführen zu können. Die Validierung der

meisten Regeln aus Abschnitt 3 sollte mit CLiXML leicht möglich sein. Allerdings führt [DJ06] gerade XML-Darstellungen von gerichteten Graphen, bei denen Knoten und Kanten analog zu EPML kodiert werden, als Beispiel an, bei dem Validierungen mit CLiXML deutlich erschwert werden.

## 5 Eigenschaftsüberprüfung mit PROLOG

Unser Ansatz zur Validierung der Eigenschaften nutzt das Programmierprinzip der logischen Programmierung, um Eigenschaften von EPML-Dateien zu validieren. Die Grundidee besteht darin, ein EPML-Dokument in eine Menge logischer Aussagen (z.B. „Es gibt eine Kante vom Knoten 1 zum Knoten 2“) zu übersetzen. Diese logischen Aussagen werden als Fakten in der logischen Programmiersprache PROLOG dargestellt. Die PROLOG-Fakten enthalten die selbe Information wie die ursprüngliche EPML-Datei und bilden einen Teil der PROLOG-Wissensbasis. Den zweiten Teil der PROLOG-Wissensbasis bilden Regeln, mit denen wir die „Sprache der EPKs“ in PROLOG „erklären“. Eine solche Regel besagt beispielsweise, dass ein Ereignis Endereignis ist, wenn es keine ausgehenden Kanten hat.

Die PROLOG-Wissensbasis enthält somit alle Informationen der ursprünglichen EPML-Datei und Regeln, die die Eigenschaften von EPK-Elementen beschreiben. Ausgehend von dieser Wissensbasis können wir nun dem PROLOG-System Fragen stellen - etwa, ob es mindestens ein Starterereignis gibt. Insbesondere können wir für jede der in Abschnitt 3 aufgeführten Regeln erfragen, ob sie erfüllt sind. Die einzelnen Schritte unseres Ansatzes sind in den folgenden Unterabschnitten dargestellt.

### 5.1 Übersetzung von EPML in PROLOG-Fakten

Die in unserem durchgehenden Beispiel genutzte EPML-Datei enthält Informationen zu den Knoten und Kanten des Graphen, der die EPK darstellt. So enthält der Abschnitt

```
<event id="10">
  <name>Kundenanfrage bearbeitet</name>
</event>
<arc id="11">
  <flow source="1" target="2"/>
</arc>
```

drei logische Aussagen:

1. Es gibt im Modell ein Ereignis mit id=10
2. Das Ereignis mit id=10 hat den Namen „Kundenanfrage bearbeiten“
3. Es gibt im Modell eine Kante vom Element mit id=1 zum Element mit id=2.

Im folgenden Listing sind die logischen Aussagen zu der in Abb. 1 dargestellten EPK, formuliert als Fakten der Sprache PROLOG, dargestellt. Die Bedeutung der Prädikate sollte ohne weitere Erklärung verständlich sein.

```

event(i_1).
elementname(i_1,'Kundenanfrage eingetroffen').
event(i_5).
elementname(i_5,'Artikel ist lieferbar').
event(i_6).
elementname(i_6,'Artikel nicht lieferbar').
event(i_10).
elementname(i_10,'Kundenanfrage bearbeitet').
function(i_2).
elementname(i_2,'Lieferstatus prüfen').
function(i_7).
elementname(i_7,'Artikel versenden').
function(i_8).
elementname(i_8,'Absage an Kunden senden').
arc(i_1,i_2).
arc(i_2,i_4).
arc(i_4,i_5).
arc(i_4,i_6).
arc(i_5,i_7).
arc(i_6,i_8).
arc(i_7,i_9).
arc(i_8,i_9).
arc(i_9,i_10).
xor(i_4).
xor(i_9).

```

Diese Fakten enthalten die selben Informationen wie Abb. 1 bzw. die zugehörige EPML-Datei. Um die Fakten aus der EPML-Datei zu generieren, benutzen wir eine einfache XSLT-Transformation, die wir im Anhang A angeben.

## 5.2 Die Regeln der Wissensbasis

Damit das PROLOG-System Schlussfolgerungen zu den aus der EPML-Datei gewonnenen Fakten ziehen kann, wird ihm ein gewisses „Grundwissen“ über EPKs in Form von logischen Regeln mitgeteilt. Die meisten dieser Regeln sind einfache Informationen zu verwendeten Bezeichnungsweisen. Dies soll an einigen Beispielregeln gezeigt werden:

```

connector(I) :- clause(and(I),true) ; clause(or(I),true);
clause(xor(I),true).

```

besagt, dass wir unter einem Verknüpfer einen AND-, XOR- oder OR-Verknüpfer verstehen.

```

no_outgoing_arcs(X) :- not(arc(X,_)).

```

besagt, dass für ein Element das Prädikat „hat keine ausgehenden Kanten“ erfüllt ist, wenn es keine Kante (d.h. kein arc-Element in der EPML-Datei) gibt, die von diesem Element ausgeht.

```
endevent(X) :- event(X),no_outgoing_arcs(X).
```

definiert ein Ereignis als Endereignis, wenn es keine ausgehenden Kanten hat.

Die komplette Regelbasis ist in Anhang B aufgeführt. Hervorzuheben ist die Kürze der Regeln: Lediglich die Regeln, die Aussagen zu Themen wie „Erreichbarkeit“ und „Zusammenhang“ treffen, benötigen mehr als eine einzelne Programmzeile, und auch diese Regeln zur Erreichbarkeit umfassen gerade einmal 13 PROLOG-Zeilen.

### 5.3 Die Validierung der EPK-Korrektheitsregeln

Nachdem durch die im vorigen Abschnitt besprochenen Regeln die für EPKs definierten Begriffe dem PROLOG-System „erklärt“ wurden, können wir nun Anfragen an das System stellen. Insbesondere können wir fragen, ob die im Abschnitt 3 aufgeführten Regeln für eine vorliegende EPK erfüllt sind. Dabei gehen wir davon aus, dass bereits geprüft wurde, dass die zu Grunde liegende EPML-Datei dem XML-Schema für EPML[MN04] entspricht. Dadurch sind die Regeln 1, 3 und das Verbot mehrfacher Kanten zwischen zwei Knoten (Teil von Regel 4) bereits „automatisch“ erfüllt.

Die Anfragen, mit denen die im Abschnitt 3 genannten Syntaxregeln für eine gegebene EPK überprüft werden können, sind im Anhang C angegeben. In der Regel sind sie so aufgebaut, dass dem PROLOG-System die Aufgabe gestellt wird, ein Beispiel zu finden, das die entsprechende Regel verletzt. Somit erhält der Benutzer nicht nur die Aussage, dass eine Syntaxregel verletzt ist, sondern es wird auch gezeigt, wo dies der Fall ist.

Einige Beispiele sollen das verdeutlichen: Als Teil von Regel 4 ist zu zeigen, dass keine Kante von a nach b existieren kann, wenn es schon eine Kante von b nach a gibt. Die entsprechende Anfrage an das PROLOG-System lautet:

```
prop4(X,Y) :- arc(X,Y),arc(Y,X).
```

Das Komma steht in PROLOG für die logische und-Verknüpfung, die angegebene Klausel drückt also die Forderung „finde zwei Elemente X und Y, für die sowohl eine Kante von X nach Y als auch eine Kante von Y nach X existiert“ aus. Ist die entsprechende Eigenschaft verletzt, werden die Knoten, die zur Verletzung führen, ausgegeben. Bei einem korrekten Modell antwortet das PROLOG-System auf die Anfrage prop4(X,Y) mit „no“, was heißt, dass keine Verletzung gefunden werden kann.

Da mit der im vorigen Kapitel definierten Regelbasis grundlegende Sprechweisen wie „ein Element X ist mit einem Element Y (möglicherweise über Verknüpfungsoperatoren) verbunden“<sup>1</sup> definiert wurden, gestalten sich die Anfragen nach Stellen, an denen eine Regel

<sup>1</sup>Diese Beziehung zwischen zwei Elementen X und Y ist im Prädikat successor definiert

verletzt wurden, recht einfach. Um die in Eigenschaft 10 aufgestellte Forderung, dass ein Ereignis immer (möglicherweise über Verknüpfungsoperatoren) mit einer Funktion verbunden ist zu prüfen, stellt man dem PROLOG-System die Frage:

```
prop10(X,Y) :- event(X), successor(X,Y), event(Y).
```

Damit wird das PROLOG-System veranlasst ein Gegenbeispiel zu suchen: Ein Ereignis, dass (möglicherweise über Verknüpfungsoperatoren) nicht mit einer Funktion, sondern mit einem Ereignis verbunden ist. Wie immer, besagt auch hier die Antwort „no“ auf die Anfrage `prop10(X,Y)`, dass kein Gegenbeispiel gefunden werden kann, also die entsprechende Eigenschaft validiert werden konnte.

## 6 Bewertung

Der im vorangehenden Abschnitt vorgestellte Ansatz folgt einem gänzlich anderen Prinzip als die in [MN03b] diskutierten Schemasprachen. Indem die tatsächlich in der EPML-Datei steckende Information in logische Aussagen übersetzt wird, kann eine logikbasierte Sprache wie PROLOG komplexe Fragestellungen zum Modell leicht beantworten. Der zu notierende Code ist ähnlich kurz wie die in [MN03b] vorgeschlagenen Schematron-Tests. Da die gängigen PROLOG-Systeme wie das von uns verwendete SWI-Prolog offene Schnittstellen zu zahlreichen anderen Programmiersprachen bieten, können die Tests leicht in andere Tools eingebunden werden.

Ein Vorteil unserer Lösung ist, dass erstmals *alle* im Abschnitt 3 aufgeführte Regeln validiert werden können. Im Gegensatz dazu ist es mit dem Ansatz von [MN03b] nicht möglich, die Regeln 2, 9, 14 und 15 zu validieren, da dies ein teilweises Durchlaufen des Graphen erfordert.

Wir widersprechen der Aussage aus [MN03a], dass dieses Durchlaufen des Graphen teuer und folglich unperformant ist. Da EPK-Modelle in der Praxis wohl höchstens eine dreistellige Zahl von Elementen haben, erweisen sich die Tests auch der Regeln 2, 9, 14 und 15 selbst an relativ großen praktischen Modellen als hinreichend schnell, obwohl wir zugeben müssen, dass unsere Prolog-Regel zum Zusammenhang von Graphen nicht besonders effizient programmiert ist. In unseren Tests wurden alle diese Eigenschaften in weniger als einer Sekunde geprüft. Hinzu kommt, dass bei der in [MN03a] vorgeschlagenen Nutzung zusätzlicher `type`-Attribute für die Elemente der EPML-Datei keineswegs auf das Durchlaufen des Graphen verzichtet werden kann. Dieser muss zwar dank der `type`-Attribute nicht mehr bei der Validierung der Syntaxregeln erfolgen, dafür aber beim Erzeugen der EPML-Datei, da schließlich zu diesem Zeitpunkt bestimmt werden muss, wie die `type`-Attribute zu belegen sind.<sup>2</sup>

---

<sup>2</sup>Theoretisch könnte man zwar beim Bauen eines EPK-Modells in einem Editor die `type`-Attribute immer sofort in dem Moment bestimmen, in dem ein neues Element gezeichnet wird. Das setzt allerdings voraus, dass beim Editieren immer nur ein Einzelement mit dem schon modellierten Teil der EPK verbunden wird. Werden Teile der EPK zunächst unabhängig voneinander modelliert, um schließlich durch Kanten verbunden zu werden, ist im Allgemeinen ein Graph-Durchlauf zur Bestimmung der `type`-Attribute notwendig.

Ein Vorteil unserer Lösung ist es, dass in den Dateien des Austauschformats EPML auf die zusätzlichen type-Attribute verzichtet werden kann. Dies führt dazu, dass das Austauschformat einfacher wird und folglich seine Verwendung einfacher implementiert werden kann. Generell halten wir die Notation der type-Attribute für nicht erstrebenswert, da diese Attribute (wenn sie denn korrekt sind, was in jedem Einzelfall ohnehin zunächst geprüft werden muss) lediglich redundante Informationen liefern.

## 7 Weiterführende Anwendungsmöglichkeiten unseres Ansatzes

Hauptziel dieses Beitrags ist es, die Nutzung logischer Programmierung zur Validierung von Eigenschaften ereignisgesteuerter Prozessketten zu zeigen. Wir wollen aber noch darauf hinweisen, dass sich die Anwendungsmöglichkeiten unserer Ideen keinesfalls auf die in Abschnitt 3 genannten Eigenschaften beschränken. Vielmehr kann man dem Prologsystem auch weitere Fragen zu anderen (statischen) Eigenschaften einer EPK stellen.

Beispielsweise kann es beim Model Checking großer EPK-Modelle sinnvoll sein, diese Modelle in kleinere Teilmodelle zu untergliedern, die dann getrennt im Model Checker untersucht werden. Damit kann sich der zu untersuchende Zustandsraum erheblich verringern. Es entsteht die Frage, an welchen Stellen man ein EPK-Modell „zerschneiden“ kann, so dass die entstehenden Teilmodelle selbst syntaktisch korrekte EPKs sind (oder zumindest durch das Hinzufügen der obligatorischen Start- und Endereignisse zu solchen ergänzt werden können.) Offenbar ist ein solches „Zerschneiden“ an solchen Kanten möglich, die die einzige Verbindung zwischen zwei Zusammenhangskomponenten des die EPK repräsentierenden Graphen darstellen. Eine solche Kante hat die Eigenschaft, dass der Graph nicht mehr zusammenhängend ist, wenn diese Kante entfernt wird. Ob das für eine Kante von X nach Y der Fall ist, prüft man leicht mit der folgenden Prolog-Regel:

```
cut_here(X,Y) :- arc(X,Y),
                (retract(arc(X,Y)),prop2,assertz(arc(X,Y)));
                (assertz(arc(X,Y),fail))).
```

In diesem Beispiel wird zunächst verlangt, dass zwischen X und Y tatsächlich eine Kante existiert, dargestellt durch den Fakt `arc(X,Y)`. Dann wird mittels `retract` der Fakt, dass diese Kante existiert, aus der Wissensbasis entfernt. Nun wird die Eigenschaft 2 aus Abschnitt 3 (Zusammenhang des Graphen) für die um die Kante  $X \rightarrow Y$  verminderte Wissensbasis getestet. Wie bei all unseren Tests, wird die Antwort „yes“ geliefert, wenn ein Gegenbeispiel gefunden wurde, also der Graph nicht (mehr) zusammenhängend ist. Die anschließenden `assertz`-Befehle dienen dazu, anschließend den ursprünglichen Fakt, dass es eine Kante von X nach Y gibt, wieder in die Wissensbasis aufzunehmen.

Weitere denkbare Anwendungsfälle sind das Erkennen von Modellfehlern (etwa nach dem in [vDMvdA06] vorgestellten Ansatz) oder das Aufspüren schlechten Modellierungsstils (z.B. von Startereignissen, von denen aus direkt in einen durch einen Split-Konnektor eingeleiteten Kontrollblock „hineingesprungen“ wird).

## 8 Zusammenfassung

In unserem Beitrag haben wir einen auf dem Prinzip der logischen Programmierung beruhenden Ansatz zur Validierung der syntaktischen Korrektheit von EPML-Dateien vorgestellt. Dieser kann im Gegensatz zu bisherigen Lösungen[MN03b] erstmals *alle* in der Literatur[Rum99, Kel99, NR02] aufgestellten Forderungen an korrekte Syntax validieren. Dabei kommt er mit einem knapperen, redundanzfreien Format der Austauschsprache EPML aus, was deren Nutzung als Austauschformat erleichtern kann.

### Literatur

- [CM01] James Clark und Murata Makoto. *RELAX NG Specification*. OASIS, 1. Auflage, December 2001.
- [Cun04] Nicolas Cuntz. Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Diplomarbeit, Universität Paderborn, 2004.
- [DJ06] Ulrich Ultes Nitsche Dominik Jungo, David Buchmann. Testing of semantic properties in XML documents. In *Proceedings of the 4th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, Paphos, Cyprus, 2006*.
- [Jel02] Rick Jelliffe. *The Schematron Assertion Language 1.5*. Academia Sinica Computing Centre, October 2002.
- [Kel99] Gerhard Keller. *SAP R/3 prozessorientiert anwenden*. Addison-Wesley, München, 1999.
- [MN03a] J. Mendling und M. Nüttgens. EPC Modelling based on Implicit Arc Types, 2003.
- [MN03b] Jan Mendling und Markus Nüttgens. EPC Syntax Validation with XML Schema Languages. In *EPK*, Seiten 19–30, 2003.
- [MN04] J. Mendling und M. Nüttgens. Exchanging EPC Business Process Models with EPML. In M. Nüttgens und J. Mendling, Hrsg., *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004*, Seiten 61–80, March 2004.
- [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.
- [Rum99] Frank J. Rump. *Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten*. B. G. Teubner Verlag Stuttgart Leipzig, 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.
- [Wor99] World Wide Web Consortium. *XML Path Language (XPath)*, November 1999.
- [Wor01a] World Wide Web Consortium. *XML Schema Part 1: Structures*, May 2001.
- [Wor01b] World Wide Web Consortium. *XML Schema Part 2: Datatypes*, May 2001.

## A XSLT-Stylesheet zur Transformation der EPML-Datei in PROLOG-Fakten

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="epc">
  <xsl:apply-templates select="event"/>
  <xsl:apply-templates select="function"/>
  <xsl:apply-templates select="arc"/>
  <xsl:apply-templates select="and"/>
  <xsl:apply-templates select="or"/>
  <xsl:apply-templates select="xor"/>
</xsl:template>

<xsl:template match="function">
  function(i_<xsl:value-of select="@id"/>).
  elementname(i_<xsl:value-of select="@id"/>,'<xsl:value-of select="name"/>').
</xsl:template>

<xsl:template match="event">
  event(i_<xsl:value-of select="@id"/>).
  elementname(i_<xsl:value-of select="@id"/>,'<xsl:value-of select="name"/>').
</xsl:template>

<xsl:template match="and">
  and(i_<xsl:value-of select="@id"/>).
</xsl:template>

<xsl:template match="or">
  or(i_<xsl:value-of select="@id"/>).
</xsl:template>

<xsl:template match="xor">
  xor(i_<xsl:value-of select="@id"/>).
</xsl:template>

<xsl:template match="arc">
  arc(i_<xsl:value-of select="flow/@source"/>,
  i_<xsl:value-of select="flow/@target"/>).
</xsl:template>

</xsl:stylesheet>
```

## B PROLOG-Regeln

```
% some useful facts about arcs
uarc(X,Y) :- arc(X,Y);arc(Y,X).
more_than_one_incoming_arcs(X) :- arc(A,X),arc(B,X),A \==B.
more_than_one_outgoing_arcs(X) :- arc(X,A),arc(X,B),A \==B.
no_incoming_arcs(X) :- not(arc(_,X)).
no_outgoing_arcs(X) :- not(arc(X,_)).
% successor(X,Y) means that X is followed by Y (possibly via some connectors)
% (Note: prop9 must be tested before calling successor(X,Y) in order to avoid
% infinite cycles.)
successor(X,Y) :- arc(X,Y).
successor(X,Y) :- arc(X,C),connector(C),successor(C,Y).
% types of elements
connector(I) :- clause(and(I),true) ; clause(or(I),true) ; clause(xor(I),true).
element(I) :- event(I);function(I);connector(I).
startevent(X) :- event(X),no_incoming_arcs(X).
endevent(X) :- event(X),no_outgoing_arcs(X).
% When using split(X) and join(X), we assume that prop8 already has been tested.
split(X) :- connector(X),more_than_one_outgoing_arcs(X).
join(X) :- connector(X),more_than_one_incoming_arcs(X).
% paths and reachability
path(A,B,Path) :- travel(A,B,[A],Q),
    reverse(Q,Path).

travel(A,B,P,[B|P]) :- arc(A,B).
travel(A,B,Visited,Path) :- arc(A,C),
    C \== B,
    \+member(C,Visited),
    travel(C,B,[C|Visited],Path).

% Neighbourhood contains all elements of List and their neighbours.
neighbourhood(List,Neighbourhood) :-
    findall(N,(member(X,List),uarc(X,N)),Neighbours),
    union(List,Neighbours,List_And_Neighbours),
    list_to_ord_set(List_And_Neighbours,Neighbourhood).

% Find the elements which are weakly connected to at least one element in List
connected_elements(List,X) :- neighbourhood(List,List), X = List.
connected_elements(List,X) :- neighbourhood(List,Neighbourhood),
    connected_elements(Neighbourhood,X).
```

## C Anfragen zur Validierung der EPK-Syntaxregeln

```
% Property 1 - The EPC is a directed graph
% Property 2 - The EPC is a coherent graph
prop2 :- element(E1),connected_elements([E1],Y),!,element(X),not(member(X,Y)).
% Property 3 - The EPC is finite
% Property 4 - There are no multiple arcs between two vertices
% (no need to check this, because they are already forbidden by the EPML Schema)
% The EPC is an antisymmetric graph
prop4(X,Y) :- arc(X,Y),arc(Y,X).
% Property 5 - The set of events is not empty and the set of functions is not empty
prop5 :- not(clause(event(_),true)).
prop5 :- not(clause(function(_),true)).
% Property 6 - Functions have exactly one incoming arc and exactly one outgoing arc.
prop6(X) :- function(X),
           (more_than_one_outgoing_arcs(X);more_than_one_incoming_arcs(X);
            no_incoming_arcs(X);no_outgoing_arcs(X)).
% Property 7 - Events have at most one incoming and at most one outgoing arc.
% (note: This would allow events with no incoming and no outgoing arc, but this
% would be detected when verifying prop 2)
prop7(X) :- event(X),
           (more_than_one_incoming_arcs(X);more_than_one_outgoing_arcs(X)).
% Property 8 - Two kinds of connectors are allowed:
%   split connectors with exactly one incoming arc and at least two outgoing arcs
%   join connectors with at least two incoming arc and exactly one outgoing arc.
%   (connectors with no incoming and no outgoing arcs are already detected
%   when verifying prop 2)
prop8(X) :- connector(X),
           more_than_one_outgoing_arcs(X),more_than_one_incoming_arcs(X).
% Property 9 - Cycles made up only of connectors are forbidden
connectors_only([L|Rest]) :- connector(L),!,connectors_only(Rest).
connectors_only([]).
prop9(Path) :- path(X,X,Path),connectors_only(Path).
% Property 10 - If an event has an outgoing arc, this arc connects the event
%   (possibly via one or more connectors) to a function.
prop10(X,Y) :- event(X),successor(X,Y),event(Y).
% Property 11- The outgoing arc of a function connects this function
%   (possibly via one or more connectors) to an event
prop11(X,Y) :- function(X),successor(X,Y),function(Y).
% Property 12 - If an event is followed by one or more connectors,
%   none of these connectors is an XOR-split or OR-split
prop12(X) :- event(X),successor(X,Y) ,
           (clause(or(Y),true) ; clause(xor(Y),true)).
% Property 13a - There is at least one start event
startevents(L) :- findall(X,startevent(X),L).
prop13a :- startevents(_) == [].
% Property 13b - There is at least one end event
endevents(L) :- findall(X,endevent(X),L).
prop13b :-endevents(_) == [].
% Property 14 - For every element X, there is a path from a start event to X
reachable_from_startevent(X) :- startevent(S),path(S,X,_).
prop14(X) :- element(X),not(startevent(X)),not(reachable_from_startevent(X)).
% Property 15 - For every element X, there is a path from X to an end event
reaches_endevent(X) :- endevent(E),path(X,E,_).
prop15(X) :- element(X),not(endevent(X)),not(reaches_endevent(X)).
```

## D Anmerkungen zum Programmcode

1. Der oben angeführte Code wurde mit SWI-Prolog, Version 5.2.13 getestet. Er sollte jedoch auch auf anderen PROLOG-Systemen lauffähig sein, ggf. müssen nicht vorhandene Builtin-Prädikate wie `list_to_ord_set` zum System hinzugefügt werden.
2. Prinzipiell ist es auch problemlos realisierbar, dass das PROLOG-System die EPML-Datei direkt einliest und verarbeitet. Das von uns verwendete System SWI-Prolog stellt hierfür die Bibliothek `sgml2p` zur Verfügung. Der Umweg über die XSLT-Transformation könnte dann entfallen. Wir haben in unserem Beitrag den Weg über die XSLT-Transformation gewählt, da so der Code leichter lesbar wird.
3. Gelegentlich ist die Reihenfolge, in der die Regeln aufgerufen werden, wichtig. Dies ist dann als Kommentar im Code vermerkt. Beispielsweise muss zunächst sichergestellt werden, dass es mindestens ein Ereignis gibt (`Property5`), bevor das entsprechende Prädikat `event` an anderer Stelle verwendet wird, um weitere Eigenschaften von Ereignissen zu prüfen.

# Nautilus Event-driven Process Chains: Syntax, Semantics, and their mapping to BPEL

Oliver Kopp, Tobias Unger, Frank Leymann

Institute of Architecture of Application Systems  
University of Stuttgart  
Universitätsstraße 38  
70569 Stuttgart

{oliver.kopp, tobias.unger, frank.leymann}@iaas.uni-stuttgart.de

**Abstract:** Nautilus Event-driven Process Chains (N-EPCs) are a variant of Event-driven process chains allowing multiple events between functions. This allows events to be used as transition conditions in a mapping to the Business Process Execution Language for Web Services (BPEL). We will give a formal definition of N-EPCs and show how they can be mapped to BPEL. A close look will be taken how connectors can be eliminated while preserving their semantics.

## 1 Introduction

Event Driven Process Chains (EPCs) were introduced in 1992 as an intuitive metamodel for process modeling [KNS92]. The business process modeling tool Nautilus [Ge06a] is using a slightly modified version of EPCs, which we call Nautilus Event-Driven Process Chains (N-EPCs). The main difference is that in N-EPCs functions and events need not alternate, allowing multiple events between functions. This enables a more detailed modeling of the control flow by allowing nested conditions such as “amount > 1 million euros and premium customer”. Figure 1 illustrates this using a simplified process for loan application processing. Surely, this can also be modeled using the traditional EPCs of [KNS92] by adding functions that execute nothing. However, the N-EPC approach makes this addition obsolete and eases reading the modeled EPCs.

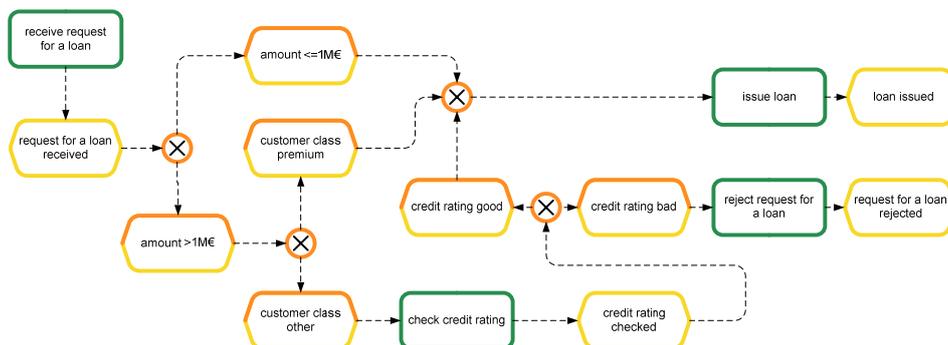


Figure 1: N-EPC describing a simplified process for loan application processing

Enabling multiple events between functions allows a detailed modeling of conditions between functions. N-EPCs are neither directly executable in BPEL compliant workflow systems nor do BPEL modeling tools support the import of N-EPCs and EPCs. On the other hand, a large number of workflow systems and modeling tools support the Business Process Execution Language for Web Services (BPEL, [An03]). Thus, we map N-EPCs to BPEL to enable both, their execution and import into BPEL tools.

In the following, we first describe the basics of N-EPCs, including their syntax and a discussion of their semantics. The main sections of this paper are devoted to the mapping to BPEL, covering in detail how events, connectors and functions get mapped.

## 2 Event-driven Process Chains in Nautilus

Nautilus Event-Driven Process Chains (N-EPCs) build the basis of the metamodel of Nautilus. An N-EPC consists of three different types of nodes: events, functions, and connectors. An event describes a state of the modeled process. A state is the result of a previous step, can trigger next steps or both. A function describes a step in the process. Connectors join or fork the control flow. A connector is either an AND, an OR, or an XOR connector. For the labeling of functions and events, the concept of objects, verbs, and states is introduced. A function is doing something with an object. After the processing, the object is in a new state. Therefore, functions are labeled with a verb and an object and events are labeled with a state and an object. Events and functions need not to be alternating. Transition from one function to another may be via multiple events. Figure 2 shows the elements of an N-EPC.

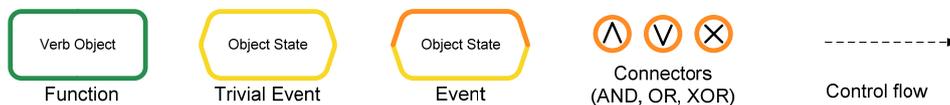


Figure 2: Elements of an N-EPC

An event directly after a function is called “trivial event”, because it always occurs after a function has been executed. The object of a trivial event is the same as in the preceding function. It is a modeling convention to use the participle of the verb as the new state of the object. For a detailed discussion of the convention see [Ge06b]. An event labeled with (o,s) occurs, if the state of the object o changes to the state s.

### 2.1 Syntax

There are several approaches for formalizing the syntax of EPCs. [NR02] and [Ki04] provide one of them. Since the metamodel of N-EPCs is different from the one introduced in [KNS92], we have to introduce a new formal definition. This formalism stays close to the one found in [Ki04].

**Notation 1 (Set of all sequences).** Let  $T$  be any set. By  $T^+$  we denote the set of all finite sequences over elements of the set  $T$ , e.g.  $T=\{0,1\}$ , then  $T^+=\{0,1,00,01,10,\dots\}$ .

**Definition 1 (N-EPC).** An N-EPC is a tuple  $M=(E,F,C,A,l,T,V,O,S)$  satisfying:

- $E,F,C$  are disjoint sets
- $E$  is a set of events
- $F$  is a set of functions
- $C$  is a set of connectors
- $A$  is a set of arcs,  $A \subseteq (E \cup F \cup C) \times (E \cup F \cup C)$ .
- $T$  is a set of terms
- $V$  is a set of verbs
- $O$  is a set of objects
- $S$  is a set of states
- $l$  is the labeling function:  $l: (E \cup F \cup C \cup V \cup O \cup S) \rightarrow (V \cup O \cup S \cup T^+)$ ,

$$l(x)=y, y \in \begin{cases} O \times S & x \in E \\ V \times O & x \in F \\ \{\emptyset, \oplus, \otimes\} & x \in C \\ T^+ & x \in V \cup O \cup S \end{cases}$$

To ease reading, we introduce following notations:

**Notation 2 (Predecessors and successors).** Let  $N$  be a set of nodes and let  $A \subseteq N \times N$  be the set of arcs. For each node  $n \in N$ ,  $\text{adj}^+(n) = \{m | (n,m) \in A\}$  denotes the successors of the node and  $\text{adj}^-(n) = \{m | (m,n) \in A\}$  denotes the predecessors of the node. If there is only one successor,  $\text{adj}_1^+(n)$  returns that successor and  $\text{adj}_1^-(n)$  that predecessor:

$$\text{adj}_1^-(n) = \begin{cases} m, m \in \text{adj}^-(n) & |\text{adj}^-(n)| = 1 \\ \perp & \text{otherwise} \end{cases}, \quad \text{adj}_1^+(n) = \begin{cases} m, m \in \text{adj}^+(n) & |\text{adj}^+(n)| = 1 \\ \perp & \text{otherwise} \end{cases}$$

“ $\perp$ ” indicates “undefined”. It is included in the definition to let  $\text{adj}_1^+(n)$  and  $\text{adj}_1^-(n)$  return a value for all  $n \in N$ .

**Notation 3 (Connected nodes).** Let  $N$  be a set of nodes and let  $A \subseteq N \times N$  be the set of arcs. For each node  $n \in N$ ,  $\text{adj}^*(n)$  denotes the transitive closure, i.e. the set of all directly or indirectly connected nodes including the node  $n$  itself.

M is a valid N-EPC if it satisfies the following conditions:

1. Each function has exactly one successor. That successor is an event<sup>1</sup>:  
 $\forall f \in F: |\text{adj}^+(f)|=1$  and  $\text{adj}_1^+(f) \in E$
2. Each function has at most one predecessor:  
 $\forall f \in F: |\text{adj}^-(f)| \leq 1$
3. Each event has at most one predecessor and at most one successor:  
 $\forall e \in E: |\text{adj}^-(e)| \leq 1$  and  $|\text{adj}^+(e)| \leq 1$
4. Events are connected by connectors, are triggered by functions or trigger functions:  
 $\forall e \in E: (|\text{adj}^-(e)|=1 \Rightarrow \text{adj}_1^-(e) \in F \cup C)$  and  $(|\text{adj}^+(e)|=1 \Rightarrow \text{adj}_1^+(e) \in F \cup C)$
5. Events and connectors are always directly or indirectly connected to a function:  
 $\forall n \in E \cup C: \exists f \in \text{adj}^*(n): f \in F.$
6. There are no self loops<sup>2</sup>:  
 $\forall c \in C: (c,c) \notin A$
7. There has to be at least one event between an OR connector with more than one successor and a function:  
 $\forall w \in \{((n_0, n_1), (n_1, n_2), \dots, (n_{j-1}, n_j)) \mid (n_i, n_{i+1}) \in A, 0 \leq i < j, n_0 \in C, l(n_0) = \bigvee, |\text{adj}^+(n_0)| > 1, n_j \in F, n_k \in E \cup C, 0 < k < j\}: \exists n_k: n_k \in E$
8. There has to be at least one event between an XOR connector with more than one successor and a function:  
 $\forall w \in \{((n_0, n_1), (n_1, n_2), \dots, (n_{j-1}, n_j)) \mid (n_i, n_{i+1}) \in A, 0 \leq i < j, n_0 \in C, l(n_0) = \bigotimes, |\text{adj}^+(n_0)| > 1, n_j \in F, n_k \in E \cup C, 0 < k < j\}: \exists n_k: n_k \in E$

We call a connector with more than one successor a “fork” and a connector with more than one predecessor a “join”. A connector can be a fork and a join if it has more than one successor and more than one predecessor. In addition, we call a connector with  $\bigwedge$  as label “AND connector”, with  $\bigvee$  as label “OR connector”, and with  $\bigotimes$  as label “XOR connector”.

## 2.2 Semantics

[Ge06b] does not contain a formal semantics of N-EPCs, leaving room for multiple interpretations. For explaining the intended semantics, we use the process folders from [vdADK02]. Process folders are comparable to tokens in a Petri net. Process folders mark the current active functions, events, connectors, and arcs in an N-EPC. An N-EPC

---

<sup>1</sup> The metamodel does not distinguish between trivial events and other events, since a trivial event is defined as the event directly following a function.

<sup>2</sup> It is sufficient to add a constraint for  $c \in C$ , because rules 1 to 4 ensure that events and functions cannot be connected to themselves.

does not contain explicit start- or end-nodes. The initial state of an N-EPC is formed by a non-empty sub-set of nodes without incoming arcs that are marked with a process folder. The end of the processing of an N-EPC is formed by process folders that are only at nodes without outgoing arcs. During execution, the process folder is passed by the nodes of the N-EPC. A function takes the process folder on its incoming arc, executes and puts the process folder on its outgoing arc. An event takes the process folder on its incoming arc and owns the process folder. As soon as it occurs, it puts the process folder on its outgoing arc. The arcs themselves do nothing with the process folder. Connectors fork and join process folders. We will first explain the semantics of forks and afterwards the semantics of joins.

Generally speaking, a fork connector ensures that the incoming process folder gets propagated to the next join or function. If a fork connector has multiple incoming edges, it is first treated as a join connector, which is explained below. If a fork connector has one incoming arc, the process folder on the incoming arc is taken and put on its outgoing arcs according to the label of the fork. If the connector is an AND fork, the folder is put on all of its outgoing arcs. If the connector is an OR fork, the folder is put on at least one of its outgoing arcs. If the connector is a XOR fork, the folder is put on one of its outgoing arcs. For XOR and OR forks, the decision, which arc is taken depends on the succeeding events. Therefore the semantics of XOR and OR forks is non-local. See Figure 3 as example: Arc 1 is active. Since the XOR connector has only one predecessor, it takes the process. If event e1 occurs, the XOR connector puts its process folder onto arc 2. If event e2, event e3 or both events occur, the process folder is via arc 3. If e1 and e2 occur, the behavior is non-deterministic: The following things can happen: An error is raised to the outside<sup>3</sup>, the XOR connector doesn't pass the process folder at all, the process folder is passed at random to arc 2 or to arc 3 or even to both arcs.

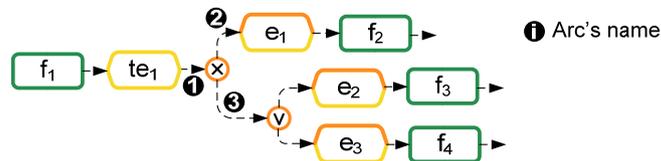


Figure 3: N-EPC<sup>4</sup> illustrating multiple events between functions

If the events are chained, the evaluation gets more complicated. Figure 4 contains an example. There, e1 and e2 can both occur. If e3 and e4 occur, the process folder gets passed to arc 2, even if both event e1 and e2 occurred. If e3 and e4 do not occur, e5 or e6 will occur so that the process folder is passed to arc 3. Therefore, the decision cannot be taken by solely looking at the first reachable events. All events on the path from the XOR connector to each function have to be regarded. This is specific to N-EPCs, since there possibly is more than one event between an XOR connector and a function.

<sup>3</sup> That means, the human reader (and executor) of the N-EPC is stopping executing the process and should talk to his manager.

<sup>4</sup> If the labeling with tuples is not important for the description, we omit that labeling and just use unique labels

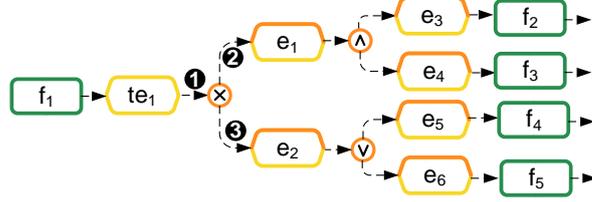


Figure 4: N-EPC illustrating multiple events between functions

We first describe the semantics informally, and then formalize it: An active XOR connector<sup>5</sup> triggers if exactly one of its outgoing arcs can trigger. An activated OR connector triggers if at least one of its outgoing arcs can trigger. An activated AND connector triggers if all of its outgoing arcs can trigger. An arc triggers if its target element can trigger. A function can always trigger. Finally, an event can trigger if it has occurred and its successor can trigger. It is important that besides the occurrence of the event its successor can trigger, since all the events on the way to the next function have to be regarded. It is not always known in advance when a certain event occurs. A state change of an object can happen because the object was modified by a function or because the modification of the object was not modeled by the N-EPC. In the latter case the event is called “external event”. An external event can be “temperature below 25°C”. The change of the object “temperature” can be modeled by a function “get temperature”, but the definition of N-EPC does not force that an object’s state can only be modified by functions. This fact makes the evaluation of a function “canTrigger” time-dependent. The formalization of this time-dependency is the subject of our current research. The formal definition of canTrigger without considering the time-dependency is as follows:

**Definition 2 (Function canTrigger):** The function canTrigger(x) returns true if the element x can trigger, false otherwise. The function eo returns true, if exactly one of its parameters is true. An event e is true if it has occurred.

$$\text{canTrigger}(x) = \begin{cases} \text{eo}(\text{canTrigger}(n_1), \dots, \text{canTrigger}(n_j)) & x \in C \text{ and } l(x) = \otimes, \text{adj}^+(x) = \{n_1, \dots, n_j\} \\ \bigvee_i (\text{canTrigger}(n_i)) & x \in C \text{ and } l(x) = \odot, \text{adj}^+(x) = \{n_i\} \\ \bigwedge_i (\text{canTrigger}(n_i)) & x \in C \text{ and } l(x) = \ominus, \text{adj}^+(x) = \{n_i\} \\ x \wedge (\text{canTrigger}(\text{adj}_1^+(x))) & x \in E \\ \text{true} & x \in F \end{cases}$$

The time-dependency has no influence on the verification of the N-EPCs, since every possible execution path in an EPC is considered for the verification (see [Me06] for an example of verification of EPCs). In the verification, a fork connector can always trigger. Therefore, the connectors pass the process folders randomly to their successors, according to the connector’s semantics. I.e. the XOR connector passes the process folder to one of its successors, the OR connector passes the process folder to either one, two,

<sup>5</sup> We use the term “trigger” as a description for the situation that the element of the EPC is active and passes the process folder to one or more outgoing arcs. An arc “triggers” if it is active and it passes the process folder to its target element.

..., or all of its successors and the AND connector passes to process folder to all of its successors without evaluating canTrigger().

It is important to note, that there is a modeling convention that the first fork following a function can always trigger. The semantics shown in [NR02] and [Ki04] share this convention since these semantics assume that a fork will always trigger.

In EPCs, the semantics of XOR and OR joins is non-local [NR02], which also applies for N-EPCs. Non-locality means that means that the XOR connector triggers if one incoming arc is active and all other incoming arcs: (a) not active, and (b) cannot become active if other elements of the EPC are triggering. The OR connector triggers if at least one incoming arc is active and all other arcs cannot become active. This informal description has been formalized in [NR02] using a transition relation. The given transition relation leads to problems in certain EPCs, first discussed in [vdADK02] and solved in [Ki04]. Figure 5 shows the sample from [vdADK02] in N-EPC notation.

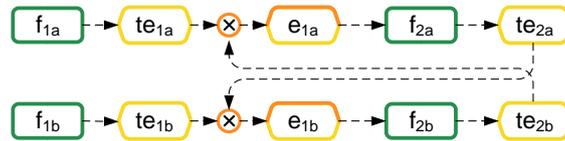


Figure 5: N-EPC showing a vicious circle

According to the semantics in [NR02], if the two XOR connectors are active, they will never propagate the process folder to their successors [vdADK02]. The mapping to BPEL will restrict the N-EPCs, preventing the occurring of any vicious circles.

Besides vicious circles, joins can be interpreted in two ways. We take the AND join as an example to illustrate them. One interpretation can be that as soon as one incoming arc becomes active, it is sure that all other incoming arcs will become active. If not, it is a modeling error. The semantics introduced in [LSW97] uses this understanding. The second interpretation of an AND join is, that it propagates the process folders if all incoming arcs are active. In all other cases it does not propagate the process folder. In particular, this case is not a modeling error. If a connector does not propagate a process folder, all of its subsequent activities that are not reachable in other ways will not be executed, too. This interpretation is shared by the semantics presented in [NR02] and [Ki04], and the informal semantics of N-EPCs. Note that the interpretation in [Ge06b] is equivalent to the semantics of the dead path elimination (DPE) if the N-EPC is acyclic. DPE forces the process graph to be acyclic and eliminates activities that cannot be reached during the execution of a BPEL process. The semantics of the DPE includes that a join synchronizes on the incoming arcs. Therefore, an XOR join directly following an AND fork will never propagate the process folders on its incoming arcs. We will give a more detailed explanation of DPE in section 3.

### 2.3 Extended Nautilus Event-driven Process Chains (N-eEPCs)

We have so far introduced the Nautilus Event-driven Process Chain, which contains events, functions, connectors, the arcs connecting them, and the labels on them. Nautilus also provides the possibility to model what information is sent and received by a function and which tool performs a function. The general concept is based on the Entity-Relationship Model [Ch76]. An entity can be a function, an information item, and a tool<sup>6</sup>. “Relations” define the relationships between entities. Nautilus allows the modeler to choose from a provided set of relation types. Table 1 lists the relation types important for the mapping to BPEL.

Entity	Relation	Entity
function	sends	information item
function	receives	information item
function	is executed by	tool

Table 1: Relation types important for the mapping to BPEL

Relating a function to another entity is optional. Thus, a function may receive something but not executed by a tool. The existence of the relation “is executed by” implies that the function is not executed by the process, but by an executer. Figuratively speaking, the function is not part of the process, but is instead called by the process. Therefore, “f sends i” means that the function f sends the information i to the process. “f receives i” means, that the process sends the information item i to the function f. If a function receives and sends information items, the order should be interpreted as “first receive, then send” [Ge06b]. It is important to note that there is no possibility for events to send or receive data.

**Definition 3 (N-eEPCs).** An N-eEPC is a tuple  $Mx=(E,F,C,A,l,T,V,O,S,I,P,s,r,x)$  and has to satisfy following properties:

- $M=(E,F,C,A,l',T,V,O,S)$  forms a valid N-EPC.  $l'$  is derived from  $l$  by restricting the preimage of  $l$  to  $(E \cup F \cup C \cup V \cup O \cup S)$ . The image remains unchanged.
- $l$  is the labeling function:  
 $l: (E \cup F \cup C \cup V \cup O \cup S \cup I \cup P) \rightarrow (V \cup O \cup S \cup T^+)$ ,  
 $l(x)=y, y \in \begin{cases} \text{as defined in definition 1} & x \in E \cup F \cup C \cup V \cup O \cup S \\ T^+ & x \in I \cup P \end{cases}$
- $I$  is the list of information items
- $P$  is the list of tools

---

<sup>6</sup> Nautilus contains 25 more elements in addition to these. We concentrate on the ones that are important for the mapping to BPEL. See [Ge06b] for a complete list of entities.

- $s$  is the function assigning the sent information items to functions<sup>7</sup>:  
 $s: F \rightarrow 2^I$
- $r$  is the function assigning the received information items to functions:  
 $r: F \rightarrow 2^I$
- $x$  is the function assigning the executing tools to functions:  
 $x: F \rightarrow 2^P$

### 3 The Business Process Execution Language for Web Services

BPEL is currently the widest-spread language to support orchestration of Web Services. It is implemented by several companies, including IBM, Microsoft, and Oracle. The current version is 1.1. In this section, a brief overview will be presented of the subset of BPEL 1.1 used by the mapping. A complete explanation of the language and its elements can be found in [An03]. Additionally, [LR05] illustrates the main concepts, and [Ou05] and [HSS05] provide a formal description.

A BPEL process make use of Web Service(s) offered by partners and is itself offered as one or more Web Service(s). These Web services are specified as port types<sup>8</sup>. The connection with a partner is modeled using a partner link, typed by a partner link type having one or two roles. Each role refers to a port type. If a BPEL process uses the partner's port type and does not offer a port type to the partner, the partner link type contains a single role element and vice versa.

A BPEL process model may use different kinds of activities. An `invoke` activity is used to invoke a partner's Web Service operation, and consists of input/output variables, a partner link and the name of the operation. If the operation is one-way, then the output variable is not specified. A `receive` activity is used to handle incoming calls to the process's Web Service operations, and consists of a variable, a partner link, and the name of the process's operation. Variables are typed, usually with WSDL message types [An03]. Such simple activities can be grouped into complex activities like `flow`.

Activities inside a flow can be connected by links that have transition conditions. Every activity with incoming links has a join condition, which as a logical operation on the status of those links. When a flow starts, it enables all immediately enclosed activities with no incoming links. Once an activity completes, the transition conditions on the outgoing links get evaluated and the status of each link is set to the result of the evaluation. The join condition of an activity is evaluated once all the incoming links have fired. If it is true, then the activity executes. If it is false, the activity is skipped and the status of its entire outgoing links is set to "false". This procedure, formally defined in [LR00], is called "dead-path-elimination" (DPE).

---

<sup>7</sup>  $2^S$  is used to refer to the power set of a set  $S$

<sup>8</sup> A port type is an interface defined with a WSDL file. For a complete description of WSDL see [Ch01].

Besides modeling control flow using flat-graph process definition approach, control flow can be modeled with a structural constructs approach or a combination of the two approaches. Structural constructs are `switch` and `sequence`. A `sequence` executes the containing activities sequentially in lexical order. A `switch` contains one or more branches with conditions assigned, and optionally a default branch. The conditions are evaluated in lexical order. As soon as a condition evaluates to true, the respective branch is taken. If no condition evaluates to true, the default branch, if existing, is executed.

BPEL Version 2.0 has just reached the public review stage of standardization [AI06]. There are no major conceptual changes regarding the navigation from 1.1 to 2.0. Thus this explanation and the presented mapping will remain valid with regards to BPEL 2.0.

## 4 Mapping N-eEPCs to BPEL

We will use the extended Nautilus Event-driven Process Chains for mapping to BPEL, since they contain annotations showing which tool executes a function and what data is needed and returned by each function. To get an idea of the mapping of N-eEPCs to BPEL we give a rough overview at first. In the subsequent paragraphs, we will give a detailed explanation of each step and a justification why the transformation was chosen as such.

The elements of the N-eEPC are used as follows: The information about the executing tools and sent and received information items is used to generate the partner's WSDL files<sup>9</sup> and the corresponding partner links. Each function becomes an operation in a port type. In the generated BPEL process the generated activities are nested in a `flow` activity to naturally reflect the structure of the structure of the EPC. Each function is transformed into a `receive`, `invoke` or `empty` activity, depending on the sent and received information and whether an executing tool is assigned. Connectors get `empty` activities having their label transformed into a join condition. The incoming and outgoing arc of an event is mapped to a link connecting the mapped predecessor and mapped successor using the event as transition condition. Events with no successor get mapped to an `empty` activity.

Having provided the high level description of the mapping, we now introduce the restrictions on the N-eEPCs which can be mapped to BPEL.

BPEL supports natural loops<sup>10</sup> having the entry node as the only exit node and BPEL does not support arbitrary cycles. On the other hand, N-eEPCs model loops implicitly. Mendling et al. [MLZ06] showed what kind of loops can be made explicit and be transformed to BPEL. Since loop detection is out of the focus of this paper, we require that the N-eEPC used in the mapping be acyclic.

---

<sup>9</sup> Every tool becomes a partner in the BPEL process. For a complete description of WSDL and it's relation to BPEL see [Ch01] and [An03]

<sup>10</sup> Natural loops are loops with a single entry node [Mu97].

**Definition 4 (Acyclicity of an N-eEPC).** A N-eEPC  $M_e=(E,F,C,A,I,T,V,O,S,I,P,s,r,x)$  is acyclic, if the N-EPC  $M=(E,F,C,A,I,T,V,O,S)$  is acyclic.

**Definition 5 (Acyclicity of an N-EPC).** A N-EPC  $M=(E,F,C,A,I,T,V,O,S)$  is acyclic, if there is no non-empty path from a connector  $c$  to the same connector:

$\nexists w: w=((n_0,n_1),(n_1,n_2),\dots,(n_{j-1},n_j)), n_0,n_j \in C, n_j=n_0, (n_i,n_{i+1}) \in A, 0 \leq i < j.$

In addition to the acyclicity we demand that the N-eEPC contains a single root and that this root is a function. Handling multiple roots introduces special handling – e.g. the detection of pick<sup>11</sup> – and other syntactical restrictions, which are out of scope of this paper.

**Definition 6 (Roots of an N-eEPC).** A N-eEPC  $M_e=(E,F,C,A,I,T,V,O,S,I,P,s,r,x)$  has the same roots as the N-EPC  $M=(E,F,C,A,I,T,V,O,S)$ :  $\text{roots}(M_e)=\text{roots}(M)$ .

**Definition 7 (Roots of an N-EPC).** Roots of a N-EPC  $M=(E,F,C,A,I,T,V,O,S)$  are the functions, events and connectors having no incoming arc:  $\text{roots}(M)=\{n \mid n \in E \cup F \cup C, |\text{adj}^-(n)|=0\}$

Information items are not annotated with a type. Therefore a fixed type has to be taken. We chose `string` as type, because most types can be serialized to a string. This leads to incompatibility if existing Web Services should be wired to the exported process. In that case, we suggest using mediation. Interface maps and data maps are one way to do mediation [IBM06]. Interface maps can be used if the name and the signature of an operation do not match. With an interface map, two different operations can be mapped to each other. For mapping data types, the data mapping can be used. A data map maps two data types to each other.

N-eEPCs allow multiple executers be related to a function. To get a clear mapping, we demand that at most one executing tool is assigned for each function.

To formally explain the mapping, we need the definition of an intermediary N-eEPC (IN-eEPC) that stores the current state of the transformation:

**Definition 8 (Intermediary N-eEPC, IN-eEPC).** An IN-eEPC  $I_M$  is a tuple  $I_M = (E,F,C,A,I,T,V,O,S,I,P,s,r,x,jc,tc)$ , where  $M_e=(E,F,C,A,I,T,V,O,S,I,P,s,r,x)$  is a N-eEPC, where no syntactical restrictions apply. E.g. a function may have several successors. In addition  $jc$  and  $tc$  are defined as follows:

- $jc$  is a function assigning the join condition to a function and a connector:  
 $jc: FUC \rightarrow \{\emptyset, \oplus, \otimes, \perp\}$
- $tc$  is a function assigning an event (interpreted as transition condition) to an arc:  
 $tc: A \rightarrow EU\{\perp\}$

---

<sup>11</sup> See [An03] for an explanation

We will first present the elementary steps of the mapping and combine them in a complete algorithm at the end. First, we will explain the mapping of events, then the mapping of connectors, and finally the mapping of functions from an N-eEPC to an IN-eEPC. Then, we map the IN-eEPC to BPEL. We will take an arbitrary N-eEPC  $M_e = (E_e, F_e, C_e, A_e, I_e, T_e, V_e, O_e, S_e, I_e, P_e, s_e, r_e, x_e)$  satisfying the conditions above as the starting point and map it to an IN-eEPC  $I_M = (E, F, C, A, I, T, V, O, S, I, P, s, r, x, jc, tc)$ .  $I_M$  is initialized as follows:

$$E := E_e, F := F_e, C := C_e, A := A_e, I := I_e, T := T_e, V := V_e, O := O_e, S := S_e, I := I_e, P := P_e, s := s_e, r := r_e, \\ x := x_e, tc(x) := \perp \quad \forall x \in A, jc(x) = \begin{cases} I(x) & x \in C \\ \perp & \text{otherwise} \end{cases}$$

#### 4.1 Mapping of events

Having an initial  $I_M$ , we can start with the transformation of events. Generally, events are mapped to transition conditions. There are two exceptions: Trivial events and events having no outgoing arcs<sup>12</sup>. Trivial events always occur after the preceding function has finished. Therefore, they do not bring additional semantics and can be removed from the N-eEPC as shown in Algorithm 1.

---

**Algorithm 1** Removal of trivial events

---

```

procedure REMOVE TRIVIALEVENTS( $I_M$ )
  for all  $f \in F$  do
     $e \leftarrow \text{adj}_1^+(f)$ 
    if  $|\text{adj}^+(e)| > 0$  then
       $A \leftarrow A \setminus \{(f,e), (e, \text{adj}_1^+(e))\} \cup \{(f, \text{adj}_1^+(e))\}$ 
    else
       $A \leftarrow A \setminus \{(f,e)\}$ 
    end if
     $E \leftarrow E \setminus \{e\}$ 
  end for
end procedure

```

---

Events having no outgoing arcs may be handled in two ways: Either remove them from the graph or transform them to `empty` activities. We decided to keep them in the resulting BPEL process instead of removing them. `empty` activities do not lead to new partner interaction, but take execution time. On the other hand, a form of documentation is lost if they are removed. Thus it is an open discussion if removal is preferable to keep them. During the modification of  $I_M$  events having no outgoing arcs are transformed to functions with no executer and no information items assigned. They will get mapped to `empty` activities during the transformation of functions.

---

<sup>12</sup> Events with no incoming arcs do not occur, since we demanded that  $M_e$  has a single root which is a function.

---

**Algorithm 2** Transformation of events without outgoing arcs

---

```
procedure TRANSFORMLEAFEVENTS( $I_M$ )  
   $E' \leftarrow E$   
  for all  $e \in E'$ ,  $|\text{adj}^+(e)|=0$  do  
     $E \leftarrow E \setminus \{e\}$   
     $F \leftarrow F \cup \{e\}$   
  end for  
end procedure
```

---

The remaining events have one incoming and one outgoing arc. They get transformed into transition conditions. With this interpretation of events, only events occurring as a result of a function can be mapped. We chose this mapping, because the metamodel of N-eEPCs does not allow annotating events with “sends”, “receives”, or “executed by” relations. The information items in N-eEPCs are not related to each other, too. E.g. “address consists of street, postal code and city”<sup>13</sup> cannot be modeled. Assume a function sending “address” and a succeeding event “(London, city)”. Without additional information, an algorithm cannot decide whether the event belongs to the function. Furthermore, we did not want to extend the metamodel, because events that receive information are a completely new concept to business analysts that know the EPC metamodel.

---

**Algorithm 3** Transformation of events to transition conditions

---

```
procedure TRANSFORMEVENTSTOTRANSITIONCONDITIONS( $I_M$ )  
   $E' \leftarrow E$   
  for all  $e \in E'$ ,  $|\text{adj}^-(e)|=1$ ,  $|\text{adj}^+(e)|=1$  do  
     $a \leftarrow \{(\text{adj}_1^-(e), \text{adj}_1^+(e))\}$   
     $A \leftarrow A \setminus \{(\text{adj}_1^-(e), e), (e, \text{adj}_1^+(e))\} \cup \{a\}$   
     $E \leftarrow E \setminus \{e\}$   
     $tc' \leftarrow tc$   
     $tc(x) \leftarrow \begin{cases} tc'(x) & x \neq a \\ e & x = a \end{cases}$   
  end for  
end procedure
```

---

The transition condition will be `object = 'state'` in the resulting BPEL file. This is sufficient for cases where the object and the information item can be identified and the effective data type of the information item is `string`. One modeling convention used is that an object and an information item describe the same entity if the label of the object equals the label of the information item. If an information item cannot be identified with an object, the transition conditions have to be edited manually after the transformation to get a valid executable BPEL process.

---

<sup>13</sup> We are aware of the fact that there are numerous variations of an address. E.g. an address can alternatively contain the state, consist of a post-office box, etc.

## 4.2 Mapping of connectors

Now all cases of the events are handled so we continue stating the handling of the connectors. As explained in the beginning of this section, each connector gets mapped to an `empty` activity. We will show that this is a straight forward mapping that can be improved to reduce the amount of activities in the generated BPEL process.

First of all, connectors having a single incoming edge and a single outgoing edge can safely be removed from  $I_M$ . As soon as their incoming arc is active, they can pass the process folders to the outgoing arc. They do not have to wait for other incoming arcs or to evaluate `canTrigger()` on the outgoing arc. If they are asked for `canTrigger()`, they can just return the value of their outgoing arc. Therefore, we remove them<sup>14</sup>.

---

**Algorithm 4** Removal of connectors not joining and not forking

---

```

procedure REMOVECONNECTORSNOTJOININGANDNOTFORKING( $I_M$ )
  while  $\exists c \in C: |adj^-(c)|=1 \wedge |adj^+(c)|=1$  do
     $A \leftarrow A \setminus \{(adj_1^-(c), c), (c, adj_1^+(c))\} \cup \{(adj_1^-(c), adj_1^+(c))\}$ 
     $C \leftarrow C \setminus \{c\}$ 
  end while
end procedure

```

---

BPEL does not provide any construct for “fork conditions”. A connector in EPCs has such an implicit fork condition. E.g. an XOR fork states that only one outgoing arc can be active. The only construct similar to a fork condition is the `switch` activity which could be represented in N-EPCs using an XOR block<sup>15</sup>. See Figure 6 for an illustration.

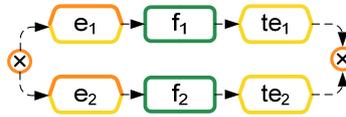


Figure 6: N-EPC modeling a switch

Instead of  $f_1$  and  $f_2$  there can be other EPC constructs allowing switches to be nested. This is the only case where BPEL supports fork conditions. There is no similar BPEL construct for AND or OR blocks. Furthermore, regarding arbitrary EPCs<sup>16</sup> the XOR block is just one of many cases. To avoid dealing with special cases, we do not use switch. As a result, the semantics of fork connectors cannot be directly mapped to BPEL. The semantics of a fork includes the assurance that an active fork will trigger at some time. This implies that the events succeeding the fork will occur in a combination allowing the fork to trigger. As a result, no fork condition is needed. Thus forks mapped to `empty` activities do not bring additional semantics to the BPEL process. They can be removed instead if following conditions apply:

<sup>14</sup> The removal is done before any events get mapped. Otherwise, transition conditions get lost.

<sup>15</sup> XOR blocks are formally defined in [Ru99]

<sup>16</sup> Called “unstructured process graph” in [MLZ06]

1. The fork  $c$  has a single incoming arc
2. Either
  - a. The incoming arc does not contain a transition condition and all outgoing arcs do not contain a transition condition
  - b. The incoming arc contains a transition condition and all outgoing arcs do not contain a transition condition
  - c. The incoming arc does not contain a transition condition and some of the outgoing arcs contain a transition condition
3. If 2a does not apply, there is no arc connecting the predecessor to one of the successors of  $c$  having a transition condition.

The predecessor of the fork will be connected with all successors of the fork. Assume  $\{g_i\}$  is the set of generated arcs. If condition 2b is fulfilled, the transition condition of the incoming arc is copied to each  $g_i$ . If condition 2c is fulfilled, the transition condition of each outgoing arc is copied to the corresponding arc  $g_i$ . Because of these copying rules the condition 3 is needed. BPEL does not allow multiple links between two activities. This is assured in  $I_M$  by the definition of  $A \subset (E \cup F \cup C) \times (E \cup F \cup C)$  (cp. definitions 1,3,8). There is the possibility to merge the transition condition of two arcs but we think the rule “at most one event forms a transition condition” eases the understanding of the generated BPEL process. The procedure is shown in algorithm 5.

BPEL supports join conditions on any activities with incoming links. Therefore, the label of the connector is used as join condition, which is transformed to a XPATH statement representing the join condition in BPEL. Using BPEL’s dead path elimination, activities not reachable will not be executed during the execution of the BPEL process.

Similar to the handling of forks, joins with a single outgoing edge can be eliminated by connecting the predecessors to the successor. The difference to the handling of the forks is the join condition. The join condition of the join connector replaces the join condition of the successor. Since the algorithm can be derived from algorithm 5, we give an example of the transformation of joins in figure 7.



Figure 7: Illustration of the elimination of join connectors

---

**Algorithm 5** Transformation of forks

---

```
function FORKSATISFIESCONDITIONS(c)
  tci ← tc((adj1-(c),c)) ≠ ⊥           ▷ For completeness: tc(⊥, c) := ⊥
  tco ← ∃s ∈ adj1+(c): tc(c,s) ≠ ⊥
  arc ← ∃s ∈ adj1+(c): ∃a ∈ A, a = (adj1-(c), s): tc(a) ≠ ⊥
  c1 ← |adj1-(c)| = 1
  c2a ← (¬tci ∧ ¬tco)
  c2b ← (tci ∧ ¬tco)
  c2c ← (¬tci ∧ tco)
  c3 ← (c2a ∨ ¬arc)
  return c1 ∧ (c2a ∨ c2b ∨ c2c) ∧ c3
end function

procedure TRANSFORMFORKS(IM)
  while ∃c ∈ C: (|adj1+(c)| > 1) ∧ FORKSATISFIESCONDITIONS(c) do
    C ← C \ {c}
    p ← adj1-(c)
    succ ← adj1+(c)
    A ← A \ {(p,c)}
    for all s ∈ succ do
      a ← (p,s)
      A ← A \ {(c,s)} ∪ a
      if tci then
        tc' ← tc
        tc(x) ←  $\begin{cases} tc'(x) & x \neq a \\ tc'((p,c)) & x = a \end{cases}$ 
      else if tco then
        tc' ← tc
        tc(x) ←  $\begin{cases} tc'(x) & x \neq a \\ tc'((c,s)) & x = a \end{cases}$ 
      end if
    end for
  end while
end procedure
```

---

### 4.3 Mapping of functions

The mapping of the functions is the last step. We will also present the mapping of arcs, information items and executors since all of them are closely related to functions.

Functions describe what is done by whom and which information is sent and received by the executor. Therefore, functions sending or receiving something and being executed by someone or something are mapped to `receive` and `invoke` activities. In all other cases, functions are mapped to an `empty` activity. The incoming and outgoing arcs are mapped to incoming and outgoing links. The outgoing link is additionally assigned a transition condition if the belonging arc has a transition condition assigned. The join condition of the generated activity depends on the value of `jc`. Assume `f` is the current function to be

mapped. If  $jc(f)=\bigvee$  then no join condition is generated, since the OR join is the default join behavior in BPEL. If  $jc(f)=\bigwedge$  then a logical `and` over all incoming links is generated. If  $jc(f)=\bigotimes$ , “exactly one” over all incoming links is expressed by logical `and`, `or` and `not` connectors. Assume  $l_1$ ,  $l_2$  and  $l_3$  are the incoming links. Then the join condition expressed in XPATH representing XOR looks as follows:

```
(( $l1 ) and ( not $l2 ) and ( not $l3 )) or
(( not $l1 ) and ( $l2 ) and ( not $l3 )) or
(( not $l1 ) and ( not $l2 ) and ( $l3 ))
```

$\$l_i$  is gets expanded to `bpws:getLinkStatus('li')` as soon as it is written into the BPEL file.

Let  $f$  be a function having an executer and incoming and outgoing information assigned.  $f$  is mapped to a `receive` if sends something and does not receive anything. The reason is that the function is not executed by the process itself but the executer assigned to the function. If someone external sends something to the process but does not get something from the process, the process has just to receive it. In all other cases, an `invoke` is generated. Assume the function  $f$  sending information  $i_1$  and receiving information  $i_2$ . Receive and send should be interpreted as “first receive, then send”<sup>17</sup>. Thus mapping to single `invoke` activity sending  $i_1$  (using the input variable) and afterwards receiving  $i_2$  (using the output variable) is enough: The executer first receives  $i_1$  and then the BPEL process is ready to receive  $i_2$ . If the function receives  $i$  and does not send anything, it is mapped to an `invoke` activity sending  $i$ . The sent and received information is used to generate the variables and message types. Assume  $r(f)=\{i_1,i_2,i_3\}$ . With this information, a message type named `i1_i2_i3` and a variable named `i1_i2_i3` of the type `i1_i2_i3` is generated<sup>18</sup>. The message type contains three parts named `i1`, `i2`, and `i3` each of them having the type `string`. The variable `i1_i2_i3` is used as the input variable. The generation of variables for  $s(f)$  follows the these rules, too.

The label of the function is used as the name for the operation: The concatenated verb and object form the name of the operation. For the generation of the WSDL files, the partner link types, the partner links and the port types, the tool executing a function is used. Each generated port type is put in a separate WSDL file. If a function was mapped to a `receive` activity, the operation is put into a port type named after the executer with the suffix `input`. The partner link and the partner link type are named `<tool>-to-process`. The generated port type is assigned to the partner link type and the partner link as `myRole`. If a function was mapped to an `invoke`, the operation is put into a port type named after the executer and assigned to the partner link `<tool>-to-process` as `partnerRole`. The partner link and its partner link type are generated, if it they had not already been generated by the mapping to an `receive` activity.

---

<sup>17</sup> cp. section 2.3

<sup>18</sup> If the message type has already been generated in a previous step, the message type and variable are not generated again. For the ordering in the name alphabetical ordering is used.

#### 4.4 The complete algorithm

Using the previous descriptions, the complete algorithm is straight-forward. The algorithm first initializes  $I_M$  with  $M$ . Connectors not joining and not forking are removed afterwards. Then the events are transformed according to section 4.1 “Mapping of events”. After this transformation, the connectors are transformed as presented in section 4.2 “Mapping of connectors”. In the last step, a depth-first search (DFS) is started from the root of  $I_M$ . During the DFS functions and connectors can be found. Events are transformed to transition conditions or to functions during the step “Mapping of events”. A connector is mapped to an `empty` activity, where the links and the join conditions are generated as stated in “Mapping of functions”. Each function gets mapped to an `empty`, `receive`, or `invoke` activity as described in section 4.3 “Mapping of functions”.

The algorithm presented in this paper was implemented in the BPEL module in Nautilus [Ged06a] and is now commercially available. The BPEL file and WSDL files it generates can be imported into a BPEL development environment like the IBM WebSphere Integration Developer. Once in such an environment, the process can be tweaked (transition conditions modified, interface and data maps added) to enable its integration into an existing infrastructure.

### 5 Related work

Mendling et al. [MLZ06] categorized the approaches of mapping EPCs to BPEL. The categorization includes a high level representation of the mapping, leaving out the generation of the type of the basic activity and leaving out the generation of port types, partner links and WSDL files. Additionally, they do not address allowing multiple events between functions because the metamodel of EPCs does not allow that. Nevertheless, our mapping falls into their “Element-Minimization” category.

Ziemann and Mendling [ZM05] presented an approach using the labels of the functions to generate `receives` and `invokes`. Our mapping derives the type of the activity from the information sent and received.

### 6 Summary and outlook

We presented the syntax and semantics of Nautilus extended Event-driven Process Chains (N-eEPCs). N-eEPCs in contrast to EPCs introduced in [KNS92], support multiple events between functions. This allows a detailed modeling of conditions between functions. We showed how multiple events between functions can be used to generate transition conditions in BPEL. The class of external events was completely dropped from the mapping, because the metamodel of N-EPCs does not offer assigning sent and received information to an event. Special care was taken to eliminate connectors to get a readable BPEL file. Another contribution of this paper is the use of “send”, “receive” and “executed by” relations to generate partner links and port types. A

commercially available implementation has been created. It produces BPEL files that can be used in known BPEL tools and systems.

Future research directions include the investigating of possibilities to support external events by N-eEPCs and the usage of function repositories to ease integration in an existing infrastructure.

### Acknowledgement

This work was supported by the German Federal Ministry of Education and Research (project number 01ISE08B).

### References

All links were followed on 2006-09-22.

- [Al06] Alves, A. et al.: Web Services Business Process Execution Language Version 2.0. Public Review Draft, 23<sup>rd</sup> August, 2006. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf>
- [An03] Andrews, T. et al.: Business Process Execution Language for Web Services version 1.1, 2003. <http://www-128.ibm.com/developerworks/library/specification/wsbpel/>
- [Ch01] Christensen, E. et al.: Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001. <http://www.w3.org/TR/wsd1>
- [Ch76] Chen, Peter P.: The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems. Volume 1. Number 1. pp. 9-36. 1976.
- [Ge06a] Gedilan Consulting GmbH: Homepage. <http://www.gedilan.com>
- [Ge06b] Gedilan Technologies GmbH: Nautilus Benutzerhandbuch. Version 4.5. 2006.
- [HSS05] Hinz, S.; Schmidt, K.; Stahl, C.: Transforming BPEL to Petri Nets. In (W.M.P. van der Aalst, W.M.P., et al.): Proceedings of the 3<sup>rd</sup> International Conference on Business Process Management (BPM 2005), volume 3649 of Lecture Notes in Computer Science, Springer, 2005; pages 220-235.
- [IBM06] IBM Cooperation. IBM WebSphere Business Process Management Version 6.0 information center. <http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/>
- [Ki04] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In (Desel, J.; Pernici, B.; Weske, M., eds.): Business Process Management, 2<sup>nd</sup> International Conference, BPM 2004. Lecture Notes in Computer Science Volume 3080, Springer, 2004; pp. 82-97
- [KNS92] Keller, G.; Nüttgens, N.; Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). In: Technical Report, Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 89, Universität des Saarlandes, 1992.
- [LR00] Leymann, F.; Roller, D.: Production Workflow - Concepts and Techniques, PTR Prentice Hall, 2000.
- [LR05] Leymann, F.; Roller, D.: Modeling business processes with BPEL4WS. In: Information Systems and E-Business Management, Volume 4, Number 3. Springer, 2005; pp. 265-284.

- [LSW97] Langner, P.; Schneider, C.; Wehler, J.: Prozeßmodellierung mit ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. In: Wirtschaftsinformatik, Volume 5, Number 39, 1997; pp. 479-489.
- [Me06] Mendling, J.; et al.: Faulty EPCs in the SAP Reference Model. In (Dustdar, S.; et al., eds.): Proceedings of the 4<sup>th</sup> International Conference Business Process Management (BPM 2006). Lecture Notes in Computer Science Volume 4102, Springer, 2006.
- [MLZ06] Mendling, J.; Lassen, K. B.; Zdun, U.: Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages. In (Lehner, F.; Nösekabel, H.; Kleinschmidt, P., eds.): Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006), Band 2, XML4BPM Track. GITO-Verlag Berlin, 2006; pp. 297-312.
- [Mu97] Muchnick, S.: Advanced Compiler Design and Implementation. Morgan Kaufmann, 1997.
- [NR02] Nüttgens, M.; Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: PROMISE 2002, Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen, GI Lecture Notes in Informatics. P-21. Gesellschaft für Informatik, 2002; pp. 64–77.
- [Ou05] Ouyang, C., et al.: Formal Semantics and Analysis of Control Flow in WS-BPEL (Revised Version). BPM Center Report BPM-05-15, BPMcenter.org, 2005.
- [Ru99] Rump, F. J.: Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten : Formalisierung, Analyse und Ausführung von EPKs. Dissertation. Stuttgart; Leipzig. Teubner, 1999.
- [vdADK02] Van der Aalst, W.; Desel, J.; Kindler, E.: On the semantics of EPCs: A vicious circle. In (Nüttgens, M.; Rump, F.J., eds): EPK 2002, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, November 2002; pp. 71–79.
- [ZM05] Ziemann, J.; Mendling, J.: EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In: Proceedings of MITIP 2005, Italy, 2005.

# Integration of EPC-related Tools with ProM

Paul Barborka, Lukas Helm, Georg Köldorfer, Jan Mendling, Gustaf Neumann  
Vienna University of Economics and Business Administration  
Augasse 2-6, A-1090 Wien, Austria  
paul@barborka.com, lukas.helm@gmx.at  
georg.koeldorfer@benet.at, {jan.mendling|neumann}@wu-wien.ac.at

Boudewijn van Dongen, Eric Verbeek, Wil van der Aalst  
Eindhoven University of Technology  
PO Box 513, NL-5600 MB Eindhoven, The Netherlands  
b.f.v.dongen@tue.nl, h.m.w.verbeek@tue.nl, w.m.p.v.d.aalst@tue.nl

**Abstract:** The heterogeneity of different formats for EPCs is a major problem for model interchange between specialized tools in practice. In this paper, we compare three different formats for storing EPCs, in particular, the proprietary formats of Microsoft Visio and ARIS Toolset as well as the tool-independent EPML format. Furthermore, we introduce the ProM framework and show using the case of a sales process model expressed in terms of an EPC that ProM is able to serve as a mediator between these formats. Beyond that, we demonstrate that the ProM framework can be used for the analysis of EPCs and to translate EPCs into YAWL models for execution or for further analysis.

## 1 Introduction

Heterogeneity of formats, tools, and notations is a notorious problem of business process management. While the heterogeneity of notations is extensively discussed in literature (see e.g. [MNN05]), there is only little attention paid to heterogeneity issues related to a single process modeling language. In this context, the availability of XML-based interchange formats plays an important role to facilitate interchange and conversion. EPCs form an example of heterogeneity issues related to a single language: there are several tools for modeling, analyzing, transforming, and managing EPCs using different representations. In order to benefit from specialized functionality, there is a strong need to exchange models between these tools.

Several of these EPC-related business process modeling tools support XML as an open standard for interchange of structured information. Such tools include *ARIS Toolset* of IDS Scheer AG, *ADONIS* of BOC GmbH, *Visio* of Microsoft Corp., *Semtalk* of Semtation GmbH, or *Bonapart* by Pikos GmbH. The heterogeneity problem in this context arises due to the fact that only some of these tools support the tool-independent EPC Markup Language (EPML) interchange format [MN06]. Most of them including e.g. ARIS and

Visio use proprietary formats that stick close to the internal information model of the tool. Therefore, an open and generic transformation platform is needed in order to facilitate the interchange of EPC business process models between these tools.

Against this background, this paper aims to demonstrate how such a transformation based integration of specialized tools can be provided in practice. As an example, we present a case study that utilizes ProM for this purpose. Section 2 presents three frequently supported interchange formats for EPCs, i.e., VDX of Microsoft Visio, AML of ARIS Toolset, and EPML as a tool-independent format. In Section 3 we give an overview of ProM. In particular, we introduce the plug-in architecture of ProM and highlight which EPC-related plug-ins are already available. Section 4 showcases a scenario involving multiple specialized tools. In this scenario, ProM will be used as an integration platform. Section 5 summarizes the paper and gives an outlook on future research.

## 2 EPC Interchange Formats

In this section, we first introduce the interchange formats of Visio (Section 2.1) and ARIS (Section 2.2). We continue with presenting the tool-independent EPML format in Section 2.3. After that we compare the three formats in Section 2.4.

### 2.1 Visio and VDX

Microsoft Visio is a general drawing and modeling tool (see e.g. [WE03]). It can be customized to support any modeling language by defining a language specific stencil. EPCs are supported by a dedicated stencil that is included in the standard distribution. Visio uses the proprietary VDX XML format to store and read models.

Figure 1 gives an overview of the Visio metamodel that is the basis for the VDX format. `VisioDocument` is the root element of a VDX file. It can contain multiple `Pages` and `Masters`. A `Page` basically represents a Visio model. It is identified by an `ID` and a `NameU` attribute. A page can contain multiple `Shapes` and `Connects`. A `Shape` describes a visual object on a Visio page. It can be a simple object such as a rectangle or a composite object such as a grouping of other shapes. A shape is identified by an `ID` and a `NameU`. Furthermore, it can carry a reference to a `Master`. In Visio a `Master` provides the link between a stencil and a page. The EPC stencil defines a master for each of the EPC element types. Via the master attribute, a shape can be related to a logical object such as e.g. an EPC event or an XOR-connector. `Connect` elements offer a mechanism to logically link two shapes by referencing them in `FromSheet` and `ToSheet` attributes. Since arcs are also shapes in Visio, a control flow sequence between two EPC elements maps to two connects: one from the first object to the arc shape and another from the arc shape to the second object.

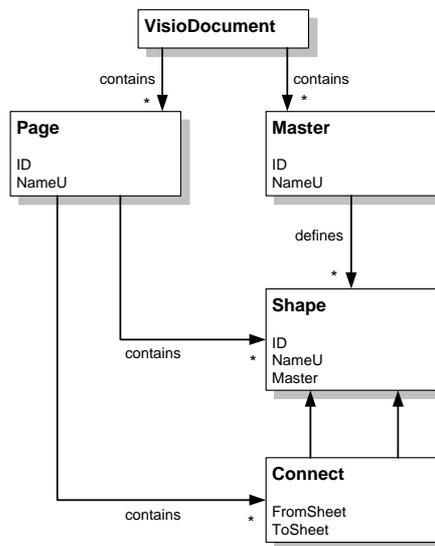


Figure 1: Visio metamodel.

## 2.2 ARIS and AML

ARIS is a specialized business process management tool that not only supports modeling, but also offers support for simulation and other types of analysis. EPCs are one of the modeling languages offered by ARIS. Individual models and a whole database of models can be written to a file in the proprietary ARIS Markup Language XML format. A complete overview of AML is given in [IDS03].

An AML file starts with an `AML` element as the root element. General information like time of creation, name of the ARIS database, language, and font style is stored in subelements of `AML`. The `Group` element, also a subelement of `AML`, is a container for all model-related information. In ARIS Toolset each `Group` element refers to a directory folder of the ARIS Explorer. A `Group` must have a unique `Group.ID` attribute and it may have multiple `AttrDef`, `ObjDef`, `Model` or further `Group` subelements as children. When the `Group` and its related directory have a name, ARIS Toolset stores it in an `AttrDef` (attribute definition) subelement whose `AttrDef.Type` attribute is set to `AT.NAME`. This is the typical mechanism used by AML to store model information. Every specific information of objects is stored in `AttrDef` or `AttrOcc` subelements of these objects (see Figure 2).

Another principle idea of ARIS Toolset, which is reflected in AML, is the separation between definition and occurrence: each model element is first defined in an abstract way and later referenced as an occurrence in a model. This allows one logical object to be included as multiple occurrences in models. Accordingly, the `Model` element contains `ObjOcc` (object occurrence) elements that refer to `ObjDef` (object definition) elements. The `ObjDef` element provides an abstract definition of an object. It has a unique `ObjDef.ID` attribute

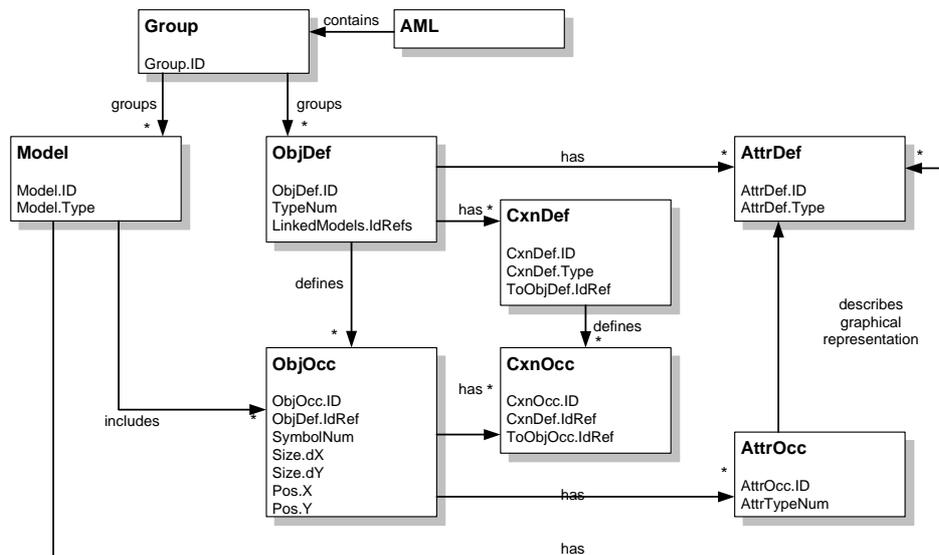


Figure 2: AML metamodel.

and a `TypeNum` attribute that refers to an object type, like e.g. EPC function or EPC event. Its `LinkedModels.IdRefs` attribute provides a list of ID-references to linked models. These can be used e.g. for hierarchical refinement of functions. `ObjDef` elements may have multiple `AttrDef` and multiple `CxnDef` subelements. `CxnDef` elements represent arcs between objects. Each `CxnDef` has a unique `CxnDef.ID` attribute, a `CxnDef.Type` attribute, and a `ToObjDef.IdRef` attribute which represents the target of the arc. Depending on the `CxnDef.Type` attribute the arc may represent control flow, information flow, or different kinds of semantic association between the objects.

A `Model` has, among others, a unique `Model.ID` and a `Model.Type` attribute. The model type, like e.g. EPC, refers to the allowed set of objects. The `Model` element may contain `AttrDef` elements to store model specific information and `ObjOcc` elements to represent graphical elements in a visual model. An object occurrence has among others a unique `ObjOcc.ID` attribute and a reference to an object definition via the `ObjDef.IdRef` attribute. The `SymbolNum` attribute refers to a graphical icon that is used to represent the object in the visual model. An EPC function would be e.g. represented by a green rectangle with radiused edges. An `ObjOcc` element may have subelements that describe its size and its position in the visual model. Furthermore the `AttrOcc` element defines how information attached via an `AttrDef` is visually represented in a model. It has a unique `AttrOcc.ID` attribute and an `AttrTypeNum` attribute that refers to its type. This type provides a syntactical link between an `AttrOcc` and an `AttrDef` element of two associated `ObjOcc` and `ObjDef` elements. Similar to object definitions `ObjOcc` may also have multiple `CxnOcc` elements. Each of them has a unique `CxnOcc.ID` attribute and a `CxnDef.IdRef` reference to an arc definition and a reference to the target of the arc via an `ObjOcc.IdRef` attribute.

## 2.3 EPML

EPML is a tool-independent interchange format for EPCs [MN06]. Mainly academic tools use EPML, but there are also commercial tools like Semtalk that provide EPML interfaces. Figure 3 gives an overview of EPML and its syntax elements. In EPML a hierarchy of EPC processes can be organized in directories. A `directory` element has a `name` attribute and it can contain other directories and/or EPC models. Each `epc` element is identified by an `epcId` attribute and has a `name` attribute. The `epcId` can be referenced by hierarchy relations attached to functions or process interfaces. Since an EPC process element might be used in two or more EPC models, there are `definitions` to establish the link between elements that are the same from a logical point of view. The `attributeTypes` element provides a container for the definition of additional structured information. An `attribute` references the type of the attribute and stores its value. An `arc` references the type of the attribute and stores its value.

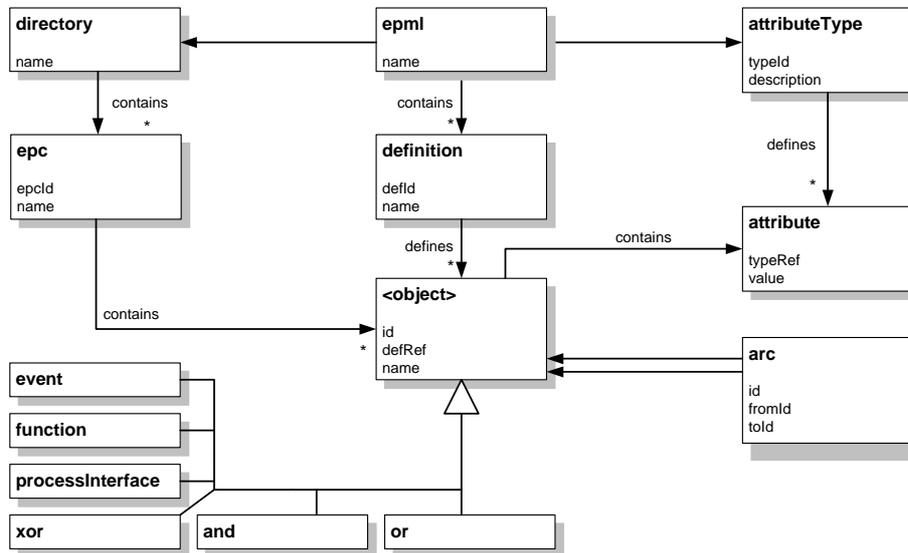


Figure 3: EPML metamodel.

In EPML each `epc` element serves as a container for all elements of the model. For each EPC element type there is a dedicated element. All elements no matter of their type have an `id` and a `name` for identification purposes, and a `defRef` if the logical element is included at different places in the model. EPML supports `event`, `function`, `processInterface`, `xor`, `and`, and `or` element types as well as some elements of extended EPCs (cf. [MN06]). Functions and process interfaces can have a reference to a linked EPC via a `linkToEpcId` attribute. All elements of an `epc` can be connected using `arc` elements with the `fromId` and `toId` attribute defining the direction. For further details on EPML the reader is referred to [MN06].

Table 1: EPC interchange formats compared

	Visio VDX	ARIS AML	EPML
EPC Model	Page	Model [Model.Type]	epc
Model Organization	Pages	Group	directory
Function	Shape [Master]	ObjOcc [SymbolNum]	function
Event	Shape [Master]	ObjOcc [SymbolNum]	event
XOR-connector	Shape [Master]	ObjOcc [SymbolNum]	xor
AND-connector	Shape [Master]	ObjOcc [SymbolNum]	and
OR-connector	Shape [Master]	ObjOcc [SymbolNum]	or
Control flow arc	Shape, Connect	CxnOcc	arc
Logical element	-	ObjDef	definition

## 2.4 Comparing the EPC Interchange Formats

Table 1 gives a comparison of the three discussed interchange formats Visio VDX, ARIS AML, and EPML. It illustrates that EPML is the only one that directly addresses the meta-model of EPCs. Both Visio and ARIS can store arbitrary graphical models; accordingly the data about whether an element is e.g. an EPC function is not stored in the metadata (as an XML element name), but in element and attribute values. In Visio such information is represented in the master section, in ARIS it is encoded in `TypeNum` and `SymbolNum` attributes. Beyond that, each of the interchange formats utilizes a different mechanism to represent arcs between model elements. In Visio the arc is a graphical element. The logical connection between shapes is established by a connect from the first shape to the arc, and from the arc to the second shape. In ARIS, each arc is a subelement of the source node holding a reference to the target node. In EPML, arcs are first class elements having two references to other elements. EPML is also the most compact format of the three. Since it does not include sophisticated layout and visual information, EPML files are much smaller than the AML and VDX files of the same model. Furthermore, a transformation from AML or EPML to Visio implies a certain loss of information since Visio is not able to represent that two or more visual elements are logically the same.

In this section, we introduced and compared three different formats to store EPCs. In the next section, we introduce the ProM framework, which is capable of reading *and* writing all of these formats. Moreover, ProM facilitates a wide variety of analysis techniques for EPCs (ranging from verification to process mining) and it is able to transform EPCs into various alternative formats and vice versa.

## 3 The ProM Framework

In this paper, the focus is on interchanging the different EPC formats in the context of the *ProM* (Process Mining) framework [DMV<sup>+</sup>05]. ProM has been developed as a tool for the *process mining* domain. Process mining aims at extracting information from event

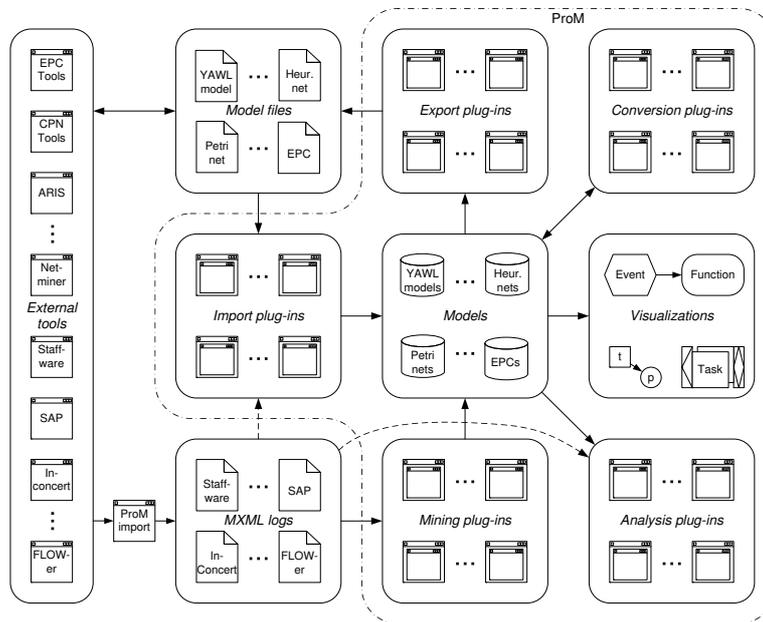


Figure 4: Overview of the ProM framework.

logs to capture the business process as it is being executed. Process mining is particularly useful in situations where events are recorded but there is no system enforcing people to work in a particular way. Consider for example a hospital where the diagnosis and treatment activities are recorded in the hospital information system, but where health-care professionals determine the “careflow”. Many process mining algorithms were developed [ADH<sup>+</sup>03, AWM04, AGL98, CW98, GBG04, GGMS05, GCC<sup>+</sup>04, Her00] and currently a variety of these techniques are supported by ProM.

Although the initial focus of ProM was on process mining, over time the functionality of ProM was extended to include other types of analysis, model conversions, model comparison, etc. This was enabled by the plug-able architecture of ProM (it is possible to add new functionality without changing the framework itself) and the fact that ProM supported multiple modeling formalisms right from the start. By applying ProM in several case studies, we got a lot of practical experiences with model interchange.

Figure 4 shows an overview of the functionality of the ProM framework. The figure shows that ProM can interact with a variety of existing systems, e.g., workflow management systems such as Staffware, Oracle BPEL, Eastman Workflow, WebSphere, InConcert, FLOWer, Caramba, and YAWL, simulation tools such as ARIS, EPC Tools, Yasper, and CPN Tools, ERP systems like PeopleSoft and SAP, analysis tools such as AGNA, Net-Miner, Viscovery, AlphaMiner, and ARIS PPM. We have used more than 20 systems to exchange process models and/or event logs with ProM. As Figure 4 shows there are ways to directly import or export models or to load logs.

ProM is open source and people can change or extend the code. Moreover, ProM offers the so-called “plug-in” concept. Plug-ins allow for the addition of new functionality by adding a plug-in rather than modifying the source code. Without knowing all details of the framework, external parties can create their own plug-ins with ease. Currently there are more than 130 plug-ins. ProM supports five kinds of plug-ins:

**Mining plug-ins** typically take a log and produce a model,

**Import plug-ins** typically import a model from file, and possibly use a log to identify the relevant objects in the model,

**Export plug-ins** typically export a model to file,

**Conversion plug-ins** typically convert one model into another, and

**Analysis plug-ins** typically analyse a model, eventually in combination with a log.

In the paper, we cannot show each of the more than 130 plug-ins. Instead we focus on EPCs, using a model for sales price calculation as reported in [BS04, p.427] as a running example (see Figure 5). The process starts with one of alternative start events and leads to a parallel execution to select articles for calculation, to select organization units, and to check the sort of the calculation. After that, the relevant calculation is determined. If the manual calculation is required, the sales price bandwidth is recorded and the sales price is calculated within that. For both automatic calculation and automatic calculation with manual confirmation, the sales price is calculated first, but in the second case it can be still subject to change. After these alternative calculations, the sales price is stored. If there is no listing process required, the price is modified for those retailers that use POS. Finally, the order set is defined to complete the process.

## 4 A Case Study on Multiple Tool Integration with ProM

In this section, we take the model as shown in Figure 5 in the Visio VDX format and show how we can utilize ProM, to enable the reuse of this model in other tools. Figure 6 shows ProM as a mediator between several formats that we cover in this case.

To start our guided tour of ProM, we take the EPC from Figure 5, as modelled in Visio. Figure 7 shows a screenshot of Visio, where the EPC is being modelled. After saving the EPC to a Visio VDX file, we loaded the EPC in ProM, as shown in Figure 8. Note that ProM loads files without their layout information. Therefore, ProM is able to generate layout information for any imported model. Since ProM was originally designed as a framework for *discovering* models from execution logs where the automatic generation of a layout is of the utmost importance, this feature has been added.

Once the EPC is loaded into the framework, many plugins are available for future processing of the EPC. As an example, we show a tailor-made verification approach, first presented in [DVA05]. This verification approach assumes that the process owner has knowledge about the underlying process and therefore is capable of answering questions



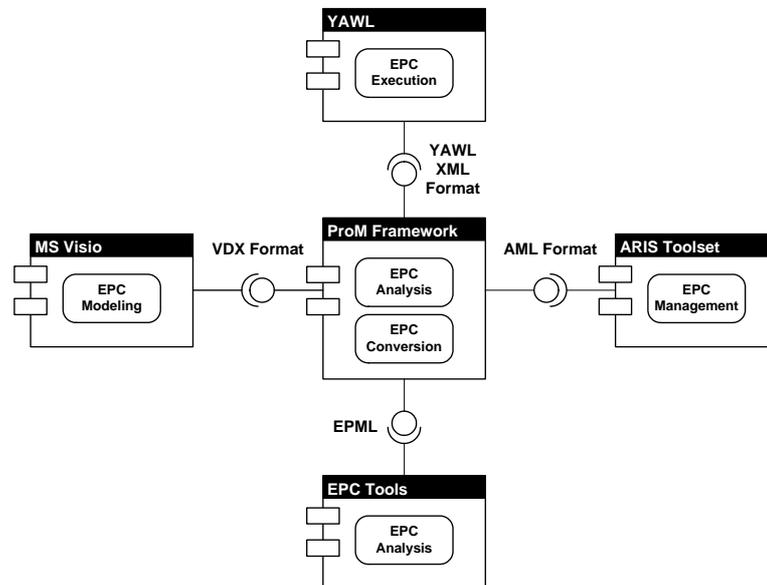


Figure 6: ProM as a mediator between different EPC formats and tools.

about the possible initializations of the process. In our case, the process can be initialized when either one of the initial events occur. Furthermore, we assume that these event cannot occur simultaneously, since that would obviously lead to a problem. The result of the EPC verification in ProM under this assumption is shown in Figure 9, where ProM indicates that there are structural errors in the EPC and the erroneous part is highlighted.

Another option is to export the EPC to an EPML file and to do the verification with EPC Tools [CK04]. EPC Tools implements EPC semantics based on the framework of Kindler [Kin03, Kin04, Kin06] which was defined to fix formal problems of the semantics reported in [NR02]. It provides a soundness check and interactive simulation capabilities. In left control panel of EPC Tools (next to the very left event in Figure 10), there is a red light indicating that the EPC is not sound. The modeler can then propagate process folders in the EPC to detect that it will not complete properly whenever automatic calculation or automatic calculation with manual confirmation is chosen. The error in this model, which was indicated by ProM and EPC Tools, now has to be repaired. Since ProM is not a modelling tool (although it does allow the user to change the type of each connector on the fly), we can now export the EPC to Visio again, update the model there and load it back into ProM. In fact, repairing the model is not so difficult, since it only requires us to change the type of one connector: if the AND-split after sales price calculated becomes and XOR, ProM now no longer indicates an error in the EPC.

Let us assume that a process designer at this point is satisfied with the model and therefore wants to upload it into the ARIS toolset. The process for doing so is simple. First, in ProM, we export the EPC to the AML format. Then, in the ARIS Toolset, we import the EPC to the database. The result of this process is shown in Figure 11. Note that again the

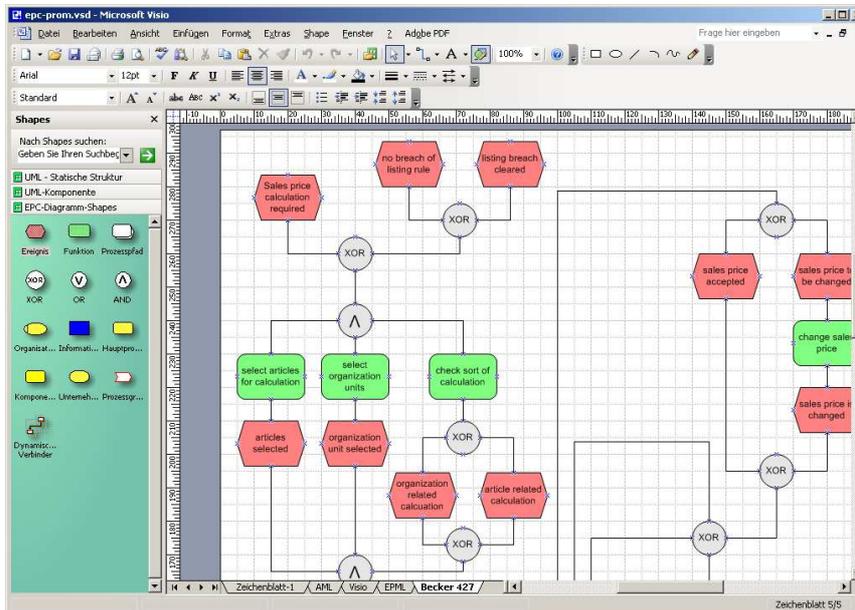


Figure 7: The example EPC modelled in Visio.

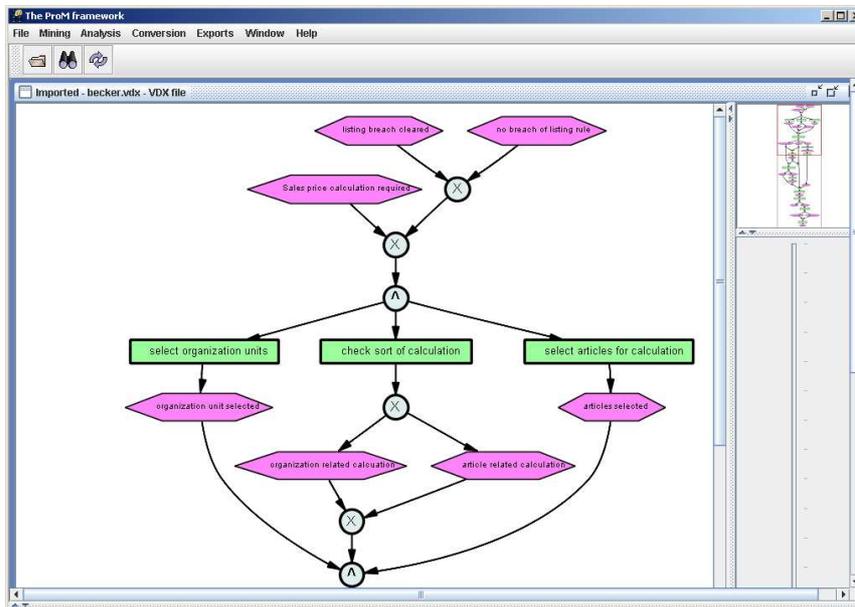


Figure 8: The example EPC imported in ProM.

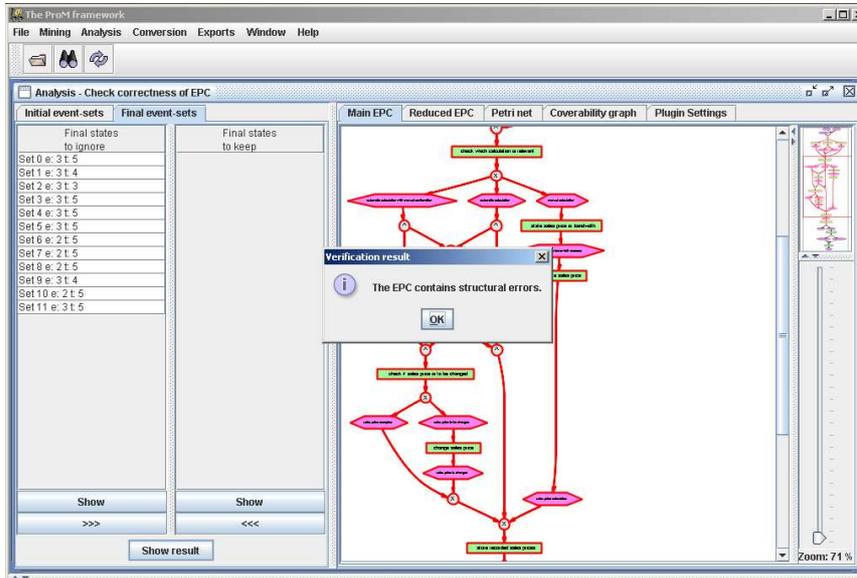


Figure 9: The result of EPC verification in Prom.

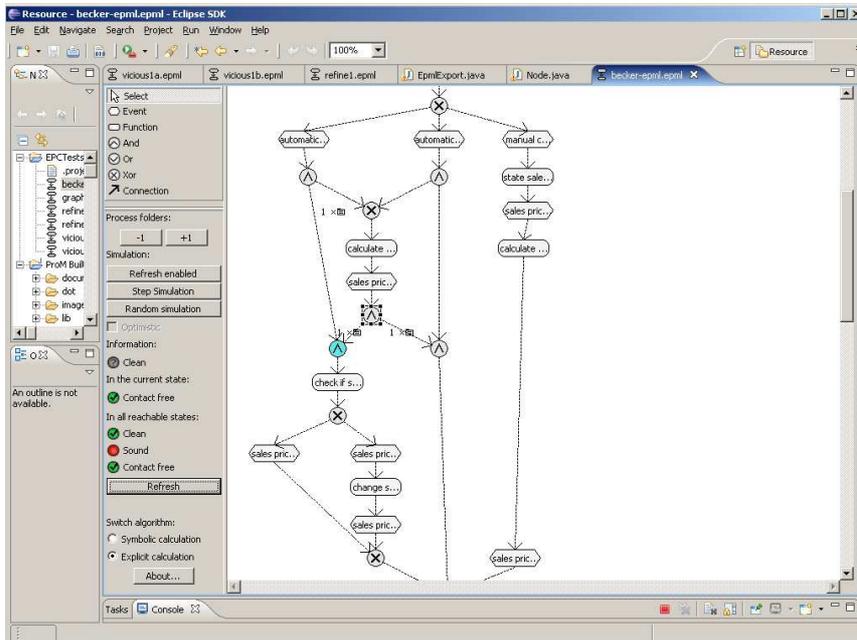


Figure 10: The result of EPC verification in EPC Tools.

layout is slightly different since ARIS provides its own layout algorithms as well.

So far, we have shown that EPCs can be imported and exported in several formats. Furthermore, we have shown that ProM provides plugins for the analysis of EPCs. However, there is one essential aspect of EPCs that we have not addressed yet, namely the execution of EPCs. Due to the OR-join in an EPC, the execution of EPCs is not straightforward. However, EPCs can directly be translated to YAWL models, which are executable in YAWL. ProM again provides a translation of EPCs to YAWL. These YAWL models can then be loaded in the YAWL engine (see [AADH04]). Figure 13 shows an activity of the sales price calculation enabled in the YAWL engine for execution. Beyond that, YAWL models can also be analyzed using a verification tool called WofYAWL regarding the relaxed soundness criterion [DR01]. The corrected example EPC is shown as a YAWL model in Figure 13. This approach has been used to verify all 604 EPCs in the SAP reference model, the results of which can be found in [MMN<sup>+</sup>06].

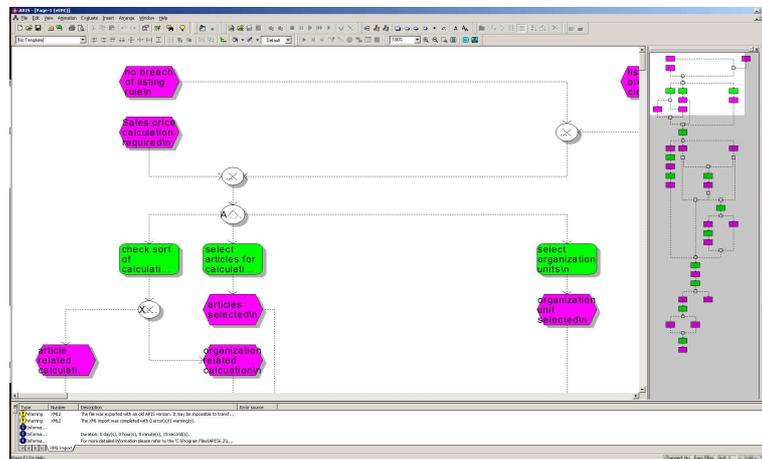


Figure 11: The corrected EPC uploaded into the ARIS Toolset

## 5 Conclusion and Future Work

The heterogeneity of different formats for EPCs is a major hinderance for model interchange between specialized tools in practice. In this paper, we compared three different formats for storing EPCs, in particular, the proprietary formats of Microsoft Visio and ARIS Toolset as well as the tool-independent EPML format. Furthermore, we introduced the ProM framework and showed using a realistic example (sales price calculation) that ProM is able to serve as a mediator between these formats. Beyond that, we demonstrated that the ProM framework can be used for the analysis of EPCs and to translate EPCs into YAWL models for execution or for further analysis. In future research, we aim to provide further EPC plug-ins for the ProM framework. First, there are several other popular busi-



Figure 12: The state sales price or bandwidth activity enabled in the YAWL engine.

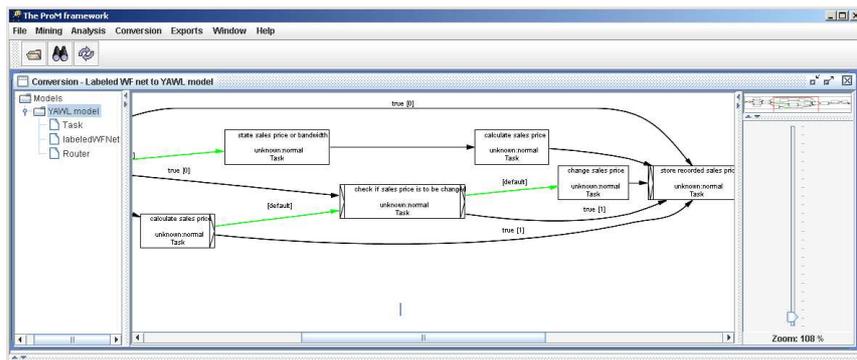


Figure 13: The corrected EPC converted to YAWL for execution.

ness process modeling tools that offer EPCs, but whose proprietary interchange formats are not yet supported. Second, we aim to provide additional EPC analysis plug-ins, e.g. for deriving EPCs from a configured C-EPC.

## References

- [AADH04] W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and Implementation of the YAWL System. In A. Persson and J. Stirna, editors, *Advanced Information Systems Engineering, Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, volume 3084 of *Lecture*

*Notes in Computer Science*, pages 142–159. Springer-Verlag, Berlin, 2004.

- [ADH<sup>+</sup>03] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
- [AGL98] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
- [AWM04] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
- [BS04] Jörg Becker and Reinhard Schütte. *Handelsinformationssysteme*. Moderne Industrie, Landsberg/Lech, 2nd edition, 2004.
- [CK04] N. Cuntz and E. Kindler. On the semantics of EPCs: Efficient calculation and simulation. In *Proceedings of the 3rd GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004)*, pages 7–26, 2004.
- [CW98] J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
- [DMV<sup>+</sup>05] B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
- [DR01] Juliane Dehnert and Peter Rittgen. Relaxed Soundness of Business Processes. In K. R. Dittrick, A. Geppert, and M. C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 151–170, Interlaken, 2001. Springer.
- [DVA05] B.F. van Dongen, H.M.W. Verbeek, and W.M.P. van der Aalst. Verification of EPCs: Using reduction rules and Petri nets. In *Conference on Advanced Information Systems Engineering (CAiSE 2005)*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, Berlin, 2005.
- [GBG04] W. Gaaloul, S. Bhiri, and C. Godart. Discovering Workflow Transactional Behavior from Event-Based Log. In R. Meersman, Z. Tari, W.M.P. van der Aalst, C. Bussler, and A. Gal et al., editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 3–18, 2004.
- [GCC<sup>+</sup>04] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
- [GGMS05] G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining and Reasoning on Workflows. *IEEE Transaction on Knowledge and Data Engineering*, 17(4):519–534, 2005.
- [Her00] J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.

- [IDS03] IDS Scheer AG. *XML-Export und -Import (ARIS 6 Collaborative Suite Version 6.2 Schnittstellenbeschreibung)*. <ftp://ftp.ids-scheer.de/pub/ARIS/HELPDESK/EXPORT/>, Juni 2003.
- [Kin03] E. Kindler. On the semantics of EPCs: A framework for resolving the vicious circle (Extended Abstract). In M. Nüttgens, F. J. Rump, editor, *Proc. of the 2nd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2003)*, Bamberg, Germany, pages 7–18, 2003.
- [Kin04] E. Kindler. On the semantics of EPCs: A Framework for resolving the vicious circle. In J. Desel and B. Pernici and M. Weske, editor, *Business Process Management, 2nd International Conference, BPM 2004*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97, 2004.
- [Kin06] Ekkart Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data Knowl. Eng.*, 56(1):23–40, 2006.
- [MMN<sup>+</sup>06] J. Mendling, M. Moser, G. Neumann, H.M.W. Verbeek, B.F. van Dongen, and W.M.P. van der Aalst. Faulty EPCs in the SAP Reference Model. In J.L. Fiadeiro S. Dustdar and A. Sheth, editors, *Proceedings of BPM 2006*, volume 4102 of *Lecture Notes in Computer Science*, page 451457, Vienna, Austria, 2006. Springer-Verlag.
- [MN06] Jan Mendling and Markus Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). *Information Systems and e-Business Management*, 4(3):245 – 263, 2006.
- [MNN05] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. *Workflow Handbook 2005*, chapter A Comparison of XML Interchange Formats for Business Process Modelling, pages 185–198. Future Strategies Inc., Lighthouse Point, FL, USA, 2005.
- [NR02] M. Nüttgens and F. J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel and M. Weske, editor, *Proceedings of Promise 2002*, Potsdam, Germany, volume 21 of *Lecture Notes in Informatics*, pages 64–77, 2002.
- [WE03] Mark H. Walker and Nanette J. Eaton. *Microsoft Office Visio 2003 Inside Out*. Microsoft Press, October 2003.

# Using BPEL processes defined by Event-driven Process Chains

Carlo Simon<sup>(1)</sup>, Jörn Freiheit<sup>(2)</sup> and Sebastian Olbrich<sup>(3)</sup>

(1) University Koblenz-Landau, simon@uni-koblenz.de

(2) Max-Planck Institute, Saarbrücken, freiheit@mpi-inf.mpg.de

(3) Philipps University at Marburg, sebastian.olbrich@zeon.de

**Abstract:** The paper discusses the usability concept of workflow services for event-driven process chains. Usability is similar to controllability, known from Workflow net based BPEL semantics theory. Based on the observation, that if business processes interact (for instance because one of them is a service), it must be possible to check whether this is structurally possible or not. In opposite to prior work, the focus is laid on the exchange of information rather than on the integration of shared events and functions. The theoretical outcome of the paper is applied to a German E-Government (web) service.

## 1 Introduction

The definition and acceptance of the *business process execution language for web-services* (BPELWS or BPEL for short, [ACD<sup>+</sup>03, Ju04]) by the key players in the software industry (especially in the field of workflow management systems) made this language a de-facto standard for the specification of business processes and services. BPEL, however, does not only standardise existing exchange languages but, moreover, puts a new concept into the centre of the considerations: the development of *service-oriented information systems' architectures* instead of monolithic ones. BPEL systems are highly distributed and have to interact, i.e. *communicate* with each other. With the introduction of this paradigm, also new challenges and questions concerning the modelling of such (distributed) systems came up: (1) What are appropriate modelling languages for such systems? (2) Which properties of such systems must be checked in order to validate their proper work.

One of the main difficulties in answering these questions is that the *specification of BPEL* is given in natural language, i.e. no mathematical interpretation of BPEL processes exists. Consequently, there does not exist a verification technique which can be derived from the BPEL specification to verify the collaboration of BPEL services. A possible solution to this is to define such a semantics after the event as discussed in Section 5. By introducing - for instance - a Workflow net semantics it is possible to answer the question whether a BPEL service can *be controlled* by another. An application of this approach obviously makes sense, if the BPEL specifications are *already given*.

Another possible starting point is a description of business processes which has *not already* been translated into BPEL specifications. This is, for example the case, if an organisation has to decide on shifting its information system to a *service-oriented architecture (SOA)*. At this moment, the services of this SOA must be identified, specified and their usability must be guaranteed. Since event-driven process chain (EPC) is one of the most popular business process modelling languages, it is both in theory and practice a highly relevant problem *to answer usability questions on the base of EPC models*.

The automatic synthesis of an environment for a modelled dynamic system - a business process or a machine - is a means to describe its *controllability*. This, however, demands a formal state semantics of the system to be controlled which is usually not given for a business environment. Nonetheless, many ideas and concepts of controllability - as shown in this paper - can be transferred also to conceptual process models although they cannot be mapped into a discrete finite state space. Since there are differences between the two approaches, we use the term *usability* for our findings for conceptual process models.

Although the consideration of this problem makes use of prior work on the integration of EPC models discussed in Section 4, this is not the ultimate solution. The integration described there is based on merging events and functions with similar meaning in single elements, i.e. the modelled processes are synchronised within these elements. A service, however, is decoupled from its user and they might operate asynchronously. Then a message passing mechanism is required rather than a synchronisation.

In order to explain the problem, to summarise the prior work, to develop a new theoretic result, and to apply this to a modelling problem, this paper is organised as follows: after an introduction to communicating workflow services in Section 2, prior work on giving a state semantics to BPEL processes to demonstrate their controllability is discussed in Section 3. Afterwards, prior work on the integration of EPC models is discussed in Section 4 in contrast to message passing between EPC models considered in Section 5. An application of the approach is given in Section 6 before the paper ends with some concluding remarks.

## 2 Communicating Workflow Services

There exist various definitions of *business processes*, that all describe business processes as interconnected activities to achieve a specific business goal [Ga83, Ja96, JB96, St01, AH02, Ga03, Ho04, SS04]. In a more specific extension of this definition, the fulfilment of customer needs is seen as the ultimate goal [Da93, HC93, Po04]. The Workflow Management Coalition (WfMC) defines in addition *workflows* as electronically supported business processes [Ho95].

Workflows are increasingly used to specify the behaviour of services and especially of web services [RSS05]. The use of services makes it possible to provide support for business processes that must be executed in a distributed, heterogeneous environment. As pointed out in [RSS05], the communication of services is no simple input/output relationship but during service execution the service has to interact with its environment. Consequently, the interface has a *life-time* and its structure might change during this time.

Currently, one of the most widely discussed languages for the specification of business processes is the *business process execution language for web services* (BPEL<sub>AWS</sub> or BPEL for short) [ACD<sup>+</sup>03, Ju04]. The language is based on formerly defined languages such as IBM's WSDL [Le01] and Microsoft's XLANG [Th01]. As the name indicates, BPEL supports the exchange of process information in web service environments. The focus lies on actually running processes rather than on general process specifications. The specification is currently under revision.

Interoperability is a topic in today's business process area. Even if BPEL processes work properly in isolation, their interaction may lead to problems like mutual waiting and thus a deadlock of the entire collaborative process. Automatic verification of interoperability correctness is crucial for guaranteeing proper behaviour of trans-organisational processes. Two problems complicate verification of collaborative processes: the merged process is usually too big to be analysed and the partners cannot be expected to publish their processes in entire detail. Thus, techniques are required to overcome these problems. In [RSS05] the notion of *controllability* has been introduced. Intuitively, a service is *controllable* if there exists a partner such that both can act and interact properly [LMSW06]. There are several approaches for providing a formal semantics for BPEL [FFK04, FGV04, St04, VA05]. Based upon these considerations, Hinz et. al. propose a translation of BPEL specifications into workflow nets [HSS05]. The purpose of this translation is less to provide a better comprehension or visualisation than to have a model which supports model checking. The transformation result, however, is only an *interpretation* since the semantics of BPEL processes is not formally defined. Assuming a workflow net semantics for BPEL, controllability can be proved automatically by constructing an automaton that describes the interaction of a workflow. This automaton contains all sufficient information while preserving inner details of the process such that correct interoperability with processes that act according to this automaton can be guaranteed. However, all these approaches are based on an interpretation of the BPEL specification only.

### 3 Controllability of Workflow net Interpretations of BPEL processes

The notion of *controllability* has been developed for BPEL processes on top of a workflow net [AH02] semantics for these processes [St04]. Using pattern for basic and structured elements of BPEL processes they are transformed automatically following canonical building rules [HSS05] into *open workflow nets* (oWFN) [MRS05]. oWFN are a more general kind of workflow nets having a *set* of input places and a *set* of output places instead of having exactly one each.

The *composition* [LMSW06] of two oWFNs  $N$  and  $M$  (denoted as  $N \oplus M$ ) results in one oWFN where the joint places and initial markings of both nets are merged. Joint places are required to be input places of the one net and output places of the other, i.e. they do not share input, output and inner places, respectively. In order to define controllability we first describe what a proper (correct) behaviour of an oWFN means. Similar to the soundness criterion for workflow nets, for oWFN the criterion weak-soundness has been developed, which intuitively states that every started process must terminate, no object within the

process may remain ignored within the process after termination and there is no futile activity within the process, i.e. for every activity there is at least one run of the process that contains this activity. An oWFN is *weak-sound* [Ma03] if from every *reachable marking* a final marking is reachable and if in every final marking only output places are marked. Having defined composition and weak-soundness, a given oWFN  $N$  is *controllable* if there exists an oWFN  $S$  such that  $N \oplus S$  is weak-sound. Then,  $S$  is also called a *strategy* of  $N$ .

For a given oWFN  $N$ , a strategy  $S$  can be generated automatically by constructing an *interaction graph* that represents the behaviour of  $S$  [We04]. The interaction graph is a directed graph where the arcs represent the incoming and outgoing messages from and to  $N$  and each node of the graph represents the state(s) of  $N$  until  $N$  sends or receives a new message. Any newly sent or received message leads to another node of the interaction graph. If all terminal nodes of the interaction graph contain only terminal states of  $N$  corresponding to the definition of weak-soundness, then  $N$  is controllable. In [MS05] it has been shown how interaction graphs can be extended in order to represent not only some but all strategies of  $N$  by using the concept of *operating guidelines*.

The disadvantage of this approach is that it makes use of the reachability set and not of the structure of the synthesised workflow net. In opposite to this, the method introduced next uses the original EPC models as input only.

## 4 EPC Model Integration

Any specification of communication between EPC models requires coupling the participating models in some form. A tight coupling by merging equally interpreted elements is reported in [SM06, MS06]. Also the origins of the presented work - like for the discussed controllability/usability - lie in the coupling of a kind of workflow nets called *module nets* which might be explained here first to provide a better understanding of Section 5.

Formal integration of processes has been reported for Petri nets in general in [BDK01] and for Process Algebra in [BPS01, Fo00b]. The absence of a proper visualisation and some restrictions concerning the semantics prohibits using Process Algebra for the modelling of businesses [Aa05]. These limitations are solved by the Semantic Process Language (SPL) which provides means for a formal specification of process sets that can be verified against non-trivial process-like properties. Since the semantics of SPL formulas (called *modules*) is defined via their Petri net implementation, their visualisation is implicitly given [Si06]. Consequently, it is also possible to simulate the models and verify their properties with all existing Petri net analysing techniques. Moreover, the approach supports stepwise development [Si05].

The formulas of SPL are built over elementary processes which specify the *occurrence* or *prohibition* of an action. Elementary processes are linked to each other using operators for *sequence* building, *alternatives*, *concurrency* and *synchronisation*, *iteration*, and *negation*. Canonical building rules generate from modules *module nets* - Petri nets with explicit *start* and *goal* transitions. Each firing sequence of such a net which reproduces the empty initial marking by firing *start* and *goal* transitions exactly once is interpreted as a *process* of the module net and, by definition, also of the original module.

These definitions have some important similarities with the issues relevant for the transformation of BPEL processes into EPC models. The constructive operators of BPEL correspond in many details to the operators of SPL. Moreover, proving in SPL makes use of the original input models and does not depend on the reachability set. The methods developed for SPL may therefore be applied to EPC models as well.

One central operator for proving in SPL is the *synchronisation* operator which defines the co-execution of the operands' processes such that they are merged in shared actions. This operator is implemented in module nets by *joining* all equally interpreted transitions (i.e. transitions which specify the occurrence or prohibition of the same action) and the *start* and *goal* transitions. It yields, in principle, in the intersection of the process sets of the participating modules concerning common actions.

Although the approach has been used already for the modelling, analysis and optimisation of business processes in some practical cases, it is - in opposite to EPC models - not established yet. It was therefore an aim to transfer the verification technique to EPC models which is in principle possible since in both cases only the original models and not a reachability analysis is used for verification. At the example of two EPC models taken from the SAP reference model, the formal integration of such models is demonstrated. Both processes describe how a customer inquiry about products is received, processed, and a quotation is created from the inquiry (the first one is taken from the Project Management branch of the SAP reference model, the second process EPC stems from the Sales and Distribution branch). Figure 1 shows the exemplary processes.

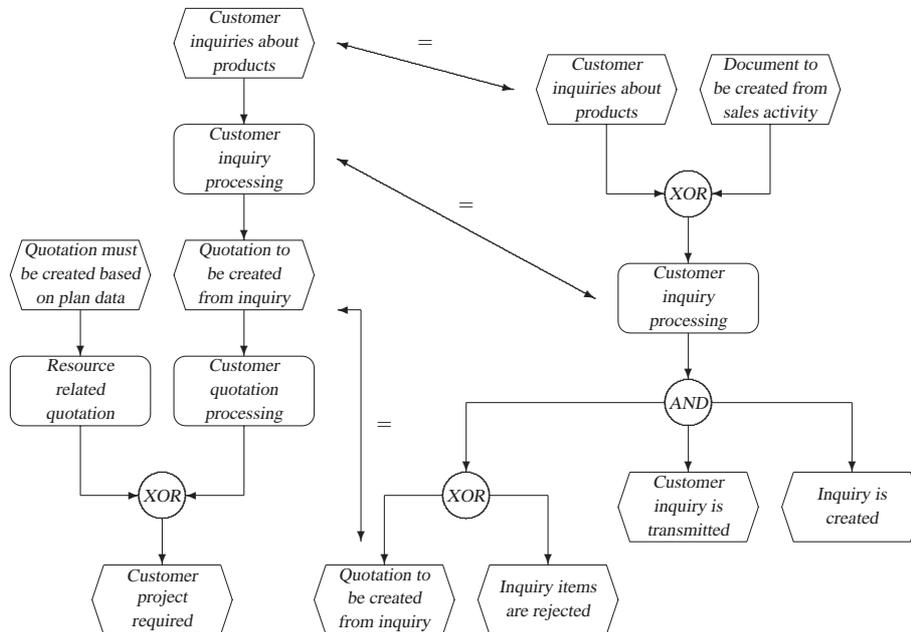


Figure 1: *Customer Inquiry* (left) and *Customer Inquiry and Quotation Processing* (right)

Indicated by equal names, the processes share two events and one function. Joined events are caused by all previous functions and triggered by all follower functions of the original EPC. Joined functions share their previous pre- and post-conditions (in terms of events). Figure 2 shows the result of the join. Redundant connectors are drawn with grey lines and can be omitted. The resulting EPC contains the intersection of the processes represented by the operands' EPCs.

## 5 Data Exchange between and Usability of EPC models

Within BPEL specifications, elementary and structured activities are distinguished. *Invoke* (sending a message) and *receive* (receiving a message) are the most important elementary activities concerned with the exchange of information between services. They are therefore in the centre of the following considerations.

A first approach to represent *invoke* and *receive* in EPC diagrams is by coupled combinations of functions and events as demonstrated in Figure 3.

In principle, information sent successfully (event *invoked*) is the start event for an asynchronous receive. However, as the implementation of the *receive* activity as an augmented workflow net in [RSS05] demonstrates, this solution does not take into account the complexity of both activities that must be operated in a workflow management system.

In a workflow management system, different instances of the same kind of process might be executed in parallel which are distinguished with the aid of a *correlation set variable*. A sent/received message must then include an identifier which enables a service to decide on whether it is addressed by the message or not. If it is not addressed, the service should return into its initial state. Before operating a received information, also a flag should be set which indicates that the message has been received properly or whether a failure has occurred. In case of a successful message transfer, also the received information should be published in an interface. Figure 4 shows the implementation of this interpretation of the processing of a *receive* as an extended EPC, i.e. besides the pure process structure also the used information objects are shown. This implementation does not include a verification of the port type which is the case in the example of the following section. Concerning the model of Figure 4 a critical remark must be made: it probably is much too close to an implementation of the *receive* activity than this is typical for an EPC model. On the other hand, it is still on a more abstract level than the respective model shown in [RSS05] which uncovers many internal details of this activity. Although this accuracy is required for a formal verification, it is obviously over-specified if the possibility to couple business processes must be discussed by modellers who want to use the service. For this purpose, the shown EPC model seems to be more appropriate.

The presented EPC model of Figure 4 explains a modeller that for successfully using/controlling the *receive* activity a proper correlation set selection is required. S/He can, moreover, recognise that the acceptance of a transmitted message is based on a further check routine which must be taken into account if the *receive* activity is used. The following section applies these observations to a practical example.

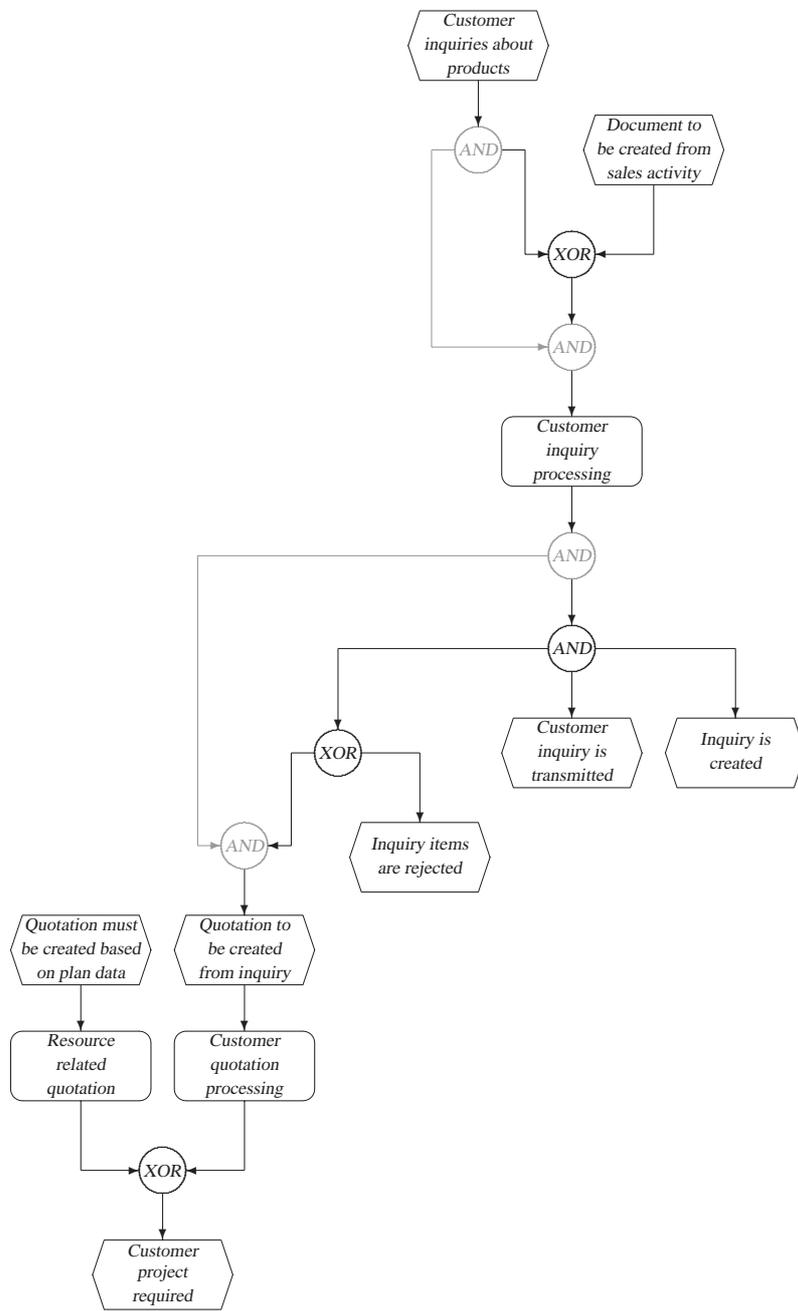


Figure 2: Integrated EPC of *Customer Inquiry* and *Customer Inquiry and Quotation Processing*

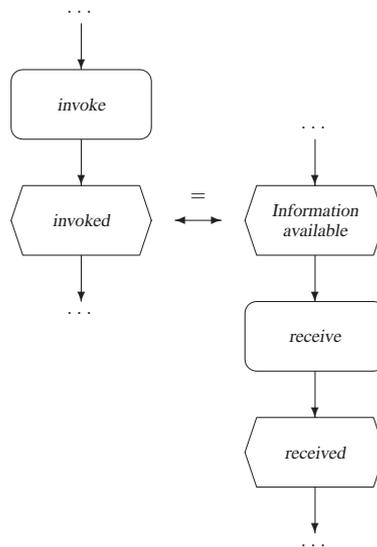


Figure 3: First implementation of *invoke* and *receive*

## 6 An E-Government Example

As a result of new political and economic agendas Public service provision in Europe has been undergoing dramatic transformation for the last couple of years. Consequently, there have been acute pressures caused by the swelling demand for both existing and new types of public services. Challenging these changes, new information and communication technologies are now opening up new opportunities to the public sector and the promotion of public services. The installation of information society applications to provide significant increase in the public sector efficiency is sub-summarised under the term of E-Government [FO00a].

On the downside, the management of the organisational change often exceeds the abilities of public institutions [SKH03] - especially the ones of smaller institutions or municipalities. On top of that, process reorganisation - necessary to offer the enhanced services in the public sector - must often have to stop short of established structures [SZ97] or legal constraints [AO05].

The introduction of web services as a solution for the establishment of E-Government could overcome those problems [SG01]. Precondition, of course, is that the technical abilities are sufficiently provided and, correspondingly, the concept of business process modelling is more and more accepted by the public authorities as well [WT03]. As a demonstration example, we analyse a web service that is currently in the evaluation period (though fully available): issuing permission on importing/exporting goods by the German federal exportation authority, the *Bundesamt für Wirtschaft und Ausfuhrkontrolle (BAFA)*.

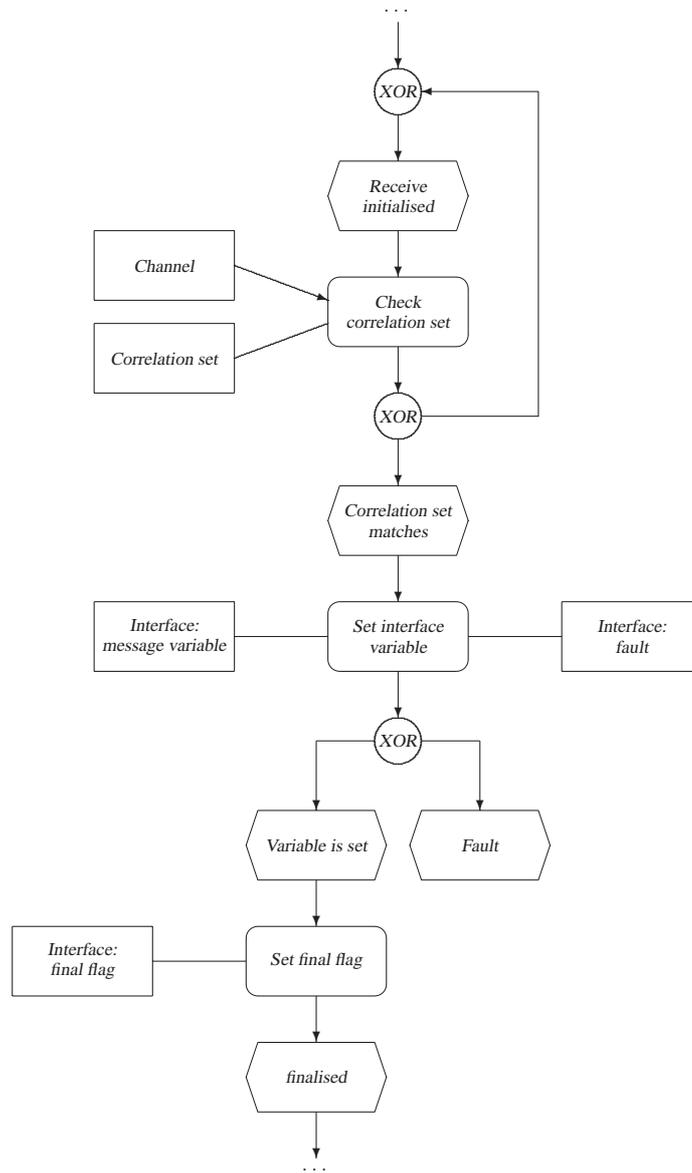


Figure 4: Implementation of *receive* specifying the internal behaviour of this activity

The BAFA is the executing authority in Germany for permissions on cross-border transactions. It takes its decisions depending on the exported/imported goods and the country the goods are exported/imported to. The BAFA does, however, not check these goods or their shipping destination/origin itself (this is duty of the customs), but rather the information on the product and whether exporting/importing of this certain item interferes with the interests of the Federal Republic of Germany or any other European Union member state. If not, an import/export certificate is issued.

To decide which products are permitted for importing/exporting from/to which country, the BAFA uses several product and country lists. The content of these lists depends on the current political debate in the European parliament and describes criteria on several categories of items that are forbidden for import/export. For example, it is not permitted to import goods to Europe that are preserved by European law (e.g. rare types of tropic wood, corals, animals) and many types of weapons or chemicals are generally forbidden for export (e.g. weapons of mass destruction). Some of these regulations might depend on a certain country or region, for example if there is an international embargo.

Though these lists are quite voluminous and change frequently, the core permission process at the BAFA is quite simple: either the applicant's information is found on a list and the import/export certificate is denied, or a specific good is free for import/export. The process is extended, if the product is more complex - for instance, a milling machine can be used to produce toys and guns as well. In case of a so called *dual-use* products, the process is extended by checking the exact purpose of the product and the background of vendor and buyer. For this, the BAFA cooperates with other institutions in various countries (i.e. customs, police, financial offices). Since the extended process bares little potential for automation, it is out of the scope of our analysis.

Already in the year 2000, the German E-Government Initiative *BundOnline 2005* uncovered the potential which lies in the automation of BAFA application processes and underlined the importance to electronically support these federal authorities' processes [BM01]. An electronic application system form (ELAN) was developed and published on the web page of the BAFA.<sup>1</sup> Since then, an applicant is able to register via a web interface and fill in the data. The procedure offers two major advantages: exporters do not have to fill in similar data more than once anymore and the BAFA could automate a large amount of its processes in the backoffice.

Yet, the information still needs to be typed in manually even though precise information of product and destination/origin is mostly electronically available in the applicant's database. In order to automate the overall process, the BAFA recently created a web service as an interface to its own electronically stored data which contains the regulations (lists) on goods and countries.<sup>2</sup> Now, the (general) information on whether permission for a specific product is needed can be answered via this web service immediately.

The class which implements the web service is called *TransmissionService*. For a user of the web service, the public methods *transferApplication()* and *submitApplication()* are of special interest. Each method requires a parameter on its call: *transferApplication()*

---

<sup>1</sup><http://www.bafa.de>

<sup>2</sup>The ELAN web service can be reached at <https://fg01.bafa.bund.de/elan/webservice>.

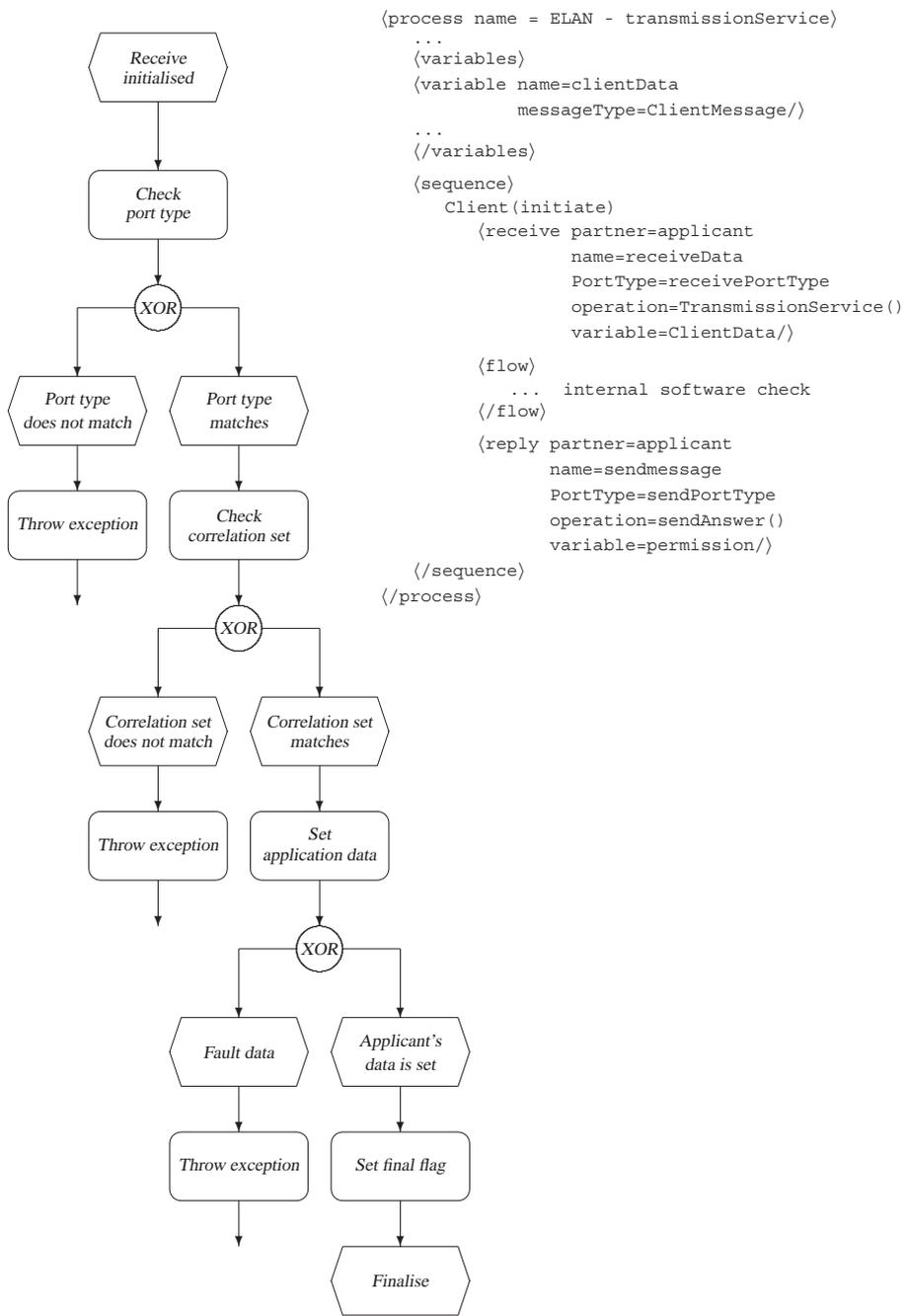


Figure 5: BPEL process of the example

requires a string which contains the application itself as an XML file, *submitApplication()* contains the user data. Since the correctness of the user data is checked before the web service is invoked, we focus on modelling the *TransmissionService*, which belongs to the implementation of *receive*. The *receive* activity is the one that differs from our theoretical models. As Figure 5 illustrates, we can derive both, BPEL process and EPC usability graph.

As explained in the previous paragraph, the *invoke* activity only occurs if the user is positively identified. Consequently, we can assume that, for the *receive* process to be usable, it needs to check the port type, the correlation set and the applicant's data (variables) as demonstrated in Figure 4. The EPC diagrams of Figure 5 and Figure 4 differ slightly in a way that every *XOR* operator in the BAFA's model explicitly throws an exception of its own and terminates the application. The more general defined BPEL model of *receive* however, allows multiple options for the occurrence of exceptions.

We are currently negotiating with the BAFA concerning the terms of publishing our models as part of their documentation of their web services on the web site of the BAFA.

## 7 Conclusion

In contrast to the concept of controllability which can be formally specified and proved (see Section 3), we have discussed a less specific approach to support a modeller in providing interfaces of interacting workflows. In this paper, for important elementary BPEL activities EPC models are presented that on the one hand show the core meaning of these activities (e.g. message passing) and on the other hand explain details that may be observed for correctly interacting workflows. In order to avoid confusion with the term controllability, we use the term usability which, intuitively, has a very similar meaning: there is a partner or environment that satisfies the input and output of a workflow such that the workflow runs and terminates correctly. Unlike the formal semantics, we have chosen a rather model-based approach to discuss BPEL constructs and their proper use for correctly interacting workflow models. As the example in Section 6 shows, the integration method presented in Section 4 can be used to compose such interacting workflows.

The amount of research projects in the sector of inter-enterprise-process-integration shows the relevance of the topic. One of these projects is the E-Justice concept of the European Union which is part of the 6th Research Framework Program funded by the action plan for eEurope 2005 by the European Commission [Co05]. Next, we plan to enhance the user-interface management for Public Services [FZ06] by our method. Thus, we hope to move closer to unified and transparent process sets over the European Union.

## References

- [Aa05] Aalst, W. M. P., van der: Pi calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "Pi hype". *BPTrends*. 3(5):1–11. 2005.
- [ACD<sup>+</sup>03] Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., D. Smith, D., Thatte, S., Trickovic, I., und Weerawarana, S. Business Process Execution Language for Web Services - Version 1.1. <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf> (last accessed 5.9.2005). May 2003.
- [AH02] Aalst, W. M. P., van der und Hee, K., van: *Workflow Management - Models, Methods, and Systems*. MIT Press. Cambridge, MA. 2002.
- [AO05] Alpar, P. und Olbrich, S.: *Legal Requirements and Modelling of Processes in e-Government*. Electronic Journal of e-Government Volume 3 Issue 3, S.107-116. 2005.
- [BDK01] Best, E., Devillers, R., und Koutny, M.: Petri net Algebra. In: Brauer, W., Rozenberg, G., und Salomaa, A. (Hrsg.), *EATCS*. Monographs in Theoretical Computer Science. Berlin. 2001. Springer.
- [BM01] BMI: *Umsetzungsplan für die eGovernment-Initiative Bund Online 2005*. Bundesministerium des Inneren, Stabsstelle Moderner Staat - Moderne Verwaltung. Kabinettsbeschluss vom 14. November 2001.
- [BPS01] Bergstra, J. A., Ponse, A., und Smolka, S. A. (Hrsg.): *Handbook of Process Algebra*. Elsevier. Amsterdam. 2001.
- [Co05] Commission, E.: Europe 2005 - an information society for all. [http://europa.eu.int/information\\_society/eeurope/2005/index\\_en.htm](http://europa.eu.int/information_society/eeurope/2005/index_en.htm) (06-07-06). 2005.
- [Da93] Davenport, T. H.: *Process Innovation: re-engineering Work through Information Technology*. Harvard Business School Press. Boston, MA. 1993.
- [FFK04] Fisteus, J. A., Fernández, L. S., und Kloss, C. D.: Formal Verification of BPEL4WS Business Collaborations. In: Bauknecht, K., Bichler, M., und Pröll, B. (Hrsg.), *E-Commerce and Web Technologies (EC-Web '04)*. volume 3182 of *Lecture Notes in Computer Science (LNCS)*. S. 76–85. Zaragoza, Spain. 2004. Springer.
- [FGV04] Farahbod, R., Glässer, U., und Vajihollahi, M.: Specification and Validation of the Business Process Execution Language for Web Services. In: Zimmermann, W. und Thalheim, B. (Hrsg.), *Abstract State Machines 2004. Advances in Theory and Practice*. volume 3052 of *Lecture Notes in Computer Science (LNCS)*. S. 78–94. Lutherstadt Wittenberg, Germany. 2004. Springer.
- [FO00a] FOEV: *Speyrer Definition von Electronic Government*. Forschungsinstitut für öffentliche Verwaltung bei der deutschen Hochschule für Verwaltungswissenschaften Speyer, <http://foev.dhv-speyer.de> (2003-01-14). 2000.
- [Fo00b] Fokkink, W.: *Introduction to Process Algebra*. Springer. Berlin. 2000.
- [FZ06] Freiheit, J. und Zangl, A.: Model-based user-interface management for public services. In: D. Remenyi (editor): *Conference proceedings of the 6th European Conference on Electronic Government (ECEG), Reading, GB S. 141-151*. 2006.
- [Ga83] Gaitanides, M.: *Prozeßorganisation*. Verlag Vahlen. München. 1983.

- [Ga03] Gadatsch, A.: *Grundkurs Geschäftsprozess-Management*. Vieweg. Wiesbaden. 3. 2003.
- [HC93] Hammer, M. und Champy, J.: *Reengineering the Cooperation*. Harper Business. New York. 1993.
- [Ho95] Hollingsworth, D.: *Workflow Management Coalition: The Workflow Reference Model*. Workflow Management Coalition. Hampshire, UK. Issue 1.1. 1995. <http://www.wfmc.org/standards/docs/tc003v11.pdf> (last accessed 5.10.2005).
- [Ho04] Hollingsworth, D.: The Workflow Reference Model 10 Years On. In: Fischer, L. (Hrsg.), *Workflow Handbook 2004*. Lighthouse Point, FL. 2004. Future Strategies Inc.
- [HSS05] Hinz, S., Schmidt, K., und Stahl, C.: Transforming BPEL to Petri Nets. In: Aalst, W. M. P., van der, Benatallah, B., Casati, F., und Curbera, F. (Hrsg.), *Business Process Management (BPM 2005)*. volume 3649 of *Lecture Notes in Computer Science (LNCS)*. S. 220–235. Nancy, France. 2005. Springer.
- [Ja96] Jablonski, S.: Anforderungen an die Modellierung von Workflows. In: Österle, H. und Vogler, P. (Hrsg.), *Praxis des Workflow-Managements*. S. 65–81. Braunschweig. 1996. Vieweg Verlag.
- [JB96] Jablonski, S. und Bussler, C.: *Workflow Management - Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press. London. 1996.
- [Ju04] Juric, M. B.: *Business Process Execution Language*. Packt Publishing. Birmingham, UK. 2004.
- [Le01] Leymann, F. Web Service Flow Language (1.0). <http://ibm.com/webservices/pdf/WSFL.pdf> (last accessed 17.10.2005). 2001.
- [LMSW06] Lohmann, N., Massuthe, P., Stahl, C., und Weinberg, D.: Analyzing Interacting BPEL Processes. In: *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*. volume 4102 of *Lecture Notes in Computer Science*. S. 17–32. Springer-Verlag. sep 2006.
- [Ma03] Martens, A.: *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. Dissertation. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II. 2003. erschienen in WiKi: Stuttgart, Berlin & Paris.
- [MRS05] Massuthe, P., Reisig, W., und Schmidt, K.: An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*. 1(3):35–43. 2005.
- [MS05] Massuthe, P. und Schmidt, K.: Operating Guidelines - an Automata-Theoretic Foundation for the Service-Oriented Architecture. In: Cai, K.-Y., Ohnishi, A., und Lau, M. (Hrsg.), *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*. S. 452–457. Melbourne, Australia. September 2005. IEEE Computer Society.
- [MS06] Mendling, J. und Simon, C.: Business Process Design by View Integration. In: Eder, J. (Hrsg.), *Proceedings: 2nd Workshop on Business Processes Design (BPD'06)*. volume 4103 of *Lecture Notes in Computer Science (LNCS)*. S. 55–64. Wien, September, 5 - 7. 2006. Springer.
- [Po04] Porter, M. E.: *Competitive Advantage*. Free Press. New York. 2004.
- [RSS05] Reisig, W., Schmidt, K., und Stahl, C.: Kommunizierende Workflow-Services modellieren und analysieren. *Informatik Forschung und Entwicklung*. 20:90–101. 2005.

- [SG01] Spahn, D. und Gisler, M.: *eGovernment: Eine Standortbestimmung*. Haupt-Verlag, Bern - Stuttgart, Wien, 2. Auflage. 2001.
- [Si05] Simon, C.: Incremental Development of Business Process Models. In: *EMISA 2005, Development Methods for Information Systems and their Application*. Number P-75 in Lecture Notes in Informatics (LNI). S. 222–235. Klagenfurt. 2005. GI.
- [Si06] Simon, C.: Integration of Planning and Production Processes. In: *Mathmod 2006, Special Session: Petrinets: Current Research Topics and their Application in Traffic Safety and Automation Engineering*. Wien, Austria. 2006.
- [SKH03] Scheer, A.-W., Kruppke, H., und Heib, R.: *E-Government - Prozessoptimierung in der öffentlichen Verwaltung*. Springer Verlag, Berlin Heidelberg, S.5. 2003.
- [SM06] Simon, C. und Mendling, J.: Verification of Forbidden Behavior in EPCs. In: Mayr, H. C. und Breu, R. (Hrsg.), *Proceedings: Modellierung 2006*. Number P-82 in Lecture Notes in Informatics (LNI). S. 233–242. Innsbruck, Austria, März, 22.-24. 2006. GI.
- [SS04] Schmelzer, H. J. und Sesselmann, W.: *Geschäftsprozessmanagement in der Praxis*. Hanser. München. 4. 2004.
- [St01] Staud, J. L.: *Geschäftsprozessanalyse*. Springer. Berlin. 2. 2001.
- [St04] Stahl, C.: A Petri Net Semantics for BPEL. Technical Report 188. Humboldt-Universität. Berlin. 2004.
- [SZ97] Snellen, I. und Zuurmond, A.: *From Bureacracy to Infocracy: Management through Information Architecture*. In: Tyler, Snellen, Zuurmond (Hrsg.), *Beyond BPR in Public Administration, Institutional Transformation in an Information Age*, Amsterdam IOS Press, S. 205-224. 1997.
- [Th01] Thatte, S. XLANG - Web Services for Business Process Design - Initial Public Draft. [http://www.gotdotnet.com/team/xml\\_wsspecs/xlang-c](http://www.gotdotnet.com/team/xml_wsspecs/xlang-c) (last accessed 17.10.2005). 2001.
- [VA05] Verbeek, H. M. W. und Aalst, W. M. P., van der: Analyzing BPEL Processes using Petri Nets. In: Marinescu, D. (Hrsg.), *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*. S. 59–78. Florida International University, Miami, Florida. 2005.
- [We04] Weinberg, D.: Analyse der Bedienbarkeit. Diplomarbeit. Humboldt-Universität zu Berlin. October 2004.
- [WT03] Wimmer, M. und Traunmüller, R.: *Geschäftsprozessmodellierung in E-Government: eine Zwischenbilanz*. eGov days 2003 Arbeitskreis Organisation, <http://falcon.ifs.uni-linz.ac.at/eGovProzessmodellierung/wimmer-traunmueller.pdf> (20-10-2004). 2003.



# Transformation of Use Cases to EPC Models

Daniel Lübke

Leibniz Universität Hannover, FG Software Engineering  
daniel.luebke@inf.uni-hannover.de

**Abstract:** Within the requirements phase of many projects, functional requirements are often documented as Use Cases. Within SOA projects, however, these Use Cases are not sufficient since they do not represent a global control flow resembling the one of a business process - an integral aspect of a SOA. Instead they are written from the point of view of a single actor. This makes an additional step necessary: Converting the Use Cases to business processes, which is – if done manually – a tremendous task. To reduce this effort wasted on conversion, this paper proposes a method for generating business processes – expressed as EPC – as from Use Cases. Thereby, the need for extensive business process modelling after gathering the requirements is eliminated.

## 1 Introduction

Service-oriented Architecture (SOA) [EMPR05] is an emerging architectural style for organising large information systems in enterprises. SOA aims to integrate software systems by exposing functionality through so-called services which can be composed later on. The composition reflects the business processes the enterprise wants to support. Therefore, business process descriptions become an integral part of the requirements SOA projects need. As has been shown [ZM05], business processes in EPC notation can be used to semi-automatically generate compositions modelled using the Business Process Execution Language (BPEL) [ACD<sup>+</sup>03]. Additionally, they can be annotated to automatically derive user interfaces which can be used in SOA applications [LLSG06].

However, prior to the project's start, business processes might not have been described yet, and if they will certainly lack specifics needed for implementing information systems on top of them: Besides the control flow through an organisation, more information about roles' needs, e.g. information concerning their background, their interaction with the system, is necessary for software development. Especially descriptions of user interaction with the system are normally missing in business process descriptions.

The missing details, which are not documented in any business process, are normally gathered in software projects using software engineering methods - more precisely requirements engineering techniques. A common technique for describing requirements for user interactive systems are Use Cases [Coc05]. Use Cases are freely-written text with only some structure, like tabular templates. They can capture requirements from the point of view of a main actor interacting with the system achieving one goal per Use Case. Therefore, the system is decomposed into many small interaction scenarios. These scenar-

ios together resemble the whole system's functionality. However, requirements captured this way are very detailed but at the same time spread over many Use Cases, making their handling very difficult. UML Use Case Diagrams [Gro04] try to bring back overview by graphically showing Use Cases and respective actors. Use Case Diagrams focus on the relationships between Use Case and Actor and between several Use Cases by their include and extend associations. However, Use Case Diagrams are not designed to illustrate the control flow between Use Cases. As such they are not suited to close the gap to the business processes. Thus, a direct business process-like and control flow centered notation is not directly available.

In order to capture the details needed for describing user interface requirements Use Cases are an appropriate option. However, to derive business processes they must be transformed into a business process model. Therefore, this paper proposes a transformation of a number of small but detailed Use Cases into a business process model. While the transformation is possible to all business process languages, Event-driven Process Chains (EPCs) are used in this paper because they can better express textual conditions using Events than UML activity diagrams with transition conditions. Furthermore, transformation techniques for converting EPC structures to BPEL compositions are available and user interface generation capabilities have been added.

In the next section of this paper Use Cases are introduced in more detail, defined and a meta-model is constructed. Afterwards, the same is done for EPCs. The third section describes the steps for transforming Use Cases to EPC models. Within the fourth section a prototype implementation is shown before an example is presented. Finally, the paper presents some related work and conclusions and an outlook is given.

## **2 Use Cases**

### **2.1 Definition**

According to Cockburn "a use case captures a contract between the stakeholders of a system about its behaviour. The use case describes the system's behavior under various conditions as the system responds to a request from one of the stakeholders, called the primary actor" [Coc05]. Therefore, Use Cases are part of system requirements, defined as a (logical) contract and describe the system interaction with the main actor, i.e. a user. However, this definition does not mention the structure of a Use Case.

Adolph and Bramble state that "Use Cases are simply stories about how people (or other things) use a system to perform some task" [AB02, p. 1]. Moreover, they have the opinion that the semi-formal nature helps structuring the requirements while at the same time not forcing the Use Case writer into too much formalism. But this definition omits a concrete Use Case structure, too.

This vagueness led to misconceptions what a Use Case actually is or looks like. Some people think that a Use Case actually is a UML Use Case Diagram which is definitely wrong. Normally, Use Cases are written in free text. This can be either without any form

or - most commonly - in a tabular form which serves as a template for all relevant Use Case attributes. An example Use Case in tabular form can be seen in figure 1.

Use Case	#3: Thesis Supervisor hands out topic
Primary Actor	Thesis Supervisor
Stakeholders	Thesis Supervisor: wants to hand out topic easily and without much paperwork Student: wants to receive topic quickly Secretary: wants easy to use/read forms for completing registration
Minimal Guarantees	A topic is only handed out once at a time
Success Guarantees	Student knows topic Supervisor knows all needed administrative information of the student
Preconditions	Student has achieved at least 80% of credit points Student has clearance from Academic Examination Office
Triggers	Student wants to sign up for a topic
Main Success Scenario	1. Supervisor checks whether the topic is still available or not 2. Supervisor reserves topic for student 3. System updates list of current thesis 4. Supervisor confirms thesis topic and student's information to the Academic Examination Office 5. System sends confirmation to student, supervisor and Academic Examination Office
Extensions	1a If topic is not available anymore, then EXIT

Figure 1: An Example Use Case

## 2.2 Usage of Use Cases

Use Cases are often used for specifying functional requirements of a software system concerning the user interaction. Use Cases provide a very powerful mechanism for this kind of usage due to the following reasons:

- **Readability:** Use Cases are written in normal language; graphical notations and figures are only used for clarifying the text. Therefore, all kinds of users can read, understand and comment on the requirements from their point of view. This is a very important property of Use Cases since user feedback and collaboration is very important in the requirements phase of a project.
- **User centric:** Use Cases are written from the point of view of an end-user of a system. Therefore, specifics can be discussed independently and directly with the corresponding person(s), e.g. in interviews, without being distracted by additional details not relevant for the actual person.
- **Template Support:** Tabular Use Cases are in themselves a guidance what requirements must be specified: Necessary and helpful attributes, like preconditions, guarantees and extensions, which are often neglected in textual requirements documents, are clearly visible and attracting the necessary attention.

- **Recognition of Error Conditions:** Use Cases encourage thinking about error and abnormal conditions [AB02, p. 2]. These are often neglected areas within requirements specifications and Use Cases are a good way to close this gap.

However Use Cases can have some drawbacks if not embedded correctly into the project:

- **Over-Specification of Requirements:** Use Cases can be very detailed. This can be beneficial for main functions of a system but can be wasted effort in non-critical functions.
- **Missing Overview and Global Control Flow:** Use Cases themselves are not only very detailed but also from the point of view of a single user. It can therefore be a tremendous task to collect and arrange use cases based on their prerequisites in an easy way which allows navigation and an explicit control flow between the Use Cases.
- **Missing Non-Functional Requirements:** Use Cases only describe the behaviour of a system, i.e. mainly the control flow. Non-functional properties, like performance and security requirements cannot be efficiently dealt with, although there are approaches like Misuse Cases [Ale03] and aspects for integrating them into Use Cases [AC03].

Cockburn lists some template variation for all kinds of projects concerning different required detail levels. This includes a template for business process modelling. Although Use Cases are often associated with object-oriented software development, they do not depend on any architectural style. Thus, they can be beneficial within SOA projects as well, as long as the system interacts with the user.

### 2.3 Meta Model

While Use Cases are a semi-formal technique which is normally used within word processors or spreadsheets, one can define a meta-model for a specific template.

People concerned with a specific functionality of a system in general can have two interests: They may be the users of the system in which they need to be obviously involved in narrowing down the requirements because they are directly concerned. Such users are called **actors** in a system. Furthermore, an actor can also be an involved computer system, like the one to be developed. All actors directly interact with the system or are the system itself.

Besides the actors other people are interested in the system: For example, the management financing the system and optimising workflows may have completely different interests than the users. Those people are called **stakeholders**. The difference between actors and stakeholders is especially obvious (and extreme) in the case of new systems replacing some of the existent workers. Stakeholders are written down with their interests in Use

Cases as well, as has been shown in the example. Because all human actors have interests in the system they operate (like usability etc.) all human actors are stakeholders for this particular Use Case as well.

The functionality represented by a Use Case is modelled as a so-called **main scenario** which is performed whenever the **trigger** occurs and if all **preconditions** are met. The scenario itself consists of **steps** which are performed by an actor. The actor can do an action which can influence or trigger other actors (like the system) or business objects (like fill out an order). These elements are the **objects** of a step in the scenario.

The main scenario is the success case of a Use Case in which each step is completed successfully. For **error conditions and other exceptions** the scenario can be split up. Each step can therefore have **extensions**. Each extension in itself is a mini-scenario with steps resulting in a recursive structure. If an extension has been performed it can either exit the Use Case or jump back to a step in the upper scenario.

Steps can reference other Use Cases for improving clarity and partitioning the requirements. If a reference to a Use Case is made from the main scenario the Use Case is said to include the other one. If it is referenced within an extension the Use Case is said to be extended by the other one.

Furthermore, certain **guarantees** must be fulfilled by a Use Case even if something goes wrong. These can be transactional guarantees, like “data is consistent all the time” or other things which must be fulfilled at all times and even in case of failures. These are modelled as **minimal guarantees** in contrast to the **success guarantees** (sometimes also called postconditions) which are achieved by a successful execution of a Use Case.

Use Case’s precondition, minimal guarantee and success guarantee are given as a set of expressions. All expressions are formulated as free text and are implicitly joined by a logical and. If two expressions are given as a precondition the precondition is satisfied if both conditions evaluate to true. In the context of this paper, it does not matter in which form (e.g. as predicates) these expressions are given. They will only be compared for equality and not for implication.

Popular templates match this meta-model. They can therefore not express parallel control flow within a Use Case. Instead parallel activities must be modelled and written down in different Use Cases. Some templates denote the actor of a step separately by having a field for each the actor and the action. This shall encourage the usage of active sentences in order to improve clarity. In the following, access to the step’s actor is assumed.

The complete meta-model modelled as a UML diagram is shown in figure 2.

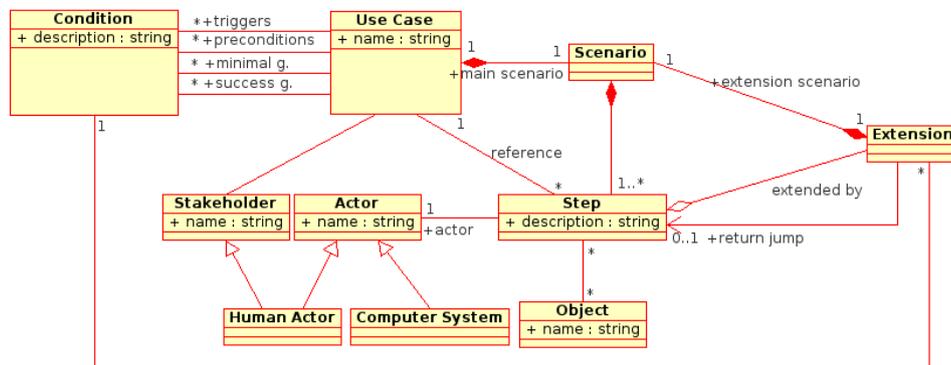


Figure 2: The Use Case meta model

### 3 Event-driven Process Chains

#### 3.1 Definition

Event-driven Process Chains (EPCs) as defined in [NR02] are a graphical model for representing business processes from an organisational point of view.

While business processes do not contain information about non-functional requirements, they do offer a coarser-grained, organisational view on the functions of a system. This means that business processes actually have strengths (global control flow and organisational perspective) where Use Cases have some of their limitations. While personalised views have been proposed [GRvdA05], such views are not as detailed as Use Cases and are not sufficient for software development. Therefore, EPCs profit as well from integration with Use Cases.

EPCs represent a business process using two main elements: events and functions. **Events** are symbolising an organisational state. They trigger functions which represent an action performed within the organisation. A **function** alters the organisational state and therefore a function results in a new event which can trigger a new function and so on. To symbolize which function results in which event and which event triggers each function these elements are connected using arcs. For using the boolean operators AND, OR and XOR, connectors can be placed in an EPC. They can be used as splits (one incoming arc and many outgoing arcs) or as joins (many incoming arcs and one outgoing arc). Furthermore, modelling many elements like organisational roles, business objects, other operators etc. have been proposed in several EPC extensions.

### 3.2 Meta Model

EPCs have been formalised by Nüttgens and Rump in [NR02]. There are certain elements an EPC **Process** can consist of: **Events**, **Functions**, **Process Indicators** and **Connectors**. Connectors are divided into **Splits** and **Joins**. Splits may only have one incoming but many outgoing arcs, i.e. associations to other elements while joins may have many incoming but only one outgoing arc. Additionally, functions can be refined by sub-processes.

A simplified meta-model is shown in figure 3. Prominently missing for conserving clarity are the restrictions for the associations, e.g. functions may only be connected to events or connectors which are ultimately followed by events. These restrictions are fully discussed in [NR02].

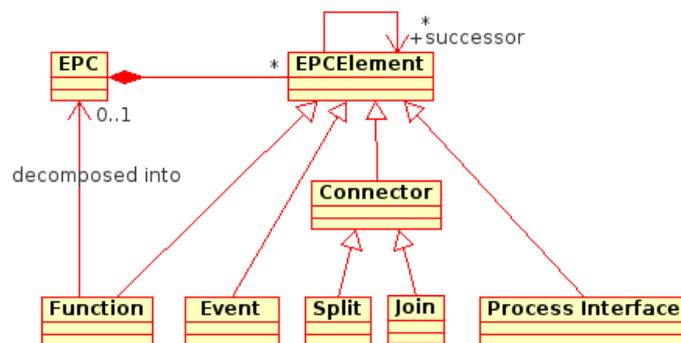


Figure 3: Simplified EPC meta model

## 4 Transformation Guidelines

The transformation of a set of Use Cases to an EPC repository is done in two steps: First, each Use Case is transformed to an EPC resulting in a set of EPCs. In the second step, the EPCs must be joined to a system of connected EPCs representing the whole process. While the first step is straight forward using some rules presented in section 4.2, there are two strategies for joining the EPCs. These strategies are presented in section 4.3.

### 4.1 Use Case Constraints

The transformation approach outlined below will work for Use Case models satisfying some constraints. Most important, our approach will not work with Use Cases having cyclic include and extend relationships.

Furthermore, our approach requires pre- and postconditions to equal literally. Since Use

Cases – including the pre- and postconditions – are written in natural language, no automatic evaluation can be made whether conditions can be satisfied by other means. Formulating conditions in a more formal way would run contrary to the Use Case goal of being understandable both by the stakeholders and the developers. Therefore, conditions should be formulated as small free text fragments. Use Case editors, like the Use Case DODE [Chr06], can offer help in consistently editing Use Cases in this regard by offering already used conditions e.g. in combo boxes. Thereby manual checks and typing errors are eliminated. Additionally, checks for literal similarity of conditions can further improve the Use Case model in this regard.

## 4.2 Use Case Transformation

For transforming Use Cases to EPCs, a set of rules is applied to the set of Use Cases (which are illustrated in figure 4:

1. **Include** and **extend** relationships of Use Cases are removed by copying the scenario and extensions of the referenced Use Case. Thereby, a flat Use Case model without any relationships is derived.
2. **Preconditions** and **triggers** are realised as events since their conditions fulfill the same role as events do. Because all preconditions must be met and the trigger must occur in order to start the Use Case, all events are joined using an AND join in the EPC if this rule results in more than one event. The first step in the main scenario is inserted after the AND join as an EPC function.
3. All steps of the **main scenario** are mapped to a linear control flow in an EPC. Each step is mapped to an EPC function. Functions are connected by using trivial OK-events. The step's actor becomes the role responsible for the created EPC function. Objects normally cannot be easily extracted from the Use Case templates and are therefore not handled by this algorithm.
4. **Success Guarantees** are like the preconditions and triggers conditions concerning a system which are to be achieved. They are mapped to EPC events which are inserted after the last function of the main scenario. Since all guarantees must hold after completion, all events (in case of multiple guarantees) are connected using an AND split.
5. **Minimal Guarantees** are discarded. These guarantees normally represent non-functional requirements which cannot be visualised using EPCs. Since they must be valid before, during and after the Use Case they do not change the system at all.
6. **Extensions** are introduced using an XOR connector. After the preceding function an XOR split is introduced which splits into the OK-event and a start event for each extension. A step with 2 extensions therefore becomes a function followed with an XOR split to 3 paths.

7. All **Extension steps** are handled like steps in the main scenario. **Extensions to extension steps** are handled recursively using these rules.
8. **Jumps** typically occurring from one extension step to a step in the main scenario are realised using XOR joins. A join is introduced before the function representing the step which is the jump target.

Below the algorithm is presented in pseudo code. All possible joins and splits are introduced first. If they are not used anywhere (e.g. by steps) these connectors are removed afterwards. The EPC is supposed to create an element on the fly if one is accessed in the EPC which is non-existent due to forward-jumps from the extensions back into the main scenario in order to make the algorithm simpler:

```
function ConvertUseCaseToEpc(UseCase)
  let EPC = new EPC(UseCase.Name);

  let UseCase = MakeFlatUseCase(UseCase)
  let LastElement = HandleTriggersAndPreconditions(
    UseCase, EPC)
  let LastElement = HandleScenario(
    UseCase.Scenario, LastElement)
  HandleGuarantees(UseCase, EPC, LastElement)
  EPC.RemoveUnnecessaryConnectors()

  return EPC
end function
```

To make the algorithm better understandable, it is split up into subroutines. The triggers and preconditions are processed first:

```
function HandleTriggersAndPreconditions(UseCase, EPC)
  let AndJoin = EPC.CreateAndJoin('TriggerAnd')

  for each C in UseCase.Preconditions
    + UseCase.Triggers
    let Event = EPC.CreateEvent(C.Name)
    Event.ConnectTo(AndJoin)
  next C

  return AndJoin
end function
```

Afterwards, the main scenario is converted. In this step, extensions are handled recursively:

```

function HandleScenario(Scenario, LastElement)
  for each Step in Scenario
    let XorJoin = EPC.CreateXorJoin('Join'+Step.Name)
    LastElement.ConnectTo(XorJoin)
    let Function = EPC.CreateFunction(Step.Name)
    Function.AddRole(Step.Actor)
    XorJoin.ConnectTo(Function)

    let XorSplit = EPC.CreateXorSplit('Split'+Step.Name)

    for each Extension in Step.Extensions
      let Event = EPC.CreateEvent(
        ExtensionStep.Condition)
      XorSplit.ConnectTo(Event)

      HandleScenario(Extension.Scenario, Event)

      if Extension.Jumps then
        Event.ConnectTo(
          EPC.Element['Join'+ExtensionStep.ReturnJump])
      end if
    next Extension
    let LastElement = EPC.CreateEvent('OK ' + Step.Name)
    XorSplit.ConnectTo(LastElement)
  next Step
end function

```

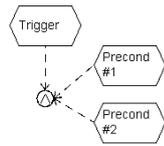
Finally, the success guarantees are appended to the end of the main scenario. Because each Use Case must achieve at least one user goal, each Use Case has at least one success guarantee:

```

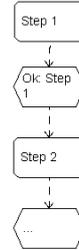
function HandleGuarantees(UseCase, EPC, LastElement)
  let AndSplit =
    EPC.CreateAndSplit('SuccessGuaranteesSplit')
  LastElement.ConnectTo(AndSplit)

  for each SG in UseCase.SuccessGuarantees
    let Event = EPC.CreateEvent(SG.Name)
    AndSplit.ConnectTo(Event)
  next SG
end function

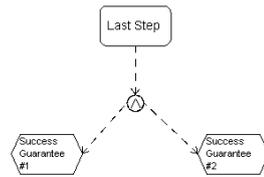
```



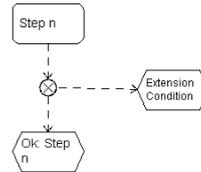
(a) Rule 2: Triggers and Pre-conditions



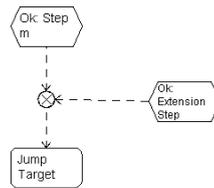
(b) Rule 3 & 7: Steps



(c) Rule 4: Success Guarantees



(d) Rule 6: Extensions



(e) Rule 8: Jumps

Figure 4: Results of Applying Transformation Rules for Use Case attributes to EPCs

### 4.3 Joining EPCs

After transforming each Use Case into an EPC, the transformation must proceed joining the single EPCs to a comprehensive EPC model. For doing this, two options are available: Either a **single, large EPC model** is built or the processes are joined using **process interfaces**.

The Use Cases are connected by using the preconditions and success guarantees. These are expressed as a set of conditions. Both join strategies work by comparing these conditions literally. A precondition can be satisfied by a literally matching success guarantee.

The first strategy joins the EPCs by looking for identically named events and constructing one large EPC. For each event its occurrence as start and end events of EPCs is looked up. All end events are replaced by an OR join, a single event and an AND split. The OR join is connected with the arcs from the removed end events. Afterwards, all start events are removed and their arcs connected to the AND split. The procedure is illustrated in figure 5.

The result of applying this strategy is a single and large EPC model. If it is not too large, the model clearly shows the organisational control flow. However, it is hard to identify the former Use Case model. Furthermore, the model can quickly become too complex. This makes manual rework necessary, e.g. refactoring the EPC using hierarchical decomposition.

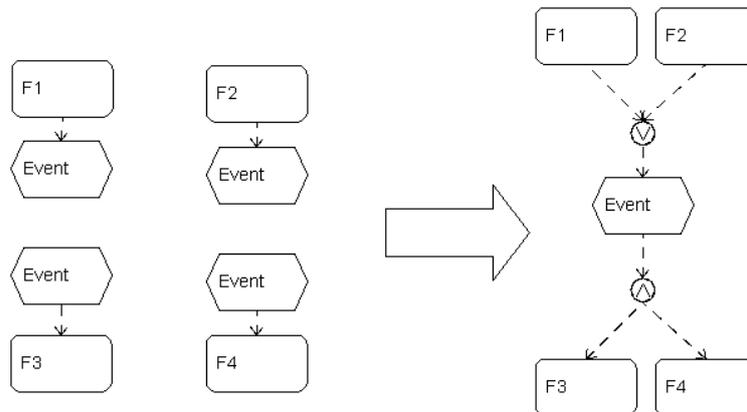


Figure 5: Joining resulting EPCs to one EPC model by Events

The second possibility for joining the EPCs is to leave all the generated EPCs intact by adding process interfaces. Thereby each process remains in an acceptable size. The process interfaces are named after the original Use Cases. This is illustrated in figure 6.

Both strategies can be beneficial in different scenarios: For communicating with isolated stakeholders and users, a small EPC with process interfaces shows only the details necessary for visualizing one Use Case. However, the organisational control flow is not as

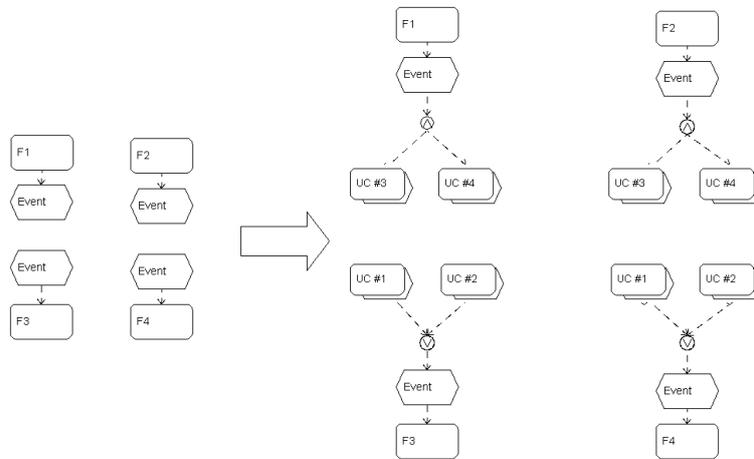


Figure 6: Joining resulting EPCs using Process Interfaces

visible as process and composition designers might like it. The organisational control flow is better visualised by a large model. Since both models are automatically (and therefore with little effort) generated, both can be used for visualising the process to different stakeholders as long as no changes are made which would render the two models inconsistent.

## 5 Prototype Implementation

Based on the outlined transformation rules, a prototype application has been developed [Die06]. The prototype consists of an XSLT stylesheet and a Java GUI front-end. The stylesheet transforms Use Cases stored within XML documents into EPCs stored in EPML [MN05].

Since there is no standardised XML format for storing Use Cases, an XML schema based on the described meta-model has been developed. It can store multiple Use Cases in different sets accompanied with all relevant project information described in the meta model.

The main logic of the transformation tool is contained in the XSLT stylesheet which implements the rules presented in section 4 and joins the EPC using process interfaces. The stylesheet can be applied to a Use Case project and will generate EPML. However, no “nice” layout of the EPCs is done; this task is left to other tools, in this case the Java front-end. The benefit from this approach is that the stylesheet can be integrated into other applications as well, which will need to lay out the Use Cases differently. For example, direct integration into Use Case editors would be beneficial, which can show the EPC for validation purposes in a small area of the screen.

The Java GUI makes using the XSLT stylesheet easy: After the user has opened a Use

Case file, the Use Cases can be opened in a tabular view as illustrated in the left hand side of the screenshot seen in figure 7. If the user chooses to transform a Use Case project, the tool applies the XSLT stylesheet and generates an EPML file. The EPML file is loaded and layouting mechanisms provided by JGraph and extended with own implementations lay out the EPC file. Afterwards the EPCs can be displayed on the right hand side as can be seen in the screenshot. The tool is able to track elements of an Use Case to elements of an EPC: Whenever the user activates a Use Case element which is visible in the EPC, the corresponding EPC element is highlighted.

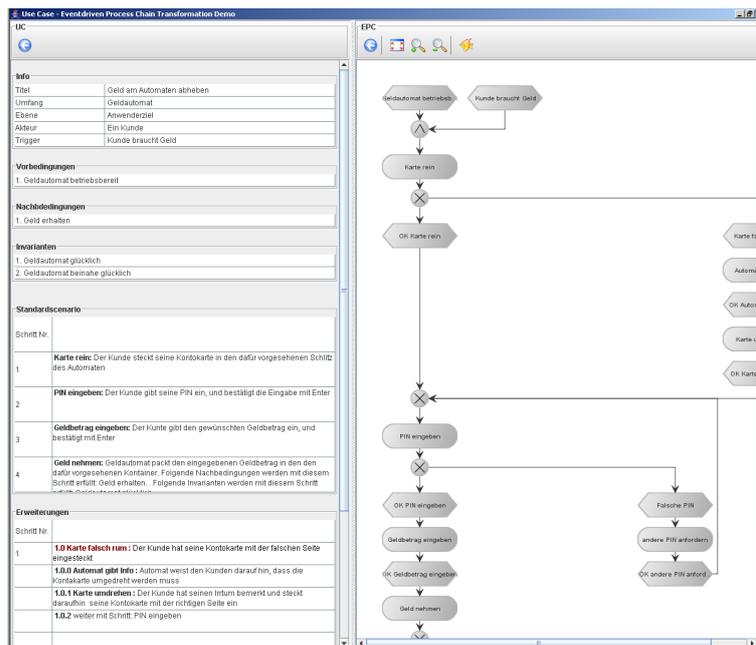


Figure 7: Screenshot of the Prototype Application

## 6 Example Use Cases

To illustrate the transformation, an example from a fictional university is presented in this section. The to be developed software is a system supporting students' theses registration. The simple registration process is documented in four Use Cases as illustrated in figure 8:

1. **Student applies for Thesis:** This Use Case describes the step a student has to make in order to apply for a thesis. The Use Case is triggered by the student's wish to write a thesis. As a result in case of success, the application is submitted to the Academic Examination Office.

Use Case	#1: Student applies for Thesis		Use Case	#2: Academic Examination Office approves Thesis			
Primary Actor	Student		Primary Actor	Student			
Stakeholders	Student: wants to apply easily Secretary (Academic Examination Office): wants easy to use/read forms for further handling registration		Stakeholders	Secretary (Academic Examination Office): wants easy to use/read forms for further handling registration Manager (Academic Examination Office): wants short handling times			
Minimal Guarantees	none		Minimal Guarantees	Student's data are handled according to regulations			
Success Guarantees	Application is submitted		Success Guarantees	Student may write Thesis			
Preconditions	none		Preconditions	none			
Triggers	Student wants to write thesis		Triggers	Application is submitted			
Main Success Scenario	1	Student	fills out form with personal data	Main Success Scenario	1	Secretary	checks if student has 80% of Credit Points
	2	Student	submits form to Academic Examination Office		2	Secretary	approves application
Extensions	none		Extensions	1a If Student has less than 80% of Credit Points then Secretary denies Application			

Use Case	#3: Student selects Topic		Use Case	#4: Supervisor approves Topic			
Primary Actor	Student		Primary Actor	Supervisor			
Stakeholders	Student: wants to have interesting topic		Stakeholders	Supervisor: wants no paperwork Secretary (Academic Examination Office): wants easy to use/read forms for further handling registration			
Minimal Guarantees	none		Minimal Guarantees	none			
Success Guarantees	Student has picked a Topic		Success Guarantees	Student has Topic			
Preconditions	none		Preconditions	Student may write Thesis Student has picked a Topic			
Triggers	Student wants to write thesis		Triggers	none			
Main Success Scenario	1	Student	looks trough list of available topics	Main Success Scenario	...		
	2	Student	chooses most interesting topic		1	Supervisor	hands out Topic
	3	Student	asks Supervisor to get the topic		...		
Extensions	none		Extensions	(left out)			

Figure 8: Use Cases for a Thesis Registration System

2. **Academic Examination Office approves Thesis:** In the next Use Case the Academic Examination Office checks if the student fulfills all prerequisites for writing a thesis. In this case, 80% of Credit Points need to be already earned. If successfully, the application is approved.
3. **Student selects Topic:** If the application is approved, the student has to choose a department and a topic for his or her thesis.
4. **Supervisor approves Topic:** Finally, the supervisor has to approve the topic the student wants to write in.

The Use Cases satisfy the given requirements: For simplification these Use Cases do not have include or extend relationships (which are therefore not cyclic) and all conditions are named consistently.

Therefore, these Use Cases can be transformed using the outlined rules. The first and second Use Cases are connected by the “Application is submitted” event. The second and forth Use Cases are connected by the “Student may write Thesis” event while the third and forth Use Cases are connected by the “Student has picked a Topic” event. The resulting EPC is shown in figure 9.

Use Cases as presented here are easy to write and similar ones have been used internally in a student project. Using the generation capabilities, possible parallel execution paths have been identified which the process participants were not aware of because they were used to the (serialised) process they were part of for years.

## 7 Application Scenarios

The presented technique for transforming Use Cases to EPC models can be applied in many contexts:

- **The original motivation of the transformation was the Derivation of Business Processes for SOA projects from Software Requirements:** Since SOA projects build upon defined business processes which are often not existent prior to the project’s start, they need to be reconstructed from the requirements. If functional requirements have been documented as Use Cases the transformation saves times and effort.
- **Validation of Use Cases:** Generation of business processes can easily show non-matching pre- and post-conditions by visualising the control flow. Thereby naming inconsistencies, missing conditions and missing Use Cases providing needed post-conditions can be detected, which leads to improved Use Cases.
- **Interview Tool for Collecting Business Processes:** Use Cases can be used by business process engineers in order to discover “hidden”, unknown and new business processes. For such Use Cases templates are available, e.g. [Coc05, p. 134]. Using Use Case templates in interviews with some stakeholders and merging them afterwards into business processes can be used as an efficient interview technique. Usage of the described transformation technique can accelerate the mining, examination and interpretation of the results. When using the generation capabilities within the interview itself, they can also be beneficial for visualising the process in order to improve and accelerate feedback as has been done with user interface sketches for software requirements [Vol06].

Possibly, there are other scenarios, but these scenarios alone demonstrate that the presented transformation technique can be handy in many situations.

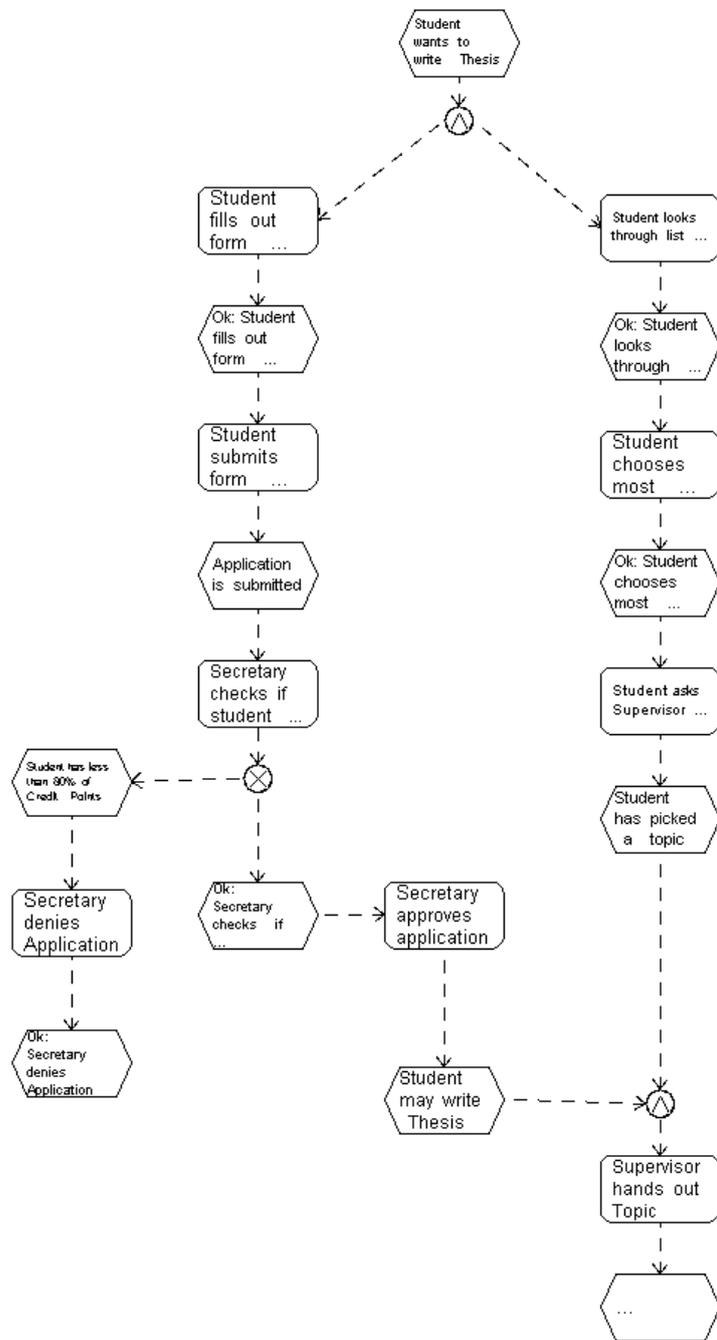


Figure 9: Resulting EPCs from the Use Cases

## 8 Related Work

Profiting from information contained in Use Cases for other models and development phases has always been a goal in software development projects, which often deal with business processes.

Cockburn himself only mentions the possibility of applying Use Cases for deriving business processes. He offers a template in [Coc05] but no rules or advise how to proceed from there.

The field of model-driven development has tried to combine the concept of Use Cases with its models. Instead of tabular and textual descriptions, UML sequence diagrams or similar models are used [Gro04]. A Use Case is consequently denoted in Use Case diagrams and refined in other models thereby eliminating the textual description. This can pose a problem when communicating with non-technical users. A process for UML-based development of business processes is given in [OWS<sup>+</sup>03]. Missing from such approaches are capabilities for expressing control flow between Use Cases and therefore the generation of business process models.

Generation of other models is often inspired by the model-driven community and is often based on UML models. The only way to achieve business process generation is to model the control flow between Use Cases in at least one additional model. With the introduction of Use Case Charts and their formalization [Whi06], it is possible to define control flow dependencies between Use Cases and refine them in UML. From such descriptions other models can be generated, e.g. Hierarchical State Machines [WJ06].

Our approach differs from the Use Case Charts by working with the textual description and impose some limitations on these. Instead of explicitly modelling the control flow in other graphical models, the control flow is expressed by the Use Cases' conditions and triggers. Therefore, we can keep the textual description in order to communicate with stakeholders while still being able to generate business processes from these descriptions.

## 9 Conclusions & Outlook

Within this paper the transformation of Use Cases into EPC models has been shown. This conversion can be automatic although manual editing for improving readability and improving event names may be necessary. This approach can be used to gather requirements for SOA projects with user interaction using well-known techniques which have proofed their usefulness in many projects.

The described transformation rules have been implemented in a prototype application allowing Use Cases to be transferred to EPC models using XSLT.

Besides the initial usage scenario in SOA-based development projects, the usage of EPCs in conjunction with Use Cases allows better overview in projects with a huge number of Use Cases, validation of pre- and post-conditions and interview support for business process modelling.

In contrast to personal business processes as described in [GRvdA05] Use Cases carry more details since they are written with exactly these details in mind and contain information about other stakeholders. Both techniques could be combined if Use Cases are connected to business functions during the generation of the business process. A user could then switch between his or her process and the Use Cases he or she is interested in.

For better integration into development projects it remains an open research question how to enable round-trip engineering. In order to achieve this goal, transformation rules from EPCs to Use Cases are necessary. However, since EPCs can have arbitrary connectors not matching the Use Case meta-model, this conversion is unlikely to be fully automatic. However, being able to do round-trips, it would be possible to map business process changes and related comments, like those gathered using Experience Forums [LS06], not only back to business processes but to Use Cases as well.

The given transformation of Use Cases to EPCs is one step for enabling the integration of proven requirements engineering techniques and their related processes with the field of business processes. Thereby, established requirements techniques can be used efficiently in SOA projects. The given transformation techniques and the prototype application will hopefully serve as a foundation for the open research questions.

## References

- [AB02] Steve Adolph and Paul Bramble. *Patterns for Effective Use Cases*. Addison-Wesley, 1st edition, August 2002.
- [AC03] J. Araujo and P. Coutinho. Identifying Aspectual Use Cases using a Viewpoint-Oriented Requirements Method. In *Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architectural Design, in conjunction with AOSD Conference 2003*, 2003.
- [ACD<sup>+</sup>03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *Business Process Execution Language for Web Services Version 1.1*, May 2003.
- [Ale03] Ian Alexander. Misuse Cases help to elicit non-functional Requirements. *Computing & Control Engineering Journal*, 14(1):40–45, February 2003.
- [Chr06] Christian Chrisp. Konzept und Werkzeug zur erfahrungsbasierten Erstellung von Use Cases. Master's thesis, Leibniz Universität Hannover, October 2006.
- [Coc05] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 14th edition, August 2005.
- [Die06] Dimitri Diegel. Konzept zur Verknüpfung von Use Cases mit ereignisgesteuerten Prozessketten. Master's thesis, Leibniz Universität Hannover, 2006.
- [EMPR05] Bill Eidson, Jonathan Maron, Greg Pavlik, and Rajesh Raheja. SOA and the Future of Application Development. In Jen-Yao Chung, George Feuerlich, and Jim Webber, editors, *Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05)*, pages 1–8, 2005.

- [Gro04] Object Management Group. Unified Modeling Language: Superstructure. WWW: <http://www.omg.org/cgi-bin/doc?formal/05-07-04>, 2004.
- [GRvdA05] Florian Gottschalk, Michael Rosemann, and Wil M.P. van der Aalst. My own process: Providing dedicated views on EPCs. In Markus Nüttgens and Frank J. Rump, editors, *EPK 2005 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, pages 156–175, 2005.
- [LLSG06] Daniel Lübke, Tim Lüecke, Kurt Schneider, and Jorge Marx Gómez. Using EPCs for Model-Driven Development of Business Applications. In Franz Lehner, Holger Nösekabel, and Peter Kleinschmidt, editors, *Multikonferenz Wirtschaftsinformatik 2006*, volume 2, pages 265–280. GITO Verlag, 2006.
- [LS06] Daniel Lübke and Kurt Schneider. Leveraging Feedback on Processes in SOA Projects. In *Proceedings of the EuroSPI 2006*, 2006.
- [MN05] Jan Mendling and Markus Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). *Information Systems and e-Business Management (ISeB)*, 4(3):245–263, July 2005.
- [NR02] Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In Jörg Desel and Mathias Weske, editors, *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen - Promise 2002*, LNI, pages 64–77. Gesellschaft für Informatik, October 2002.
- [OWS<sup>+</sup>03] Bernd Oestereich, Christian Weiss, Claudia Schröder, Tim Weilkens, and Alexander Lenhard. *Objektorientierte Geschäftsprozessmodellierung mit der UML*. d.punkt Verlag, 2003.
- [Vol06] Carl Volhardt. Werkzeug zur Unterstützung von Interviews in der Prozessmodellierung. Bachelor's Thesis; WWW: [http://www.se.uni-hannover.de/documents/studthesis/BSc/Carl\\_Volhard-Werkzeug\\_zur\\_Unterstuetzung\\_von\\_Interviews\\_in\\_der\\_Prozessmodellierung.pdf](http://www.se.uni-hannover.de/documents/studthesis/BSc/Carl_Volhard-Werkzeug_zur_Unterstuetzung_von_Interviews_in_der_Prozessmodellierung.pdf), 2006.
- [Whi06] Jon Whittle. A Formal Semantics of Use Case Charts. Technical Report ISE-TR-06-02, George Mason University, <http://www.ise.gmu.edu/techrep>, 2006.
- [WJ06] Jon Whittle and Praveen K. Jayaraman. Generating Hierarchical State Machines from Use Cases. In Martin Glinz and Robyn Lutz, editors, *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 19–28. IEEE Computer Society, 2006.
- [ZM05] Jörg Ziemann and Jan Mendling. EPC-based Modelling of BPEL Processes: A Pragmatic Transformation Approach. In *Proceedings of the 7th International Conference "Modern Information Technology in the Innovation Processes of the Industrial Enterprises" (MITIP 2005)*, 2005.

# Verifying Properties of (Timed) Event Driven Process Chains by Transformation to Hybrid Automata

Stefan Denne\*

German Research Center for Artificial Intelligence  
Stuhlsatzenhausweg 3  
66123 Saarbrücken

**Abstract:** Event-driven Process Chains (EPCs) are a commonly used modelling technique for design and documentation of business processes. Although EPCs have an easy-to-understand notation, specifying entire information systems leads to rather large and complex models. Questions like for instance the termination of a process (within some given time)—easy to answer for small EPCs—can hardly be answered for those models. Nevertheless, questions like these can be vital for the execution of the described business processes. Whereas simulation might be able to give a hint on whether the process terminates, only verification can give such guarantees.

In this paper we introduce a method to verify properties of what we call *Timed EPCs* (EPCs annotated with time attributes). We transform Timed EPC to (hybrid) automata and thereby define EPCs formally. Based on the formal definition, properties of EPCs (like e. g. is the ending event always reached within 20 time units) can be verified by transforming these properties to corresponding properties of the resulting automata. The transformation of EPCs and properties works fully automatic. The ultimate verification takes place in utilising commonly available verification tools.

## 1 Introduction and Overview

Verification is a method to gain assertions on a system's behaviour. As for EPCs, verification is rarely used. And if at all, with a focus on the correctness of the EPCs. In our approach verification enables the user to get information about the validity of the design of the modelled EPC. Shortcomings can be revealed, quality as well as performance can be assured at a desired level. With *user defined properties* in the sense of Rump [Rum98] it becomes possible for the user to prove assertions 'the account will always be checked before any withdrawal is granted'. If the proof fails, a counter example in form of a path in the EPC can be provided. Since we focus on *Timed EPC* the user will even be able to verify time properties as, for instance, 'the process chain always terminates within 30 seconds'.

In this paper we will introduce a method for the verification of properties of Timed EPCs. Verification can be done in two ways. One is to define semantics on EPCs directly (as it has been done by Nüttgens and Rump [NR02]). The second possibility is to utilise

---

\*stefan.denne@dfki.de

transformation to models that are formally defined and for which it is well known how verification is performed (even supported by tools). We chose the latter approach and give a formal transformation of EPCs to hybrid automata. (Thereby formal semantics of EPCs are defined indirectly.)

This paper will essentially concentrate on the basic transformation ideas even if this means to somewhat restrict the EPCs under consideration. That is to say we define, in terms of a context free grammar, a suitable subset of EPCs which covers a broad variety of models without being meant to be comprehensive.

This work has been done formally (cp. [Den06]) but for this paper we will abstract from technical details and illustrate the transformation of EPCs and corresponding properties using figures and examples. By the transfer of Formal Methods to the field of business processes and the use of ready-to-go tools, the verification—even of timed properties—is possible with reasonable effort.

This paper is organised as follows: we first describe the restriction of the EPCs we use and introduce the aspects of time we will consider. Then we formulate and classify the properties we can verify by the transformation to automata. Afterwards we introduce the grammar that we use for the representation of EPC. In the succeeding sections, we define what we mean by automata. The transformation is described in several steps according to the grammars used. We will not present the technical details but the overall plot using examples to illustrate the transformation process and the results. We also show how to describe and transform the properties of the EPCs. In Section 4 we give a brief glance at some features not presented in this paper. The last but one section gives a brief overview about related work. We conclude the paper with a summary about what has been achieved and what can be done as future work.

## **2 Modelling**

In this section, we introduce the different modelling techniques we use in our approach. First, the EPCs that will be transformed are characterised. Then we introduce the automata that are the target of the transformation.

### **2.1 Representation of EPCs**

In the representation of the EPCs we use we try to be as close as possible to commonly used EPCs. Nevertheless, we will use a restricted form of EPCs in order to minimise the number of transformation rules necessary. We will not deal with open ends, this means, concurrent chains join in any case. Likewise, we do not have joins without splits before, that means we only have one starting event. Additionally the inner structure of concurrent and alternative chains will not allow two (or a multiplicity of two) elements (functions or events). Further we will only consider two different cases of connector combinations, namely XOR–XOR and AND–AND connector pairs (we do not consider OR-OR connector pairs). We call

EPCs with identical connector combinations *regular*.

Despite these restrictions the grammar introduced below fulfils the informal definition of EPC given by [KNS92], and [HKS93].

### 2.1.1 The use of Time within EPCs

Initially, time has intentionally not been considered in EPC. By leaving out time, modelling with EPCs does not have to consider problems that time involves. For example, simultaneous functions and events do not occur,<sup>1</sup> since a (temporal) ‘before’ and ‘after’ does not exist.

The the simulation facility integrated in the ARIS Toolset<sup>2</sup> uses given time attributes to calculate process ratios, for example the minimal and the maximal time to pass from the starting to the ending event of the chain. Functions can be annotated with time slots describing a ‘stage’ in the progress of a function. *Setup time* is the amount of time required to get ready to execute a function, *Process time* is the time needed for its execution.<sup>3</sup> Looking for example at a machine that heats metal before bending it, one can identify these two stages; setup time: heating the metal and process time: bending it. The time attributes are related to each other. For instance ‘setup time’ lies before ‘process time’.<sup>4</sup>

We call EPCs that have time attributes *Timed Event-Driven Process Chains*.

### 2.1.2 Properties of EPCs

Properties of an EPC can be described as situations (or as sequences of situations) that—in terms of events and functions—occur or do not occur in the EPC.

Imagine a shop selling books. Before any order is processed it shall be checked whether the book is in the stock to guarantee a delivery date. In order to guarantee this check, an appropriate property must hold for the model. In other words: if one can verify that such a property holds, the delivery date can be guaranteed. If the model is small, this might be easy to check, but if the model is complex or it is large, it might not be obvious that this check is performed in all cases.

As in Bernard et al. ([BBF<sup>+</sup>01, 79, 83, 91, 103]), properties can be classified in *Reachability properties*: is a situation<sup>5</sup> reachable under some circumstances? *Safety properties*: will an undesirable situation never occur (under certain condition)? *Liveness properties*: will a

---

<sup>1</sup>If an event  $E_1$  occurs *and* an event  $E_2$  occurs, this ‘and’ is a logical ‘and’.

<sup>2</sup>The ARIS Toolset is part of the ARIS Design Platform. (IDS Scheer AG)

<sup>3</sup>Other time attributes are: *Waiting time* — the time that needs to pass before a function can be started (for example because a machine is occupied before) and *Transfer time* — the time that is needed to enter a function (for instance to pass some material to the current function). ‘Transfer time’ is an attribute that is assigned on arcs leading to a function. Thus, we interpret ‘transfer time’ to belong to the (following) function.

<sup>4</sup>‘Transfer time’ lies before ‘waiting time’, ‘waiting time’ lies before ‘setup time’, ‘setup time’ before ‘process time’. ‘Waiting time’, ‘setup time’ and ‘process time’ are associated to a function.

<sup>5</sup>A situation can be a (boolean) combination of some events or functions.

situation ultimately occur? *Fairness properties*: will a situation, under certain conditions occur (or fail to occur) infinitely often?

In general, properties of EPCs describe a behaviour that corresponds to certain sequences of events and functions.

Accordingly to Rump ([Rum98, 120]) there are typical types of (user defined) properties concerning elements (events and functions) of EPCs. The elements  $E_1$  and  $E_2$  can either be events or functions.

1. There is a concrete sequence represented by the EPC-schema, in which  $E_1$  is activated.
2.  $E_1$  is activated in any possible sequence.
3. If  $E_1$  is activated in a sequence, then  $E_2$  will sometime be activated.
4.  $E_2$  will never be activated, after  $E_1$  has been activated.
5. In a sequence of the business process  $E_1$  and  $E_2$  will never be activated both, irrespective of their order
6. The element  $E_1$  will only be activated once in one sequence.

Note that the first two questions are reachability properties and the following are liveness properties. The term ‘activated’ corresponds to ‘reachable’ in our terminology.

In general, temporal logics are eligible to express properties of that kind. We use a CTL-like<sup>6</sup> temporal logic to describe these properties (see Section 3.3).

For Timed EPCs, the properties above can be enriched by assertions about time. One may ask for example: Is there a sequence where a situation  $E_1$  is reached in less than five time units? Or the property might be:  $E_2$  is always reached within 20 time units. (See Section 3.5).

A formal description of CTL and TCTL is presented in [Den06].

Another type of property is the maximal and/or minimal time that is needed between some elements (events or functions) within the EPC. In comparison to the properties described so far, a question about the validity does not lead to a ‘yes’ or ‘no’, but to a parameter that is the maximal or minimal time. For instance: ‘There is a sequence, where  $E_1$  is reached in less than  $P$  time units.’ The parameter  $P$  then represents the maximum in focus.

### 2.1.3 A Grammar for Timed EPCs

The EPCs we use are syntactically defined by a context free grammar. Context free grammars define (context free) languages using a set of derivation rules. This definition is constructive. By the application of the rules of the grammar a sentence of the language may be generated.

---

<sup>6</sup>CTL is an abbreviation for Computation Tree Logic

The advantage of the constructive approach is that the rules of the grammar describe how a proper EPC looks like. Only such EPCs can be constructed that were defined by the grammar.

In this paper we focus on Timed EPCs. Every time attribute introduced in Section 2.1.1 represents a duration and is associated to a function. We introduce two different time attributes only: ‘setup time’ and ‘processing time’, as we only show how this is done in principle. These time attributes appear in a fixed order: ‘setup time’ lies before ‘processing time’. We will keep this order in the grammar. Only ‘processing time’ is mandatory, therefore we need two rules; one that contains ‘processing time’ only (rule TS10) and a second that contains both, ‘set-up time’ and ‘processing time’ (rule TS9).

Of course there are many different grammars possible that have other or additional rules that allow the construction of EPCs where different elements (like time attributes, organisational units) are possible<sup>7</sup>

$\mathbf{G}_{TS} = (N_{TS}, T_{TS}, R_{TS}, \text{EPC})$  with  $N_{TS} = \{\text{EPC}, \text{E-PART}, \text{F-PART}, \text{T-ATTR}\}$ ,  $T_{TS} = \{\text{Event}, \text{Function}, \text{AND}, \text{XOR}, (, ), \text{Setuptime}, \text{Processtime}, \rightarrow\}$  and

$R_{TS} = \{\text{EPC}$	$ ::= \text{Event} \rightarrow \text{F-PART} \rightarrow \text{Event}$	(TS1)
$\text{E-PART}$	$ ::= \text{E-PART} \rightarrow \text{F-PART} \rightarrow \text{E-PART}$	(TS2)
$\text{F-PART}$	$ ::= \text{F-PART} \rightarrow \text{E-PART} \rightarrow \text{F-PART}$	(TS3)
$\text{E-PART}$	$ ::= \text{Event}$	(TS4)
$\text{F-PART}$	$ ::= \text{Function (T-ATTR)}$	(TS5)
$\text{E-PART}$	$ ::= \text{AND (E-PART,E-PART)}$	(TS6)
$\text{F-PART}$	$ ::= \text{AND (F-PART,F-PART)}$	(TS7)
$\text{E-PART}$	$ ::= \text{XOR (E-PART,E-PART)}$	(TS8)
$\text{T-ATTR}$	$ ::= \text{Setuptime, Processtime}$	(TS9)
$\text{T-ATTR}$	$ ::= \text{Processtime}$	(TS10)

The rules TS1 to TS3 allow the construction of linear chains of an arbitrary length. (TS1 assures that each chain will start and end by an event.) Concurrent branches are introduced by the rules TS6 and TS7, TS8 constructs alternative branches. Events are derived by TS4. The rule TS5 extends a function with a list of time attributes.

## 2.2 Automata

Automata are finite graphs whose nodes correspond to global states. Such global states represent some sort of general observational situation, as, for instance, the heater is on or the the heater is off. Automata change there states by travelling along a transition that connects two states.

In this section we only give a informal introduction to automata as far as they were used

<sup>7</sup>In [Den06] a grammar including organisational units and time attributes is defined and used for the transformation into timed automata.

for the transformation. Formal details can be found in [Den06].

### 2.2.1 Communicating Automata

We use communicating automata to model a system by modelling its components. Each component is represented by an automaton and can be rather simple. But as the components are able to communicate with each other the behaviour of one component can depend on the behaviour of some other. Thus, the behaviour of the system is the result of the composition of the behaviours of the components. Consequently, a system with simple components can have a quite complicate behaviour.

We prefer a modular approach to keep the transformation straightforward. Each of the rules of the context free grammars will be transformed into a small set of automata. All sets are composed to a system representing the behaviour of the original EPC.

### 2.2.2 Composition of Communicating Automata

When building systems out of components we have to define if and how they interact. If there is no interaction at all the behaviour of the resulting system is the combination of the independent behaviours of the components and, as the behaviour of each component is determined by the set of states, the system behaviour is the *Cartesian product* of the states of the automata.

Automata can interact by synchronising their discrete steps. The synchronised automata cannot perform a step without the other performing a corresponding step as well. With the system behaviour being a combination of the components, the number of system states is reduced compared to the non-synchronised system. If two automata  $\mathcal{A}$  and  $\mathcal{B}$  share the same synchronisation label  $l$ , then a transition of  $\mathcal{A}$  labelled with  $l!$  (or  $l?$ ) must be accompanied by a transition of  $\mathcal{B}$  labelled with  $l?$  ( $l!$ ) too.

Figure 1 shows two automata that synchronise via a corresponding synchronisation label  $l$ . Both of them start concurrently in their initial location ( $n_1$  and  $n_5$ ). While automaton  $\mathcal{A}$  can travel from location  $n_1$  to location  $n_2$ ,  $\mathcal{B}$  being in  $n_5$  cannot leave because it has to synchronise with some partner having a corresponding label ( $l_1!$ ). So,  $\mathcal{A}$  being in location  $n_2$  and  $\mathcal{B}$  in  $n_5$ , the two automata can only perform a common step leading from  $n_2$  to  $n_3$  and from  $n_5$  to  $n_6$ . The subsequent transitions can be taken independently.

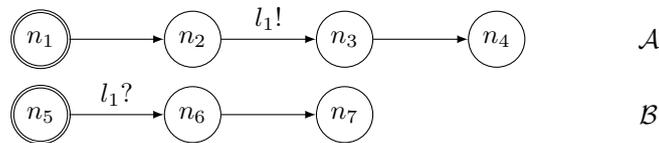


Figure 1: Two synchronising automata

Because we use binary synchronisation (one automaton sending a signal with one —out of several other—receiving it), only two automata out of the set (of automata) can syn-

chronise by a *compatible label*<sup>8</sup>. As our approach requires to synchronise more than two automata we use label sets. When travelling along a transition all the labels in the label sets have to be synchronised regardless of the number of automata to be synchronised.

Here a brief example for compatible label sets: the three label sets  $L_1 = \{l!\}$ ,  $L_2 = \{l?, m!\}$  and  $L_3 = \{m?\}$  are compatible, because every label in the different label sets has a compatible partner.

The composition of two automata leads to a new automaton. Each location of the composed automaton is a tuple combination of two locations of the original automata. The starting location of the automaton  $\mathcal{C}$   $\langle n_1, n_5 \rangle$  is a combination of the automata  $\mathcal{A}$  and  $\mathcal{B}$ . The following two steps of  $\mathcal{C}$  are the only possible steps the combined composed system can perform. (The first step is restricted as only the automaton  $\mathcal{A}$  can perform a step, the second step is the one that is synchronised via the label  $l$ ). The labels are no longer required after the composition.

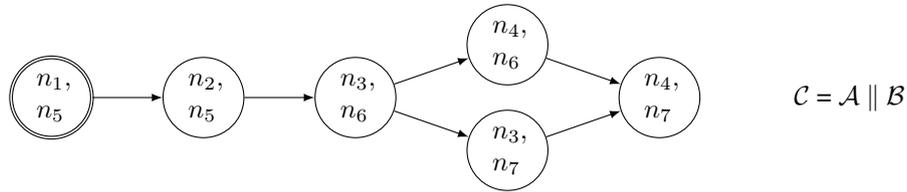


Figure 2: Composition of two communicating automata

### 2.2.3 Timed Automata

Timed Automata (a special kind of hybrid automata) are an extension of the communicating automata. The global states of communicating automata were discrete. That means after the entry into a state no further action happens while the automaton remains within the state. Within the global states of timed automata some continuous activity takes place. A *clock* (or chronometer or watch) defined in a location, rises continuously (by a rate of 1). Depending on the value of that clock the automaton can change from one state to the other travelling along a transition. These transitions are usually guarded with some constraint formula<sup>9</sup>, that is required to hold if the transition is supposed to be taken. Similarly, nodes have some attached constraint formula that describes an invariant for this very node, this means, some property that has to be true while the system resides in this node. The dynamics of the systems behaviour, on the other hand, is given by a description of how the data changes with time. Additionally, transitions are annotated with a general assignment that

<sup>8</sup>Compatible labels have the same name and one has a exclamation mark and the other a question mark. The exclamation mark has the meaning of ‘sending’ and the question mark the meaning of ‘receiving’. If there is more than a question mark as a possible partner, the exclamation mark can arbitrarily synchronise with one of the question marks available.

<sup>9</sup>A constraint formula is for instance a inequality that limits a *clock* to an certain value like  $c \leq 5$ .

is responsible for the discrete action to be performed by taking the transition (For example resetting a clock to zero).

Timed automata can behave in two ways: (1) by the change of one state into another (by passing a transition) or (2) by the passing of time, which changes the value of a clock.

Figure 3 shows an example of an timed automaton. The automaton has a clock  $t$  that is set to zero when passing the transition  $\langle n_1, n_2 \rangle$ . The invariant  $t \leq 5$  forces that the location has to be left when the clock has reached 5 time units and the guard  $t \geq 4$  hinders to leave the location before the clock  $t$  is at least equal to 4 time units. Thus the location  $n_2$  can be left when the clock  $t$  is at least 4 and must be left as soon as it is 5. The automaton does not synchronise with other automata therefore the label set is empty and the transitions do not have labels.

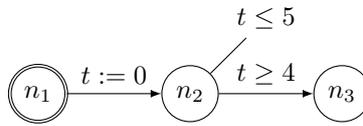


Figure 3: A timed automaton

#### 2.2.4 Composition of Timed Automata

The composition of timed automata is similar to the composition of communicating automata. The transitions with labels are the synchronisation points of the automata. For the composed location the invariant is the conjunction of the invariants of the location of the original automata. In order that the new transition (that leads from the actual tuple of the composed automaton to the following tuple) can be passed all guards of the transitions of the original automata (for that step) have to be true. The actions for the new transition are just the set of all actions of the original transitions (that would have been taken).

### 3 Transformation

The transformation of Event-driven Process Chains to automata is defined recursively on the structure of EPCs by transformation rules. This structure is immediately introduced by the grammar rules that allow to construct EPCs. The processing of a transformation rule leads to a set of new automata or modifies an existing set of automata in order to model the behaviour of the EPC.

Basically, functions are transformed into automata locations and events are transformed into transitions. We keep information about which function or event is related to which location or transition by a function.

We will not give the formal transformation rules and technical details of the transformation here, they can be found in [Den06].

### 3.1 Transformation of Linear Sequential Chains

To illustrate how the transformation works, we first show how sequential chains are transformed. These chains are constructed using the rules TS1 to TS5 of the grammar  $\mathbf{G}_{TS}$ .

In the following, the transformation rule for the transformation of TS1 will be presented in detail and an example will show the complete transformation of a sequential chain. The examples used will have no time attributes<sup>10</sup>.

We assume that the set of automata  $\mathfrak{B}$  is already transformed (see Figure 4 (1)). The dashed arrow symbolises that the structure of the transformed set is not relevant (and not available) for the transformation of the current transformation step.

The transformation rule for TS1 creates two automata (each of them in a created set) that represent the starting and ending event and modifies the existing automata in a way that the sets are 'executed' linearly after each other. This is done by inserting compatible synchronisation labels (Figure 4 (2)).

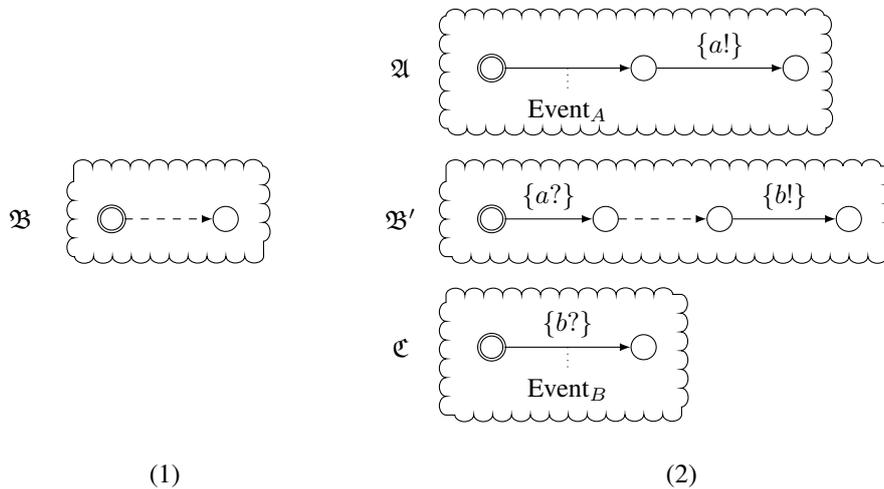


Figure 4: Sketch of the transformation of the rule TS1

EXAMPLE 1 (TRANSFORMATION OF A SIMPLE SEQUENCE) As an example we transform the following simple sequence (Figure 5) into automata.



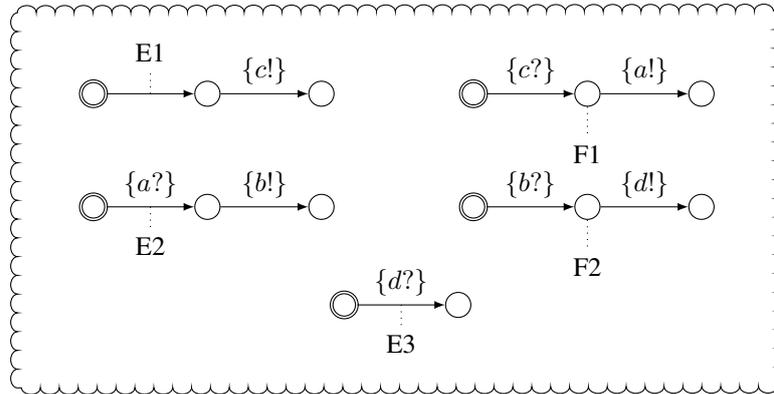
Figure 5: A simple linear sequence

The following sequence shows how the example is constructed using the rules of the grammar  $\mathbf{G}_{TS}$ .

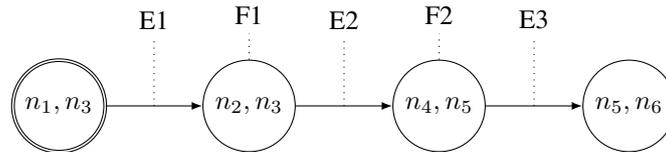
<sup>10</sup>Formally this means that instead of the rule TS5 the following rule is used: F-PART ::= Function.

$EPC \Rightarrow E_1 \rightarrow F\text{-PART}_1 \rightarrow E_3$  (rule TS1)  
 $F\text{-PART}_1 \Rightarrow F\text{-PART}_2 \rightarrow E\text{-PART}_1 \rightarrow F\text{-PART}_3$  (TS3)  
 $F\text{-PART}_2 \Rightarrow F_1$  (TS5)     $E\text{-PART}_1 \Rightarrow E_2$  (TS4)     $F\text{-PART}_2 \Rightarrow F_2$  (TS5)

The transformed set of automata looks as follows:



Composing the set of automata, we get the following automaton:



The sequence of referenced events and functions corresponds to the sequence of the original EPC (of Figure 5).

### 3.2 Transformation of Concurrent and Alternative Chains

Concurrent and alternative chains occur when EPCs have AND and XOR branches. For the transformation of these branches we introduce the concept of *Schedulers*. A scheduler is a regular automaton, but its locations and transitions do not have a reference to any event or function. Schedulers control other automata. A scheduler synchronises the starting and the ending of the ‘underlying’ automata. In the case of concurrent chains the first and the last location are to be synchronised by the scheduler; on alternative chains the scheduler synchronises but one chain only.

Two concurrent branches of an EPC have to be started and ended synchronously. This behaviour is transferred to automata. The two sets of automata are controlled with a scheduler that synchronises both chains. Using two label sets  $\{a!, c!\}$  and  $\{b!, d!\}$  the scheduler

$S_{\text{AND}}$  synchronises the resulting automata sets each representing the two branches. The two scheduled sets  $\mathfrak{A}$  and  $\mathfrak{B}$  have correspondent labels. (Figure 6).

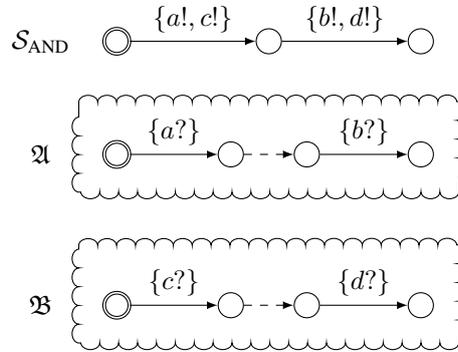


Figure 6: The scheduling of an AND-part

Each of the XOR-branches is an alternative path within the EPC. So, the scheduler must alternatively synchronise with one of the transformed sets representing a branch of the original EPC.

Let us suppose the scheduler has the label  $a!$  at the start (and  $b!$  at the end) and each of the sets ( $\mathfrak{A}$ ,  $\mathfrak{B}$ ) has the same label  $a?$  (and  $b?$ ). Then there is only one compatible pair  $a!$  and  $a?$  at a time. Hence only one of the sets will be synchronised with the scheduler  $S_{\text{XOR}}$ . A sketch of the transformation is shown in Figure 7.

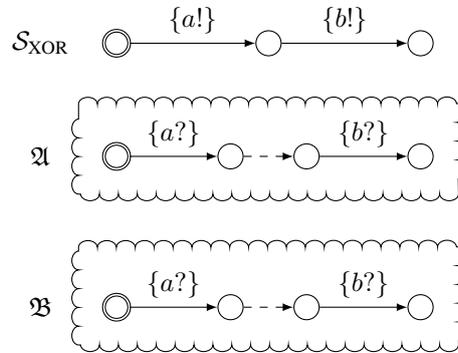


Figure 7: The scheduling of an XOR-part

**EXAMPLE 2 (TRANSFORMATION OF CONCURRENT AND ALTERNATIVE CHAINS)** Given is the EPC shown in figure 8. We omit the details of the transformation here and show the resulting set of automata in Figure 9. The named locations were used to show the transformation of properties (Section 3.3). The composition of the resulting automata is depicted

in Figure 10. We omit the details but show the reference to the original EPC. The original EPC ‘reappears’ when travelling along the locations and transitions.

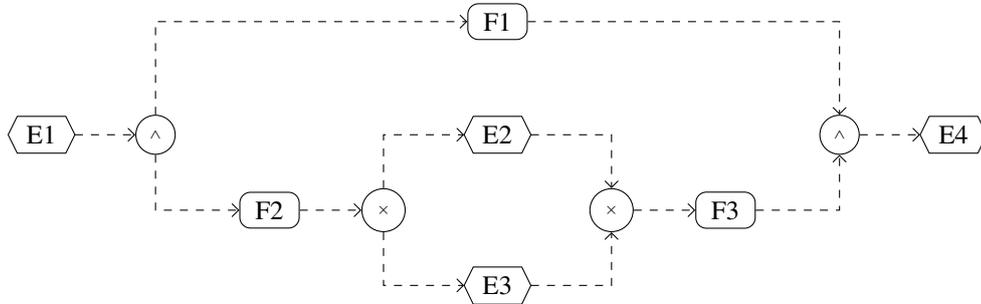


Figure 8: Example 2 — Concurrent Chains

### 3.3 Transformation of Properties of EPCs

In 2.1.2 we have raised several kinds of properties. With respect to EPCs these questions can be expressed in CTL<sup>11</sup>. Here an example.

EXAMPLE 3 (EXPRESSING PROPERTIES OF EPCs USING CTL) The properties described in this example refer to the EPC of Figure 8.

	Description	CTL-formula
(1)	Is the event $E_3$ reachable?	$EF E_3$
(2)	Is the (end) event $E_4$ always reached?	$AF E_4$
(3)	If the (start) event $E_1$ is reached then inevitably the (end) event $E_4$ is reached too.	$AG(E_1 \Rightarrow AF(E_4))$
(4)	Either $E_2$ or $E_3$ are reached (never both)	$(AG((E_1 \Rightarrow AG(\neg E_2)) \wedge AG(E_2 \Rightarrow AG(\neg E_1))))$

Intuitively  $AF$ ,  $EF$ ,  $AG$ ,  $EG$ ,  $AU$  and  $EU$ , mean “inevitably”, “possibly”, “always”, “possibly always”, “inevitably until” and “possibly until”.

As CTL is also used for describing the properties of automata, the transformation can easily be done. Recall, that events and functions are related to transitions and locations. Using that relation, a property of an EPC is transformed directly to a property of the resulting automata.

<sup>11</sup>Here, we use a CTL-like language since we do not use all temporal qualifiers.

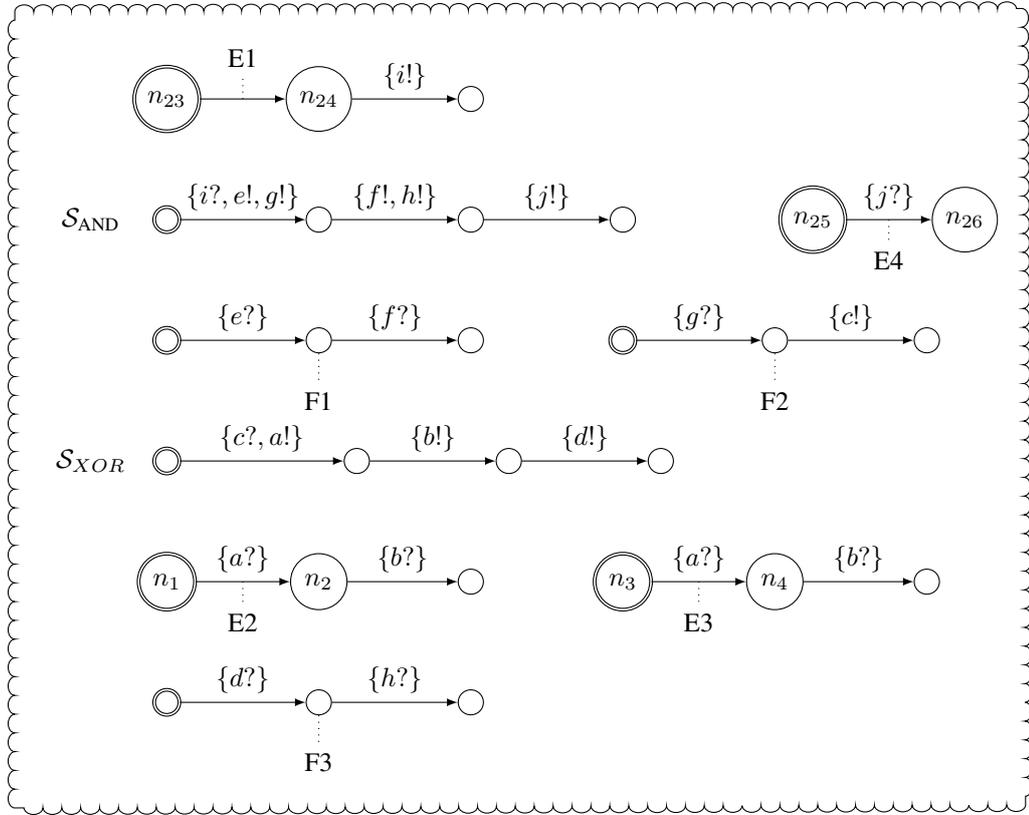


Figure 9: Transformed set of automata of Example 2

Then this property can be proved or rejected (with a counterexample) using a model checker like UPPAAL ([BLL<sup>+</sup>95]) or HyTech ([HH95]).

The properties above are transformed to properties referring to transitions of the transformed set of automata:

- (1)  $EF \langle n_3, n_4 \rangle$
- (2)  $AF \langle n_{25}, n_{26} \rangle$
- (3)  $AG(\langle n_{23}, n_{24} \rangle \Rightarrow AF(\langle n_{25}, n_{26} \rangle))$
- (4)  $(AG(\langle n_1, n_2 \rangle \Rightarrow AG(\neg \langle n_3, n_4 \rangle)) \wedge AG(\langle n_3, n_4 \rangle \Rightarrow AG(\neg \langle n_1, n_2 \rangle)))$

### 3.4 Transformation of Timed EPCs

In EPCs the time we are dealing with is represented by attributes attached to functions.

Timed automata use (reset) clocks, invariants and guards to deal with time. The time

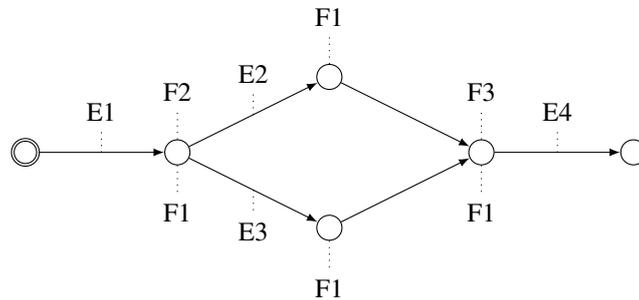


Figure 10: Composed automaton of Example 2

spent in each location can be restricted by defining how long to stay and when to leave a location.

For the transformation of Timed EPC we use so-called *Urgent Transitions*. In general, time can pass in a location when no time restriction is given. If no invariant forces to leave a location it is possible to stay there infinitely (long). It is not mandatory to take a transition. Urgent transitions change this behaviour. An urgent transition must be taken 'as soon as possible'. This means, if a guard of an urgent transition is true (what is the case if there is no guard given explicitly) it must be taken immediately. Thus, no time will ever pass in a location with an urgent transition if no time restrictions are given.

**Transformation of Functions** Figure 11 (1) shows the transformation of a function F1 that has no time attributes. The transformation of such a function leads to (a set of automata with) a single automaton having two locations and one transition. The second location is associated with the corresponding function.

If the function F1 has a processing time  $p$  of 5 time units a clock  $c$  is inserted. The clock is set to 0 when entering the location  $n$  (by the assign  $c := 0$ ). A guard (on an inserted transition) hinders the location  $n$  to be left until 5 time units passed. An invariant forces that the location  $n$  is left when more than 5 time units passed (see Figure 11 (2)).

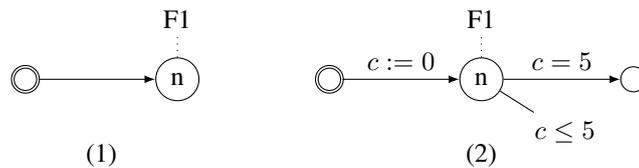


Figure 11: Changes of the transformation when a has got a time attribute.

**Transformation of Events** There is no need to change the transformation of events. (As we now use urgent transitions no time will pass in location that are not guarded.)

Here is an example to show how an EPC with ‘setup-time’ ( $s$ ) and ‘process time’ ( $p$ ) is transformed.

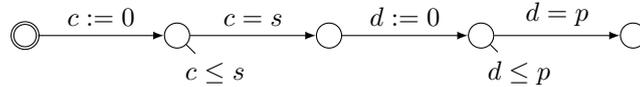


Figure 12: Transformation of ‘setup-time’ and ‘process time’

### 3.5 Transformation of Properties of Timed EPCs

A so-called *Timed Property* is a normal property provided with additional time constraints. For example: ‘ $E_4$  is inevitably reached and the clock  $t$  is greater or equal to 20’. Timed properties can be expressed using TCTL. In TCTL this property is expressed as:  $AF(E_4 \wedge t \geq 20)$ .  $t \geq 20$  is a so-called *Constraint Formula*<sup>12</sup>

The clock that is referred to, is not part of the system modelled. It can be imagined as a ‘global’ clock that is started at the beginning of the EPC. The time spent in the functions of the EPC is then sum up in this ‘virtual’ clock.

Basically, this is also done when a timed property is transformed: A new clock is introduced at the ‘very beginning’ and reset. The ‘very beginning’ is the transition that corresponds to the transformed starting event. (If the property consists of several clocks each of them must be inserted analogously.) After this preparatory work, the property can be translated in terms of automata.

#### Minimum and Maximum Durations

Another kind of properties proposed are minimum and maximum durations. These durations refer to a selected starting and ending element in the EPC. In general, it is interesting to know, how long an entire business process lasts at least and at most. We will consider the starting event  $E_{\text{start}}$  and ending event  $E_{\text{end}}$  of the EPC for explaining the method.

First, a new automaton  $\mathcal{A}$  with three locations and two transitions is created. This automaton is synchronised with the (already transformed) set of automata. The first transition is synchronised with the transformation that is associated with  $E_{\text{start}}$ . The last transformation is synchronised with the transformation that is associated with  $E_{\text{end}}$ . A new clock is reset travelling along the first transition. Figure 13 shows an example of a suchlike automaton, having a clock  $t$  and two labels  $z_1$  and  $z_2$ . (The compatible labels  $z_1!$  and  $z_2!$  are inserted to the transitions associated with  $E_{\text{start}}$  and  $E_{\text{end}}$ .)

<sup>12</sup>A *Constraint Formula* is either  $\top$  (truth) or  $\perp$  (falsity) or is an equality or inequality between constraint terms. Examples:  $t \leq 5$ ,  $p = a + 17$ , etc. A *Constraint Term* is either a variable (out of a fixed variable set) or a real valued constant or some arithmetic operation (addition, subtraction and multiplication) of some (real valued) constants. Examples:  $x$ ,  $x + 5$ ,  $17$ ,  $17 + 9 - 144 * 6$ ,  $x * 5 - 3$ .

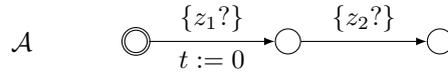


Figure 13: The automaton for the determination of minimum and maximum durations

The property used to determine the maximum or minimum duration of the EPC contains a parameter. Using an appropriate model checker (like for instance HyTech), the model checker tries to find a value for the parameter, so that the property holds.

In general these properties have the following form: ‘Is it inevitably that a location  $n$  (or transition  $\langle m_1, m_2 \rangle$ ) is reached and the clock  $t \leq p$ ’ ( $p$  then is the maximum), or ‘Is it inevitably that a location  $n$  (or transition  $\langle m_1, m_2 \rangle$ ) is reached and the clock  $t \geq p$ ’ ( $p$  then is the minimum). Since we consider the starting and ending event of the EPC, the transition  $\langle m_1, m_2 \rangle$  of the property corresponds to the transformed ending event.

This method can be generalised to select arbitrary elements (functions and events) as starting and ending points. The use of events has just been described and illustrated in the upper example. When using functions there are two cases: (1) If the function is the starting point then the left transition of the resulting automaton must be selected to synchronise with. (2) Otherwise, if it is the ending point then the right transition must be selected.

## 4 Additional Features

### 4.1 Back-transformation of Automata to EPC

Back-transforming the transformed set of automata to an EPC provides a sort of valuation or validation of the transformation and shows the modeller the ‘interpretation’ of the transformed EPC. He/She can compare the original EPC with the back-transformed EPC to compare the differences between the two models.

Because verification of some property can yield to some counter-example (provided by the verification tool), back-transformation can be used to detect where the EPC has a possible design flaws.

### 4.2 Actor Separation

We defined a process we called *Actor Separation* that utilises the annotation of EPCs with organisational units to extract EPCs that describe the business process under consideration from the perspective of selected organisational units. The resulting automaton is specific to one organisational unit with all the advantages of simulation and verification.

Actor separation can be used in two ways: (1) It is often interesting to show that some properties are (still) valid for this ‘sub-model’ of the original EPC. (2) By back-transforming

such a local perspective to an EPC one gets a customised view that concerns the organisational unit(s) in focus alone. This tailored EPC can be used as guidance for the responsibility of the actors (persons, offices, etc.) in the entire business process.

## 5 Related Work

To our knowledge there are only two approaches that use verification to get proven assertions about EPCs.

Van der Aalst [vdA99], defines two properties *regular* and *sound* and claims soundness to be a minimal requirement, because it guarantees that the process chain is free of potential deadlocks and livelocks. Further properties like *well-structuredness* (that is equivalent to our restriction to regular EPCs without the OR-connector) may be used to indicate design flaws of the EPC.

Rump ([Rum98]) distinguishes between *general properties* and *user defined properties*. Whereas the first aim at the correctness of the EPC, also indicating the origin of the design flaw, the latter allow to define arbitrary assertions that can be expressed using CTL. Provided with operational semantics, the defined EPC is transformed into a reachability graph where general properties concerning the structure of the EPC are verified. User defined properties are expressed using CTL and can be verified by the use of a model checker (for instance SMV).

Commonly used is the transformation to Petri nets (e. g. [CS92], [Rod97] [MR00],[Rit99, Rit00], [Deh02], [LSW98]). Elements and structures of an EPC are transformed into elements and structures of Petri nets.

A problem that arises when transforming EPCs to Petri nets, namely the non-locality of join-connectors, was presented in van der Aalst et al. ([vdADK02]) and was solved by Kindler ([Kin03]). Based on these semantics Cuntz built a tool ([Cun04]) that can be used to simulate EPCs answering the questions of cleanness, contact-freeness and deadlock-freeness.

Two other approaches are situated in the context of workflow management. The first approach transfers EPC to state and activity charts and focuses on the specification and verification of the workflows ([WWD<sup>+</sup>97]). The second approach ([CKSW01]) transforms *extended* EPCs (eEPC)<sup>13</sup> to a language called *Flow Definition Language*, that is used to specify workflows in the workflow management system *MQSeries Workflow* (IBM).

Operational semantics on EPCs have been defined by Nüttgens and Rump ([NR02]). They use a state based approach with tokens travelling over the state graph.

---

<sup>13</sup>Extended EPC have various additional modelling elements and are extended by a lot of additional attributes. eEPC are for instance provided by the ARIS Toolset.

## 6 Conclusion and Future Work

### 6.1 Conclusion

In this paper we introduced a method for the verification of properties of (Timed) EPC. Properties in this understanding are defined by the modeller and are a kind of quality assurance needed in the real world process.

In a first step we take communicating automata as a target description language. Simple linear chains, but also concurrent and alternative chains are then transformed into communicating automata. Properties on these EPCs can be expressed using a variant of CTL (Computation Tree Logic), a temporal logic language. And the transformation into properties on communicating automata allows the verification of such properties within the framework of EPCs.

Using the framework of *timed automata* the transformation and verification of timed EPCs is realised. The time attributes in the EPC express real-time durations of the modelled system. By an adequate transformation, properties such as maximal overall time and minimal overall time of the process (chain) can be found and/or verified. A timed extension of CTL, namely Timed CTL, is used to express these properties.

The presentation of the principle of the transformation (of EPCs and corresponding properties) is given priority over the completeness of the EPCs transformed.

A tool for the transformation of EPC and a (corresponding) tool for verification of automata are currently being developed. Using this tool the transformation of larger and elaborated examples are possible.

### 6.2 Future Work

Although, we did not consider all the EPCs currently used in practice, the grammars used allow to define a huge variety of EPCs. Nevertheless, there are obvious extensions that will get the approach closer to practice. For instance the missing OR-connector or process interfaces are elements that were commonly used. Missing structure are multiple starting and ending events, multinary connections and hierarchical EPCs.<sup>14</sup>

Beside these obvious extensions some more visionary extensions are possible.<sup>15</sup>

One idea is to extend EPC by arbitrary resources. We annotate functions with ‘effort’ instead of ‘time’ and add some role/position that can be adopted by several (different) persons. Then we can calculate the time necessary to perform the function with respect to the number of persons, as it is practised in project management. Or we can ask for properties like: ‘How many persons are necessary to perform the business process in 20 time units?’ The extension to continuously changing arbitrary resources means that we

---

<sup>14</sup>Transforming this elements is work in progress. E. g. OR can easily be transformed in a similar way, using its logical equivalent:  $x \text{ OR } y \equiv (x \text{ AND } y) \text{ XOR } (x \text{ XOR } y)$ .

<sup>15</sup>More extensions can be found in [Den06].

need to chose *Hybrid Automata*.<sup>16</sup>

The extension of automata by *Abstract Data Types* introduces the possibility to specify and verify properties regarding the content of information. Abstract Data Types can be understood as a generalisation for *Informational Objects* already used to represent data in EPCs.

## References

- [BBF<sup>+</sup>01] Béatrice Bérard, Michel Bidot, Alain Finkel, Francois Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, Berlin, New York, 2001.
- [BLL<sup>+</sup>95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - A Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems*, pages 232–243, 1995.
- [CKSW01] David Christensen, Achim Kraiss, Anja Syri, and Gerhard Weikum. Automatische Übersetzung von Geschäftsprozessmodellen in ausführbare Workflows. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung*, pages 505–513, London, UK, 2001. Springer-Verlag.
- [CS92] R. Chen and A.-W. Scheer. Modellierung von Prozeßketten mittels Petri-Netz-Theorie. In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, number 107. Institut für Wirtschaftsinformatik (IW<sub>i</sub>), 1992.
- [Cun04] Nicolas Cuntz. Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Master's thesis, Universität Paderborn, Juni 2004.
- [Deh02] Juliane Dehnert. Making EPCs fit for Workflow Management. In *Proceedings of the GI-Workshop EPK 2002*, pages 51–69, 2002.
- [Den06] Stefan M. R. Denne. Verifying Properties of Event-driven Process Chains by Transformation to Hybrid Automata. Master's thesis, Saarland University, April 2006. Available on request to the author.
- [HH95] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell HYbrid TECHnology Tool. In *Hybrid Systems II*, number 999 in Lecture Notes in Computer Science, pages 265–293. Berlin, New York, 1995.
- [HKS93] W. Hoffmann, J. Kirsch, and A.-W. Scheer. Modellierung mit Ereignisgesteuerten Prozeßketten (Methodenhandbuch). In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, number 107. Institut für Wirtschaftsinformatik (IW<sub>i</sub>), Saarbrücken, 1993.
- [Kin03] Ekkart Kindler. On the semantics of EPCs: A framework for resolving the vicious circle. In Markus Nüttgens and Frank J. Rump, editors, *Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, pages 71–80, 2003.

---

<sup>16</sup>In the example just mentioned we are not forced to switch to hybrid automata, as an extension of timed automata, namely automata with rated clocks can be used. In general such kinds specialised approaches do not succeed.

- [KNS92] G. Keller, Markus Nüttgens, and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, number 89. Institut für Wirtschaftsinformatik (IWi), 1992.
- [LSW98] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In Jörg Desel and M Silva, editors, *Application and Theory of Petri Nets*, pages 286–305, 1998.
- [MR00] Daniel Mold and Jörg Rodenhagen. Ereignisgesteuerte Prozeßketten und Petrinetze zur Modellierung von Workflows. In *Visuelle Verhaltensmodellierung verteilter und nebenläufiger Software-Systeme, Proceedings 8. Workshop des Arbeitskreises "Grundlagen objektorientierter Programmierung" der GI-Fachgruppe*, pages 57–63, 2000.
- [NR02] Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, number P-21, 2002.
- [Rit99] Peter Rittgen. Modified EPCs and Their Formal Semantics. In *Arbeitsberichte des Institutes für Wirtschaftsinformatik*, number 19. Koblenz-Landau, 1999.
- [Rit00] Peter Rittgen. Quo vadis EPK in ARIS? In *Wirtschaftsinformatik*, number 42, pages 27–35. 2000.
- [Rod97] J Rodenhagen. Darstellung ereignisgesteuerter Prozeßketten (EPK) mit Hilfe von Petrinetzen. Master's thesis, Universität Hamburg, 1997.
- [Rum98] Frank J. Rump. *Durchgängiges Management von Geschäftsprozessen auf Basis ereignisgesteuerter Geschäftsprozeßketten*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 1998.
- [vdA99] W. M. P. van der Aalst. Formalization and Verification of Event-driven Process Chains. In *Information and Software Technology*, number 41, pages 639–650. 10 1999.
- [vdADK02] W. M. P. van der Aalst, Jörg Desel, and Ekkard Kindler. On the semantics of EPCs: A vicious circle. In Markus Nüttgens and Frank J. Rump, editors, *Proceedings of the GI-Workshop EPK*, pages 71–79, 2002.
- [WWD<sup>+</sup>97] Gerhard Weikum, Dirk Wodtke, Angelika Kotz Dittrich, Peter Muth, and Jeanine Weißfels. Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR. *Inform., Forsch. Entwickl.*, 12(2):61–71, 1997.

# A Comparison of (e)EPCs and UML 2 Activity Diagrams

Harald Störrle

Universität Innsbruck, Institut für Informatik  
Technikerstrasse 21a, A-6020 Innsbruck, Österreich  
Harald.Stoerrle@uibk.ac.at

**Abstract:** In this paper, Event Process Chains (EPCs) and activity diagrams (ADs) of the Unified Modeling Language (UML) are compared with respect to (1) their syntax and its expressiveness, (2) their semantic domains and problems, and (3) their pragmatics and application conditions. The comparison is based on industrial experience and a survey of the research literature. Our conclusion is that while earlier versions of the UML did not provide sufficient means for modeling of business processes, the current version does. Since UML provides additional benefits over EPCs when it comes to software development as a consequence of business process modeling, we predict that for these applications, UML ADs will prevail over EPCs.

## 1 Introduction

### 1.1 Motivation

In the Unified Modeling Language version 1.x (UML 1.x, [OMG03]), Activity diagrams (ADs) have played a very limited role, since they were only an alternative notation for the same concepts already defined for the state machine notation. With the recent version 2.0 of UML (UML 2, [OMG05, Stö05]) this has changed. The concept (i. e., meta class) of Activity has been introduced as a substrate for the semantics. The expressiveness of ADs has been strongly increased, and their semantics has been upgraded from the run-to-completion interleaving semantics of UML state machines to a “Petri-like” approach.

This change of the UML specification is motivated by the strong need for a notation within the UML framework that allows the modeling of business processes. This, however, is the main domain of Event Process Chains (EPCs, [Sch98, Sch95]), and one wonders how the two notations actually compare—can ADs take the place of EPCs in real world applications?

### 1.2 Approach

To answer this question, the syntax and its expressiveness, the semantic domains and problems, and the pragmatics and application conditions are examined for both notations. We

have not taken into account methodological aspects, in particular, the general approaches, that is, the methodology of ARIS and the associated tool support by the ARIS toolset are

### 1.3 Related Work

To our knowledge, there is no systematic in-depth comparison of EPCs and UML 2 ADs so far. Since UML 1.x is now obsolete, so are comparisons between EPCs and UML 1.x ADs. There are superficial comparisons like [Stö05, p. 252], and there might be internal concept papers in commercial organisations.

## 2 Syntax

EPCs are often considered an integral part of the ARIS method and toolset. The UML, on the other hand, is just a notation, not a method (as it was the case for UML predecessors like OMT and OOSE). By definition, methodological concerns are excluded from the subject domain belonging to UML. This separation of concerns is considered a major milestone in the UML community, since it allowed the standardized of the language independent of other, more complex issues. These need to be addressed in real life work too, of course, but focusing on one issue first has been helpful, as the tremendous success of UML over the last decade underlines.

In this paper, thus, we disregard methodological aspects. In this section, we start by comparing ADs and EPCs by their respective notational elements. This section is structured into basic syntax, data flow, exceptions, and complex nodes.

### 2.1 Basic Syntax

Both ADs and EPCs provide elementary actions, events, and parallel and optional control flows. Both notations allow procedure-call-like refinements. See Figure 1 and 2 for a tabular comparison of the basic syntax of ADs and EPCs.

Figures 1 and 2 makes it obvious, that ADs allow many more notational variants than EPCs. For instance, AD forks/joins have not only the basic logical operators and, or, xor, but arbitrary logical formulas. Similarly, arbitrary conditions may be imposed on case distinctions (decision nodes). ADs distinguish between the opening and closing of alternative and concurrent branches (though, unfortunately, the notation allows mixing the two), which facilitates the definition of well-nestedness of operators, a notorious problem for EPCs. In ADs, it is possible to distinguish between different kinds of final nodes (terminating the whole action vs. terminating only on concurrent flow), and different kinds of events (send, receive), while there is only the event-notion for start, stop, and all kinds of events in EPCs.

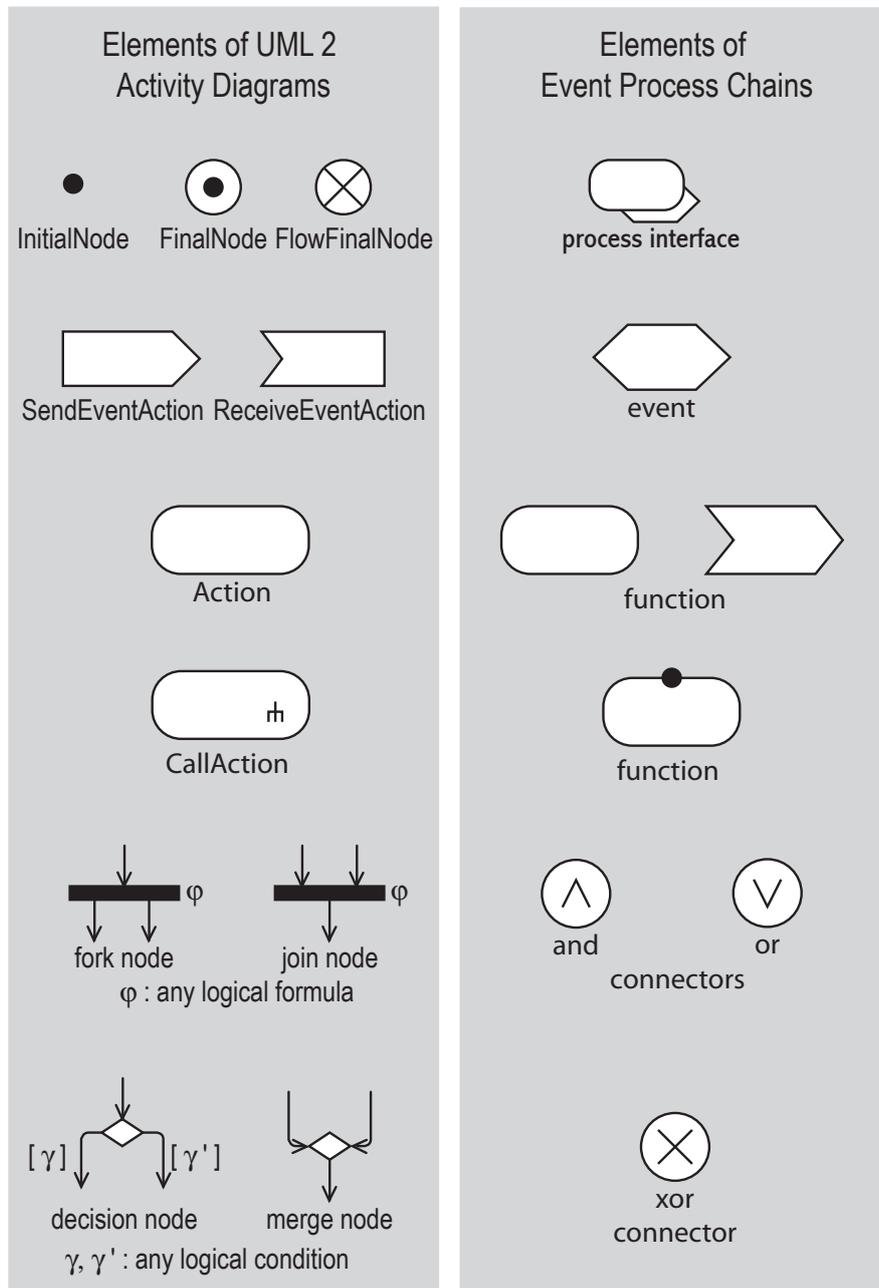


Figure 1: Comparing elementary ADs and EPCs: actions, events, and basic control nodes.

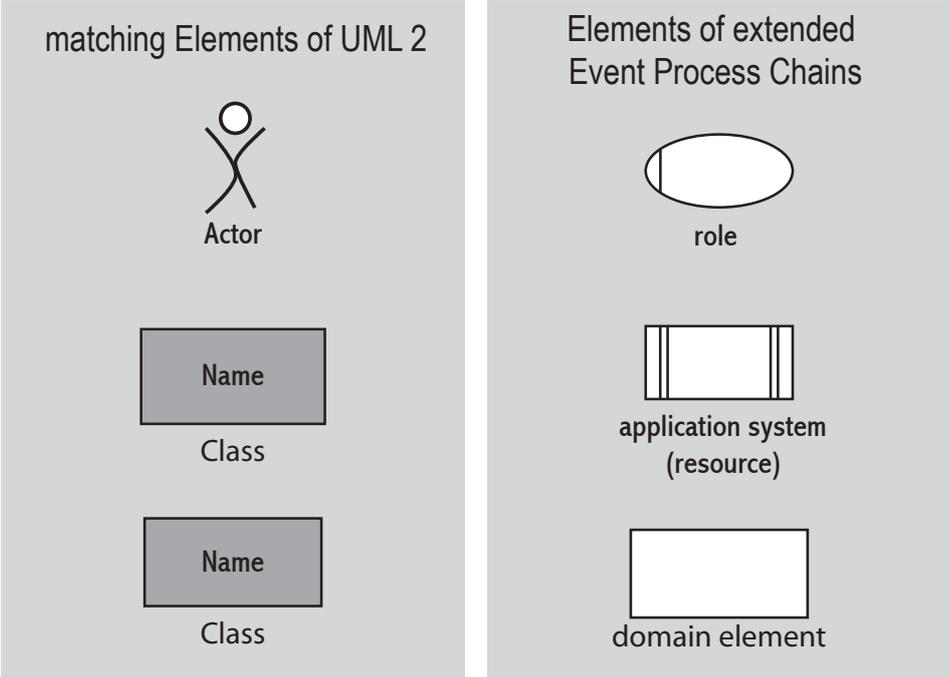


Figure 2: Comparing elementary ADs and eEPCs: actors, classes, and systems.

## 2.2 Data Flow

Regular EPCs do not provide facilities to model data flow, but extended EPCs (eEPCs) do, though only for the inputs and outputs of individual functions, not the flow as such. In ADs, on the other hand, a confusing variety of alternative notations with identical semantics is available (see Figure 3). Additionally, ADs provide means to model buffering strategies, weights, selection criteria and transformation actions for the edges connecting data and functions.

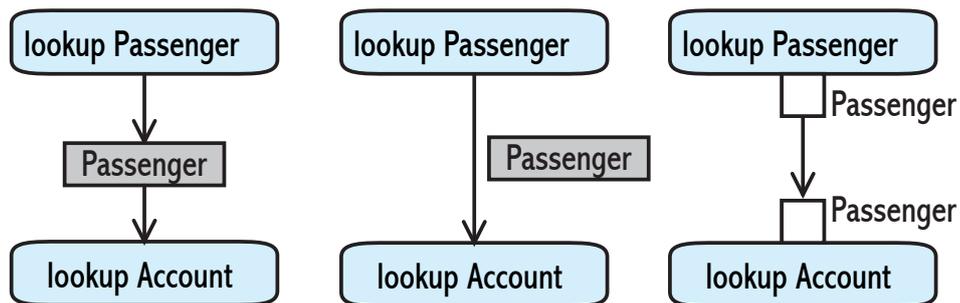


Figure 3: Notational variants for data flow in UML 2 ADs: all three notations mean the same.

## 2.3 Exceptions

Probably the most novel feature of ADs are exceptions (see Figure 4), which are rather similar to the programming language concept. In practice, there are sometimes ill formed EPCs that misuse calls/refinements as a kind of GoTos (see Figure 5). This can and has been used to simulate exception-like behavior in an unstructured way. Of course, such models are very error-prone, and this kind of notational abuse should be prohibited.

## 2.4 Complex nodes

Another new concept of UML 2 ADs are so called structured activity nodes. They comprise structured nodes and expansion regions. Structured nodes correspond to some constructs of structured programming, most notably loops, and conditional. Expansion regions provide notations for different kinds of concurrent execution of actions (cf. [Stö04c]). There are no comparable concepts in the (e)EPC notation.

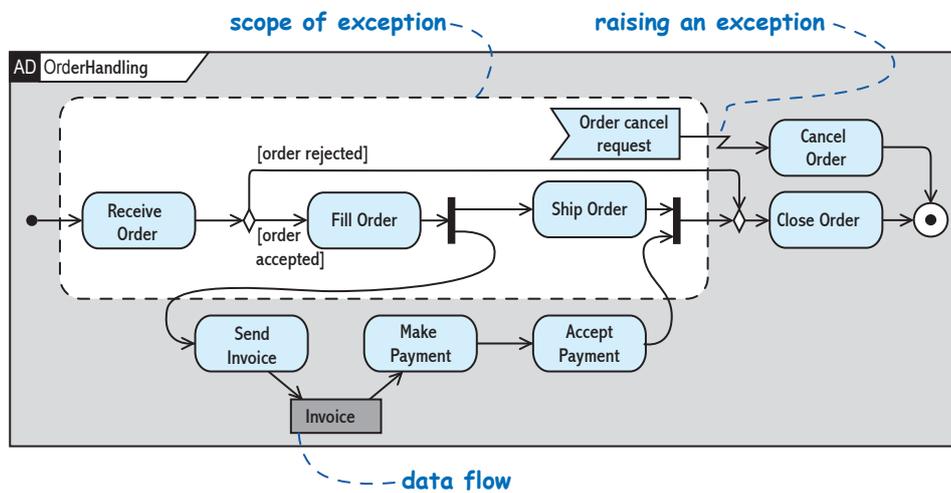


Figure 4: An AD with an exception and with data flow (taken from [OMG05]).

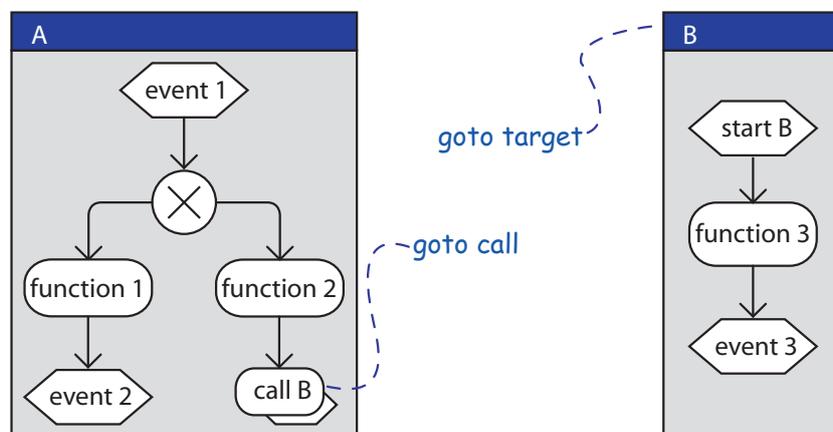


Figure 5: Abusing refinement/procedure call as goto.

## 2.5 Syntax comparison summary

All EPC concepts are available in ADs, often even in different variants. On the other hand, many AD concepts are not available in EPCs. The same is true for the annotations on elements. Therefore, the AD notation is much richer and thus much more expressive than the EPC notation.

## 3 Semantics

Both EPCs and ADs refer to the same semantic domain, namely Petri-nets. Both notations are defined in an informal way only, so that a formal semantics has to be defined separately. Many such approaches have been proposed for EPCs over the years (cf. e. g. [ADK03, DAV05]). For UML 1.x ADs, there have been some approaches, which are now obsolete, of course. For UML 2.0 ADs, not very much work has been done so far (in particular [Stö04b, Stö04a, BG04], or see e. g. the related work section in [Stö04c] for a complete summary).

Unfortunately, both notations imply a number of semantic problems, though this number is probably smaller for EPCs due to the smaller number of concepts and notational variants, which result in less trouble defining a formal semantics. Interestingly, some semantic problems occur for both notations in a very similar way (cf. the analogous findings in [CK04] and [SH05] on non-local semantics). It might be an interesting question to find out, whether a solution for one notation also works for the other.

In terms of formality, there is not much difference—both notations are informal. While the number of semantical issues seems to be smaller for EPCs, the syntax of ADs is more precisely defined due to the UML's meta modeling approach. Unless adequate formal semantics have been defined and generally agreed upon for both (e)EPCs and ADs, no comparison of the semantics is complete.

## 4 Pragmatics

### 4.1 Transition from analysis to implementation

In the syntax comparison above, we have concluded that ADs are much more expressive than EPCs. Concerning the pragmatics, notational variability is a mixed blessing, though: on the one hand, great expressiveness certainly is helpful to capture a wide range of processes more precisely. However, in the absence of a precise semantics, this might be a false precision. Also, one might suspect that the plenitude of concepts and notations in ADs make them more difficult to learn and comprehend. Whether this is an advantage for EPCs or for ADs depends on the application area in which they are used. There are a number of such potential application areas.

- analysis tasks like business process modeling
- design tasks like workflow modeling
- implementation tasks like program modeling

Business process modeling is the application domain EPCs have been created for (see [KNS92, LSW97]), and it has also been the most important driving force behind the enhancement and standardization of ADs in UML 2. Both notations have been used for this purpose in practice, though there are not very many experience reports on using UML 2 ADs yet. It is obvious though, that the notational richness is more of a hindrance for this type of task, due to the kind of people usually involved. However, it is of course possible to tailor ADs such that only the expressive means of EPCs are available. In fact, initial practical experience in industrial projects suggests that there are some notions and notations of UML 2 ADs that are quite natural for domain experts with little computer-science know how, such as the concept of exception. This seems to be very intuitive for many people.

Both EPCs and ADs have also been applied successfully to workflow modeling. Here, the notational richness of ADs is already an advantage, since there are design decisions that can not be represented in EPCs—one would have to enhance the notation or rely on a proprietary tool to support design tasks. Enhancements, of course, are much more difficult than restrictions.

The notational richness of ADs turns into a strong advantage when it comes to program modeling, that is, creating models on level of detail similar to programs. This could be done either constructively in order to create a running system or analytically in order to visualize programming language code such as BPEL, Java, and, of course, ABAP. Regular EPCs or extended EPCs are definitely too weak to be useful for such applications. It is of course possible to add more and more concepts and additional information to (e)EPCs, but then, the simplicity and ease of use that is their particular advantage would be lost, too.

Obviously, it is easier to go from analysis to design and to implementation using only one notation and only one toolset. This is the case for ADs which are a well integrated part of the UML. Thus it is very easy to proceed from a pure process description into other views concerned with data, domain architecture, software architecture and so on. Also, refining analysis level processes into design level processes is easier when the same notation is used for both.

Recent approaches to transform (e)EPCs into languages like the Business Process Execution Language (BPEL, [ACD<sup>+</sup>03]) or workflow definition languages, suffer from the same problem. While such approaches definitely improve the viability of using (e)EPCs as a front end to implementing service oriented architectures, they also imply another language interface with more opportunities for errors and more difficult traceability.

As a direct consequence of the previous considerations, it follows that ADs are better suited for software development projects than (e)EPCs. For business process reengineering purposes, on the other hand, using (e)EPCs has a small advantage in that no tailoring is required. This section is summarized in Figure 6.

NOTATION	APPLICABILITY IN LIFECYCLE PHASE		
	ANALYSIS	DESIGN	IMPLEMENTATION
(e)EPCs	✓	(✓) (requires extension)	–
UML 2 ADs	(✓) (requires restriction)	✓	✓

Figure 6: Comparing EPCs and ADs by the phases of the software life cycle in which they are applicable.

## 4.2 Tools

Over the last 10 years, the number and quality of UML tools has increased dramatically. Today, there is a broad range of several hundred tools, from open-source to high-end industrial tool suites, from mere drawing tools to integrated development environments with team support, version and configuration control, code generation, consistency checkers, report generators and so on.

For (e)EPCs, on the other hand, there are only a few tools, most notably of course the ARIS toolset, but there is now also the EPC TOOLS open source initiative [CK]. Generally, EPC tools tended to also provide more advanced functionality like simulation and analysis of models which was absent in UML models.

With XMI, there has been a standardized data exchange format for many years. Even though standard compliance has been less than perfect, effective data exchange between different tools is a reality today for UML. The standardization of UML and its exchange format XMI are primarily driven by the Object Management Group (OMG). Since the OMG is just an industrial consortium its documents are not standards in the proper sense. However, the previous version of the UML (1.4.2) and the accompanying XMI version have been standardized by the International Standards Organisation (ISO) recently (ISO/IEC 19501:2005 and ISO/IEC 19503:2005), and the OMG also pursues the updating of these standards to more recent versions of UML.

For (e)EPCs on the other hand, there have been far less tools so that data exchange has not been as much of a problem as it had been for UML. Still, there has been demand for a general exchange format, and with EPML (cf. [MN04]), there is now a practical proposal, including tool support. This format seems to be widely accepted, though so far it lacks standardization even by the community or an industry consortium.

NOTATION	NUMBER OF TOOLS	TOOL SUPPORT	
		EXCHANGE FORMAT	STANDARDIZATION
(e)EPCs	~ 10	EPML	proprietary
UML 2 ADs	> 200	XMI	UML 1.4.2 / ISO 19501:2005 XMI / ISO 19503:2005

Figure 7: Comparing EPCs and ADs by the available tool support.

## 5 Discussion

For projects restricted to domain analysis, EPCs and tailored (i. e., simplified) ADs are about level-headed. Wherever code is to be produced at some point, however, ADs have an edge over EPCs as they allow seamless integration of analysis with design and implementation.

There are a number of general advantages ADs have over EPCs:

- UML ADs are part of a standard that is internationally accepted and developed, while EPCs are mainly used in Germany, and in particular in organisations using SAP software.
- There are so much more tools for ADs than there are for EPCs, that there is a great likeliness that there are better and cheaper UML tools than there are EPC tools. The more advanced functionalities traditionally associated with some EPC tools (e. g. simulation, model analysis, consistency checking and so on) are nowadays also found and often superseded in UML tools.
- Since UML is the de facto lingua franca of software engineering, all professional software engineers (and most computer scientists) will have had at least some exposure to UML, but frequently none to (e)EPCs. Therefore, UML ADs are a more viable choice in a software development project than (e)EPCs.
- Also, there is a much greater choice of books and commercial trainings, a larger body of scientific work, industrial experiences, and best practices for ADs than there is for EPCs.

None of these reasons is entirely compelling by itself—together, however, they make it likely that ADs will supersede (e)EPCs in the long run. Even for specialties of EPCs, where they currently still have advantages over ADs (certain tools, familiarity in certain communities etc.), ADs will spread to become the standard notation. How long this process may take remains to be seen.

## References

- [ACD<sup>+</sup>03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Golland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services (v1.1), 2003. available at <http://www.ebpm1.org/bpel4ws.htm>.
- [ADK03] Wil van der Aalst, Jörg Desel, and Ekkart Kindler. On the semantics of EPCs: A Framework for resolving the vicious circle (extended abstract). In Markus Nüttgens and Frank J. Rump, editors, *Proc. 2. Ws. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK'03)*, pages 71–79. Gesellschaft für Informatik e.V. 2003. available at [www.epk-community.de](http://www.epk-community.de).
- [BG04] Conrad Bock and Michael Gruninger. PSL: A semantic domain for flow models. *Intl. J. Software and Systems Modeling*, Online First, 2004.
- [CK] Nicolas Cuntz and Ekkart Kindler. The EPC Tools. available at [www.cs.upb.de/cs/kindler/Forschung/EPCTools](http://www.cs.upb.de/cs/kindler/Forschung/EPCTools).
- [CK04] Nicolas Cuntz and Ekkart Kindler. On the semantics of EPCs: Efficient calculation and simulation. In Markus Nüttgens and Frank J. Rump, editors, *Proc. 3. Ws. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK'04)*, pages 7–26. Gesellschaft für Informatik e.V. 2004. available at [www.epk-community.de](http://www.epk-community.de).
- [DAV05] B.F. van Dongen, Wil M.P. van der Aalst, and H.M.W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In Oscar Pastor and J. Falcao e Cunha, editors, *Proc. 17th Intl. Conf. Advanced Information Systems Engineering (CAiSE'05)*, number 3520 in LNCS, pages 372–386. Springer Verlag, 2005.
- [KNS92] Gerd Keller, Markus Nüttgens, and August-Wilhelm Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerte Prozessketten (EPK)". Technical Report 89, Institut für Wirtschaftsinformatik, Uni Saarbrücken, 1992. available at <http://www.iwi.uni-sb.de/iwi-hefte/heft089.pdf>.
- [LSW97] P. Langner, C. Schneider, and K. Wehler. Prozeßmodellierung mit ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. *Wirtschaftsinformatik*, 39(5):479–489, 1997.
- [MN04] Jan Mendling and Markus Nüttgens, editors. *Proc. 1st GI Ws. XML Interchange Formats for Business Process Management (XML4BPM)*, March 2004.
- [OMG03] OMG. OMG Unified Modeling Language Specification (adopted formal specification, version 1.5). Technical report, Object Management Group, March 2003.
- [OMG05] OMG. UML 2.0 Superstructure Specification (formal/05-07-04). Technical report, Object Management Group, August 2005. available at [www.omg.org](http://www.omg.org), downloaded at September 19<sup>th</sup>, 2005.
- [Sch95] August-Wilhelm Scheer. *Business Process Engineering. Reference Models for Industrial Enterprises*. Springer Verlag, 1995.
- [Sch98] August-Wilhelm Scheer. *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. Springer Verlag, 3<sup>rd</sup> edition, 1998.
- [SH05] Harald Störle and Jan Hendrik Hausmann. Obstacles on the Way Towards a Formal Semantics of UML 2.0 Activities. In Klaus Pohl, editor, *Proc. Natl. Germ. Conf. Software-Engineering 2005 (SE'05)*, number P-64 in Lecture Notes in Informatics, pages 117–128. Gesellschaft für Informatik e.V. 2005.

- [Stö04a] Harald Störrle. Semantics and Verification of Data-Flow in UML 2.0 Activities. In Mark Minas, editor, *Proc. Intl. Ws. on Visual Languages and Formal Methods (VLFM'04)*, pages 38–52. IEEE Press, 2004. available at [www.pst.informatik.uni-muenchen.de/~stoerrle](http://www.pst.informatik.uni-muenchen.de/~stoerrle).
- [Stö04b] Harald Störrle. Semantics of Control-Flow in UML 2.0 Activities. In Paolo Bottoni, Chris Hundhausen, Stefano Levaldi, and Genny Tortora, editors, *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 235–242. IEEE Computer Society, 2004.
- [Stö04c] Harald Störrle. Semantics of Expansion Nodes in UML 2.0 Activities. *Nordic Journal of Computing*, 11(3):1–24, 2004.
- [Stö05] Harald Störrle. *UML 2 erfolgreich einsetzen*. Addison-Wesley, 2005.

# Berechnung von Komplexitätsmetriken für ereignisgesteuerte Prozessketten

Volker Gruhn, Ralf Laue, Frank Meyer  
{gruhn, laue}@eBUS.informatik.uni-leipzig.de  
Lehrstuhl für Angewandte Telematik und E-Business\*  
Universität Leipzig, Fakultät für Informatik

**Abstract:** Ereignisgesteuerte Prozessketten werden dazu eingesetzt, betriebliche Abläufe zu modellieren. Der wohl wichtigste Anwendungsfall besteht darin, dass diese Modelle zur Kommunikation zwischen verschiedenen Personen genutzt werden. Ziel dieser Kommunikation ist es etwa, die modellierten Abläufe zu verstehen oder Verbesserungspotential aufzudecken. Modelle, die von Menschen verstanden und ggf. geändert werden müssen, sollten so beschaffen sein, dass sie leicht verständlich sind. In den vergangenen Monaten gab es zahlreiche Veröffentlichungen, die Komplexitätsmetriken für Geschäftsprozessmodelle vorschlugen, um zu definieren, wie man „leichte Verständlichkeit“ messen kann [Car05b, GL06, MMN<sup>+</sup>06, ARGP06, CMNR06, Car06]. In diesem Beitrag wird eine Auswahl der dort vorgeschlagenen Metriken kurz beschrieben und ein Werkzeug vorgestellt, mit dem Komplexitätsmetriken für EPKs gemessen werden können.

## 1 Einführung und verwandte Literatur

Um die Arbeit mit ereignisgesteuerten Prozessketten (EPK) zu erleichtern und Fehler oder Missverständnisse auszuschließen, sollten die Modelle möglichst leicht verständlich sein. Diese Forderung führt zu dem Wunsch, Faktoren, die die Verständlichkeit von Modellen beeinflussen, zu messen. Mit anderen Worten: Gesucht sind Metriken, die eine Aussage über die Komplexität von Modellen liefern. Ein Modellierer kann aus den Messungen solcher Metriken Hinweise darauf entnehmen, wann ein Modell vereinfacht werden sollte, was beispielsweise durch das Zerlegen in mehrere Teilmodelle möglich sein kann [RV04]. Wir betonen aber, dass aus den Ergebnissen der Metrikberechnungen keinesfalls schon automatisch Vorgaben für Modellverbesserungen abgeleitet werden können. Die berechneten Metriken können dem Modellierer lediglich Hinweise auf mögliche Verbesserungen geben. Dieser muss dann auf Grund seiner Erfahrung insbesondere entscheiden, ob durch die Metriken eine hohe Komplexität des Prozesses selbst oder des Prozessmodells erkannt wurde und die entsprechenden Schlussfolgerungen zur Verbesserung des Prozesses oder des Prozessmodells ziehen. Ebenso betonen wir, dass keine noch so gute Komplexitätsme-

---

\*Der Lehrstuhl für Angewandte Telematik und E-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG

trik alle für die Verständlichkeit einer EPK notwendigen Aspekte messen kann; ein offensichtliches Beispiel sind Verständnisprobleme, die sich aus einer schlechten Benennung von Funktionen und Ereignissen ergeben.

Für Software in höheren Programmiersprachen sind zahlreiche Komplexitätsmetriken bekannt, die seit Jahren erfolgreich eingesetzt werden. Verschiedene Forschungsgruppen untersuchten nun, wie sich diese bekannten und erfolgreich angewendeten Ansätze auf Geschäftsprozessmodelle übertragen lassen. Cardoso[Car05b] schlug eine Verallgemeinerung der aus der Software-Komplexitätsmessung bekannten zyklomatischen Komplexität [McC76] vor, die wir in Abschnitt 2.3 besprechen werden.

Gruhn und Laue [GL06] sowie Cardoso et al. [CMNR06] gaben eine Übersicht über Ansätze zur Komplexitätsmessung von Software und ihre Übertragung auf Geschäftsprozessmodelle. Rolón et al. [ARGP06] schlagen verschiedene Zähl- und Verhältnismetriken für die Sprache BPMN vor. All diese Arbeiten entstanden unabhängig voneinander in verschiedenen Forschergruppen in den vergangenen zwölf Monaten - sicher ein Zeichen dafür, dass die Komplexitätsmessung von Geschäftsprozessmodellen ein aktuelles und relevantes Thema ist.

Keine der bisher genannten Arbeiten liefert eine umfassende Validierung der Tauglichkeit der vorgeschlagenen Metriken. Den ersten Schritt hierzu unternahmen Mendling et al. [MMN<sup>+</sup>06], die EPK-Modelle des SAP R/3 Referenzmodells testeten und untersuchten, welcher Zusammenhang zwischen verschiedenen Komplexitätsmetriken und Modellfehlern besteht.

Während bei allen bisher genannten Arbeiten zur Messung der Modellkomplexität nahezu ausschließlich der Kontrollfluss eines Geschäftsprozessmodells betrachtet wurde, schlägt [Car06] erstmals Metriken vor, die auch den Dokumentenfluss und die Nutzung von Ressourcen in die Betrachtungen einbeziehen.

Die wichtigsten der in den genannten Veröffentlichungen vorgeschlagenen Metriken werden im Abschnitt 2 diskutiert.

Bisher sind unseres Wissens kaum Werkzeuge verfügbar, die Komplexitätsmetriken von Geschäftsprozessmodellen messen können. Zu nennen ist in diesem Zusammenhang Argo/ UML[RR00], das u.a. Layoutprobleme in UML-Diagrammen erkennt. Einen Ansatz für UML-Aktivitätsdiagramme präsentiert [Gro04]; hier werden einerseits bekannte Stilprobleme[Amb03] erkannt, andererseits einige Zählmetriken sowie bei Aktivitätsdiagrammen die zyklomatische Komplexität (siehe Abschnitt 2.3) berechnet.

In unserem Beitrag stellen wir das Werkzeug EPCMetrics vor, das verschiedene Komplexitätsmetriken für EPK-Modelle berechnen kann.

## 2 Die Metriken

In diesem Abschnitt werden in der Literatur beschriebene Komplexitätsmetriken für Geschäftsprozessmodelle beschrieben sowie einige weitere eingeführt. Alle hier vorgestellten Metriken wurden im Werkzeug EPCMetrics, das in den späteren Abschnitten vorgestellt

wird, implementiert. In diesem Abschnitt soll nur eine grobe Übersicht über die Metriken gegeben werden. Für die Details möchten wir auf die jeweils angeführte Literatur verweisen. Eine umfassende Bewertung der vorgeschlagenen Metriken ist nicht Gegenstand dieses Beitrags. Wir wollen jedoch das Werkzeug EPCMetrics in unserer weiteren Forschung dazu nutzen, Untersuchungen zur Tauglichkeit der Metriken anzustellen.

## 2.1 Zählmetriken

Das einfachste Komplexitätsmaß für Software ist eine Zählung der Lines of Code. Es misst also einfach die Länge des Quelltextes oder die Zahl der ausführbaren Anweisungen. Es liegt nahe, ein analoges Größenmaß als einfache Metrik für den Umfang einer EPK zu nehmen. Als entsprechende Maße wurden die Zahl der Aktivitäten oder die Zahl der Aktivitäten und Konnektoren vorgeschlagen[GL06, CMNR06]. In [MMN<sup>+</sup>06] wurden weitere Zählmaße untersucht und die Vermutung bestätigt, dass eine hohe Zahl von Join-Konnektoren und eine hohe Zahl von inneren Ereignissen (also solchen, die weder Start- noch Endereignis sind) die Wahrscheinlichkeit für Fehler im Modell erhöht.

## 2.2 Verhältnismetriken

Man kann vermuten, dass eine hohe „Konnektorendichte“ im Modell das Verständnis erschwert und die Wahrscheinlichkeit struktureller Fehler in EPKs (z.B. Deadlocks) erhöht. Dies kann mit Metriken gemessen werden, die die Zahl der Konnektoren ins Verhältnis setzen zur Zahl der Funktionen bzw. zur Zahl der Funktionen und internen Ereignisse. Ebenso kann es sinnvoll sein, die in den kommenden Abschnitten vorgestellten Metriken für die Zahl der Zyklen im Modell (siehe Abschnitt 2.4), der unstrukturierten Elemente (siehe Abschnitt 2.5) oder zum Layout (siehe Abschnitt 2.7) ins Verhältnis zur Zahl der Funktionen [und der internen Ereignisse] zu setzen, um statt einer absoluten Zahl die „Häufigkeit“ von Zyklen etc. zu messen.

## 2.3 Kontrollflusskomplexität

McCabe[McC76] schlug die zyklomatische Zahl als Komplexitätsmetrik für Computerprogramme vor. Sie geht vom Kontrollflussgraphen aus und zählt die möglichen Wege, die im Kontrollflussgraphen zurückgelegt werden können. Für die exakte Definition dieser Metrik verweisen wir auf die umfangreiche Literatur[McC76, Kan02]; informell reicht es zu sagen, dass die zyklomatische Zahl eines Kontrollflussgraphen der Zahl der binären Entscheidungen (also der `if`-Statements in einer höheren Programmiersprache) plus 1 ist. Nicht-binäre Entscheidungen (also etwa `select` oder `case`-Statements in einer höheren Programmiersprache) mit  $n$  möglichen Ausgängen werden wie  $n-1$  binäre Entscheidungen behandelt. Die McCabe-Metrik gibt also Aufschluss über die Zahl der Verzweigungen im

Kontrollflussgraphen.

Cardoso[Car05b] leitete von der McCabe-Metrik eine analoge Metrik für Geschäftsprozessmodelle ab. Um diese zu bestimmen, ist es lediglich nötig, für jeden Split im Modell die Zahl der möglichen verschiedenen Kontrollflüsse, die von diesem Split ausgehen können, zu addieren.

Die Zahl der verschiedenen Kontrollflüsse ist

- für einen AND-Split: 1 (Es müssen immer alle folgenden Pfade parallel bearbeitet werden.)
- für einen XOR-Split mit  $n$  Ausgängen:  $n$  (Genau einer der  $n$  Pfade muss genommen werden, dafür gibt es gerade  $n$  Auswahlmöglichkeiten.)
- für einen OR-Split mit  $n$  Ausgängen:  $2^n - 1$  (Das entspricht den Möglichkeiten, mindestens einen und höchstens  $n$  zu durchlaufende Pfade auszuwählen.)

Ein erster (wenn auch wegen seines geringen Umfang noch wenig aussagekräftiger) Labortest von Cardoso legt nahe, dass diese Metrik geeignet ist, die (subjektiv empfundene) Komplexität eines Geschäftsprozessmodells zu messen[Car05b]. Unzulänglichkeiten dieser Metrik wurden in [MMN<sup>+</sup>06] und [GL06] diskutiert.

Analog zu der von Cardoso vorgeschlagenen Metrik betrachtet [MMN<sup>+</sup>06] eine Metrik, die analog für jeden Join-Konnektor die Zahl der möglichen ankommenden Kontrollflüsse bestimmt, die dieser Join-Konnektor verarbeiten kann.

## 2.4 Metriken für Zyklen

Um untersuchen zu können, inwiefern Zyklen im Modell Einfluss auf Verständlichkeit und Fehlerrate haben, haben wir drei Metriken implementiert, die Zyklen im Modell untersuchen. Die Metrik NCycles bestimmt einfach die Zahl der elementaren Zyklen[Joh75]. Diese Zahl ist allerdings wenig aussagekräftig: In der in Abb. 1 gezeigten EPK etwa kann der Zyklus auf vier verschiedene Arten durchlaufen werden:  $A \rightarrow B \rightarrow E1/E2 \rightarrow C \rightarrow D \rightarrow E \rightarrow F1/F2 \rightarrow A$ , so dass die Metrik vier elementare Zyklen zählt.

Dieses Manko ist in der Metrik NCycleSets behoben. Hierfür werden Zyklen, die die gleichen Einstiegs- und Ausstiegsknoten haben sowie deren Nachfolgermenge der Ausstiegsknoten übereinstimmen, zu einer Äquivalenzklasse zusammengefasst. NCycleSets zählt diese Äquivalenzklassen, so dass z.B. die Zyklen im Modell in Abb. 1 die Metrik nur um 1 statt um 4 erhöhen. Schließlich versucht die Metrik CycleSetsComplexity, „Unstrukturiertheit“ in Zyklen zu messen: Für jeden zusätzlichen Einstiegs- und Ausstiegsknoten in einer Äquivalenzklasse von Zyklen wird ebenso 1 zur Metrik hinzugezählt wie für jeden Fall von Überschneidungen zwischen Zyklen in verschiedenen Äquivalenzklassen.

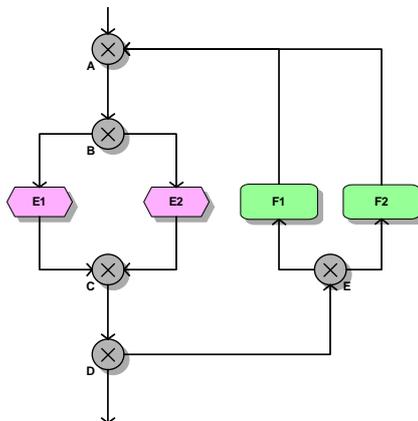


Abbildung 1: Beispiel für Zyklen

## 2.5 Metrik zur Unstrukturiertheit

Strukturelle Fehler (wie Deadlocks) in einer EPK lassen sich stets auf fehlende Korrespondenz zwischen Split- und Join-Konnektoren zurückführen. Für EPKs, deren Split- und Join-Konnektoren nur sauber verschachtelte Kontrollblöcke bilden, kann gezeigt werden, dass sie frei von solchen Fehlern sind [LSW97, SO00, CS94]. Allerdings schränkt die Forderung, nur wohlstrukturierte Modelle zu verwenden, den Modellierer in der Praxis zu stark ein. Van der Aalst hebt in [van99] hervor, dass unstrukturierte Modellelemente nicht notwendig zu Modellfehlern führen, aber dennoch wenn möglich vermieden werden sollen. In [MMN<sup>+</sup>06] wurde vorgeschlagen, das Missverhältnis zwischen Split- und Join-Konnektoren durch den Quotienten *Zahl der Joins/Zahl der Splits* zu messen. Diese Metrik ist in EPCMetrics als RJoinsSplits implementiert. Wir befürchten allerdings, dass sie in der Praxis von eher geringem Nutzen ist. Ein Grund dafür ist, dass der Typ der Konnektoren, eine häufige Fehlerquelle, nicht beachtet wird. Noch schwerer wiegt, dass man eine EPK mit einer lokalen Unstrukturiertheit, die sich in einem Join-Split-Verhältnis größer als 1 niederschlägt, „verbessern“ kann, indem man an anderer Stelle ein weiteres unstrukturiertes Modellelement einfügt, das mehr Splits als Joins enthält.

Wir haben in EPCMetrics eine Metrik implementiert, die zunächst versucht, den zu einem Split-Konnektor  $s$  passenden Join-Konnektor  $j(s)$  zu finden bzw. umgekehrt zu einem Join-Konnektor den passenden Split-Konnektor. Aus Platzgründen diskutieren wir hier nur die Suche nach einem Join-Konnektor zu einem Split-Konnektor  $s$ , der nicht (wie dies in Abb. 2 a) der Fall ist) eine Iteration abschließt. Die anderen Fälle werden von EPCMetrics gesondert behandelt und sind im Quelltext der Klasse Unstructured ausführlich kommentiert.

Um  $j(s)$  zu bestimmen, suchen wir einen Knoten, der ausgehend von  $s$  auf mindestens zwei Pfaden, die außer  $s$  und  $j(s)$  keine gemeinsamen Knoten haben, erreichbar ist. Außerdem wird von diesen Pfaden gefordert, dass an keiner Stelle entlang des Pfades in einen Zyklus eingetreten wird. Dann nennen wir die von  $s$  eingeleitete Kontrollflussstruktur (oder kürzer:  $s$  selber) *unstrukturiert*, wenn das zugehörige  $j(s)$  nicht oder nicht eindeutig bestimmt

ist (vgl. Abb. 2 b) und 2 c)). Ebenso heißt  $s$  unstrukturiert, wenn zwar  $j(s)$  eindeutig bestimmt ist, aber nicht alle der folgenden Aussagen gelten:

- Die Typen von  $s$  und  $j(s)$  stimmen überein.
- Alle Pfade von  $s$  zu einem Endereignis gehen durch  $j(s)$ .
- Alle Pfade von einem Startereignis zu  $j(s)$  gehen durch  $s$ .
- Alle Pfade von  $s$  zu  $s$  gehen durch  $j(s)$ .
- Alle Pfade von  $j(s)$  zu  $j(s)$  gehen durch  $s$ .

Es lässt sich leicht induktiv zeigen, dass für EPKs, die lt. [LSW97] oder [CS94] wohlstrukturiert sind, die genannten Eigenschaften gelten, so dass für solche EPKs die Zahl der unstrukturierten Konnektoren nach unserer Metrik 0 ist.

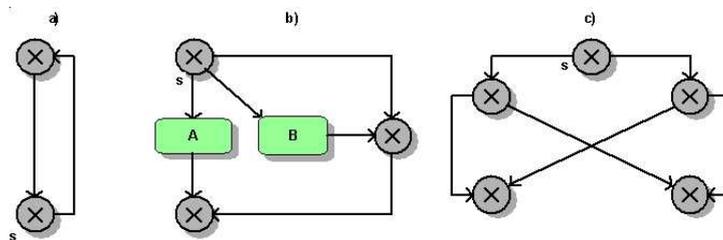


Abbildung 2: Unstrukturiertheit

## 2.6 Verschachtelungstiefe

Harrison und Magel[HM81] schlugen ein Software-Komplexitätsmaß vor, das berücksichtigt, dass verschachtelte Kontrollstrukturen schwerer verständlich sind als lineare. Die von ihnen vorgeschlagene Metrik zählt zunächst einmal die Anweisungen im Programm. Für jede Entscheidungsanweisung (*if-then*) wird jedoch zusätzlich noch die Zahl der Anweisungen, die im Einflussbereich des mit diesem *if-then* beginnenden Kontrollblocks liegen, hinzugezählt. Somit fallen Anweisungen innerhalb eines verschachtelten Kontrollblocks stärker ins Gewicht. Für die formale Definition der Metrik verweisen wir auf [HM81]. Eine analoge Metrik für EPKs wurde in EPCMetrics in der Klasse NestingLevel implementiert: Hier werden jeweils die Split-Konnektoren, die im Einflussbereich eines anderen Split-Konnektors liegen, gezählt.

## 2.7 Metriken zum grafischen Layout

Gutes Layout trägt maßgeblich zur Verständlichkeit von grafischen Modellen wie EPKs bei [AF06, Amb03]. Zwei Aspekte des grafischen Modelllayouts werden mit EPCMetrics gemessen: EdgesIntersections zählt, wie oft sich zwei Kanten in einem Modell überschneiden, was den Lesefluss behindern kann. WritingDirection zählt, wie oft von der Hauptleserichtung (in EPKs üblicherweise von oben nach unten, seltener von links nach rechts) abgewichen wird. Dies sind nur zwei Aspekte guten Layouts. Zahlreiche weitere könnten ergänzt werden, etwa zur Ausrichtung der Kanten von Modellelementen oder zur Bestimmung dicht nebeneinander parallel verlaufender Kontrollflusspfeile, die den Leser verwirren können.

## 2.8 Zusammenfassung der vorgestellten Metriken

Tabelle 1 fasst die in den obigen Abschnitten aufgeführten Komplexitätsmetriken zusammen, nennt die Klassennamen, unter denen die jeweilige Metrik in unserem Werkzeug EPCMetrics implementiert ist, und gibt weiterführende Literatur an.

# 3 Das Metrikwerkzeug

Zur Berechnung der in den vergangenen Abschnitten eingeführten Komplexitätsmetriken für EPKs haben wir ein in der Sprache Java geschriebenes Werkzeug EPCMetrics entwickelt. Dieses kann entweder alleinstehend als Batchprogramm aufgerufen werden oder in die grafische Oberfläche des EPK-Editors und -Simulators EPCTools [Cun04] integriert werden. EPCMetrics steht unter der freien GNU General Public License und kann von unserer Website [Lau06] heruntergeladen werden.

## 3.1 Installation und Benutzung

EPCMetrics kann in das Werkzeug EPCTools [Cun04] integriert werden. EPCTools ist ein in der Sprache Java geschriebener Editor und Simulator für EPK-Modelle, der als Erweiterung für die Entwicklungsumgebung Eclipse entwickelt wurde.

Abb. 3 zeigt einen Screenshot der Integration der Metrikberechnung in die grafische Oberfläche von EPCTools. Die Metriken werden für das im Editor dargestellte EPK-Modell berechnet. Überschreiten die berechneten Metriken vorgegebene Schwellwerte, wird dies dem Modellierer mitgeteilt: Ein gelber Hintergrund steht für eine hohe, ein roter Hintergrund für sehr hohe Komplexität der EPK, zu sehen in den ersten beiden Zeilen des rechten unteren Fensters in Abb. 3. Dass verschiedene Metriken, wie in der Abbildung gezeigt, zu unterschiedlichen Aussagen zum Grad der Komplexität kommen können, soll dabei nicht

Klassenname	Metrik	Literatur
NConnectors	Zahl der Konnektoren	
NEdges	Zahl der Kanten	[MMN <sup>+</sup> 06]
NEndEvents	Zahl der Endereignisse	[MMN <sup>+</sup> 06] [ARGP06]
NEvents	Zahl der inneren Ereignisse	[MMN <sup>+</sup> 06] [ARGP06]
NStartEvents	Zahl der Startereignisse	[MMN <sup>+</sup> 06] [ARGP06]
NFunctions	Zahl der Funktionen	[GL06] [CMNR06] [MMN <sup>+</sup> 06] [ARGP06]
NFunctions Connectors	Zahl der Funktionen und Konnektoren	[CMNR06] [MMN <sup>+</sup> 06]
NJoinAnd	Zahl der And-Joins	[MMN <sup>+</sup> 06]
NJoinOr	Zahl der Or-Joins	[MMN <sup>+</sup> 06]
NJoinXor	Zahl der Xor-Joins	[MMN <sup>+</sup> 06]
NJoins	Zahl der Join-Konnektoren	[MMN <sup>+</sup> 06]
NSplitAnd	Zahl der And-Splits	[MMN <sup>+</sup> 06]
NSplitOr	Zahl der Or-Splits	[MMN <sup>+</sup> 06]
NSplitXor	Zahl der Xor-Splits	[MMN <sup>+</sup> 06]
NSplits	Zahl der Split-Konnektoren	[MMN <sup>+</sup> 06]
RConnectors EventsFunctions	Verhältnis zwischen der Zahl der Konnektoren und der Zahl der Funktionen und internen Ereignisse	
RConnectors Functions	Verhältnis zwischen der Zahl der Konnektoren und der Zahl der Funktionen	
RJoinsSplits	Verhältnis zwischen der Zahl der Joins und der Zahl der Splits	[MMN <sup>+</sup> 06]
CFC	Kontrollfluss-Komplexität der Split-Konnektoren	[McC76] [LK02] [GL06] [Car05b] [CMNR06] [MMN <sup>+</sup> 06] [Car05a]
JC	Kontrollfluss-Komplexität der Join-Konnektoren	[MMN <sup>+</sup> 06]
CycleSets Complexity	siehe Abschn. 2.4	
NCycles	Zahl der elementaren Zyklen	[Joh75]
NCycleSets	siehe Abschn. 2.4	
NestingLevel	Zahl der Splits, die im Einflussbereich eines anderen Splits liegen	[HM81]
Unstructured	siehe Abschn. 2.5	
Edges Intersections	Layout: Überschneidungen von Kontrollflusskanten	
Writing Direction	Layout: Abweichungen von der Hauptleserichtung	

Tabelle 1: Komplexitätsmetriken

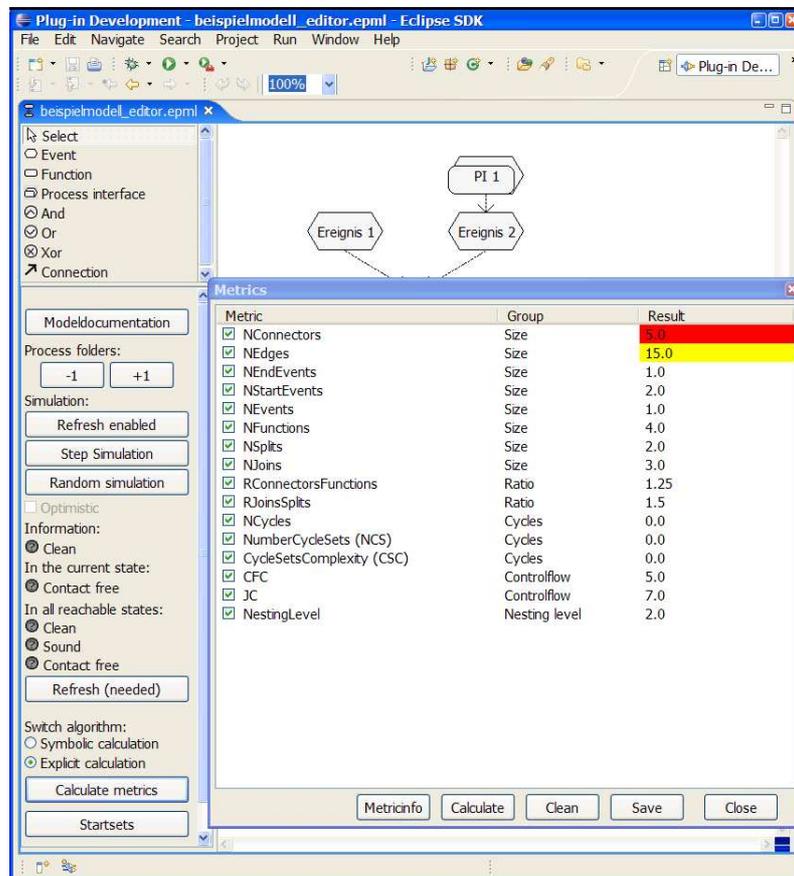


Abbildung 3: Screenshot der erweiterten Version von EPCTools

verwundern, da sie verschiedene Aspekte der Komplexität messen.

Die Schwellwerte werden, wie auch die Auswahl der Metriken selbst, über eine Konfigurationsdatei (*metrics\_configuration.xml*) konfiguriert, die sich im Rootverzeichnis des Plugins befindet. Neben der Anzeige der berechneten Metriken sieht unser Werkzeug auch vor, die Ergebnisse der Metrikberechnungen für ein Modell in einer Reportdatei zu speichern.

Neben der Verwendung innerhalb von EPCTools kann unser Werkzeug auch als Batchprogramm aufgerufen werden. Das ist nützlich, wenn Komplexitätsmetriken für mehrere als gespeicherte Dateien vorliegende EPKs berechnet werden sollten. Die Ergebnisse der Berechnungen sowie einige statistische Angaben über das Modell werden dann wieder in einer Reportdatei gespeichert. Die in Abschnitt 3.4 vorgestellten Schnittstellen zu Werkzeugen von Drittanbietern erlauben es ebenfalls, zusätzlich semantische Analysen des Modells durchzuführen und deren Ergebnisse ebenfalls in der Reportdatei abzuspeichern.

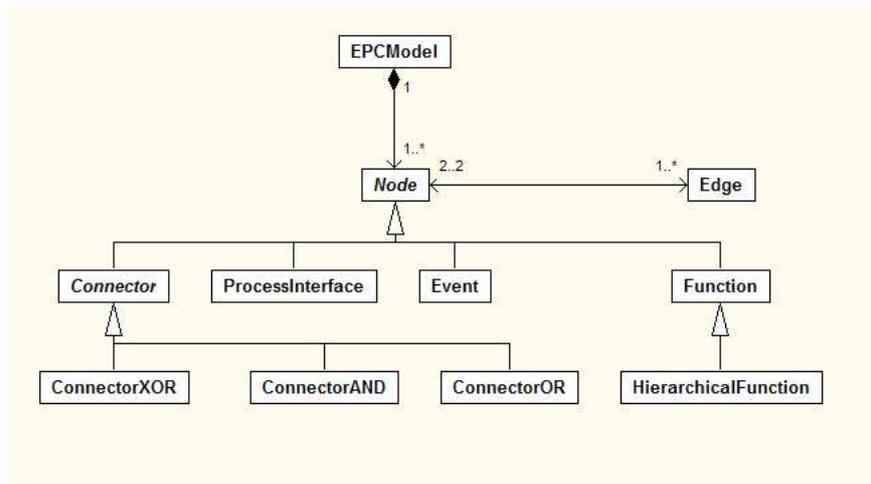


Abbildung 4: Klassenstruktur des Datenmodells

### 3.2 Datenmodell

Die Struktur des internen Datenmodells von EPCMetrics ist in Abb. 4 dargestellt. Wie aus der Abbildung ersichtlich, werden als Knotentypen Ereignisse, Funktionen, Verknüpfungsoperatoren, Prozesswegweiser sowie hierarchische Funktionen unterstützt. *EPCModel* ist die zentrale Klasse des Datenmodells. Sie stellt u.a. eine Reihe von Methoden zur Verfügung, um auf bestimmte Mengen von Knoten des Modells wie Startknoten und Verknüpfungsoperatoren zuzugreifen, die bei der Initialisierung der Datenstruktur berechnet werden. An Informationen zum grafischen Layout eines EPK-Modells werden vom Datenmodell zur Zeit nur Angaben zum Verlauf der Kontrollflusspfeile unterstützt.

Das Datenmodell eines EPK-Modells wird in einer Containerklasse verwaltet. Diese verwaltet neben dem Datenmodell ein Objekt mit syntaktischen Eigenschaften des Modells, die Resultate der Metriken für das Modell sowie optional ein Objekt mit Informationen über semantische Eigenschaften des Modells. Dargestellt ist die Architektur in Abb. 5.

### 3.3 Erweitern um eigene Metriken

Damit das Werkzeug leicht um weitere Metriken erweitert werden kann, wurde ein Interface definiert, das von jeder Metrik implementiert werden muss. Die Vererbungshierarchie ist in Abb. 6 dargestellt. Die abstrakte Klasse *AbstMetric* implementiert diverse get- und set-Methoden und wird von allen instanziierten Metrikklassen geerbt. Hauptmethode des Interfaces ist *calculateMetric(EPCModel): MetricResult*. Diese berechnet für ein gegebenes EPK-Modell die Metrik und liefert das Ergebnis in einer Containerklasse zurück.

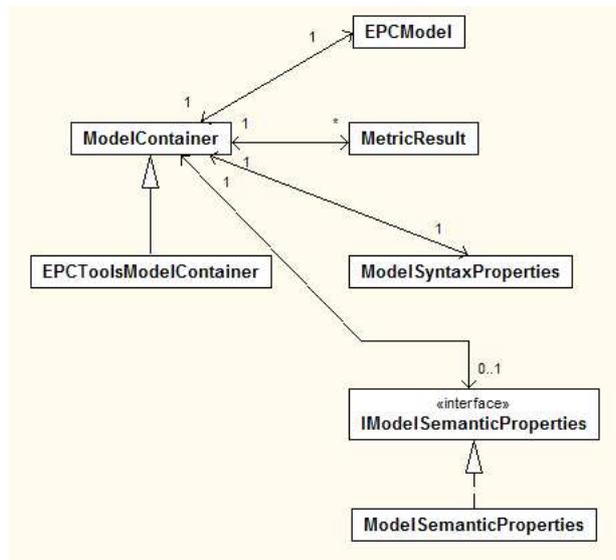


Abbildung 5: Klassenstruktur der internen Verwaltung eines Modells

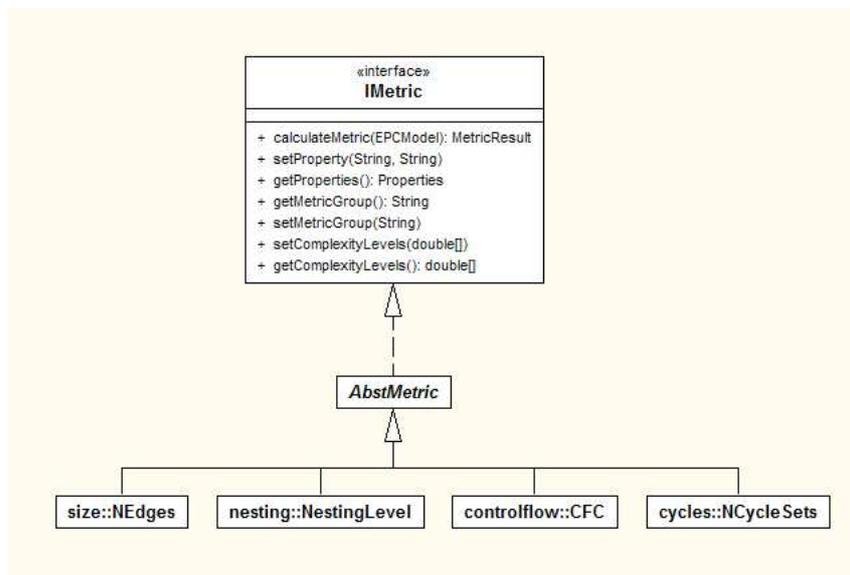


Abbildung 6: Architektur der Metrikimplementierung (Die Abbildung zeigt aus Übersichtsgründen nur einen Ausschnitt der implementierten Metriken)

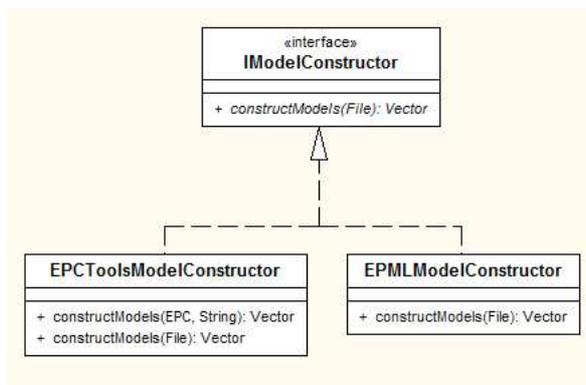


Abbildung 7: Architektur der Konstruktion der Datenmodelle

### 3.4 Schnittstellen zu anderen Werkzeugen

EPCMetrics wurde mit einer offenen Architektur entworfen, die Schnittstellen zu anderen Werkzeugen erlaubt. Die entscheidende Rolle spielen hierbei die Konstruktionsklassen. Diese sind dafür verantwortlich, EPK-Quelldateien zu lesen und in eine interne Datenstruktur umzuwandeln. Derzeit können Dateien im Austauschformat EPML[MN04] gelesen werden. Sollen weitere Dateiformate wie das vom ARIS Toolset verwendete XML-Format AML gelesen werden, muss hierfür lediglich eine entsprechende Konstruktionsklasse geschrieben werden.

Die Architektur der Modellkonstruktion ist in Abb. 7 dargestellt. Alle Konstruktionsklassen implementieren das Interface *IModelConstructor*. Dieses definiert die Methode *constructFiles(File): Vector*. Die implementierenden Klassen bilden bei Aufruf dieser Methode für die EPK-Modelle aus dem Fileobjekt der Datenquelle die intern verwendeten Datenmodelle sowie die zugehörigen Container. Die Rückgabe der Methode besteht aus den Containern der konstruierten EPK-Modelle.

Zur Zeit sind zwei Konstruktionsklassen implementiert:

1. *de.ulpz.ebus.epc.metrics.util.modelutils.EPMLModelConstructor*:  
Diese Konstruktionsklasse ist für Dateien im EPML-Format geeignet. Für jedes EPK-Modell in einer EPML-Datei wird ein eigenes Datenmodell konstruiert. Falls die EPK-Modelle durch Prozesswegweiser und hierarchische Funktionen verbunden sind, werden nur die Metriken für jedes einzelne in der EPML-Datei definierte Teilmodell berechnet. Eine Prüfung semantischer Eigenschaften eines Modells ist nicht möglich.
2. *de.ulpz.ebus.epc.metrics.util.modelutils.EPCToolsModelConstructor*:  
Diese Konstruktionsklasse ist ebenfalls für Dateien im EPML-Format geeignet. Als Parser wird hier der Parser von EPCTools[Cun04] verwendet. Dadurch wird zu-

nächst ein Modell der EPK in der von EPCTools verwendeten internen Datenstruktur erzeugt. Aus dieser wird dann ein Modell der EPK in der von EPCMetrics verwendeten Datenstruktur berechnet. Da die EPK in der EPCTools-Datenstruktur vorliegt, kann der Model-Checker der EPCTools zum Testen semantische Eigenschaften der EPK (Sauberkeit, Soundness u.a., vgl. [Cun04]) benutzt werden. Da EPCTools keine Prozesswegweiser und hierarchischen Funktionen unterstützt, kann diese Konstruktionsklasse nur mit EPML-Dateien verwendet werden, die nur ein einziges EPK-Modell enthalten.

## 4 Zusammenfassung

In diesem Beitrag haben wir ein Werkzeug vorgestellt, mit dessen Hilfe verschiedene in der Literatur vorgeschlagene Komplexitätsmetriken für ereignisgesteuerte Prozessketten berechnet werden können. Neben den wesentlichen in der Literatur vorgeschlagenen Metriken können auch eigene hinzugefügt werden. Bei einer Anwendung eines solchen Werkzeuges in der Praxis sollte man sich sinnvollerweise auf die Berechnung weniger Metriken beschränken. Überschreiten diese einen gewissen Schwellwert, der von Anwendungsfall zu Anwendungsfall durchaus variieren kann, sollte dies dem Modellierer signalisiert werden. In unserer weiteren Forschung wollen wir das Werkzeug dazu nutzen, festzustellen, wie Metriken voneinander abhängen und welchen Einfluss sie auf Fehler im Modell oder das Verstehen von Modellen haben. Dadurch soll auch die in diesem Beitrag noch unbeantwortete Frage untersucht werden, welche der vorgestellten Metriken am geeignetsten für den Praxiseinsatz sind.

## Literatur

- [AF06] Kathrin Amann und Andreas Fleischmann. Bewertung der Verständlichkeit graphischer Modelle. In *GI-Konferenz Modellierung 2006*, Seiten 281–284, 2006.
- [Amb03] Scott W. Ambler. *The Elements of UML Style*. Cambridge University Press, 2003.
- [ARGP06] Elvira Rolón Aguilar, Francisco Ruiz, Félix García und Mario Piattini. Applying Software Metrics to evaluate Business Process Models. *CLEI Electron. J.*, 9, 2006.
- [Car05a] Jorge Cardoso. Control-flow Complexity Measurement of Processes and Weyuker's Properties. In *6th International Enformatika Conference, International Academy of Sciences, 26-28 October 2005, Budapest, Hungary*, Jgg. 8, Seiten 213–218, 2005.
- [Car05b] Jorge Cardoso. How to Measure the Control-flow Complexity of Web Processes and Workflows. In *The Workflow Handbook*, Seiten 199–212, 2005.
- [Car06] Jorge Cardoso. Complexity Analysis of BPEL Web Processes. *Software Process: Improvement and Practice Journal*, to appear 2006.
- [CMNR06] J. Cardoso, J. Mendling, G. Neumann und H. Reijers. A Discourse on Complexity of Process Models. In *Proceedings of the BPM 2006 Workshops, Workshop on Business Process Design BPI 2006, Vienna, Austria, 2006*.

- [CS94] R. Chen und A.W. Scheer. Modellierung von Prozessketten mittels Petri-Netz-Theorie. *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, (107), 1994.
- [Cun04] Nicolas Cuntz. Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Diplomarbeit, Universität Paderborn, 2004.
- [GL06] Volker Gruhn und Ralf Laue. Complexity Metrics for Business Process Models. In *9th International Conference on Business Information Systems (BIS 2006)*, Klagenfurt, Austria, 2006.
- [Gro04] Richard Gronback. Model Validation: Applying Audits and Metrics to UML Models. In *Borland Conference, September 11-15, 2004, San Jose, California, USA*, 2004.
- [HM81] Warren A. Harrison und Kenneth I. Magel. A complexity measure based on nesting level. *SIGPLAN Not.*, 16(3):63–74, 1981.
- [Joh75] Donald B. Johnson. Finding All the Elementary Circuits of a Directed Graph. *SIAM J. Comput.*, 4(1):77–84, 1975.
- [Kan02] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [Lau06] Ralf Laue. [ebug.informatik.uni-leipzig.de/~laue](http://ebug.informatik.uni-leipzig.de/~laue), 2006.
- [LK02] Antti Latva-Koivisto. Finding a Complexity Measure for Business Process Models. Bericht, Systems Analysis Laboratory, Helsinki University of Technology, 2002.
- [LSW97] P. Langner, C. Schneider und J. Wehler. Ereignisgesteuerte Prozessketten und Petri-Netze. *Berichte des Fachbereichs Informatik der Universität Hamburg*, (106), 1997.
- [McC76] Thomas J. McCabe. A Complexity Measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976.
- [MMN<sup>+</sup>06] J. Mendling, M. Moser, G. Neumann, H. M. W. Verbeek, B. F. van Dongen und Wil M.P. van der Aalst. A Quantitative Analysis of Faulty EPCs in the SAP Reference Model. Bericht BPM-06-08, BPM Center Report, BPMcenter.org, 2006.
- [MN04] J. Mendling und M. Nüttgens. Exchanging EPC Business Process Models with EPML. In M. Nüttgens und J. Mendling, Hrsg., *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004*, Seiten 61–80, March 2004.
- [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.
- [RV04] Hajo A. Reijers und Irene T. P. Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In *Business Process Management*, Seiten 290–305, 2004.
- [SO00] Wasim Sadiq und Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, June 2000.
- [van99] Wil M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.