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+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+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+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$^+$06] und [GL06] diskutiert.

Analog zu der von Cardoso vorgeschlagenen Metrik betrachtet [MMN$^+$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.

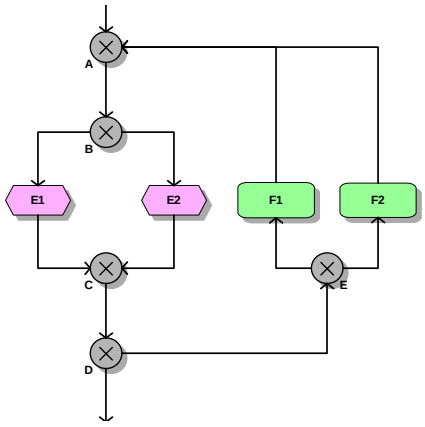

Abbildung 1: Beispiel für Zyklen

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$^+$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 eine 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.

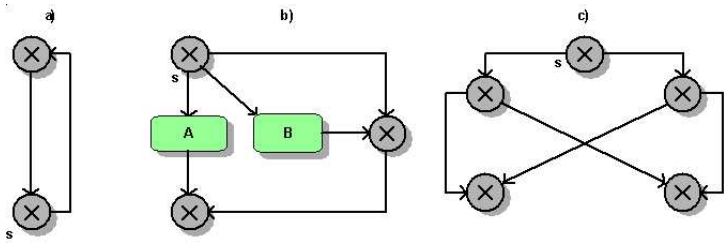

Abbildung 2: Unstrukturiertheit

2.6 Verschachtelungstiefe

Harrison und Magel[HM81] schlugen ein Software-Komplexitätsmaß vor, dass 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+06]
NEndEvents	Zahl der Endereignisse	[MMN+06] [ARGP06]
NEvents	Zahl der inneren Ereignisse	[MMN+06] [ARGP06]
NStartEvents	Zahl der Startereignisse	[MMN+06] [ARGP06]
NFunctions	Zahl der Funktionen	[GL06] [CMNR06] [MMN+06] [ARGP06]
NFunctions Connectors	Zahl der Funktionen und Konnektoren	[CMNR06] [MMN+06]
NJoinAnd	Zahl der And-Joins	[MMN+06]
NJoinOr	Zahl der Or-Joins	[MMN+06]
NJoinXor	Zahl der Xor-Joins	[MMN+06]
NJoins	Zahl der Join-Konnektoren	[MMN+06]
NSplitAnd	Zahl der And-Splits	[MMN+06]
NSplitOr	Zahl der Or-Splits	[MMN+06]
NSplitXor	Zahl der Xor-Splits	[MMN+06]
NSplits	Zahl der Split-Konnektoren	[MMN+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+06]
CFC	Kontrollfluss-Komplexität der Split-Konnektoren	[McC76] [LK02] [GL06] [Car05b] [CMNR06] [MMN+06] [Car05a]
JC	Kontrollfluss-Komplexität der Join-Konnektoren	[MMN+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 Hautpleserichtung	

Tabelle 1: Komplexitätsmetriken

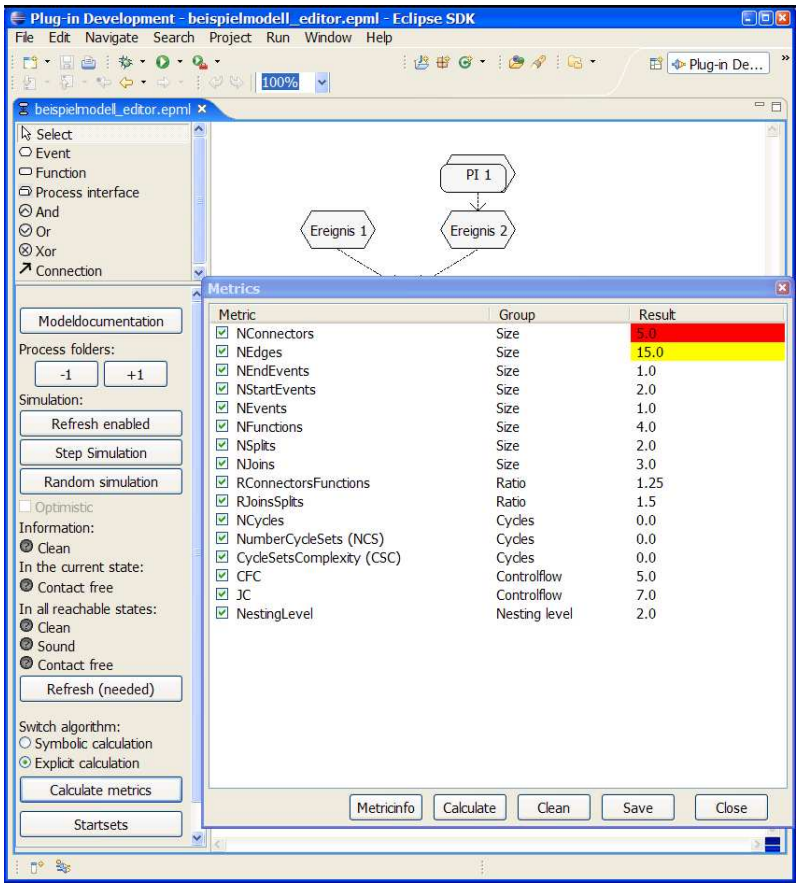

Abbildung 3: Screenshot der erweiterten Version von EPCTools

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.

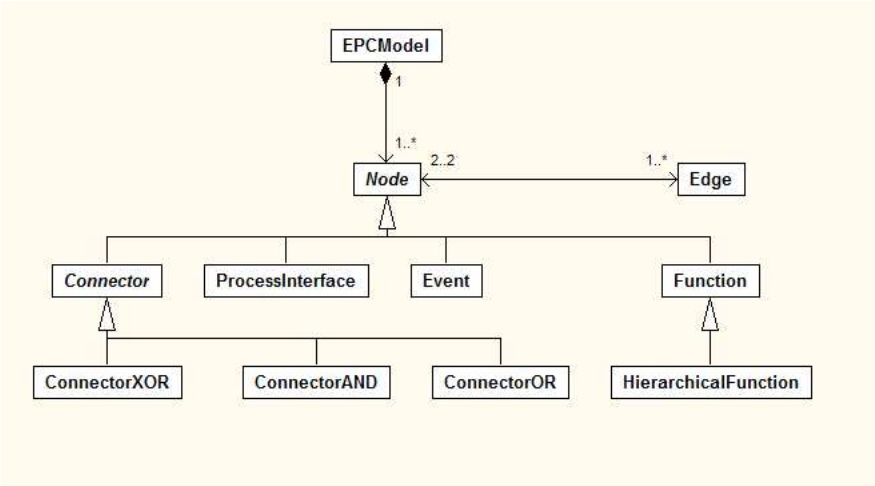

Abbildung 4: Klassenstruktur des Datenmodells

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 instanziierbaren 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.

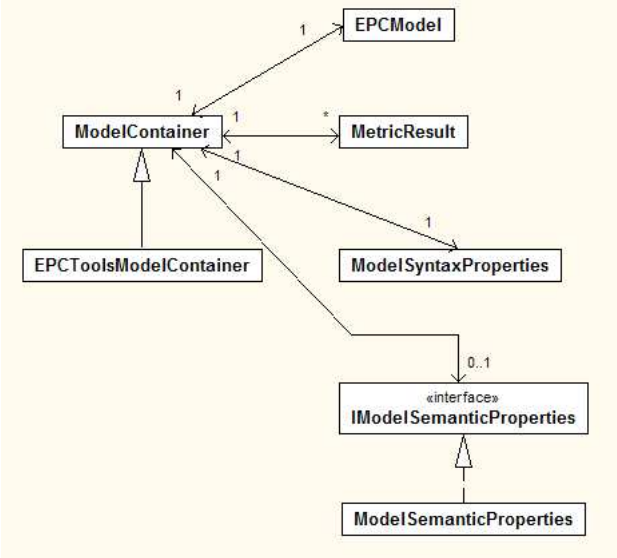

Abbildung 5: Klassenstruktur der internen Verwaltung eines Modells

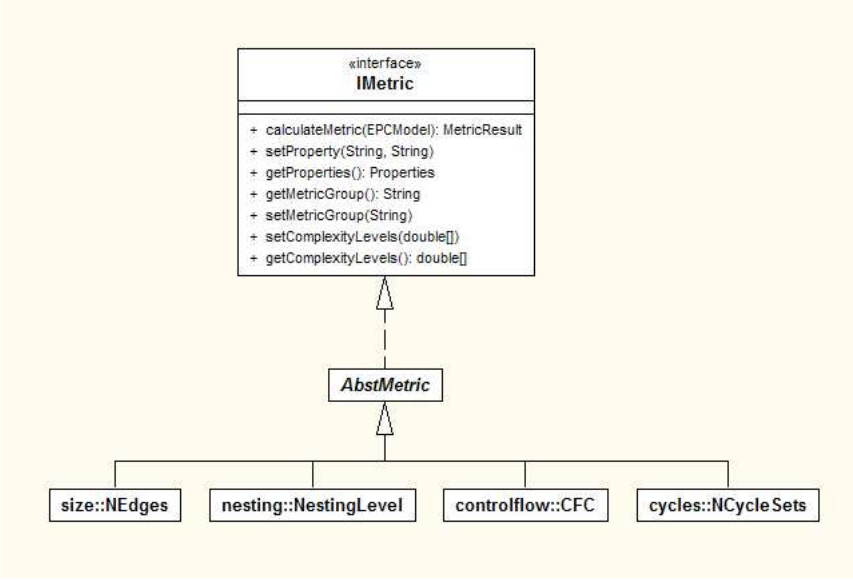

Abbildung 6: Architektur der Metrikimplementation (Die Abbildung zeigt aus Übersichtsgründen nur einen Ausschnitt der implementierten Metriken)

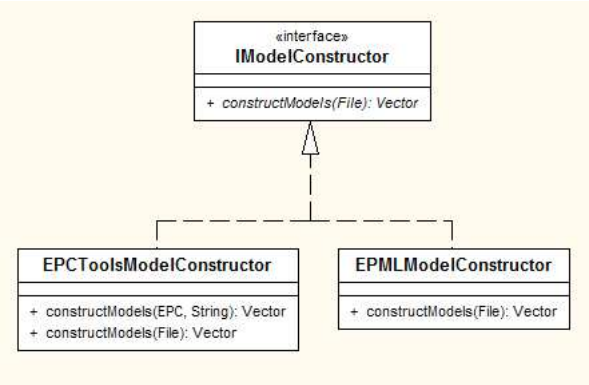

Abbildung 7: Architektur der Konstruktion der Datenmodelle

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. ebus.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+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.