Andreas Oberweis, Ralf Reussner (Hrsg.): Modellierung 2016
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 173

Konzeptionelle Modellierung ausführbarer Event Processing Networks für das Event-driven Business Process Management

Stefan Gabriel[1], Christian Janiesch[2]

Abstract: Unternehmensübergreifende Geschäftsprozesse müssen nicht nur interne, sondern auch externe Ereignisse berücksichtigen, um adäquat auf auftretende Situationen reagieren zu können. Eine kurze Reaktionszeit auf Basis von Event-driven BPM-Systemen verbessert hierbei den Entscheidungsspielraum. Die Planung derartiger Systeme auf Basis von BPM- und CEP-Technologie ist derzeit allerdings nicht ohne frühzeitige Festlegung auf proprietäre Technologie möglich. Um dem zu begegnen, schlagen wir eine Sprache und Architektur zur konzeptionellen Modellierung und Serialisierung in ausführbaren Code von CEP-Modellen für das Event-driven BPM vor. Wir zeigen, wie eine entsprechende Implementierung auf Basis einer herstellerunabhängigen CEP-Modellierungssprache und Notation, umgesetzt in der Open-Source-Modellierungs-Plattform Oryx, für die Esper Event Processing Language aussehen kann und demonstrieren diese an einem Beispiel.

Keywords: Event-driven Business Process Management, Complex Event Processing, Complex Event Processing Model & Notation, Model-driven Engineering.

1 Einleitung

Die Maßnahmen und Entscheidungen zur Reaktion auf auftretende Geschäftsvorfälle werden insbesondere durch die Latenzzeiten in der Datenübermittlung und Datenverarbeitung beeinflusst. Je mehr Zeit zwischen dem Auftreten der geschäftsrelevanten Ereignisse und der getroffenen Maßnahme vergeht, desto niedriger ist in der Regel der geschäftliche Nutzen der Entscheidung [MS10; OJ15]. Jedes Unternehmen sollte daher bestrebt sein, seine Geschäftswerte zu optimieren und die Latenzzeiten zu reduzieren. Aus Sicht der IT lässt sich dies bspw. durch den Einsatz ereignisgesteuerter Technologien im Sinne eines Business Activity Monitoring (BAM) [Ga02] oder weiterführend eines Event-driven Business Process Management (EdBPM) [Ja12, Kr14] reduzieren.

Existierende Lösungen haben aber gemein, dass sie jeweils nur einen Ausschnitt des Gesamtproblems lösen. Wesentliche Faktoren stellen Hindernisse für die Benutzung solcher Lösungen dar. Zu diesen Faktoren zählen: fehlende offene Standards [BD10], Abhängigkeit vom eingesetzten (rudimentären) Produkt [Ja12] und die Abstraktion zwischen Modellierung und technischer Umsetzung [Du13].

[1] Karlsruher Institut für Technologie, Kaiserstraße 12, 76131 Karlsruhe, stefan-gabriel@web.de
[2] Julius-Maximilians-Universität Würzburg, Juniorprofessur für Information Management, Sanderring 2, 97070 Würzburg, christian.janiesch@uni-wuerzburg.de

Für die Entwicklung von Complex Event Processing (CEP)- und EdBPM-Systemen sind weitere Ansätze bereits vorhanden: [EN11] beschreiben eine grundlegende Idee für die semantische Benutzung ereignisverarbeitender Konstrukte. [Fr12] haben dies graphisch und in Anlehnung an BPMN [OM13] für die Anwendung im BAM umgesetzt. [BD10] erläutern den theoretischen Aufbau und praktische Herangehensweise zur Einführung ereignisgesteuerter Systeme in Organisationen. Verwandte Arbeiten im Bereich der Integration von BPM(N) und CEP finden sich auch bei [De07; Ku10]. Hier geht es aber weniger um CEP im engeren Sinne als um die Verbesserung des Event-Handlings in BPMN. [Vi14] schlägt die Event Processing Model and Notation ([moby-]EPMN) vor. Es handelt sich allerdings nicht um eine geeignete Methode zur konzeptionellem, semantischen Modellierung von Event Processing Networks (EPN). Die Notation dient im Wesentlichen dazu, Konjunktion, Disjunktion, Sequenzen, und funktionale Annotationen vorzunehmen. Eine umfangreiche Übersicht zum EdBPM findet sich auch bei [Kr14].

Im vorliegenden Beitrag stellen wir eine Software-Architektur als Basis für das EdBPM vor, die es erlaubt konzeptionell und anbieterunabhängig EPN für das CEP zu modellieren und diese automatisch in entsprechenden CEP-Code zu transformieren. Wir haben dafür die Vorabreiten von [Fr12] auf CEP erweitert und eine Transformationsschicht geschaffen, die diese anbieterunabhängigen Modelle beispielhaft in Esper-Code übersetzt, der dann automatisiert zum Monitoring von Prozessen eingesetzt werden kann.

2 Konzeptionelle Modellierung von Event Processing Networks

Die grundlegende Idee zum zugrundeliegenden Meta-Modell der Modellierungssprache basiert auf [Fr12]. Die nachfolgende Spezifikation des Meta-Modells (siehe Abb. 1), die Notation und zugehörige Semantiken, Umsetzung in einer Programmiersprache und Serialisierung sind eine Weiterentwicklung dieser Grundidee.

Die Modellierung eines *EventProcessingNetwork* entspricht der Modellierung eines EPN. Ein Modell wird sequentiell erstellt und die einzelnen Ereignisse gelangen ebenso sequentiell in die entsprechenden *EventStreams*.

Alle Knoten werden über die abstrakte Klasse *FlowNode* weiter verfeinert und entsprechend alle Kanten über die abstrakte Klasse *Edge*. Als Flussknoten stehen die Adapter und alle Event Processing Agents (EPA) zur Verfügung. Diese Arten werden als abstrakte Klasse definiert. Die Klasse *EventProcessingAgent* hat drei Spezialsierungen mit weiteren Konkretisierungen basierend auf [EN11]: die konkrete Klasse *Filter*, die abstrakten Klassen *Transformation* und *PatternDetect*. Die Klasse Transformation wird von den EPAs *Aggregate*, *Translate*, *Split* und *Compose* konkretisiert. Die Klasse *Enrich* ist ein Sonderfall von Translate und erbt daher von dieser Klasse. Ein Objekt der Klasse PatternDetect beinhaltet genau ein Objekt der Klasse *Pattern*. Die Klasse Pattern spezialisiert elementare Muster durch die abstrakte Klasse *BasicPattern* und dimensionale Muster durch die abstrakte Klasse *DimensionalPattern*. Der Filter-EPA kann neben dem Filtern von Informationen auch Ereignisse projizieren. Dabei kann ein Filter-Objekt

keine bis alle Ereignisattribute projizieren und definiert dadurch einen eigenen Ereignistyp für den ausgehenden EventStream.

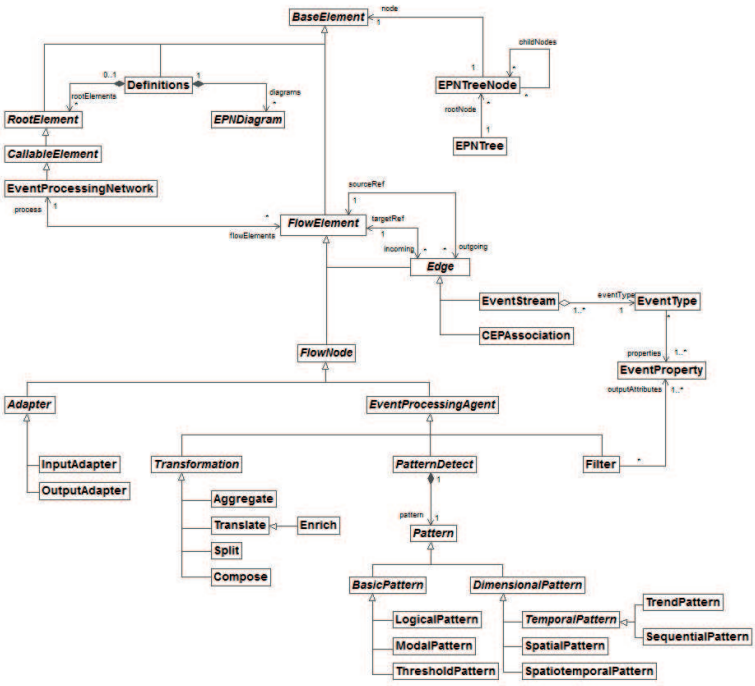

Abb. 1: Complex Event Processing Model & Notation Meta-Modell (Ausschnitt)

Die Modellierungssprache kann an BPMN angedockt werden, um EdBPM-Prozesse zu modellieren. Die folgenden wesentlichen Notationselemente in Abb. 2 geben einen Überblick darüber, wie Ereignisse verteilt, verarbeitet und benutzt werden, um Bezüge zwischen Ereignissen herzuleiten.

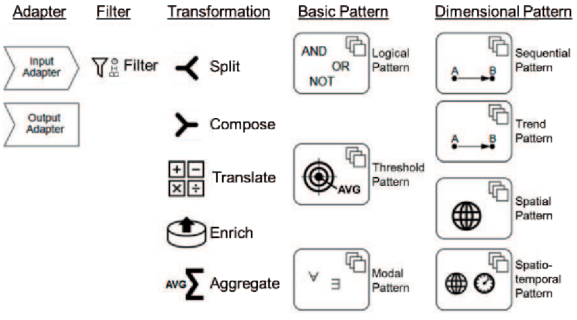

Abb. 2: Notationsübersicht (Ausschnitt)

3 Softwarearchitektur und Implementierung

Die Implementierung der Systemarchitektur folgt der üblichen Dreiteilung im CEP von Producer, Processor und Consumer [Lu02; EN11]. Wir haben dazu drei Komponenten implementiert: Die CEP-Applikation, die Event-Kommunikation und ein Modellierungswerkzeug.

Die CEP-Applikation ist der Kern der entwickelten Software und bindet die CEP-Engine und weitere Werkzeuge an. Das Projekt ist die CEP-Applikation und beinhaltet den Web Service zur Registrierung von Statements in der Esper Event Processing Language (EPL), implementiert die Schnittstellendefinition und enthält die Bereitstellungslogik des Registrierungs-Web-Services. Als CEP-Engine kommt hier Esper 5.1 zum Einsatz. Weiterhin wird eine MySQL-Datenbank betrieben, mit einem entsprechenden Connector an die Komponente angebunden und über eine eigene entwickelte Komponente zur Event-Kommunikation gekoppelt. Weiterhin beinhaltet die Applikation die Schnittstellen zu den Web Services zur Registrierung der EPL-Statements und zur Bereitstellung der Key Performance Indicators (KPI). Die Schnittstelle bietet Operationen an, um EPL-Statements an der CEP-Engine als einzelne oder gebündelte Anweisungen zu registrieren, und deren erzeugten KPI-Web-Services zu identifizieren und deren Daten bereitzustellen. Die Schnittstelle definiert wie der KPI-Web-Service seinen Wert bereitstellt.

Die Event-Kommunikation ist für die Ereignisübertragung zwischen den Ereignisproduzenten und der Ereignisverarbeitung zuständig. Sie enthält die nachrichtenorientierte Middleware (MOM) und implementiert das Publish-Subscribe-Entwurfsmuster als Event Channel. Die Komponenten benutzt die Tools Apache ActiveMQ, Apache CXF WS-Notification und Simple Logging Facade for Java (SLF4J). Als MOM wird Apache ActiveMQ eingesetzt. Die Implementierung des Publish-Subscribe-Entwurfsmuster von Apache CXF WS-Notification wurde für den speziellen Einsatz für die Software erweitert. Die MOM dient dem Entwurfsmuster zur Nachrichtenübermittlung. Das SLF4J ist ein Hilfstool für die Middleware, die Funktionalität zum Logging beinhaltet.

Als Modellierungswerkzeug wird der Oryx-Editor eingesetzt [Or12]. Der Oryx-Editor dient als Oberfläche einer ganzheitlichen Lösung, die sich von der Modellierung entsprechender Prozesse über die Serialisierung in EPL-Statements bis zur Veröffentlichung von KPIs als Web Services erstreckt. Grundsätzlich besteht der Oryx-Editor aus einem Frontend und Backend, die beide als Applikation auf einem Web-Server ausgeführt werden. Das Frontend ist über einen Browser abrufbar und bedienbar. Die Komponente benutzt die Tools Apache Tomcat 7 als Webserver, PostgreSQL Database als Datenbankserver für den Oryx-Editor und unsere CEP-Applikation. Der Web-Server und der Datenbankserver sind die Grundvoraussetzung für den Betrieb des Oryx-Editors. Das Stencil Set ist der Datensatz für die Benutzung der Modellierungssprache im Editor. Ein Stencil Set besteht aus der grafischen Darstellung der Sprachelemente, einer konkreten Beschreibung der Elemente mit Eigenschaften und Eingabemöglichkeiten und der Regeldefinition für die Sprache [Pe07]. Hierdurch wird das Meta-Modell syntaktisch als auch semantisch implementiert.

4 Serialisierung der konzeptionellen Modelle

Im Backend ist das Meta-Modell implementiert und alle Flusselemente werden als Klassen abgebildet. Diese Klassenstruktur ist eine allgemeine Zwischenstruktur des Diagramms, bevor es in die speziellen Elemente des Meta-Modells übersetzt wird. Factory-Klassen erzeugen die entsprechenden Objekte der implementierten Klassenstruktur anhand der Diagrammrepräsentation. Spezielle Klassen organisieren die Serialisierung eines Modell-Elements für die konkrete EPL der CEP-Engine Esper.

Das Frontend speichert die Diagramme anhand der Definition des entsprechenden Stencil Sets ab. Die Datenrepräsentation wird im Backend für die Konvertierung in die interne Struktur benötigt. Der Austausch der Daten zwischen Frontend und Backend erfolgt über ein Servlet. Es steuert die Konvertierung in die interne Struktur an und übersetzt die interne Struktur der Meta-Modelle in EPL. Weiterhin tritt das Servlet als Vermittler zwischen dem Frontend und Registrierungsservice der CEP-Applikation auf.

Nachdem die Modelle erzeugt wurden, müssen die Anweisungen zur Ereignisverarbeitung aus dem Modell zur CEP-Engine gelangen. Abb. 3 zeigt den Ablauf der Daten von der Modellierung bis zur Registrierung in der Engine.

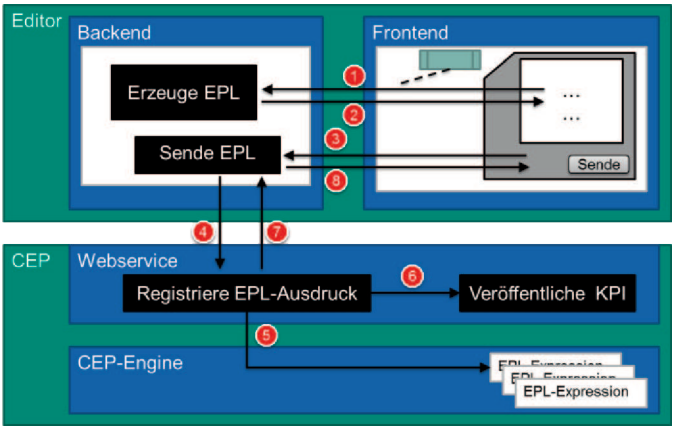

Abb. 3: Datenaustausch zwischen den Softwarekomponenten

Im Frontend des Oryx-Editors wird das entsprechende Plug-In ausgeführt und öffnet ein Dialogfenster zum Generieren der EPL. Dieses schickt die Diagrammdaten an das ein Servlet im Backend (1). Dort werden die Daten konvertiert und in entsprechende EPL-Statements übersetzt (2). Die EPL-Statements werden an das Servlet im Backend geschickt (3). Das Servlet sendet die Statements weiter an den Registrierungs-Web-Service der CEP-Applikation (4). Die Statements werden anschließend in der CEP-Engine registriert (5) und durch die modellierten KPIs als Web Service veröffentlicht (6). Danach sendet der Registrierungs-Web-Service die Adressen der erzeugten KPI-Web-Services an das Servlet zurück (7), der sie ans Dialogfenster im Frontend weiterleitet (8).

5 Implementierungsbeispiel

Das folgende Szenario beschreibt einen Kühllager-Monitoring-Prozess bei dem die Kühllagertemperatur beobachtet und bei Unregelmäßigkeiten reagiert wird (siehe Abb. 4). Die Schnelligkeit der Reaktion ist von Bedeutung, da Güter in Kühllagern verderblich sind.

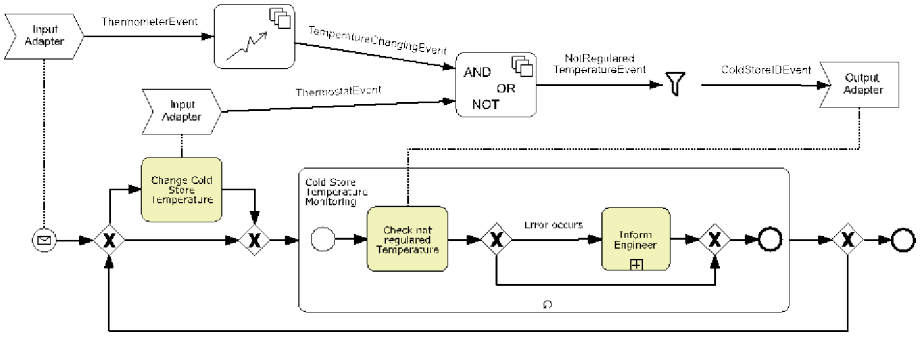

Abb. 4: Prozess *Kühllager-Monitoring* in BPMN mit CEPMN-Ergänzungen

Die Ereignisverarbeitung läuft wie folgt ab: Das Ereignis *Temperatur* des Typs *ThermometerEvent* geht in das System ein. Das *Trend*-Pattern untersucht die Temperaturereignisse auf eine streng steigende Temperatur. Ist dies der Fall, wird das letzte Ereignis des Trends als Ereignistyp *TemperatureChangingEvent* an ein *Logical*-Pattern weitergeleitet. Liegt diesem Pattern kein Ereignis *Thermostatregulierung* vom Typ *ThermostatEvent* vor, erzeugt es innerhalb des Zeitfensters ein Ereignis zur nichtregulierten Temperaturänderung vom Typ *NotRegularedTemperatureEvent*. Zum Schluss wird das Event auf die Eigenschaft *ColdStoreID* reduziert. Diese Eigenschaft ist der Wert der resultierenden KPI. Der Output-Adapter benennt mit seinem Namen den zugehörigen KPI-Web-Service. Parallel dazu wird ein Kühllager-Monitoring-Prozess in BPMN dargestellt, der durch die Echtzeitauswertungen über eine konkrete Aktivität beeinflusst werden kann. So wird deutlich an welcher Stelle im Prozess Daten ausgelesen und verwendet werden.

Die Serialisierung der einzelnen EPA und die Adapter werden in der Esper EPL generiert. Ein EPN beginnt mit mindestens einem Input- und endet mit mindestens einem Output-Adapter. Die Anweisung `INSERT INTO` gibt an, dass die Ergebnisse in einen EventStream weitergeleitet werden, der gleichzeitig neu erzeugt wird. Auf diese Weise wird der korrekte Event-Fluss im EPN zwischen den verschiedenen EPA sichergestellt.

Die Syntax `SELECT`, `FROM` und `WHERE` ist in der EPL gleichbedeutend mit den Konstrukten aus SQL. Über die Angabe `as` wird eine ausgehende Event-Eigenschaft benannt. Die Trend- und Logical-Pattern wenden eine Mustererkennung von Ereignissen und lösen dann ein Ereignis aus, wenn in ihren angegebenen Zeitfenstern die definierten Bedingungen erfüllt sind. Im Logical-Pattern heißen die Variablen gleich den Ereignis-

typen, die jeweils nach EventStreams benannt sind. Die Serialisierung hat folgende Form:

```
// this is an InputAdapter
INSERT INTO a8db2f2b
SELECT * FROM ThermometerEvent

// this is a TrendPattern
INSERT INTO 2705dfd2
SELECT {a.ColdStoreID, b.ColdStoreID} as ColdStoreID,
       {a.Temperature, b.Temperature} as Temperature
FROM pattern[every a=a8db2f2b ->
                  b=a8db2f2b(Temperature > a.Temperature)].win:
                                          time(20 second) output last

// this is an InputAdapter
INSERT INTO b9f185a9
SELECT * FROM ThermostatEvent

// this is a LogicalPattern
INSERT INTO a93f9b48
SELECT b9f185a9.EmployeeID as EmployeeID,
       b9f185a9.ColdStoreID as ColdStoreID,
       b9f185a9.Temperature as Temperature
FROM pattern[every 2705dfd2=2705dfd2 AND NOT b9f185a9=b9f185a9
            (ColdStoreID=2705dfd2.ColdStoreID)].win:time(30 minute)

// this is a Filter
INSERT INTO 9071b7fc
SELECT ColdStoreId as ColdStoreId
FROM a93f9b48

// this is an OutputAdapter
SELECT ColdStoreID as 9071b7fc
FROM 9071b7fc
```

6 Zusammenfassung und Ausblick

Existierende Produkte für das Echtzeit-Monitoring und -Management von Geschäftsprozessen bieten derzeit keinen offenen Ansatz, der die Planung und Entwicklung von EdBPM-Prozessen ohne die vorgelagerte Entscheidung für einen Anbieter proprietärer Software abbildet. Wir schlagen daher eine anbieterunabhängige Modellierung von EdBPM vor. Das Meta-Modell und die grafische Notation orientieren sich am Standard BPMN 2.0. Wir haben dabei sichergestellt, dass die Modellierung nicht nur ein Selbstzweck ist, sondern legen darüber hinaus dar, wie eine Transformation und Serialisierung vom Modell zu einem ausführbaren CEP-Statement möglich ist. Wir zeigen dies am Beispiel der Event-Verarbeitungssprache Esper EPL und der offenen CEP-Engine Esper und illustrieren das Ergebnis an einem Anwendungsbeispiel. Neben der Modellierung serialisiert die Software die resultierenden Modelle automatisiert in EPL-Statements und registriert diese in der CEP-Engine. Die spezifizierten KPI werden so automatisch und in Echtzeit über Web Services verfügbar. Während die gegenwärtige Implementierung auf

Esper basiert, ist es aber auch denkbar, vergleichbare auf SQL basierende CEP-Sprachen wie die Continuous Computation Language (CCL) oder Continuous Query Language (CQL) zu nutzen.

Literaturverzeichnis

[BD10] Bruns, R.; Dunkel, J.: Event-Driven Architectur. Springer, Berlin, 2010.

[De07] Decker, G.; Grosskopf, A.; Barros, A.: A Graphical Notation for Modeling Complex Events in Business Processes. In Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC), Annapolis, MD, 2007, S. 27.

[Du13] Dumas, M.; La Rosa, M.; Mendling, J.; Reijers, H.: Fundamentals of Business Process Management. Springer, Heidelberg, 2013.

[EN11] Etzion, O.; Niblett, P.: Event Processing in Action. Manning Publications, Cincinnati, OH, 2010.

[Fr12] Friedenstab, J.-P.; Janiesch, C.; Matzner, M.; Müller, O.: Extending BPMN for Business Activity Monitoring. In Proceedings of the 45th Hawai'i International Conference on System Sciences (HICSS), Maui, HI, 2012, S. 4158-4167.

[Ga02] Gartner Inc.: Business Activity Monitoring: Calm Before the Storm, 2002, http://www.gartner.com/resources/105500/105562/105562.pdf, Abruf am 21.10.2015.

[Ja12] Janiesch, C.; Matzner, M.; Müller, O.: Beyond Process Monitoring: A Proof-of-Concept of Event-driven Business Activity Management. Business Process Management Journal, 18 (4) 2012, S. 625-643.

[Kr14] Krumeich, J.; Weis, B.; Werth, D.; Loos, P.: Event-Driven Business Process Management: Where are we now? A Comprehensive Synthesis and Analysis of Literature. Business Process Management Journal, 20 (4) 2014, S. 615-633.

[Ku10] Kunz, S.; Fickinger, T.; Prescher, J.; Spengler, K.: Managing Complex Event Processes with Business Process Modeling Notation. In Proceedings of the 2nd International Workshop on BPMN. LNBIP Vol. 67, Potsdam, 2010, S. 78-90.

[Lu02] Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional, Boston, MA, 2002.

[MS10] zur Muehlen, M.; Shapiro, R.: Business Process Analytics. In Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture. Vol. 2 (vom Brocke, J.; Rosemann, M., Eds.), Springer, Berlin, 2010.

[OJ15] Olsson, L.; Janiesch, C.: Real-time Business Intelligence und Action Distance: Ein konzeptionelles Framework zur Auswahl von BI-Software. In Proceedings of the 12. Internationale Tagung Wirtschaftsinformatik (WI), Osnabrück, 2015, S. 691-705.

[OM13] Object Management Group Inc.: Business Process Model and Notation (BPMN) Version 2.0.2, 2013, http://www.omg.org/spec/BPMN/2.0.2/PDF, Abruf am 18.06.2015.

[Or12] Oryx Editor. https://code.google.com/p/oryx-editor/, Abruf am: 21.10.2015.

[Pe07] Peters, N.: Oryx. Stencil Set Specification, https://oryx-editor.googlecode.com/files/OryxSSS.pdf, Abruf am 21.10.2015.

[Vi14] Vidačković, K.: Eine Methode zur Entwicklung dynamischer Geschäftsprozesse auf Basis von Ereignisverarbeitung. Dissertation. Universität Stuttgart, Stuttgart, 2014.