Kopplung von zeitdiskreten, domänenspezifischen Simulationsmodellen an Prozessmodelle der BPMN 2.0

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat.
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

eingereicht am Fachbereich Informatik von

Dipl.-Inf. Philip Joschko

aus Heidelberg

Hamburg, März 2014

Tag der Disputation: 18. August 2014

Folgende Gutachter empfehlen die Annahme der Dissertation:

Name: Prof. Dr.-Ing. Bernd Page

Name: Prof. Dr.-Ing. Matthias Riebisch

Zusammenfassung

Geschäftsprozesse sind naturgemäß einer Umgebung ausgesetzt. Sie besitzen nicht nur einen Ein- und Ausgang, sondern interagieren auch zur Laufzeit mit anderen Systemen, welche mit den Mitteln der Prozessmodellierung nicht adäquat repräsentierbar sind. Dies können entweder im jeweiligem Betrieb befindliche Systeme, wie Maschinen oder Lager, oder auch externe Systeme, wie Logistiknetze oder Wettereinflüsse sein. Die Interaktionsbeziehungen zwischen Prozessen und umliegenden Systemen sind in einigen Fällen bidirektional und kybernetisch. Bei der Simulation von Geschäftsprozessen wurde diesem Aspekt bisher keine Rechnung getragen, obwohl diese Systemumgebung für eine umfassende Leistungsmessung der Prozesse branchenabhängig eine entscheidende Rolle spielen kann.

Die vorliegende Arbeit beschäftigt sich mit der Verknüpfung von in der BPMN 2.0 modellierten Prozessen mit domänenspezifischen Simulationsmodellen. Als Voraussetzung für die verknüpften Modelle wird lediglich gestellt, dass diese zeitdiskret sind und auf den gleichen Simulationskern wie die Prozesssimulation zugreifen. Zu diesem Zweck werden diejenigen Elemente der BPMN 2.0 identifiziert, die unter Einhaltung der Spezifikation der OMG erweitert werden können, um Interaktionsbeziehungen zu domänenspezifischen Systemen darzustellen. Die Verknüpfungen zu den Teilmodellen können hierdurch in bestehende Prozessdefinitionen integriert werden. Es wird erläutert, welche Arten von Interaktionsbeziehungen zwischen heterogenen Teilmodellen und Prozessmodellen grundsätzlich darstellbar und für die Simulation nutzbar sind. Eine Benutzeroberfläche wird entworfen, über welche der Anwender die Kopplung von Prozessmodellen an konkrete Teilmodelle oder Entitäten der Teilmodellen konfigurieren kann, so dass dynamische Interaktionsbeziehungen während der Simulation ermöglicht werden. Ein Mechanismus zur entsprechenden Parametrierung, Generierung und Verknüpfung der Simulationsmodelle wird vorgestellt. Hierbei werden die Prinzipien der komponentenbasierten Entwicklung angewandt. Dadurch entsteht ein Softwarerahmenwerk, welches durch Hinzufügen von Erweiterungen ohne Änderung an der bestehenden Code-Basis für verschiedene Branchenlösungen angepasst werden kann. Die Konzepte werden prototypisch mit der Rahmenanwendung Empinia und der Simulationsbibliothek DESMO-J implementiert. Anhand eines Fallbeispiels aus dem Forschungsprojekt „Systemoptimierung Offshore Wind“ wird der Prototyp und somit die vorgestellten Konzepte evaluiert.

Das Ergebnis ist ein für verschiedene Branchen adaptierbares Kollaborationsrahmenwerk. Es eignet sich zur kostengünstigen und risikolosen Leistungsmessung von Prozessen mit Hilfe von Simulationsexperimenten im Rahmen von Business Process Improvement-Projekten, bei denen domänenspezifische Teilmodelle eine entscheidende Rolle für das Laufzeitverhalten der Prozesse spielen. Die Erweiterbarkeit der Software vergrößert dabei das mögliche Einsatzfeld der Prozesssimulation im erheblichen Maße.

Abstract

Intrinsically, business processes are exposed to an environment. In addition to their input and output, they interact with other systems at runtime, which are not adequately representable with the traditional means of process modeling. These may either be internal systems applied in particular companies, such as machines or stocks, or external systems, such as logistics networks or weather conditions. In some cases, interaction relationships between processes and surrounding systems may be bidirectional and cybernetic. In business process simulation this has not been sufficiently considered so far, even though the system environment may play a decisive role for comprehensive performance measurements of processes.

This work is concerned with linking up processes modeled by BPMN 2.0 with domain-specific simulation models. The only preconditions for the linked models are that they ought to be time-discrete and that they shall refer to the same simulation core as the process simulation does. For this purpose, those elements of BPMN 2.0 are identified which can be extended while complying the specification of the OMG, to represent the interaction relationships with domain-specific systems. Hereby, links to partial models can be integrated into existing process definitions. It is elucidated, which types of interaction relationships between heterogeneous models are generally representable and utilizable for the simulation. A user interface is designed, through which the user can configure the coupling of process models with concrete partial models or entities out of these partial models in such a way that dynamic interaction relationships are enabled during the simulation. A mechanism for the corresponding parametrization, generation, and linkage of simulation models is presented. The principles of component-based development are applied consistently. From this approach, a software framework was obtained that can be adjusted for different industries by adding extensions without changing the existing code base. The concepts are prototypically implemented using the framework application Empinia and the simulation library DESMO-J. The prototype and the presented concepts are evaluated by means of a case study from the research project “System Optimization Offshore Wind”.

The result is a collaboration framework that is adaptable for various industries and that is suitable for cost-efficient and risk-free performance measurement of processes with the help of simulation experiments within business improvement projects in which domain-specific partial models play a decisive role for the runtime behavior of the processes. Thereby, the expandability of the software increases the potential range of application significantly.

Danksagung

Es gibt eine Vielzahl von Menschen an unserer Universität, denen ich aus verschiedensten Gründen zu Dank verpflichtet bin, denn sie alle habe auf ihre Art und Weise dazu beigetragen, dass ich diese Arbeit heute abgeben kann.

Prof. Dr.-Ing. Bernd Page danke ich für das entgegengebrachte Vertrauen so viele spannende Projekte bearbeiten zu dürfen, in denen ich zahlreiche Erfahrung sammelte und aus denen zahlreiche Publikationen und diese Arbeit entstanden.

Prof. Dr. Volker Wohlgemuth, Prof. Dr. Andreas Fleischer und Prof. Dr.-Ing. Matthias Riebisch danke ich für ihre konstruktive Kritik und die guten Hinweise, die mir geholfen haben, den richtigen Fokus bei meiner wissenschaftlichen Tätigkeit zu finden.

Meinen Kollegen Dr. Johannes Göbel und Andi Widok danke ich für Rat und Tat mit dem sie mir jederzeit zur Seite standen und besonders für die Entlastung in unseren Forschungs- und Lehrvorhaben in den letzten Monaten, die es mir ermöglicht hat, mich zum Schluss ganz wesentlich auf die Fertigstellung dieser Arbeit zu konzentrieren.

Meinen (teils ehemaligen) Mitdoktoranden Dr. Nicolas Denz, Arne Koors, Sven Kruse, Dr. Julia Fix, Peter Krehan und Milena Andonova danke ich aufrichtig für das Anhören und Diskutieren meiner Vorträge auf unseren Doktorandenexkursionen.

Ich danke Dr. Claudia Wyrwoll für die fröhlichen Stunden auf unserem Campus, die mich mit viel positiver Energie versorgt haben – insbesondere beim Gänsefüttern und Slacklinen an unserem kleinen Biotop. Doreen Jirak danke ich für ihr immenses Einfühlungsvermögen und die vielen guten Ratschläge, mit denen sie mich stets neu motivierte und natürlich für die Kekse und den Kaffee.

Saskia Greiner und Susanne Appel von der Hochschule Bremen danke ich für die hervorragende Zusammenarbeit in unserm FuE-Projekt SystOp Offshore Wind, welches mir die Fallstudie und wertvolle Inspiration zu dieser Doktorarbeit geliefert hat.

Meinen ehemaligen und noch aktiven studentischen Mitarbeitern Tilmann Stehle, Johannes Haan, Tim Janz, Christian Ritter, Hennes Schäfer, Fred Sun, Cornelia Mengel, Dominik Koch und Alexander Oeser danke ich für ihre hervorragende Arbeiten. Nur mit einem solch starken Team war es mir möglich, erfolgreich die vielen interessanten Projekte zu bearbeiten, die mir an diesem Ort vergönnt waren.

Dr. Dorothea Pieper danke ich, weil sie sich trotz biologischen Fachhintergrunds durch diese Arbeit eines Informatikers durchgebissen hat, um sie Korrektur zu lesen. Ich danke ihr, dass sie sich alle meine Gedanken so aufmerksam anhört, egal ob sie sich um Privates oder die Wissenschaft drehen. Und ganz besonders herzlich danke ich ihr dafür, dass sie mein Leben auf so vielfältige Weise bereichert. Schön dass es dich gibt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABPMP	Association of Business Process Management Professionals
ARIS	Architektur integrierter Informationssysteme
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
BPMI	Business Process Management Initiative
BPI	Business Process Improvement
BUIS	Betriebliches Umweltinformationssystem
CAB	Cabinet, Dateiendung
CAD	Computer Aided Design
DES	Discrete Event Simulation
DESMO-J	Discrete-Event Simulation and Modelling in Java
DLL	Dynamic Link Library, Dateiendung
EABPM	European Association of Business Process Management
EPK	Ereignisorientierte Prozessketten
FIFO	First in first out
GEF	Graphical Editor Framework
GPO	Geschäftsprozessoptimierung
HTML	Hypertext Markup Language
IoC	Inversion of Control
IEEE	Institute of Electrical and Electronics Engineers
ISO	Internationale Organisation für Normung
LIFO	Last in first out
MDI	Multiple Document Interface
MVC	Model View Controller
OMG	Object Management Group
OWP	Offshore-Windpark
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modelling Language
XML	Extensible Markup Language

1. Einführung

Nur ein Narr macht keine Experimente.

Charles Darwin.

1.1. Ausgangslage

Durch den steigenden Marktdruck der sich ausweitenden Globalisierung und immer kürzeren Produktlebenszyklen spielt das strukturierte Management von Geschäftsprozessen innerhalb von Unternehmen eine zunehmend größere Rolle (vgl. Komus (2011)). Im Zuge des Prozessmanagements sollen Prozesse kundenfreundlicher, effizienter und kostengünstiger gestaltet werden. Hierfür ist nach einer anfänglichen Analyse meist eine Umstrukturierung notwendig. Dabei können Kernprozesse, die einen direkten Nutzen für den Kunden erzeugen, Stützprozesse, welche zur Ausführung der Kernprozesse notwendig sind, und Managementprozesse, welche sich mit der Strukturierung und Leitung eines Unternehmens befassen, betrachtet werden. Grundlage der Betrachtung ist die Modellierung der Prozesse mit Hilfe geeigneter grafischer Notationen wie der *Business Process Modell and Notation 2.0* (BPMN), durch welche die sequentielle Abfolge von notwendigen Aktivitäten beschrieben werden kann. Diese Prozessmodelle dienen im ersten Schritt als Kommunikationsgrundlage und zur Wissenserhaltung im Unternehmen. Auch für kleinere und mittlere Unternehmen ist dies inzwischen ein wichtiges Thema geworden (vgl. Pingel und Schmidt (2007)). Die Modellierung eröffnet zudem eine Vielzahl weiterer Möglichkeiten, welche von der automatisierten Steuerung bis hin zur Anwendung von Analysemethoden zur Bewertung der Prozesse reichen.

Zur Umstrukturierung von Prozessen eignet sich die rein statische Betrachtung von Prozessdefinitionen nur eingeschränkt, denn für eine fundierte Verbesserung der Prozesse muss das dynamische Laufzeitverhalten betrachtet werden (vgl. Tumay (1995), S. 55). Für eine Analyse des Laufzeitverhaltens bietet sich als Alternative zur Beobachtung der real laufenden Prozesse die Simulationstechnik an. Ihr Potential ist in diesem Bereich

sehr groß, denn zu einem breiten Spektrum von Fragestellungen kann die Simulation entscheidungsunterstützende Kennzahlen liefern. Unter anderem ermöglicht sie den kurzfristigen, kostengünstigen und risikolosen Vergleich von Prozessvarianten. Für neue Prozesse können bereits vor ihrer Einführung zu erwartende Performanzkriterien bestimmt werden (vgl. Tumay (1996), S. 93). Ressourcenanalysen ermöglichen die Bestimmung der Auslastung von Mitarbeitern, Maschinen, Räumlichkeiten und weiteren Ressourcen; mögliche Engpässe werden hierbei aufgedeckt (vgl. Rücker (2008b), S.96). Die Kombination mit der Prozesskostenrechnung ermöglicht die aufwandsgerechte Zuordnung von leistungsmengenneutralen Kostenarten. Fehler in Prozessdefinitionen können aufgedeckt werden. Eine rein analytische Betrachung wird der Komplexität der vorhandenen Prozesse häufig nicht gerecht, wenn erst durch eine Betrachtung der zeitlichen Aspekte des Laufzeitverhaltens kritische Prozessbestandteile (z.B. Rückkopplungen oder temporär auftretende Engpässe) aufgedeckt werden können.

Die Simulation von Geschäftsprozessen wird daher seit den Neunziger Jahren zur Leistungsmessung von Prozessdefinitionen verwendet, um eine Entscheidungsunterstützung bei der Neuplanung von Prozessen zu bieten (vgl. Bradley et al. (1995)). Dies wurde bereits in verschiedenen Arbeiten basierend auf verschiedenen Notationen beschrieben (z.B. Schmauder und Schmidt (1998), Enstone und Clark (2006), Müller (2012)). Auch zur Simulation von in der relativen neuen Notation BPMN 2.0 dargestellten Prozessen gibt es bereits Darstellungen und Softwarelösungen (Onggo und Karpat (2011), Joschko et al. (2012)).

1.2. Problemstellung

Innerhalb eines Betriebes werden jedoch auch Systeme verwendet, die sich nicht mit Hilfe von Geschäftsprozessmodellierungssprachen beschreiben lassen. Hierzu gehören zum Beispiel die Organisationsstrukturen des Betriebs mit seinen Ressourcen wie Mitarbeitern, technischen Fertigungsanlagen sowie Lagerhaltungs- oder Logistiksystemen. Die Prozesse innerhalb einer Unternehmung interagieren mit diesen Systemen. Alle betrieblichen Prozesse sind zudem naturgemäß einer Umgebung ausgesetzt (vgl. Wohlgemuth (2005)). Der Input- und der Output von Prozessen ist im Zuge der Wertschöpfung an Lieferanten und Kunden geknüpft. Auch zur Laufzeit sind Prozesse Umgebungseinflüssen ausgesetzt. Beispiele dafür sind Interaktionen mit anderen Organisationen, die wirtschaftliche Lage des betreffenden Sektors, Umwelt- und Wettereinflüsse, Logistiknetze oder die Nutzung von externen technischen Systemen. Die Dauer der Prozesse, ihre Erfolgsaussichten und die Häufigkeit der Prozesse ist direkt abhängig vom Zustand dieser Systeme. Umgekehrt wirken die Prozesse der Unternehmung auf den Zustand solcher Systeme ein. Eine bidirektionale, kybernetische Kopplung liegt vor.

Der Einfluss dieser Umgebung auf die Prozesse kann unterschiedlich stark sein. Während der Einfluss in einigen Domänen grundsätzlich eher gering ist (etwa bei Verwaltungsprozessen einer Behörde), gibt es Domänen, die im besonderen Maße ihrer Umgebung ausgesetzt sind. Hierzu zählt unter anderem die Offshore-Windenergiebranche. Deren Instandhaltungsarbeiten wirken auf den Zustand der einzelnen Windenergieanlagen und der Zustand der Windenergieanlagen bestimmt die Notwendigkeit der Instandhaltungsarbeiten. Eine Rückkopplung zwischen Prozessen und Windenergieanlagen entsteht. Die Ausführbarkeit der Prozesse ist außerdem abhängig von der prognostizierten und gegenwärtigen Wetterlage.

Um auch solche Prozesse fundiert mit Hilfe der Simulationstechnik zu analysieren, sollten die relevanten Teilsysteme innerhalb einer Unternehmung und die Systemumgebung in die Analyse des Laufzeitverhaltens einbezogen werden. Falls eine kybernetische Beziehung zwischen Prozessen und Umgebung besteht, also die Auswirkungen der Prozesse auf die Teilmodelle zu einer Rückkopplung und somit zu einer Wirkung der Teilmodelle auf die Prozesse führen, können nur durch eine zeitgleiche Simulation realistische Ergebnisse produziert werden. Auf der Modellierungsebene müssen hierfür Modelle der Teilsysteme mit den Prozessmodellen verknüpft werden, so dass eine Interaktion zur Laufzeit der Simulation möglich wird. Um eine solche Verknüpfung der Modelle zu realisieren, sind domänenspezifische, für die jeweils benötigten Teilmodelle entworfene Erweiterungen der verwendeten Prozessmodellierungssprache notwendig. Flexible Kopplungsmechanismen müssen Interaktionsbeziehungen nicht nur mit anderen Modellen, sondern auch mit konkreten Entitäten innerhalb dieser Modelle ermöglichen.

Nicht zwingend sind die Prozessmodelle dabei selbst der Untersuchungsgegenstand. Es ist denkbar, dass ein externes System daraufhin untersucht werden soll, wie sich die Prozesse des Betriebs auf dieses System auswirken. So kann beispielsweise die erwartete Energieerzeugung des Offshore-Windparks untersucht werden, in dem die Instandhaltungsprozesse mit Hilfe von Geschäftsprozessmodellierungssprachen beschrieben werden – der Windpark selbst jedoch mit einer anderen, dafür besser geeigneten Modellierungstechnik. Die an das Prozessmodell gekoppelten domänenspezfischen Modelle werden in diesem Fall nicht mehr zur Systemumgebung degradiert, sondern sind als gleichwertiger oder gar hervorzuhebender Untersuchungsgegenstand zu verstehen.

Der Integration von domänenspezifischen Teilmodellen in die Laufzeitanalyse von Prozessen wurde bisher nicht genügend Aufmerksamkeit gewidmet. Es existieren bis dato keine Lösungen, die eine branchenabhängige Anpassung der Software erlauben, um die Integration von Teilmodellen in die Simulationsanalyse zu ermöglichen. Alle auf Geschäftsprozessnotationen beruhenden Lösungen beschränken sich auf die Simulation der Prozesse ohne eine Betrachtung der relevanten Systemumgebung. Bisher muss auf generellere Simulationswerkzeuge zurückgegriffen werden, die flexibel genug sind, um auch die Systemumgebung abzubilden. Diese bieten jedoch keine Geschäftsprozesseditoren,

mit denen existierende Prozessdefinitionen importiert oder Prozesse adäquat abgebildet werden können. Diese Lücke soll in dieser Arbeit geschlossen werden.

1.3. Zielsetzung

Diese Arbeit beschäftigt sich mit der Konzeption einer erweiterbaren BPMN 2.0-Bibliothek zur Simulation von Geschäftsprozessen unter Betrachtung ihrer Prozessumgebung. Diese Prozessumgebung soll durch heterogene Teilmodelle repräsentierbar sein, welche nicht adäquat in der BPMN 2.0 beschrieben werden können. Zur Verknüpfung von Prozessen mit Teilmodellen werden diejenigen Elemente der BPMN 2.0 identifiziert, welche sinnvoll domänenspezifisch erweitert werden können, um eine Kopplung im Sinne der Simulation zu ermöglichen. Ein Konzept zur Integration von domänenspezifischen Erweiterungen wird bereitgestellt. Zielsetzung dieser Arbeit ist es somit, Konzepte für domänenspezifische Softwarelösungen zur Simulation von Geschäftsprozessen zu erarbeiten.

Die Teilmodelle, die die Systemumgebung eines Prozesses repräsentieren, können sehr unterschiedlich strukturiert sein. Über die interne Struktur der jeweiligen Teilmodelle ist bei der Erstellung dieser Arbeit nichts bekannt, sie sind bei der Erstellung des Konzeptes als Black Box anzusehen. Die in Kapitel 5 beschriebene ereignisorientierte Schnittstelle zur Interaktion setzt lediglich voraus, dass die Modelle auf der Zeitachse diskretisierbar sind.

Die Editoren, die zur Manipulation der domänenspezifischen Teilmodelle verwendet werden, können dabei so unterschiedlich ausfallen, wie die Teilmodelle selbst. Generische Lösungen zur Beschreibung von Teilmodellen basierend auf Notationen wie Unified Modeling Language (UML) oder Discrete Event System Specification (DEVS) sind zwar denkbar, erfordern vom Anwender aber die zusätzliche Einarbeitung in die entsprechende Notation und sind daher weniger geeignet, um entsprechende Modelldefinitionen gemeinsam mit Domänenexperten zu validieren. Ein generischer Ansatz zur Beschreibung der Teilmodelle wird in dieser Arbeit nicht verfolgt. Stattdessen sollen Entwickler Erweiterungen einbinden können, die in der entsprechenden Branche bereits bekannt sind (z.B. Geoinformationssysteme zur Definition von räumlichen Daten). Anstelle von existierenden Komponenten können auch auf die jeweilige Domäne zugeschnittene Editoren zur Verfügung gestellt werden. Je nach Anwendungsfall sind graphische Editoren (zum Beispiel in der speziellen Variante der graphbasierten Editoren) oder auch formularbasierte Editoren geeignet. Der Modellierer soll hier nicht mehr die Struktur des jeweiligen Domänenmodells beschreiben, sondern nur die benötigten Parameter und die genaue Ausprägung seiner Modellinstanz. Auch der Import von bestehenden Datensätzen aus

betrieblichen Datenbanken wird mit einer solch speziell anzupassenden Editorkomponenten möglich, was eine wesentliche Voraussetzung für die Durchführbarkeit von Simulationsstudien in der Praxis ist.

Diese Teilmodelle sollen mit den Prozessmodellen verknüpfbar sein, um die Interaktion zur Laufzeit der Simulation zu ermöglichen. Die Verknüpfung der Teilmodelle stellt dabei Herausforderungen auf drei Ebenen:

- Auf *Modellierungsebene* werden mögliche Erweiterungspunkte für die BPMN 2.0 identifiziert. Dabei soll die Spezifikation der BPMN 2.0 strikt eingehalten werden, so dass die Austauschbarkeit der verwendeten Modelleditoren gewahrt bleibt. Die Erweiterungspunkte ermöglichen die Bereitstellung von domänenspezifischen Modellelementen, welche Schnittstellen zu den heterogenen Simulationsmodellen repräsentieren.
- Um *Simulationsexperimente* durchführen zu können, muss identifiziert werden, welche Arten von Interaktion zwischen einem Prozessmodell und seiner Umgebung erforderlich sind und durch entsprechende Prozesselemente dargestellt werden können. Es muss geklärt werden, auf welche Art der Benutzer diese Verknüpfungen näher beschreiben kann oder muss, um zur Laufzeit der Simulation eine dynamische Interaktion zu ermöglichen. Die Simulation von Geschäftsprozessen in der für sie relevanten Systemumgebung soll hiermit ermöglicht werden.
- Mit Hilfe der *komponentenbasierten Software-Entwicklung* soll ein adaptives Rahmenwerk bereitgestellt werden, welches über Erweiterungspunkte die Entwicklung von domänenspezifischen Branchenlösungen ermöglicht. Die bestehende Code-Basis des Rahmenwerkes soll dabei nicht angepasst werden müssen. Stattdessen soll lediglich durch Hinzufügen von neuen Komponenten die Kopplung mit weiteren Modelltypen ermöglicht und damit die Einsatzmöglichkeit der Software vergrößert werden.

Die Möglichkeit zur Leistungsmessung von Prozessdefinitionen mittels Simulation unter Einbeziehung der domänenspezifischen Systemumgebung wird damit geschaffen. Ein erweiterbares Rahmenwerk zur Implementation von Branchenlösungen wird somit bereitgestellt.

Um den Fokus dieser Arbeit auf diese Verknüpfung von Prozessmodellen mit domänenspezifischen Teilmodellen legen zu können, werden zwei grundlegende Annahmen getroffen: Zunächst sollen die betreffenden Teilmodelle diskretisierbar auf der Zeitachse sein, da die BPMN eine ereignisbasierte Repräsentation von Prozessemodellen ist. Die Kopplung von kontinuierlichen mit zeitdiskreten Modellen wird in anderen Arbeiten betrachtet. Des Weiteren wird davon ausgegangen, dass alle Teilmodelle Zugriff auf denselben Simulationskern haben und sich somit eine Ereignisliste teilen. Die Kopplung von

unterschiedlichen Simulationskernen oder auch die verteilte Ausführung von Simulationsexperimenten wird ebenfalls in anderen Arbeiten behandelt.

1.4. Struktur der Arbeit

Diese Arbeit beginnt mit einer Einführung in die Geschäftsprozessmodellierung. Die Modellierung ist der Simulation grundsätzlich vorgelagert und bringt eigene Anforderungen und Zielsetzungen mit sich. Ein Betrieb wird als System betrachtet und an diesem Beispiel an das Thema Modellierung herangeführt. Das übergeordnete Thema Business Process Management und die Personen, die mit Geschäftsprozessen und deren Modellen in Berührung kommen, beschrieben. Das Kapitel 2 schließt mit einer Betrachtung der Notation BPMN 2.0, sowie ihrer Grenzen und Erweiterungsmöglichkeiten.

In Kapitel 3 wird die Analysemethode Simulation vorgestellt. Nach einer kurzen Erläuterung der ereignisdiskreten Simulation werden das Potential und die möglichen Ziele der Geschäftsprozesssimulation benannt. Die historische Entwicklung der Geschäftprozesssimulation seit Beginn der Neunziger Jahre und die für diese Arbeit entscheidenden Meilensteine werden erläutert. Nach eine kurzen Vorstellung von einigen ausgewählten Simulationswerkzeugen, wird eine BPMN-Simulationsbibliothek und somit die grundsätzliche Funktionsweise der BPMN-Simulation vorgestellt. Diese Bibliothek wurde federführend vom Autor dieser Arbeit entwickelt und dient als Grundlage für die prototypische Implementation der hier vorgestellten Konzepte.

Verschiedene Ansätze zur Kopplung und Erweiterung von Modellen werden in Kapitel 4 untersucht. Hierzu werden Beispiel aus der Modellierung, aus der Softwaretechnik und aus der Simulation herangezogen. Verwandte Arbeiten werden beleuchtet, um Anregungen und Anforderungen zu übernehmen. Die komponentenbasierte Entwicklung als Ansatz, Erweiterungen für bestehende Anwendungen softwaretechnisch umzusetzen, wird erläutert und exemplarische Anwendungen aus der Simulation werden aufgeführt.

Das Kernstück dieser Arbeit bildet Kapitel 5. Zunächst wird analysiert, in welchen Beziehungen ein Geschäftsprozess zu seiner Umgebung stehen kann. Die Möglichkeiten, dies generell abbilden zu können werden sondiert. Die Notation BPMN 2.0 wird auf ihre Möglichkeiten hin untersucht, diese verschiedenen Arten von Beziehungen eingängig abbilden zu können. Die entwickelten Konzepte zur Erweiterung der BPMN 2.0 mit dem Ziele der Herstellung von Interaktionsbeziehungen mit domänenspezifischen Teilmodellen werden hergeleitet. Die Minimalanforderungen zur Einbettung von domänenspezifischen Teilmodellen in ein Simulationsrahmenwerk werden benannt. Neben der Erweiterung auf Modellierungsebene werden diejenigen Mechanismen identifiziert und erläutert,

welche eine Verknüpfung der Modelle zur Laufzeit der Simulation realisieren. Das Vorgehen zur Erstellung einer domänenspezifischen Erweiterungsbibliothek mit der Zielsetzung der Durchführung von Simulationsexperimenten und die dafür vorgesehenen Rollen werden erläutert.

In Kapitel 6 werden die Konzepte prototypisch mit einem Plug-in-Rahmenwerk umgesetzt. Die zur Verfügung stehenden Mechanismen des Rahmenwerks und deren Nutzung für eine konkrete Implementation der zur Realisierung der Erweiterungsmechanismen notwendigen Klassen werden vorgestellt. Die exemplarischen Quellcodeauszüge im Anhang werden erläutert. Ein Ansatz für eine Benutzeroberfläche und dessen Umsetzung wird aufgezeigt.

An der prototypische Implementation wird eine Fallstudie durchgeführt. Kapitel 7 beschreibt ein Anwendungsbeispiel aus der Offshore-Windernergiebranche, dessen Anforderungen dem Forschungs- und Entwicklungsprojekt „System Optimierung Offshore Wind“ entnommen wurden. Die Anforderung an zwei Erweiterungsbibliotheken zur Integration von Windpark- und Wettermodellen und deren Umsetzung werden beschrieben. Erneut werden Quellcodeauszüge und die erstellte domänenspezifische Benutzeroberfläche vorgestellt. Die Praxistauglichkeit der Konzepte wird anhand eines Beispielprozesses zur Instandsetzung von Windkraftanlagen evaluiert. Schließlich werden noch zwei weitere Erweiterungen, die auf den Konzepten dieser Arbeit beruhen, vorgestellt, um die Flexibilität des Ansatzes aufzuzeigen.

Die Ergebnisse der Arbeit werden in Kapitel 8 diskutiert. Die innovativen Beiträge werden zusammengefasst. Die Grenzen der Arbeit werden kritisch beleuchtet und Anwendungsmöglichkeiten aufgezeigt. Die Arbeit schließt mit einem Ausblick auf Aspekte, die in künftigen Arbeiten zu untersuchen sind.

An einigen Stellen im Text werden Methoden-, Klassen- und Paketnamen benannt. Diese werden durch eine nichtproportionale Schriftart kenntlich gemacht: `Beispieltext für einen Quellcode-Bestandteil`

Wenn Begriffe eingeführt werden, die besonders entscheidend für den betreffenden Absatz sind, werden diese in einer kursiven Schriftart dargestellt: *Beispieltext für eine Begriffseinführung*

2. Geschäftsprozessmodellierung

Das Kapitel erläutert relevante Begriffe aus dem Umfeld der Geschäftsprozessmodellierung, die zum Verständnis der Arbeit notwendig sind. Es beschreibt dabei diejenigen motivierenden Aspekte, die zu den hier gewählten Ansätzen geführt haben. Zunächst wird der Begriff System anhand eines Betriebes erläutert. Hierauf aufbauend werden die Grundlagen der Modellierung beschrieben. Anschließend werden Geschäftsprozesse als Teil des Betriebes definiert und betrachtet. Um die Motivation und die Ziele der Geschäftsprozessmodellierung zu verdeutlichen, wird auch das übergeordnete Thema Prozessmanagement angesprochen. Letztlich werden die Anforderungen an Modellierungssprachen für Geschäftsprozesse dargestellt. Es wird begründet, warum die Wahl in dieser Arbeit auf die BPMN 2.0 fiel und was die hier relevanten Aspekte dieser Notation sind.

2.1. Modellierung

Dieser Abschnitt erläutert zunächst den Begriff System mit besonderem Augenmerk auf den Betrieb. Anschließend wird der Begriff der Modellbildung erläutert. Dies ist notwendig, um sich dem Themengebiet der Geschäftsprozessenmodellierung, welches erst im nächsten Abschnitt beschrieben wird, fundiert zu nähern.

2.1.1. System und Betrieb

Ein System ist eine Komposition aus miteinander in Interaktionsbeziehung stehenden Elementen. Diese Elemente haben Eigenschaften bzw. Attribute. Sofern diese veränderlich sind, werden sie als Zustandsvariablen bezeichnet. Eine Interaktionsbeziehung liegt vor, „wenn der Zustand eines Elementes den Zustand eines anderen Elementes kausal beeinflusst“ (Page (1991), S.2f). Die Elemente eines Systems können selbst wiederum Systeme sein. Bossel nennt ein Objekt ein System, wenn es bestimmte Bedingungen erfüllt:

1. Das Objekt erfüllt eine bestimmte Funktion, d.h. es läßt sich durch einen Systemzweck definieren.

2. Das Objekt besteht aus einer bestimmten Konstellation von Systemelementen und Wirkungsverknüpfungen (Relationen).

3. Das Objekt verliert bzw. verändert seine Systemidentität, wenn seine Systemintegrität zerstört wird.

(vgl. Bossel (1992), S.16)

Bei dynamischen Systemen verändert sich der Zustand der Elemente über die Zeit. Durch Wirkungsverknüpfungen zwischen den Elementen eines dynamischen Systems kann es zu sogenannten Rückkopplungen kommen. Diese liegen vor, wenn ein Element seinen Zustand direkt selbst beeinflusst, oder es den Zustand eines anderen Elementes so beeinflusst, dass dessen eventuell zeitlich versetzte Reaktion wiederum eine Zustandsänderung am ursprünglichen Element bewirkt. Liegt eine solche Rückkopplung vor, so wird es als kybernetisch bezeichnet (vgl. Page (1991), S.3).

Ein Betrieb ist ein dynamisches System. Der Systemzweck ist die Wertschöpfung des Betriebes. Die Systemelemente sind einzelne Abteilungen wie Buchhaltung, Produktion, Lager, Personalabteilung, Versand, Einkauf oder Marketing. Deren Systemelemente sind wiederum Mitarbeiter und technische Hilfsmittel. All diese Systemelemente führen Prozesse aus, die Auswirkungen auf andere Systemelemente haben. Da deren Reaktion sich auf die ursprünglich handelnden Akteure auswirken kann, handelt es sich um ein kybernetisches System.

In Abbildung 2.1 wird das System „Betrieb“ schematisch als gerichteter Graph dargestellt. Die Knoten stellen die einzelnen Systemelemente und somit Abteilungen dar; die Kanten repräsentieren die Relationen innerhalb des Systems. Letztere beschreiben die Beziehungen und die Richtung der Einflussnahme zwischen den Elementen.

Ein System hat eine eindeutig definierte Systemgrenze. Wenn ein Einfluss von der Außenwelt oder auf die Außenwelt (der sogenannten Systemumgebung) vorliegt, so wird dies entsprechend als Systemeingang und Systemausgang beschrieben und das System als offen bezeichnet. Ist es hingegen ohne Einflussnahme von der Systemumgebung abgeschottet, handelt es sich um ein geschlossen System. Reale Systeme sind in der Regel immer offen und können nur durch starke Abstraktion als geschlossen angesehen werden (vgl. Page (1991), S.3).

Grundsätzlich ist ein Betrieb ein offenes System, da zur Wertschöpfung eine Interaktion mit der Außenwelt erfolgen muss. Ein- und Ausgänge sind im Zuge der Wertschöpfung die Kunden und Lieferanten. Aber auch mit Behörden oder Branchenverbänden muss in der Regel interagiert werden, denn eine weitere Zielsetzung des Unternehmens ist die Einhaltung von Regeln und Gesetzesvorgaben.

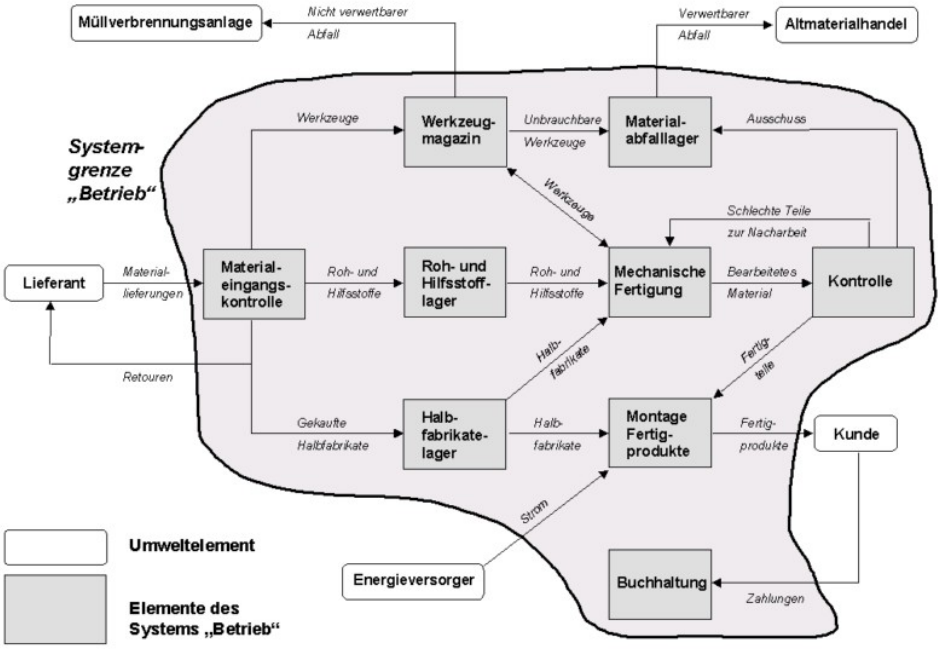


Abbildung 2.1.: „Ein Betrieb als System am Beispiel einer Fabrikation“. Abbildung aus: Wohlgemuth (2005), S. 55

Es gibt noch weitere Ein- und Ausgänge bei einem Betrieb: Jedes Unternehmen ist den aktuellen Markt- und Arbeitsmarktbedingungen ausgeliefert. Der Absatz der eigenen Produkte wird schwieriger, wenn der Konkurrenzdruck durch Mitbewerber steigt. Die eigenen Arbeitskräfte können teurer werden, wenn zum Beispiel die Qualität der Ausbildung an den Universitäten sinkt und dadurch weniger entsprechend ausgebildete Leute verfügbar sind. Politische Unruhen können Wirtschaftskrisen auslösen. Viele Betriebe sind Wetterbedingungen ausgesetzt. Das bei besonders starkem Wintereinbruch die Mitarbeiter zu spät zur Arbeit kommen, ist ein Einfluss von dem in der Regel abstrahiert werden kann. Für Unternehmen aus der Agrarindustrie oder für Energieerzeugungsunternehmen, welche Windkraft- oder Solaranlagen betreiben, ist der direkte Einfluss des Wetters auf die Erreichung der Unternehmensziele jedoch ein entscheidender Faktor. Auch in der Gastronomie, für Getränkehersteller und -lieferanten ist die aktuelle Wetterlage entscheidend für die Auftragslage.

2.1.2. Definition Modelle

Durch die Identifikation eines Systems und seiner Systemgrenzen kann ein Problembereich aus der Realität eingegrenzt werden. Dies ist der erste Schritt zur Modellbildung (vgl. Wohlgemuth (2005), S.12). Ein Modell ist eine für einen bestimmten Zweck erstellte, vereinfachte Abbildung eines Systems. Das abgebildete System wird als Original bezeichnet.

Die folgenden Merkmale kennzeichnen ein Modell:

- Das Abbildungsmerkmal besagt, dass ein Modell selbst ein System ist, welches das Originalsystem nachbildet, ihm aus einer gewissen Perspektive also hinreichend ähnlich ist. Dabei findet eine Abstraktion und Idealisierung statt.

- Das Verkürzungsmerkmal besagt, dass ein Modell ein Original nicht vollständig repräsentiert, sondern nur einen ausgewählten Ausschnitt. Dieser Ausschnitt ist von der Zielsetzung der Modellbildung abhängig, aber auch von der subjektiven Sichtweise des Modellierers.

- Das pragmatische Merkmal besagt, das die Art des Modells, der modellierte Ausschnitt aus dem Original und der Detaillierungsgrad jeweils am Zweck des Modelles ausgerichtet sind. Somit kann es von einem Original viele verschiedene Modelle geben, die für ihren jeweiligen Zweck repräsentativ für das System stehen.

(vgl. Häuslein (1993), S.8)

Modelle werden mit unterschiedlichen Zielsetzungen erstellt eingesetzt werden (vgl. Horton (2003)). Je nach Zielsetzung muss das Modell bestimmte Details enthalten, während von anderen abstrahiert werden kann.

Modelle helfen dabei ein Originalsystem zu erklären, es besser verständlich zu machen oder verstehen zu lernen. Sie können als Trainings- oder Ausbildungsgrundlage genutzt werden, um eine Person mit einem bestimmmten System vertraut zu machen. Bei Entwurf von neuen oder der Veränderung bestehender Systeme werden Modelle als Gestaltungshilfe eingesetzt, um die Planungsprozesse zu unterstützen. Ebenso werden sie eingesetzt, um Prognosen über ein System und seine zukünftigen Zustände anzustellen und sein Verhalten unter bestimmten Bedingungen abzuschätzen. Schließlich werden sie genutzt, um Eingangsgrößen und Parameter zu finden, die in einem möglichst optimalen Systemzustand oder -verhalten münden sollen. Das Originalsystem soll letztendlich auf diese Weise optimiert werden. (vgl. Page und Kreutzer (2005), S.6f; Wohlgemuth (2005), S.17; Klückmann (2009), S. 15)

2.1.3. Kategorisierung von Modellen

Unabhängig von ihrer Zielsetzung, lassen sich Modelle anhand von verschiedenen Kriterien kategorisieren:

Page (1991) klassizifiert Modelle nach Art der Zustandsübergänge. Er unterscheidet zunächst statische Modelle ohne Zustandsübergänge von dynamischen Modellen mit Zustandsveränderungen. Diskrete Modelle ändern ihre Zustände immer schlagartig zu einem bestimmten Zeitpunkt, während kontinuierliche Modelle fließende Übergänge besitzen. Bei deterministischen Modellen folgt auf einen gegebenen Zustand immer ein bestimmter Folgezustand. Bei stochastischen Modellen sind diese Zustandsübergänge zusätzlich abhängig von zufälligen Einflüssen.

Das Abbildungsmedium, welches für ein Modell gewählt wird, hängt nicht mit der realen Beschaffenheit des Originals zusammen. Materielle Modelle gibt es zum Beispiel in der Automobilindustrie. Dort werden Modelle aus Holz gebaut, um die Form eines neuen Autos im Windkanal zu testen. Für eine rechnergestützte Systemanalyse werden immaterielle, formalisierte Modelle benötigt. Diese lassen sich unterteilen in analytische Modelle und Simulationsmodelle. Für diese Arbeit relevant sind die im folgenden Kapitel beschriebenen Simulationsmodelle. Formalisierte Modelle können durch Algorithmen beschrieben werden. Geschäftsprozesse lassen sich gut mit Hilfe von grafischen Notationen darstellen (vgl. Abschnitt 2.2.5). Sofern die graphischen Modelle ausreichend formalisiert sind, können auch sie durch geeignete Softwarewerkzeuge (z.B. CAD-Systeme) zur Weiterverarbeitung genutzt werden.

Wenn es sich um dynamische Modelle handelt, deren Zustandsübergänge computergestützt berechnet werden, können diese nach Art der Ausführung klassifiziert werden. Sie können auf einem Prozessorkern, auf mehreren Kernen (Multi-Threading) oder verteilt im Netz (vgl. Kunert (2010)) berechnet werden.

In dieser Arbeit werden dynamische, zeitdiskrete und stochastische Modelle betrachtet. Es handelt sich im Computermodelle, welche sowohl mit Hilfe von graphischen Notationen (betrifft die Geschäftsprozesse) als auch programmatisch mit beliebigen Algorithmen definiert werden (betrifft die domänenspezifische Umgebung). Es wird zudem nur die Errechnung auf einem Prozessorkern betrachtet, welche (für die Simulation typisch, vgl. Kapitel 3) nicht realzeitsynchron erfolgen muss.

2.2. Geschäftsprozesse

2.2.1. Definition Geschäftsprozess

Ein wichtiger Bestandteil eines Betriebes sind die betrieblichen Prozesse. Nach DIN EN ISO 9000-2000 3.41 besteht ein Prozess „aus mehreren in Wechselbeziehung oder Wechselwirkung stehende Tätigkeiten, welche Eingaben in Ergebnisse umwandeln. Die Eingaben für einen Prozess sind üblicherweise Ergebnisse aus anderen Prozessen. In der Regel werden die Prozesse in einer Organisation geplant unter beherrschten Bedingungen durchgeführt, um Mehrwert zu schaffen.“

Unter einer Organisation sind sowohl gewinnorientierte Unternehmen zu verstehen, als auch Non-Profit-Organisationen, wie zum Beispiel Vereine, Stiftungen, Verwaltungen oder Behörden. Der Begriff „Organisation“ ist allerdings doppeldeutig, da er neben der Institution selbst auch auf die funktionale Ordnung innerhalb einer Institution abzielen kann. Um Missverständnissen vorzubeugen wird daher in dieser Arbeit bevorzugt der Begriff „Unternehmen“ verwendet. Dies soll aber andere Formen von Organisationen nicht ausschließen, denn auch sie können Vorteile aus den hier genannten Konzepten erzielen.

Ein *Geschäftsprozess* (engl.: Business Process) ist eine Menge von Tätigkeiten, die ein bestimmtes Unternehmensziel verfolgen. Er generiert durch seine Ausführung immer ein konkretes Ergebnis, hierbei werden i.d.R. neue Daten oder Dokumente generiert. Das Ergebis wird als Output bezeichnet. Nicht immer wird das angestrebte Prozessziel erreicht. In diesem Kontext ist auch der Abbruch eines Prozesses als Ergebnis zu betrachten. Prozesse werden durch ein jeweils zu spezifizierendes Ereignis, wie z.B. dem Eingang einer Nachricht, ausgelöst. Es wird als Input bezeichnet und übermittelt i.d.R. bestimmte Daten oder Dokumente, auf denen der Prozess arbeitet. Wenn der Auslöser eines Prozesses auch der Empfänger des Ergebnisses ist, wird von einem End-to-End Prozess gesprochen (vgl. EABPM (2009)).

Die Tätigkeiten stehen in einer Beziehung zueinander. Sie können sequentiell, nebenläufig oder hierarchisch von Menschen oder Maschinen ausgeführt werden. Ihre Abarbeitung verursacht monetäre Kosten und verbraucht Zeit. Aber auch weitere Kostenarten, wie der Verbrauch von Material, sind üblich. Auch die einzelnen Tätigkeiten haben immer einen Input und einen Output. Die erste Tätigkeit in einem Prozess erhält den Input des Prozesses. Die letzte Tätigkeit erzeugt den Output des Prozesses. Die Tätigkeiten dazwischen erzeugen Zwischenergebnisse.

In einem Unternehmen laufen Prozesse wiederholt nach dem gleichen Schema ab. Der Begriff Prozess umfasst daher sowohl die Definition des entsprechenden Prozesses, als auch die ausgeführte Instanz. Um klarer herauszustellen was gemeint ist, können die

Begriffe Prozessinstanz für den tatsächlich ausgeführten Prozess und Prozessdefinition für das Modell verwendet werden.

Innerhalb eines Unternehmens kann zwischen Kern-, Stütz- und Managementprozessen unterschieden werden (vgl. Seidlmeier (2010), S.3). Kernprozesse dienen der eigentlichen Wertschöpfung des Unternehmens. Sie erzeugen einen direkten Nutzen für den Kunden, wie zum Beispiel ein Produktionsprozess. Stützprozesse sind zur Ausführung der Kernprozesse notwendig, erzeugen aber selbst keinen direkten Kundennutzen (zum Beispiel Buchhaltungsprozesse). Managementprozesse dienen der Strukturierung der Abläufe in einer Organisation, zum Beispiel die Strategiefindung oder das Projektmanagement. Häufig sind Managementprozesse nur einmalig ausgeführte Prozesse, während Kern- und Stützprozesse ständig wiederholt werden und teilweise auch mehrere Instanzen parallel zueinander laufen (vgl. Schwarz et al. (2001), S.3).

2.2.2. Prozessmanagement

Prozessmanagment (engl: Business Process Management, Abk.: BPM) ist der systematische Ansatz, um „Prozesse zu erfassen, zu gestalten, auszuführen, zu dokumentieren, zu messen, zu überwachen und zu steuern und damit nachhaltig die mit der Unternehmensstratgie abgestimmten Ziele zu erreichen“ (EABPM (2009)). Es handelt sich um eine Managementdisziplin zur strategischen Entwicklung des Unternehmens. Diese grenzt sich ab zur klassischen, funktional ausgelegten Sichtweise auf Unternehmen. Der Fokus liegt nicht mehr auf den Funktionen der einzelnen Abteilungen, sondern auf der Wertschöpfungskette und ist somit mehr an den Kunden orientiert, als an den hierarchischen Strukturen eines Unternehmens. Prozessmanagement beinhaltet hierfür verschiedene Wissensgebiete, die zielgerichtet eingesetzt werden sollen.

Bei der *Prozessanalyse* werden Prozessen in ihre Bestandteile zerlegt und deren Strukturen erfasst. Ziel ist es, ein gemeinsames Verständnis der Prozesse zu erreichen. Das Ergebnis ist eine Prozessdokumentation in Form von Modellen.

Im Fokus sämtlicher Aktivitäten des BPM steht somit die *Prozessmodellierung*. Dies geschieht in der Regel mit Hilfe grafischer Notationen, wie zum Beispiel der in dieser Arbeit verwendeten BPMN 2.0. Die hierfür notwendigen Techniken werden in Abschnitt 2.2.5 erläutert.

Bei der *Prozessleistungsmessung* werden Qualitäts- und Effizienzkriterien definiert und an den laufenden Prozessen permanent überwacht und bewertet. Zur Leistungsmessung kann anstelle der Beobachtung real ablaufender Prozesse auch die Simulation von Prozessmodellen genutzt werden. Auf die Simulation von Geschäftsprozessen wird in Kapitel 3.1 eingegangen.

Gegebenenfalls wird eine Verbesserung der gewählten Kennzahlen angestrebt. Das *Prozessdesign* beschäftigt sich mit dem planmäßigen Entwurf veränderter oder neuer Prozesse. Hierbei werden auch die Schnittstellen der Prozesse betrachtet, zum Beispiel Wechselwirkungen mit anderen Prozessen, Anforderungen an IT-Systeme oder Implementation von Kontrollpunkten zur Leistungsmessung. Das Themengebiet der *Prozesseinführung* befasst sich schließlich mit dem Change Management beim Umstieg auf neue Prozesse.

Viele Softwarehersteller haben Tools entwickelt, die das BPM unterstützen. Diese werden meist als Business Process Management Systeme (BPMS) bezeichnet und beinhalten Modellierungs-, Steuerungs-, Analyse- und Messinstrumente. Ein Vorteil von speziellen Werkzeugen zur Geschäftsprozessmodellierung ist, dass sie die Syntax und Semantik der entsprechenden Notation verstehen und den Modellierer somit durch automatische Konformitätsprüfungen unterstützen können. Sie können in der Regel den zeitlichen Ablauf eines Prozesses animieren und helfen so Fehler im Entwurf zu identifizieren. Die Modelle können an eine Workflow Enginge zur automatisierten Ausführung übergeben werden, welche eine automatisierte Leistungsmessung durchführen kann. Einige wenige Tools enthalten die Möglichkeit, mit Hilfe der Simulationstechnik Kennzahlen zu den Prozessen zu berechnen.

Abbildung 2.2 zeigt die Ergebnisse einer Studie von der global agierenden Unternehmensberatung Bearing Point aus dem Jahre 2012 (vgl. Bearing Point (2012)). Hier wurde untersucht, welche Erwartungen mit der Einführung von BPM verknüpft werden. Die meistgenannten Ziele sind die Steigerung der Effizienz, die Erhöhung der Transparenz, das Heben von Standisierungspotentialen, die Verbesserung der Qualität, die Steigerung der Kundenorientierung und das Aufdecken von Schwachstellen und Bottlenecks.

Nicht für alle Unternehmen ist BPM geeignet. Bearing Point empfiehlt in derselben Studie, vor der Einführung von BPM die Reife eines Unternehmens für das BPM zu prüfen. Hierbei spielen Kriterien wie die Unternehmensgröße, die Unternehmensstruktur, aber auch die Branche der Unternehmung eine entscheidende Rolle.

2.2.3. Ziele der Prozessmodellierung

Auch ohne die Orientierung an der Philosophie des BPM werden Prozesse in Unternehmen erfasst. In diesen Unternehmen werden bewusst oder unbewusst nur Ausschnitte aus dem Themenpool des BPM angewandt.

Im ersten Schritt dienen Prozessdefinitionen immer als Kommunikationsgrundlage. Es gilt, einen Handlungsleitfaden für die Beschäftigten bereitzustellen, der Selbstkontrolle und Orientierung ermöglichen soll. Sie dient dem Wissenserhalt in einem Unternehmen, um den Gefahren, die Mitarbeiterfluktuation mit sich bringt, vorzubeugen. Neue Mitarbeiter können leichter geschult werden oder sich notfalls eigenständig einen Überblick

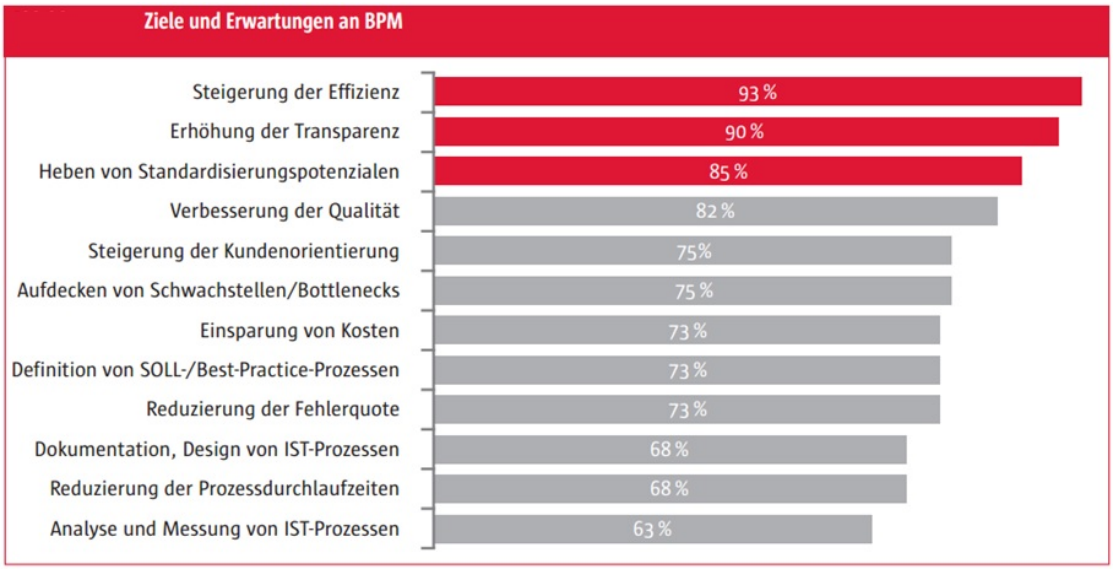


Abbildung 2.2.: „Ziele und Erwartungen an BPM“. Abbildung aus: Bearing Point (2012)

verschaffen. Prozessdefinitionen erleichtern es, über Prozesse zu sprechen und bilden somit die Grundlage für eine selbstkritische Reflektion der eigenen Handlungsweisen.

Diese Kommunikationsgrundlage bildet die Basis für alle weiteren, mit der Modellierung verknüpften Zielsetzungen. So kann mit Hilfe der Prozessdefinition der Nachweis für normkonforme Prozesse erbracht werden, welche für die Zulassung der Geschäftsaktivitäten in einigen Branchen als Qualitätsnachweis erforderlich sind.

Mit Hilfe der Prozessdefinitionen lassen sich Interaktionen mit Partnern, etwa Kunden oder Lieferanten abstimmen. Die eigenen Prozesse müssen teilweise mit Prozesses der Partner interagieren. Mit einer geeigneten Modellierungssprache wie der BPMN lässt sich die Interaktion zwischen zwei Prozessen darstellen und eine Schnittstellenbeschreibung generieren, aus der hervorgeht, wer zu welchem Zeitpunkt welche Zuarbeiten von dem Partner erwarten darf. In einigen Branchen wie der Automobilbranche existieren Referenzprozessmodelle, um Standards für diese Schnittstellen zu schaffen. So wird beispielsweise festgelegt, auf welcher Seite eine Qualitätskontrolle zu erfolgen hat (vgl. Allweyer (2009), S.60).

Einige Unternehmen nutzen die Prozessdefinitionen als Grundlage für eine automatisierte Steuerung der Prozessinstanzen mit Hilfe einer Workflow Engine. Voraussetzung ist eine ausreichend formalisierte Modellierungssprache, für welche das dynamische Laufzeitverhalten exakt spezifiert ist. Ziel hierbei ist es, Prozesse teilautomatisiert ablaufen zu lassen. In diesem Kontext werden Prozesse meist als Workflow bezeichnet. Die

Workflow Engine startet für jeden anfallen Geschäftsprozess eine Worklow-Instanz. Es wird unterschieden zwischen komplett automatisierbaren Tätigkeiten, Tätigkeiten die von Menschen mit IT-Unterstützung durchgeführt werden und manuellen Tätigkeiten. Die Mitarbeiter werden dabei auf Aufgaben aufmerksam gemacht und bekommen die jeweils notwendigen Informationen und Dokumente zur Verfügung gestellt. Auf Basis der zugrundeliegenden Daten und der Prozessdefinition kann die Workflow Engine bei Verzweigungen selbständig die korrekte Ausführungsvariante des Prozesses wählen. Hierdurch soll die Fehleranfälligkeit verringert und die Effizienz gesteigert werden. Auch das Monitoring der Prozesse wird hierdurch möglich. So kann für jede Prozessinstanz überprüft werden, an welcher Stelle sie gerade steht. Kennzahlen zur Ausführungsdauer einzelner Aktivitäten oder Prozesse können bereitgestellt und aufbereitet werden.

Wenn mit den Prozessdefinitionen eine Kommunikationsgrundlage geschaffen ist, kann das Bedürfnis entstehen, die Prozesse zu verbessern. Dies wird im deutschen als Geschäftsprozessoptimierung (Abk.: GPO) bezeichnet, was jedoch ein unpassender Begriff ist, denn es wird kein Optimum der Prozesse im mathematischen Sinn bestimmt, sondern lediglich eine verbesserte Variante gesucht. Der englische Begriff Business Process Improvement (Abk.: BPI) ist eine treffendere Bezeichnung (vgl. Stehle (2014)) und wird daher in dieser Arbeit verwendet. Es gibt in der Literatur eine Vielzahl von Richtlinien für eine effizientere Gestaltung von Prozessen. In Youngblood (1994) werden 32 Wege zur Verbesserung von Prozessen aufgezählt, unter anderem: Eliminierung mehrfach vorhandener Aktivitäten (z.B. wiederholter Prüfungen), Zusammenfassung von zusammengehörigen Aktivitäten, Wechsel des Sachbearbeites vermeiden, Prozessschritten parallelisieren, Outsourcing von ineffizienten Aktivitäten, Vereinfachung der Prozesse oder Bildung von multi-funktionalen Team zur Abarbeitung von Aufgaben.

Die Abbildung 2.3 zeigt das strukturelle Vorgehen zur Verbesserung von Geschäftsprozessen. In der Vorbereitungsphase muss zunächst ein geeingetes Projektteam zusammengestellt werden, mit dessen Hilfe die relevanten Kennzahlen der Prozesse spezifiziert werden, um die Prozessalternativen vergleichen zu können. Stehle (2014) hat folgende typische Kennzahlen zur Leistungsmessung identifziert:

- Durchlaufzeit
- Prozesseffizienz
- Prozessqualität
- Kundenzufriedenheit
- Termintreue
- Kosten

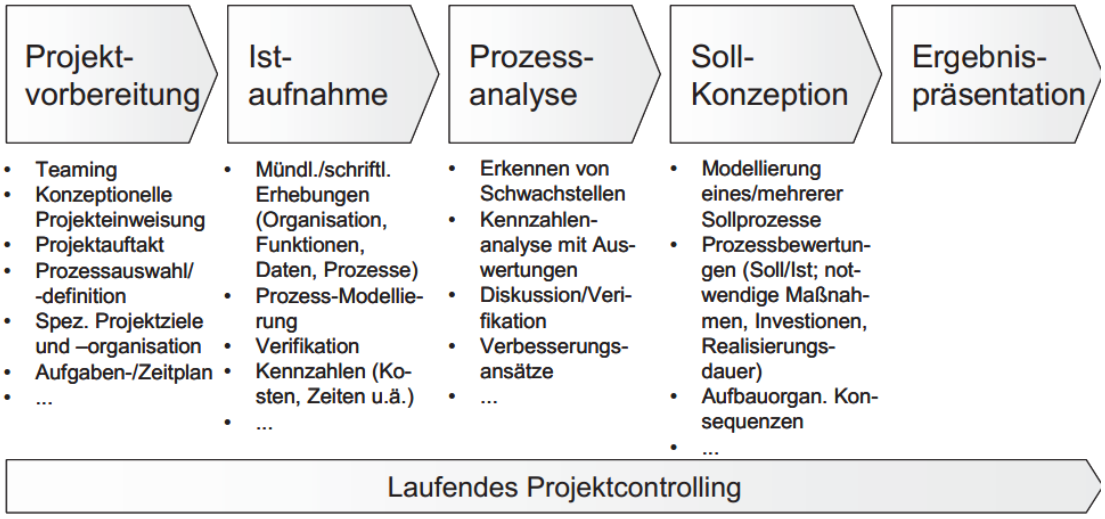


Abbildung 2.3.: BPI-Projektphasen, Originaltitel: „GPO-Projektphasen“. Abbildug aus: Seidlmeier (2010)

Auf die Vorbereitungsphase folgt die Aufnahme der bestehenden Prozesse, sofern dies noch längst geschehen ist. In der Phase der Analyse werden die festgelegten Kennzahlen quantifiziert. Dies kann im Rahmen der Prozessleistungsmessung zum Beispiel mit Monitoring-Tools erfolgen. Eine weitere Möglichkeit ist die Leistungsmessung mit Hilfe der Simulationstechnik. In der nächsten Phase müssen die Soll-Prozesse konzipiert werden. Dies geschieht abermals durch die Modellierung von Prozessdefinitionen.

Ohne die Simulationstechnik ist vor der Einführung der Prozesse keine Leistungsmessung dieser Alternativen möglich. Es kann daher nur gemutmaßt werden, dass die veränderten Prozesse bessere Performanzkriterien liefern. Die Simulation ermöglicht es jedoch, bereits vor der eventuell risikobehafteten und kostenintensiven Einführung von Prozessen mögliche Alternativen zu untersuchen (vgl. Kapitel 3). Die Aussagekraft der Simulationstechnik im Rahmen der Leistungsmessung und des BPI für Unternehmen zu verbessern, ist die motivierende Zielsetzung dieser Arbeit.

Wenn Unternehmen bei der Definition ihrer Prozesse auf entsprechend leistungsstarke Modellierungssprachen und Werkzeugen gesetzt haben, so entwickelt sich nicht selten erst ihm Rahmen der Erfassung der eigenen Prozesse den Wunsch, diese Prozessdefinitionen zum Beispiel für die technische Workflow-Umsetzung oder für Analysen mit Hilfe der Simulation zu nutzen.

2.2.4. Personen und Rollen

Mit Geschäftsprozessen kommen diverse Personem mit unterschiedlichem fachlichem Hintergrund in Berührung. Die Anzahl von definierbaren Rollen erscheint in einigen Lehrbüchern beliebig groß. Die Association of Business Process Management Professionsals (ABPMP) hat sogar über 150 verschiedene Rollen identifiziert. Dieser Abschnitt soll eine Vorstellung von der Heterogenität der Personen geben, die mit Geschäftsprozessen umgehen müssen, da dies eine wesentliche Voraussetzung für das Verständnis der Notwendigkeit von speziellen Modellierungssprachen und auch für einige Aspekte der dieser Arbeit zugrundeliegenden Anforderungen ist.

Der *Prozessbeteiligte* führt Tätigkeiten innerhalb eines Prozesses aus. Diese Person kann sehr unterschiedliche Positionen innerhalb eines Unternehmens haben. Ebenso sind der Ausbildungsstand und die Erfahrung divers.

Der *Prozessverantwortliche* ist verantwortlich für die fachlich korrekte Umsetzung von Prozessen. Es handelt sich meist um Abteilungs- oder Teamleiter mit einem höheren Stand an Erfahrung. Auch ohne eine explizite Einführung von BPM sind diese beiden Rollen in jeder Unternehmung vorhanden.

Die Einführung des Prozessmanagments bringt weitere Rollen mit sich:

Prozesscontroller überwachen und bewerten Prozesse anhand von Kennzahlen. Sie ergreifen Maßnahmen, wenn bei wichtigen Prozessen zeitkritische Zustände auftreten, sie ineffizient ablaufen oder in bestimmten Abteilungen zu hohe Kosten bei der Durchführung der Prozesse anfallen. Sie gehören tendenziell der Managementebenen an und haben eine entsprechend betriebswirtschaftliche Ausbildung.

Prozessanalysten können Prozesse erfassen und dokumentieren. Sie werden bei der Einführung von BPM gebraucht, um Prozessdokumentationen der Ist-Prozesse zu erstellen. Sie haben Erfahrung bei der Erfassung, sind zur Modellierung von Prozessen ausgebildet und beherrschen die Anwendung von speziellen Werkzeugen.

Prozessberater und*Prozessdesigner* entwerfen Soll-Prozesse. Beide verfügen über einen ähnlichen Ausbildungsstand wie Prozessanalysten, haben jedoch zusätzlich genaue Vorstellungen von den fachlichen Abläufen. Der Prozessberater versteht sich auf die Anwendung von speziellen Methoden zur Absicherung des Prozessdesigns, wie zum Beispiel der Simulationstechnik. Diese Rollen sind nicht zwangsläufig innerhalb eines Unternehmens vorhanden. Häufig wird ihre Leistung bei Unternehmensberatungen eingekauft.

Prozess Developer sind in der Lage, Prozesse in technischen Umgebungen automatisiert ausführbar zu implementieren. Es handelt sich meistens um Softwaretechniker mit entsprechend fundierten IT-Kenntnissen und fundierten Kenntnissen in Modellierungssprachen (siehe Abschnitt 2.2.5).

Häufig haben also Personen Kontakt mit Geschäftsprozessen die keine spezielle Ausbildung im Bereich des BPM besitzen. Sie haben teilweise besondere fachliche Qualifikationen, aber keine speziellen Kenntnisse über Modellierungssprachen. Einige haben betriebswirtschaftliche Kenntnisse und Fähigkeiten aus dem Bereich des Managements. Andere haben besonders gute IT-Kenntnisse. Nur selten sind Menschen mit spezifischen Kenntnissen aus dem Bereich des BPM in einer Unternehmung vorhanden.

2.2.5. Notwendigkeit von Modellierungssprachen

Geschäftsprozesse sind zeitdiskrete, stochastische, dynamische Systeme. Die Ausführungsreihenfolge von Tätigkeiten innerhalb eines Geschäftsprozesses lässt sich grundsätzlich als Flussdiagramm darstellen. Es wird also eine mehr oder weniger formalisierte graphische Notation genutzt. Tätigkeiten werden dabei als Rechtecke dargestellt. Gerichtete Kanten geben die Ausführungsreihenfolge an. „Zur genauen Darstellung komplexerer Prozesse mit allen relevanten Aspekten, wie Verzweigungsregeln, Ereignissen, ausführenden Organisationseinheiten, Datenflüssen usw. genügt dies nicht“ (vgl. Allweyer (2009), S. 8). Daher werden spezielle Notationen verwendet, die an die Anforderungen von Geschäftsprozessen angepasst sind. Die Basisanforderungen lassen sich aus der Definition von Geschäftsprozessen ableiten. Es werden mindestens Symbole für Tätigkeiten, Input und Output benötigt. Da Tätigkeiten sequentiell oder nebenläufig ablaufen können und verschiedene Prozessvarianten auf Entscheidungen innerhalb des Prozesses beruhen, werden auch Verzweigungen und Zusammenführungen benötigt.

In Frank und Laak (2003) werden die Anforderungen an Modellierungssprachen für Geschäftsprozesse wie folgt kategorisiert:

Anwenderbezogene Anforderungen beziehen sich auf die Einfachheit, Verständlichkeit und Anschaulichkeit. Wie in Abschnitt 2.2.4 beschrieben, kann der fachliche Hintergrund der beteiligten Akteure sehr unterschiedlich sein. Jeder dieser Rollen soll in der Lage sein, die Modelle zu lesen und gemäß ihrer Bedürfnissen zu nutzen. Eine anschauliche Modellierung ist eine wesentliche Voraussetzung, um Akteuren mit unterschiedlichem fachlichem Hintergrund eine gemeinsame Kommunikationsgrundlage zu bieten.

Formale Anforderungen beinhalten die Korrektheit, die Vollständigkeit, die Einheitlichkeit und Redundanzfreiheit. Dies sind die Voraussetzungen für eine Systematische Analyse des Geschäftsprozesses, z.B. mit dem Ziel der Optimierung durch Prozessanalysten, für die automatisierte Steuerung und Prozessüberwachung von Prozessen und für die Leistungsmessung mit Hilfe der Simulationstechnik.

Anwendungsbezogene Anforderungen beinhalten die Mächtigkeit und die Operationalisierbarkeit. Dies sind zwingende Voraussetzungen, um eine automatisierbare Ausführung oder Simulationsexperimente zu ermöglichen. Die Aufgabenangemessenheit hingegen

versetzt den Modellierer in die Lage, „alle als relevant erachteten Eigenschaften eines darzustellenden Sachverhalts in der gewünschten Detaillierung und Präzision zu beschreiben“ (Frank und Laak (2003)). Hierzu zählt die Möglichkeit, spezielle Sachverhalte aus einer Domäne mit geeigneten Mitteln ausdrücken zu können.

Becker et al. (2011) formulieren die Anforderungen an die erstellten Modelle und somit implizit auch Anforderungen an die Modellierunssprache. Die Modelle sollen syntaktisch und semantisch Korrekt sein, was eine ausreichende Formalisierung der Sprache voraussetzt. Die Modelle sollen keine irrelevanten Details enthalten, was eine grundsätzliche Voraussetzung für Modellierungsprojekte ist (vgl. Abschnitt 2.1). Sie sollen leicht verständlich für den Betrachter sein, was eine entsprechend intuitiv verständliche Modellierungsprache voraussetzt. Damit die Prozessmodelle innerhalb eines Unternehmens vergleichbar sind, sollen Modellierungskonventionen geschaffen und eingehalten werden. Letztlich wird gefordert, dass die Prozessmodelle wohldefinierte Schnittstellen zu anderen Modeltypen enthalten - eine Forderung der diese Arbeit im Zuge der Simulationsmodellierung nachkommt.

Für die Definition von automatisierbaren Prozessen (Workflows) müssen die Prozessdefinitionen in der Regel sehr viel detaillierter beschrieben werden als bei Prozessdefinitionen, die lediglich als fachliche Kommunikationsgrundlage dienen (vgl. Komus (2011)). Jede Ausnahmesituation und jeder mögliche Fehler muss hierbei bedacht werden. Außerdem werden auch spezielle Anforderungen an die Modellierungssprache gestellt. So muss die Möglichkeit zur Interaktion mit der bestehenden IT-Infrastruktur gewährleistet sein, zum Beispiel durch die Bereitstellung von speziellen Webservice-Aktivitäten (vgl. EABPM (2009), S. 41). Es gibt einige Sprachen wie die BPEL oder die Windows Workflow Foundation, die ausschließlich auf die Definition von automatisierbaren Workflows abzielen. Es gibt jedoch auch Sprachen, welche sowohl der fachlichen als auch der technischen Modellierung dienen. Diese haben den Vorteil, dass fachliche Prozessdefinitionen direkt in Workflow-Definitionen überführt und auch die Workflow-Definitionen noch von den verschiedenen Rollen in dem Unternehmen verstanden werden können. Solche Sprachen eignen sich auch im besonderen Maße für die Durchführung von Simulationsstudien.

Eine Notation, welche all diese Anfoderungen erfüllt, ist Voraussetzung für die Beschreibung domänenspezifischer Geschäftsprozesse, welche von Beteiligten mit unterschiedlichem Hintergrund verstanden wird[1] und zugleich die Leistungsmessung mit Hilfe der Simulationstechnik ermöglicht.

[1] In Abschnitt 5.4 werden die für die Anwendung der in dieser Arbeit beschriebenen Rollen aufbauend auf der Darstellung in 2.2.4 aufgelistet.

2.2.6. Vergleich von Modellierungssprachen

Allen Geschäftsprozessmodellierungssprachen sind die grundlegenden Anforderungen von Geschäftsprozessen gemein. Sie bieten Elemente für Tätigkeiten, Input, Output und Verzweigungen.

In List und Korherr (2006) wurde ein Meta-Modell für Geschäftsprozessmodellierungssprachen entwickelt, um den Vergleich von Sprachen zu ermöglichen. Dieses wurde sogleich auf sieben Notationen angewendet. Vier der sieben Sprachen werden in diesem Abschnitt beleuchtet. Dies dient der Darstellung, warum sich in dieser Arbeit auf die BPMN 2.0 konzentriert wurde und kein allgemeinerer Ansatz für alle Modellierungssprachen gesucht wurde. Die restlichen drei Sprachen werden aus Relevanzgründen nicht angesprochen.

Petri-Netze wurden erstmals in Petri (1962) beschrieben. Sie sind nicht speziell auf Geschäftsprozesse zugeschnitten, werden aber durchaus auch zur Modellierung und Simulation von Arbeitsabläufen verwendet (vgl. Licher et al. (2001), S. 490). Es handelt sich um Flussdiagramme, in denen der aktuelle Zustand eines Systems durch Hinterlegung von Marken auf einzelnen Elementen repräsentiert wird. Sie eignen sich sehr gut zur formalen Analyse von Systemen und zum Nachweis oder zur Widerlegung von bestimmten Eigenschaften wie Verklemmungsfreiheit, Lebendigkeit, Terminiertheit, Beschränktheit oder Reversibilität. Da Petri-Netze jedoch keine speziellen Symbole für typische Elemente von Geschäftsprozessen haben, erfüllen sie in den meisten Fällen nicht die Anforderung der Aufgabenangemessenheit und Verständlichkeit. So wird in List und Korherr (2006) angemerkt, dass sie keine Elemente zur Darstellung von Organisationseinheiten bieten, nicht zwischen Sequenz- und Nachrichtenfluss unterscheiden und keine Darstellung von benötigten Rollen ermöglichen. Zahlreiche andere Notationen haben jedoch das grundlegende Prinzip der Platzierung von Marken auf einzelnen Elementen von den Petri-Netzen übernommen. Dazu gehören auch die folgenden Sprachen.

Ereignisgesteuerte Prozessketten (EPK) sind eine sehr simple Sprache zur Beschreibung von Geschäftsprozessen. Sie wurde im Jahr 1992 an der Universität Saarland entwickelt (vgl. Keller et al. (1992), S. 11ff), und ist ein wesentliches Element des ARIS-Konzeptes. Da sie nur mit Ereignissen, Tätigkeiten (Funktionen) und Verzweigungen gerade diejenigen Elemente besitzen, die zur Beschreibung von Geschäftsprozessen zwingend notwendig sind, sind sie sehr einfach zu erlernen. Jedoch ist ihre Aussagekraft aufgrund der geringen Anzahl von Elementen recht gering. Komplexe Zusammenhänge lassen sich nur schwer darstellen. Sie bieten keinen eingängigen Symbolsatz, der auf den ersten Blick die Bedeutung von einzelnen Prozessabschnitten verdeutlichen könnte. EPK sind vorranging in Deutschland verbreitet, aber die internationale Zusammenarbeit und Verflechtung lässt zunehmend auch deutsche Unternehmen von EPK zur aussagekräftigeren BPMN migrieren (vgl. Decker et al. (2009)). Es gibt Erweiterungen zur EPK wie die erweiterte

Ereignisgesteuerte Prozessketten (eEPK) .Wegen des eingeschränkten Symbolsatzes sind EPK für die vorliegende Arbeit ungeeignet.

Von der Object Management Group (OMG) wurde die *Unified Modeling Language* (UML) als graphische Modellierungssprache zur Spezifikation und Dokumentation von Software-Artefakten entwickelt. Für verschiedene Anwendungsfälle aus dem Bereich der Softwaredokumentation stehen insgesamt 14 Modelltypen zur Verfügung. Die Aktivitätsdiagramme der UML eignen sich für eine technische Beschreibung von Prozessen. Ihre Ausführungssemantik ist exakt spezifiziert, wodurch automatische Ausleitungen von ausführbaren Prozessen möglich werden. Der Prozess Developer findet dort alle notwendigen Informationen für eine Implementation von ausführbaren Prozessen. Die Notation darf nach bestimmten Regeln erweitert werden. Hierauf aufbauend wurde zum Beispiel eine Erweiterung zur Definition von Simulationsmodellen konzipiert (vgl. Knaak und Page (2006) und Sandu (2007)). Jedoch ist diese Sprache primär für die Softwareentwicklung konzipiert und ihre Elemente wenig intuitiv. Sie eignet sich nicht als Kommunikationsgrundlage zwischen Prozessbeteiligten, Prozessverantwortlichen und Managementebene (vgl. Loos und Allweyer (1998)). Sie wird daher nicht zur Darstellung von Geschäftsprozessen gewählt, sofern die in Abschnitt 2.2.3 genannten fachlichen Ziele verfolgt werden, sondern zur Beschreibung von technischen Anforderungen eines Prozesses. Daher entfällt auch diese Sprache als mögliche Grundlage für diese Arbeit.

Die *Business Process Modell and Notation* (BPMN) in der Version 2.0 ist auf die Modellierung von Geschäftsprozesse zugeschnitten. Sie bietet ein reichhaltiges, zumeist intuitives Symbolset zur Beschreibung typischer Ereignisse und Aktivitäten. Die Unterscheidung zwischen Sequenz-, Nachrichten- und Assoziationskanten verdeutlicht den Unterschied zwischen Prozessfluss, Interaktion zwischen zwei Prozessinstanzen sowie Objekt- und Datenfluss. Sie bietet mit den angehefteten Ereignissen und Ereignisunterprozessen die Möglichkeit, Abbruchbedingungen für Prozesse und Aktivitäten zu beschreiben. Die Notation wurde mit der Zielsetzung entworfen, von den verschiedenen beteiligten Rollen in einem Unternehmen leicht verstanden zu werden. Durch ihre formal exakt spezifizierte Ausführungssemantik ist sie sowohl für die Beschreibung von fachlichen Prozessdefinitionen als auch von technischen Workflow-Beschreibungen geeignet. Ihre Elemente sind durch eigene Attribute erweiterbar und die Darstellung darf an vorgegebenen Stellen angepasst werden. Obwohl die Simulationsanalyse nicht Teil der Spezifikation ist, eignet sich diese Sprache optimal für die Erreichung der Ziele dieser Arbeit.

2.3. Prozessmodellierungssprache BPMN 2.0

Die für diese Arbeit wesentlichen Aspekte der BPMN 2.0 werden in dem folgenden Abschnitt beleuchtet. Er ist nicht als Lehrbuch über die BPMN aufzufassen, da einige

Aspekte außer Acht gelassen wurden, die für diese Arbeit nicht relevant sind. Um einen breiteren Überblick über die BPMN 2.0 zu erlangen, sei an dieser Stelle Allweyer (2009) empfohlen.

2.3.1. Grundlegendes

Die BPMN 2.0 wurde vom IBM-Mitarbeiter Stephen A. White im Jahre 2001 entwickelt und von der Business Process Magament Initiative (BPMI) im Jahr 2004 in der Version 1.0 veröffentlicht. Die BPMI fusionierte im Jahr 2005 mit der OMG. Seither übernimmt die OMG die Pflege und Weiterentwicklung der BPMN (vgl. Freund und Rücker (2012)). Anfang 2011 wurde die BPMN in ihrer Version 2.0 offiziell veröffentlicht (vgl. OMG (2011)), wobei erste Publikationen zu dieser Version schon im Jahre 2009 erschienen sind (vgl. Allweyer (2009)).

Die BPMN 2.0 unterscheidet sich von der BPMN 1.0 durch einige Merkmale, von denen zwei besonders relevant für diese Arbeit sind:

- Die BPMN 2.0 hat eine formal festgelegte Ausführungssemantik. Dies wurde mit der Zielsetzung eingeführt, die Ausführbarkeit in automatisierten Workflow-Systemen zu ermöglichen und zu vereinheitlichen. Obwohl die Simulation selbst nicht Teil der BPMN-Spezifikation ist, ist diese wohldefinierte Ausführbarkeit eine wesentliche Voraussetzung für die Ausführung innerhalb von Simulationsexperimenten.
- Die BPMN 2.0 ist erweiterbar. An verschiedenen Stellen dürfen eigene Elemente eingeführt werden und an allen Elemente dürfen eigene Attribute definiert werden (vgl. Abschnitt 2.3.8). Dies ist eine wesentliche Voraussetzungen für die Erweiterungen, die in dieser Arbeit vorgenommen werden.

Durch die Spezifikation ist eine Strukturdefinition (XSD) zur Persistierung als XML-Datei vorgegeben (vgl. Haan (2012)). Dies ermöglicht die Anzeige und Bearbeitung eines Modells durch verschiedene Software-Werkzeuge. Die Einführung eigener Elemente ist nur im beschränkten Umfang möglich, wenn die Austauschbarkeit des Werkzeuges gewahrt bleiben soll (siehe Abschnitt 2.3.8).

Die BPMN 2.0 bietet insgesamt drei Diagrammtypen. Zur Modellierung von Geschäftsprozessen werden Kollaborationsdiagramme verwendet. Diese Diagrammform ist Gegenstand dieser Arbeit und wird in den folgenden Abschnitten beschrieben. Mit Hilfe von Konversationsdiagrammen wird dargestellt, welche Akteure eine Tätigkeiten kooperativ abwickeln. Choreographiediagramme bieten die Möglichkeit, die zeitliche Abfolge von kooperativen Tätigkeiten zu modellieren. Konversationsdiagramme und Choreographiediagramme sind nicht Gegenstand dieser Arbeit.

Da die Art der vorhandenen Elemente entscheidend für die Auswahl der Kopplungsmechanismen an externe Modelle ist, wird im folgenden ein grober Überblick über die vorhandenen Elemente gegeben. Es werden nur diejenigen Ausprägungen angesprochen, die relevant für diese Arbeit sind.

2.3.2. Aktivitäten

Tätigkeiten werden in der BPMN mit dem Element Aktivität dargestellt (vgl. OMG (2011), S. 156ff). Aktivitäten werden von Menschen oder Maschinen ausgeführt und haben eine endliche Dauer. Wie in Abbildung 2.5 gezeigt, werden sie durch ein Rechteck mit abgerundeten Ecken symbolisiert. Neben Prozessen benötigen auch Aktivitäten einen Input und erzeugen einen Output, wobei die explizite Modellierung hiervon nicht zwingend erforderlich ist. Zwingend erforderlich ist jedoch, dass der Sequenzfluss durch eine gerichtete Kante mit durchgezogener Linie dargestellt wird und die Ausführungsreihenfolge der Elemente darstellt (vgl. Workflow Pattern Sequenz in van der Aalst (2000)).

Die BPMN unterscheidet eine Reihe von Aktivitäten. Benutzer-Aktivitäten und manuelle Aktivitäten werden von Menschen ausgeführt, Service-, Geschäftsregel- und Skript-Aktivitäten von Maschinen. Es gibt Aktivitäten zur Darstellung von Kommunikationsbeziehungen die in sendende und empfangende Aktivitäten unterschieden werden. Wie in Abbildung 2.4 zu sehen, werden diese Aktivitäten durch ein Symbol in der oberen linken Ecke der Aktivität unterschieden.

Weitere Symbole am unteren Rand der Aktivität unterscheiden weitere Arten von Aktivitäten. Unterprozessaktivitäten referenzieren einen Unterprozess, der wiederum Aktivitäten enthält, die die Ausführung der Unterprozessaktivität im Detail beschreiben. Dies ermöglich eine hierarchische Modellierung von Prozessen. Drei Arten von Markierungen geben zudem an, dass eine Aktivität wiederholt ausgeführt werden muss, bevor der Sequenzfluss fortgesetzt wird: die parallele Aktivität, die sequentielle Aktivität und die Wiederholungsaktivität.

2.3.3. Ereignisse

Ereignisse unterscheiden sich von Aktivitäten dadurch, dass sie keine Ressourcen belegen, weil sie keine Zeit verbrauchen. Die BPMN 2.0 bietet eine Vielzahl von verschiedenen Ereignistypen und schreibt vor, in welchem Kontext sie verwendet werden dürfen (vgl. OMG (2011), S. 233ff). Dies ermöglicht eine sehr genaue, aber dennoch verständliche Modellierung. Da Ereignisse wie in Abbildung 2.5 gezeigt mit Hilfe von Symbolen (innerhalb eines Kreises) repräsentiert werden, ist ihre Interpretation in den meisten Fällen

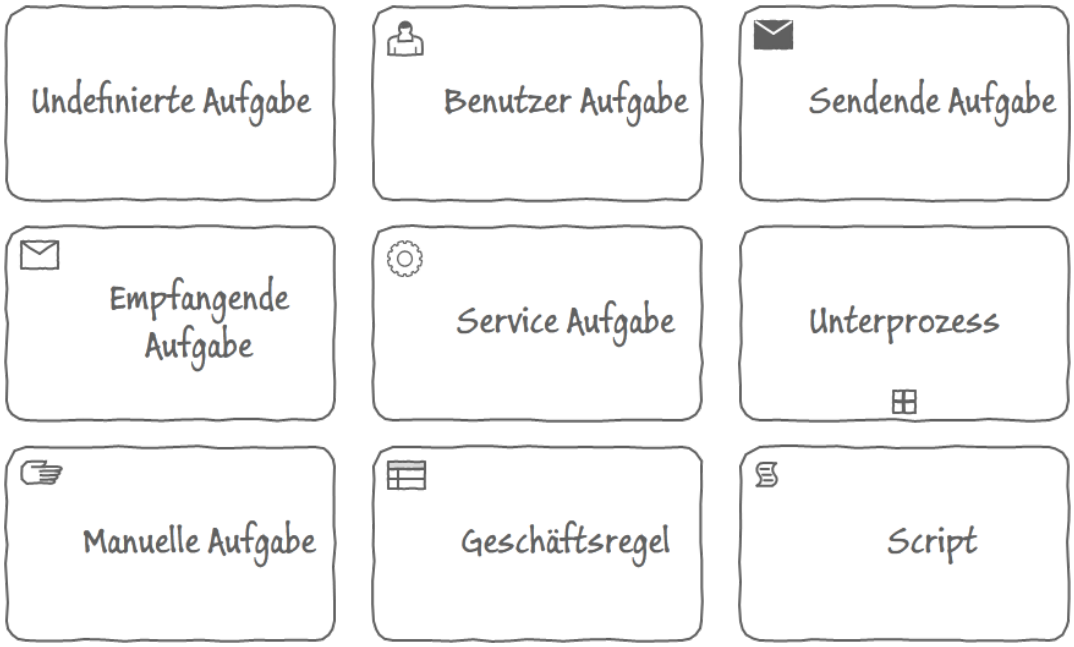


Abbildung 2.4.: Aktivitätstypen und ihre Symbole in der BPMN 2.0

intuitiv möglich. Abhängig von der Art der Ereignisse haben sie unterschiedliche Auswirkungen auf das Laufzeitverhalten der Prozesse.

Ereignisse werden unterschieden in sendende und empfangende Ereignisse. Sendende Ereignisse werden durch ein schwarzes Symbol dargestellt, empfangende durch ein weißes. Empfangende Zwischenereignisse müssen ggf. auf das Eintreten des Ereignisses warten, führen deshalb zu Wartezeiten innerhalb eines Prozesses und somit zu einer Verlängerung von dessen Laufzeit. Sie belegen während dieser Wartezeit jedoch keine Ressourcen. So kann zum Beispiel bei Zeitgeberereignissen angegeben werden, zu welchem konkreten Zeitpunkt das Ereignis ausgelöst wird.

Die meisten empfangenden Ereignisse korrespondieren mit sendenden Ereignissen des gleichen Typs. Sendende Zwischenereignisse werden zeitverzugslos ausgeführt. Sie haben keine Auswirkung auf die Laufzeit des betreffenden Prozesspfades. Sie beeinflussen nur die Laufzeit des empfangenden Prozesses. Das sendende Ereignis kann im selben Prozess, in einem Unterprozess oder in einer parallel ablaufenden Prozessinstanz eintreten.

In der BPMN werden für sendende und empfangende Ereignisse verschiedene Ereignistypen eingeführt. Sie sollen jeweils in genau definierten Fällen verwendet werden. Nachrichtenereignisse beschreiben den zielgerichteten Austausch von Informationen zwischen

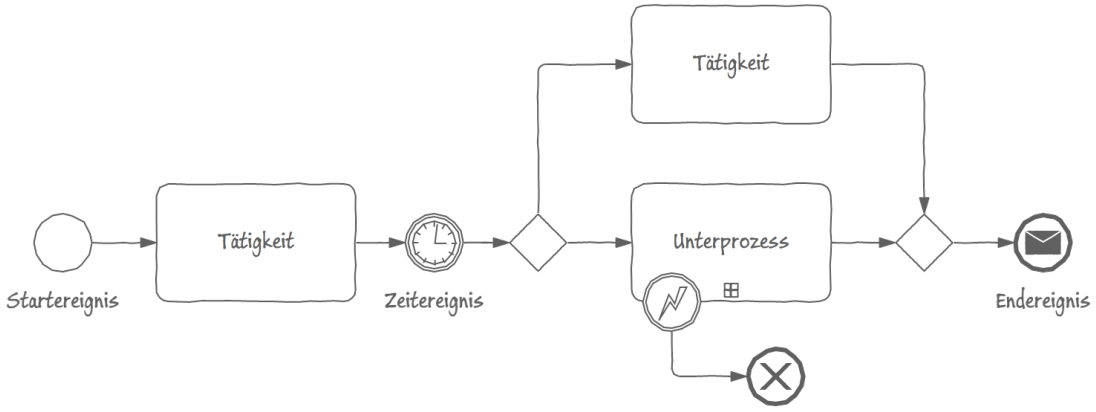


Abbildung 2.5.: BPMN-Prozess mit Ereignissen, Gateways und Aktivitäten

zwei Prozessinstanzen. Nachrichtenflüssen werden zusätzlich dargestellt als gerichtete Kante mit gestrichelter Linie. Damit werden Nachrichtenereignisse und auch Nachrichtenaktivitäten verknüpft, um den genauen Empfänger zu spezifizieren. Es empfangen jedoch nicht alle zur Prozessdefinition gehörenden Prozessinstanzen die Nachricht, sondern jeweils nur diejenige Instanz, die dem sendenden Prozess zugeordnet ist. So wird z.B. die Antwort auf eine Kundenanfrage stets an den anfragenden Kunden geschickt.

Signalereignisse beschreiben ebenfalls den Austausch von Informationen. Das sendende Ereignis ist jedoch ein Broadcast, welches von allen Prozessinstanzen empfangen wird, die ein entsprechendes Signalempfangsereignis haben. Signalereignisse und alle anderen Ereignistypen werden nicht mit Nachrichtenkanten verknüpft.

Fehlerereignisse beschreiben das Auftreten eines technischen und Eskalationsereignisse das Auftreten eines fachlichen Fehlers. Abbildung 2.6 zeigt die 12 verschiedenen Ausprägungen von Ereignissen in der BPMN 2.0-Spezifikation. Für eine genaue Beschreibung der verschiedenen Ereignistypen sei auf OMG (2011) verwiesen.

In der BPMN wird zudem unterschieden, ob es sich um ein Ereignis handelt, welches einen Prozess startet (vgl. OMG (2011), S. 240ff), einen Prozess beendet (vgl. OMG (2011), S. 247ff) oder innerhalb eines Prozesses ausgelöst wird(vgl. OMG (2011), S. 251ff).

Startereignisse geben an, wann Prozesse begonnen werden und wie häufig diese ausgeführt werden. Hier sind nur die generellen und die empfangenden Ereignistypen erlaubt. Sie bilden die direkte Verknüpfung zum Input eines Geschäftsprozesses. Endereignisse geben dementsprechend das Ergebnis eines Prozesses und somit die direkte Verknüpfung zum Output an. Sie gibt es ausschließlich in der generellen und in der senden-

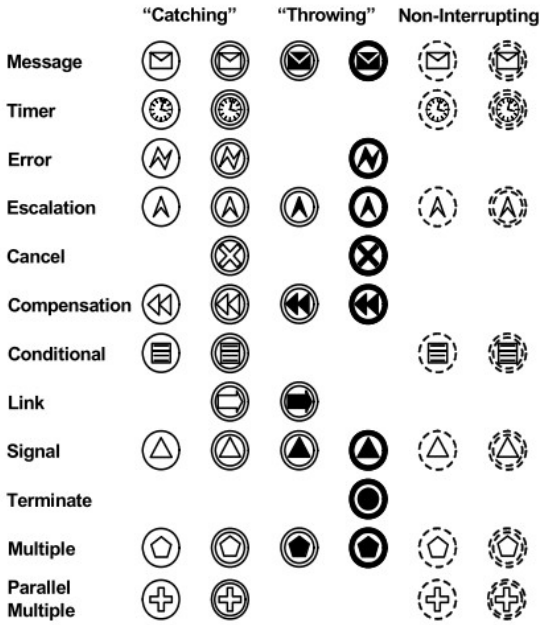


Abbildung 2.6.: Ereignistypen und ihre Symbole in der BPMN 2.0, Originaltitel: „Types of events and their markers“. Abbildung aus: OMG (2011)

den Variante. Bei Zwischenereignissen können sowohl empfangende als auch sendende Ereignisse auftreten.

Zusätzlich bietet die BPMN die Möglichkeit, mit Hilfe von an Aktivitäten angehefteten Ereignissen eine Aktivität abzubrechen. Bei angehefteten Ereignissen sind nur die empfangenden Typen erlaubt. Sie haben direkten Einfluss auf die Dauer einer Aktivität und somit auf die Dauer der Prozesslaufzeit. Das Beschreiben solcher Abbruchbedingungen findet sich in anderen Prozessmodellierungssprachen nicht. Es handelt sich um ein höheres Modellierungskonstrukt, deren Eintreten in anderen Sprachen nur umständlich beschrieben werden kann.

Angeheftete Ereignisse sind auch in einer nicht unterbrechenden Variante erlaubt. Sie können nur während der Ausführung der betreffenden Aktivität ausgelöst werden, brechen diese jedoch nicht ab, sondern erzeugen stattdessen einen neuen Teilprozess.

2.3.4. Gateways

Mit Hilfe von Gateways können unterschiedliche Varianten bei der Ausführung eines Prozesses beschrieben werden (vgl. OMG (2011), S. 287ff). Hier werden Entscheidungen

getroffen, welchem Pfad innerhalb eines Prozesses gefolgt werden soll. Gateways können als Verzweigung und als Zusammenführung verwendet werden.

Bei parallelen, verzweigenden Gateways wird auf jedem ausgehenden Pfad ein Teilprozess erzeugt. Es handelt sich um das Workflow Pattern AND-Split (vgl. van der Aalst (2000)). An einem zusammenführenden, parallelen Gateway (Workflow Pattern Synchronisation in van der Aalst (2000)) wird entsprechend darauf gewartet, bis über jeden eingehend Sequenzfluss ein Teilprozess angekommen ist.

Bei exklusiven, verzweigenden Gateways werden an den ausgehenden Sequenzflüssen Bedingungen überprüft. Es wird maximal einem ausgehenden Pfad gefolgt. Da es sich um ein XOR-Gateway (vgl. Workflow Pattern Exclusive Choice in van der Aalst (2000))handelt, darf nur eine der Bedingungen wahr sein. Das ereignisbasierte Gateway verhält sich ähnlich, jedoch werden anstatt von Bedingungen Ereignisse angegeben.

In welcher Sprache die Bedingungen formuliert werden und mit welchen Mechanismen überprüft wird, welchem Pfad gefolgt wird, wird von der BPMN-Spezifikation nicht vorgeschrieben. Bei rein fachlichen Modellen zur Kommunikationsgrundlage kann eine Bedingung in natürlicher Sprache angegeben werden. Bei ausführbaren Modellen mit dem Ziel der Workflow-Steuerung oder der Simulation, muss eine maschinell prüfbare Bedingung angegeben werden. In welcher Sprache die zu prüfende Bedingung beschrieben wird, ist in der BPMN-Spezifikation also nicht festgelegt. Sie schlägt jedoch die XML-basierte Abfragesprache XPath für die Formulierung von Bedingungen für Gateways und Mehrfachaktivitäten vor (vgl. Allweyer (2009), S. 94).

Die inklusive Verzweigung benötigt ebenso wie die exklusive Verzweigung Bedingungen an jeder ausgehenden Kante. Da es sich um ein OR-Split (vgl. van der Aalst (2000)) handelt, dürfen hier mehrere Bedingungen wahr sein. Jedem Pfad mit wahrer Bedingung wird gefolgt.

Auf die Beschreibung des komplexen Gateways, des exklusiven ereignisbasierten Gateways (nicht zu verwechseln mit dem ereignisbasierten Gateway) und des parallelen ereignisbasierten Gateways wird aus Relevanzgründen verzichtet und stattdessen auf die Spezifikaion der BPMN 2.0 verwiesen.

2.3.5. Pools und Swimlanes

Pools und Lanes repräsentieren Organisationseinheiten, in denen ein Prozess ausgeführt wird. Sie bilden einen Rahmen um die Aktivitäten, Ereignisse, Gateways und Sequenzkanten einer Prozessdefinition.

Eine Prozessdefinition muss sich vollständig innerhalb eines Pools befinden. Sequenzkanten dürfen nicht aus einem Pool herausführen. Durch Nachrichtenflüsse kann die Interaktion zwischen zwei Prozessen, welche durch verschiedene Pools dargestellt werden, repräsentiert werden. So kann zum Beispiel die Interaktion zwischen einer Organisation und ihren Kunden oder Lieferanten dargestellt werden. Dies ermöglicht die Ableitung der in Abschnitt 2.2.3 erwähnten Interaktionsschnittstelle.

Swimlanes unterteilen Pools in untergliederte Organisationseinheiten. Ein Prozess kann mehrfach die Swimlane wechseln. Hierdurch kann übersichtlich dargestellt werden, welche Abteilung in einem Unternehmen die in der Swimlane enthaltenen Aktivitäten ausführen muss.

2.3.6. Assoziationen, Datenobjekte und Artefakte

Der BPMN-Sequenzfluss beschränkt sich auf die reine Ausführungsreihenfolge, die weitergegebenen Marken tragen keine Informationen und sind somit nicht vergleichbar mit Dokumenten oder Werkstücken, die durch den Prozess wandern (vgl. Allweyer (2009), S.21). Mit Hilfe von Assioziationskanten können solcherart Objekte jedoch mit Flusselementen assoziiert werden.

Die Spezifikation sieht Datenobjekte und Datenbanken vor (vgl. OMG (2011), S. 206ff), mit welchen der Informations- und Datenfluss eines Prozesses angegeben werden kann. Mit Hilfe der Datenobjekte kann explizit der Output und der benötigte Input einer Aktivität definiert werden. Wenn die Modellierung mit Datenobjekten genutzt wird, so können nur die jeweils assoziierten Elemente auf ein Datenobjekt zugreifen. Das gleiche gilt für Datenbanken. Der Unterschied ist hier, dass ein Datenobjekt flüchtig ist und i.d.R. nur innerhalb einer Prozessinstanz genutzt wird, während die Datenbank einen permanenten Speicher, auf den aus verschiedenen Prozessinstanzen zugegriffen wird, darstellt und somit eine indirekte Interaktion ermöglicht. Bei beiden Objekten müssen gerichtete Assioziationskanten genutzt werden, um einen Lese- von einem Schreibzugriff zu unterscheiden.

Eine Elementgruppe namens Artefakte wird ebenfalls mit Assoziationskanten mit den Flusselementen verknüpft. Hier sind auch ungerichtete Kanten erlaubt. Die Spezifikation sieht die Ausprägung der Annotation vor, mit dessen Hilfe Textelemente an Elemente geknüpft werden können. Hierdurch kann zum Beispiel in der Prozessdefinition die Wiederholungsbedingung für Wiederholungsaktivitäten in die grafische Prozessdefinition integriert werden. Die Definition eigener Artefakte ist explizit erlaubt (vgl. Abschnitt 2.3.8.

2.3.7. Grenzen der Notation

Die BPMN 2.0 beschreibt wenige Möglichkeiten zur Interaktion der Prozesse mit anderen Systemen, welche zudem allesamt auf die Ausführung des Prozesses als Workflow konzipiert sind. Die Service-Aktivität ist in der Lage, einen beliebigen Webservice aufzurufen. Skript-Aktivitäten können Programmcode bearbeiten, der die Interaktion mit anderen Systemen beinhalten kann. Nachrichtenaktivitäten können Nachrichten an andere Systeme senden oder von ihnen empfangen. Benutzeraktivitäten fordern Benutzereingaben und bilden somit die Schnittstelle zur Interaktion mit natürlichen Personen. Datenbankobjekte erlauben das Lesen oder Schreiben von Daten in einer Datenbank. Alle anderen Elemente wirken sich auch bei der Ausführung des Workflows nur auf die Prozessinstanz selbst aus und sind nicht zur Interaktion mit anderen Systemen vorgesehen.

Alle Elemente haben zusätzliche, von ihrem Typ abhängige Attribute. In der BPMN sind primär technische Attribute vorgesehen, wie sie zur Ausführung des Modells als Workflow benötigt werden. So besitzen Sequenzkanten zum Beispiel ein Attribut zur Beschreibung einer Bedingung, die an Verzweigungen ausgewertet wird. Nachrichtenflüsse besitzen ein Attribut zur Beschreibung der zu transportierenden Daten und Mehrfachaktivitäten ein Attribut zur Beschreibung der Ausführungshäufigkeit. Fachliche Attribute „für Zeiten, Kostensätze, Kapazitäten usw. sind nicht vorgesehen“(vgl. Allweyer (2009), S. 24).

So beinhaltet die BPMN 2.0 explizit keine Attribute zur Beschreibung von Simulationeigenschaften (vgl. OMG (2011), S. 22). Die Ausführungssemantik ist zwar exakt festgeschrieben, jedoch mit der Zielsetzung der automatisierten Steuerung von Prozessen. Für die Simulation notwendige Parameter sind in der Spezifikation nicht enthalten. Ein Austausch von um Simulationsparameter angereicherten Modellen zwischen verschiedenen Softwaretools ist nicht möglich. Ein Vorschlag, die BPMN um Simulationsparameter anzureichern, findet sich in Abschnitt 3.4. Auch ist die Verbindung mehrerer Prozesse über gleichnamige End- und Startereignisse zwar in der Regel intuitiv verständlich und in Workflow-Systemen und Simulation umsetzbar, aber in der Spezifikation nicht formalisiert (vgl. Allweyer (2009), S.71).

Die Beschreibung der beteiligten Rollen an Aktivitäten hat sehr enge Grenzen. Es ist lediglich bei Benutzeraktivitäten ein Attribut für die ausführende Person vorgesehen, welches aber nicht näher spezifiziert ist. Bei anderen Aktivitäten kann die Organisationseinheit über den Namen der Swimlane angegeben werden, ein explizites Attribut, welches zur Laufzeit ausgewertet werden könnte, ist hier nicht vorgesehen. Eine Unterscheidung gemäß (vgl. Smith et al. (2007)), (vgl. Hightower (2008)) oder (vgl. Blokdijk und Menken (2008)) nach Durchführungsverantwortung, Kostenverantwortung, Fachverantwortung, Informationsrecht, Unterstützung oder Genehmigung entfällt in der BPMN 2.0 komplett. Auch die Zurordnung mehrerer Beteiligter an einer Aktivität ist in Kol-

laborationsdiagramme nicht möglich, hierfür sind jedoch die in Abschnitt 2.3.1 kurz erwähnten Choreographie- und Konversationsdiagramme vorgesehen.

2.3.8. Erweiterbarkeit der Notation

Weder die Simulation noch die Kopplung mit anderen Modellen ist durch die BPMN-Spezifikation vorgegeben. In der BPMN 2.0-Spezifikation wird aber exakt vorgesehen, dass und welche Erweiterungen oder Modifikationen möglich sind (vgl. OMG (2011), S. 44). Es ist zwingend notwendig, sich an diese Vorgaben zu halten und die entsprechend vorgegebene XML-Struktur einzuhalten, wenn sichergestellt werden soll, dass die Modelle auch in anderen Softwarewerkzeugen anzeigbar und manipulierbar bleiben.

Jedes Element besitzt ein Feld mit Erweiterungsattributen. Hier können völlig frei eigene Attribute für Elemente vorgegeben werden. Dies wird in Abschnitt 3.4 genutzt, um die für die Simulation benötigten Eigenschaften (auch ohne Kopplung an andere Teilmodelle) zu hinterlegen. Wird das Modell in ein zweites Softwarewerkzeug geladen, welches diese Attribute nicht kennt, so ignoriert es diese bei der Bearbeitung des Modells und legt sie bei einer erneuten Persistierung des manipulierten Modells wieder in der XML-Datei an der ursprünglichen Stelle ab. Wenn das Modell nach der Manipulation wieder mit der ursprünglichen Software geladen wird, findet diese die Erweiterungsattribute wieder vor und kann damit arbeiten. Es dürfen an keiner anderen Stelle Attribute hinzugefügt werden, sonst entspricht die XML-Datei nicht mit der Vorgabe aus der XSD-Datei und es wird zu Problemen beim Serialisieren und Deserialisieren der Modelle in verschiedenen Werkzeugen kommen.

Die Einführung völlig neuartiger Flusselemente ist nicht erlaubt, es dürfen also nur Ereignisse, Aktivitäten, Sequenzkanten und Gateways genutzt werden. Die Einführung von eigenen Artefakten ist hingegen erlaubt (vgl. OMG (2011), S. 66). Die Spezifikation sieht dabei explizit vor, dass Artefakte mit Objekten aus anderen Systemen verbunden werden können. Es ist dabei erlaubt, sowohl gerichtete als auch ungerichtete Assoziationskanten zu verwenden. Eigene Kantentypen dürfen nicht eingeführt werden.

Die visuelle Darstellung wird wie folgt eingeschränkt:

- An der grundsätzlichen Darstellung von Aktivitäten durch Rechtecke und Ereignissen durch Kreise darf nichts verändert werden.
- In Aktivitäten und Ereignissen dürfen eigene Symbole zur Visualisierung von Erweiterungsattributen eingeführt werden, solange eine Verwechslung mit bestehenden Symbolen ausgeschlossen ist.

- Bei Ereignissen muss darauf geachtet werden, dass schwarze (oder dunkle) Symbole sendende Ereignisse repräsentieren und weiße (oder helle) Symbole empfangende Ereignisse.
- Die Verwendung beliebiger Farben und Schriftarten bei allen Elementen ist erlaubt.
- Die Änderungen der Linienart von Kanten und Ereignissen ist nicht zulässig, da dies die Bedeutung der Elemente verändern würde.

(vgl. OMG (2011), S.8)

Somit werden an verschiedenen Stellen Erweiterungen oder Modifikationen erlaubt, die die Zielsetzung dieser Arbeit ermöglichen. Allweyer weißt darauf hin, dass ein zu exzessiver Gebrauch der Erweiterungsmöglichkeiten sich negativ auf das gemeinsame Verständnis des Standards und der Austauschbarkeit der Modellierungswerkzeuge auswirken kann Allweyer (2009) (S. 156). Auf diesen Aspekt wird bei der Konzeption dieser Arbeit ein hoher Wert gelegt.

3. Simulation von Geschäftsprozessen

Dieses Kapitel befasst sich mit der Simulation von Geschäftsprozessen. Die grundlegenden Begriffe und die Ziele der Simulation werden erläutert. Die historische Entwicklung angefangen mit der Verwendung von Werkzeugen ohne Unterstützung grafischer Modelleditoren bis hin zum aktuellen Stand von Wissenschaft und Technik im Bereich der Simulation von BPMN 2.0-Prozessen wird aufgezeigt. Eine Auswahl vorhandener Simulationswerkzeuge und insbesondere die dieser Arbeit als Grundlage dienende BPMN-2.0-Simulationsbibliothek werden vorgestellt.

3.1. Simulation

Um das dynamische Laufzeitverhalten eines Systems unter bestimmten Bedingungen zu untersuchen, können Experimente durchgeführt werden. Nicht immer ist es möglich oder ratsam, diese Experimente direkt am Originalsystem durchzuführen. Werden sie stellvertretend an einem Modell durchgeführt, so wird von Simulation gesprochen (vgl. Page (1991), S.7).

Die Zielsetzungen entsprechen zunächst den in 2.1 genannten Modellierungszielen, jedoch wird hier, im Gegensatz zu einer statischen Betrachtung von Modellbeschreibungen, das dynamische Laufzeitverhalten unter Betrachtung der Zustandsübergänge im Zeitverlauf betrachtet. Dies ermöglicht die in Abschnitt 2.2.3 angesprochene Prozessleistungsmessung, die nun alternativ zur Betrachtung der real laufenden Prozesse auch durch eine Simulation durchgeführt werden kann.

Wenn Simulationsexperimente mit einem immateriellen, formalen Modell am Rechner durchgeführt werden[2], wird zwischen zwei grundsätzlich verschiedenen Modellierungsansätzen anhand der Art der Zustandsübergänge unterschieden. Wie in Abschnitt 2.1.3 beschrieben, gibt es diskrete und kontinuierliche Modelle und dementsprechend auch unterschiedliche Simulationsansätze. Im Rahmen dieser Arbeit wird nur die diskrete

[2] Auch mit materiellen Simulationsmodellen lassen sich Experimente durchführen. Ein Beispiel hierfür ist der Wellensimulationskanal des Forschungsinstituts Küste der Universität Hannover, mit dessen Hilfe der Einfluss der Meere auf die Küste untersucht wird.

Simulation betrachtet, daher entfällt eine genauere Betrachtung von kontinuierlicher Simulation[3].

Geschäftsprozessmodelle werden durch Ereignisse angestoßen. Die Tätigkeiten laufen sequentiell ab und haben einen definierten Anfang und ein definiertes Ende, an welchen jeweils die Zustandsänderungen erfolgen. Sie beinhalten aufgrund der Struktur ihrer Beschreibung ausschließlich diskrete Zustandsübergänge und fallen somit in den Bereich der diskreten Modellbildung und Simulation. Für die Abbildung der Systemumgebung wurde in der Beschreibung der Zielsetzung der Arbeit (Abschnitt 1.3) bereits eingeschränkt, nur diskrete Modelle zu betrachten. Werden kontinuierliche Modellbeschreibungen für die Abbildung der Systemumgebung verwendet, so müssen diejenigen entscheidenden Zustandsübergänge, die sich auf die Prozessmodelle auswirken können, entsprechend diskretisiert werden.

Bei der Berechnung von Simulationsexperimenten ist der Fortschritt der Zeit nicht an den realen, kontinuierlichen Zeitfluss gekoppelt, sondern an die Leistungsfähigkeit des verwendeten Prozessors. Dabei werden nur die Zustandsänderungen betrachtet. Findet zwischen zwei Ereignissen kein weiteres Ereignis statt, verstreicht diese Zeit sprunghaft, ohne Rechenzeit zu benötigen. Die Simulation eines kompletten Geschäftsjahres kann theoretisch in wenigen Sekunden oder Minuten erfolgen. Der Zeitaufwand ist abhängig von der Komplexität des Modells und der Anzahl der zu berechnenden Zustandsübergänge. Die Berechnung kann bei Bedarf zwar realzeitsynchron erfolgen (vgl. Klückmann (2009)), das ist jedoch nur notwendig, wenn das Simulationsmodell mit einem System interagieren soll, welches nur in Realzeit arbeiten kann. In Joschko et al. (2009) wird dieses Prinzip angewendet, um mit Hilfe von Containerterminalmodellen die in Realzeit laufenden Steuerungssysteme des Terminals zu testen.

3.1.1. Simulationsparadigmen

Im Bereich der diskreten Simulation ist die Discrete Event Simulation (DES) am weitesten verbreitet. Sie ist ein heuristisches Verfahren und errechnet keine optimalen Lösungen für ein Problem, sondern versucht sich durch systematisches Experimentieren an eine gute oder praktikable Lösung anzunähern.

Sehr weit verbreitet haben sich der prozessorientierte und der ereignisorientierte Ansatz. Während bei der prozessorientierten Sicht der gesamte Lebenszyklus einer Entität als ein Prozess abgebildet wird, werden bei der ereignisorientierten Sichtweise die einzelnen

[3] Bei der kontinuierlichen Simulation wird ein System mittels Differentialgleichungen beschrieben, um in sich geschlossene Wirkungsketten zu untersuchen. Anwendung findet dies häufig in naturwissenschaftlichen Bereichen wie der Chemie, der Biologie oder der Mechanik. Der derzeit am weitesten verbreitete Ansatz hierfür ist System Dynamcis.

Zustandsänderungen als Ereignisse abgebildet. In beiden Fällen ist eine Ereignisliste vorhanden, welche eine Art Terminplaner (*Scheduler*) repräsentiert. Auf ihr wird eingetragen, zu welchem Zeitpunkt ein Ereignis stattfindet bzw. ein Prozess fortsetzt. In den *Scheduler* eingetragen werden diese Zeitpunkte jeweils von vorher stattfindenden Ereignissen bzw. Prozessen. Mindestens ein initialisierendes Startereignis muss bei der Modellbildung definiert werden (vgl. Page et al. (2000))S.11-18.

Grundsätzlich können sich beide Ansätze gegenseitig ersetzen oder ergänzen. Ein prozessorientiertes Modell lässt sich auch ereignisorientiert beschreiben und umgekehrt. Hierfür müssen Prozesse in die konkreten Zustandsänderungen zerlegt und durch Ereignisse ausgedrückt werden. Für eine Aktivität im Rahmen der Prozessmodellierung gäbe es dementsprechnd ein Startereignis und ein Endereignis. Der Unterschied zwischen den beiden Paradigmen liegt in der Sichtweise des Modellierers. Während er bei der prozessorientierten Sicht die Perspektive des abzuarbeitenden Prozesses einnimmt, um dessen Lebenszyklus zu beschreiben („Froschperspektive“), nimmt er bei der ereignisorientierten Sicht eher eine Perspektive von oben („Vogelperspektive“) ein, um die vorhandenen Zustandsänderungen global zu definiereen.

Geschäftsprozessmodellierung kann prozessorientiert stattfinden. Hier wird der Lebenslauf eines Prozesses als Iteration über die vorhandenen Modellelemente beschrieben. Dies schließt jedoch nicht aus, dass der ereignisorientierte Ansatz für eine Simulation verwendet werden kann. Jedoch sind weniger Transformationen an dem Modell nötig, wenn dem prozessorientierten Ansatz auch bei der Simulation gefolgt wird.

3.1.2. Stochastische Experimente

Ein wesentlicher Bestandteil für die Bildung eines diskreten Simulationsmodells sind stochastische Verteilungen. Sie werden genutzt, um realistische Schwankungen in einem System abzubilden, deren Abbildung deterministisch nicht möglich ist oder zu aufwendig wäre (vgl. Tumay (1996)). So sind menschliche Arbeitszeiten niemals konstant. Die Vielzahl der Einflüsse beginnt schon bei der persönlichen Befindlichkeit des Betroffenen, wie Krankheit, sozialen Probleme oder Müdigkeit. Arbeitnehmer können konzentriert arbeiten oder durch Kollegen abgelenkt werden. Daher werden menschliche Tätigkeiten niemals als Konstanten abgebildet, sondern zum Beispiel mit Hilfe einer Normalverteilung. Auch technische Anlagen verhalten sich nicht deterministisch. Fehler in den Anlagen sind nicht exakt vorherzusehen, jedoch steigt häufig mit zunehmender Laufzeit die Wahrscheinlichkeit für einen Ausfall der Anlage. Dies lässt sich zum Beispiel mit einer Weibull-Verteilung abbilden.

Um die geeigneten stochastischen Verteilungen abzubilden, sind gewisse Grundkenntnisse der Simulation und der Stochastik notwendig. Die Wahl der richtigen Verteilung und

der korrekten Parameter ist abhängig von der Quantität und Qualität der zur Verfügung stehenden Daten. Sehr häufig ist diese Datenlage die größte Hürde in einem Simulationsprojekt. Es gibt verschiedene Methoden, um solche Daten zu erheben. Die Laufzeit von Tätigkeiten kann eine Weile beobachtet werden, Prozessbeteiligte können befragt oder die Daten aus Monitoring-Systemen exportiert werden. Die so erhobenen empirischen Daten müssen analysiert werden, die korrekte Verteilung muss ausgewählt und die korrekten Parameter gefunden werden. In der Praxis werden folgende Verteilungstypen besonders häufig verwendet:

- Normalverteilungen sind in der Theorie besonders geeignet, um menschliche Tätigkeiten abzubilden. Allerdings fällt es häufig schwer, in Interviews die korrekten Parameter zu erfassen. Leichter ist dies, wenn aus Monitoring-Systemen reale Messdaten gezogen werden können.

- Dreiecksverteilungen eignen sich gut, wenn die Daten mit Hilfe von Interviews erhoben werden. Die Angabe von Minima, Maxima und Durchschnittswerten fällt erfahrenen Mitarbeitern in der Regel leicht. Es ist zu beachten, dass der Peek der Dreiecksverteilung den Modus darstellt und nicht den Mittelwert. Wenn das Delta zwischen Maximum und Mittelwert wesentlich von dem Delta zwischen Minima und Mittelwert abweicht, muss der korrekte Parameter für den Modus zunächst errechnet werden.

- Gleichverteilungen können gewählt werden, wenn lediglich eine Bandbreite der möglichen Dauer bestimmt werden kann, aber über die genaue Verteilung wenig bekannt ist.

Die Simulationskomponente erzeugt auf Basis parametrierter Verteilungstypen Pseudo-Zufallszahlen. Da es sich bei einem Computer um eine deterministische Maschine handelt, ist er nicht in der Lage echte Zufallszahlen zu erzeugen, sondern errechnet synthetisch hergestellte Pseudo-Zufallszahlen, welche nach bestimmten Algorithmen aus einem Startwert eine reproduzierbare Zufallszahlenfolge errechnen. Dieser Startwert wird als Seed bezeichnet. Ein Seed kann unter Zuhilfenahme der Systemuhr erzeugt werden, in dem aus der aktuellen Zeit in Millisekunden ein Wert errechnet wird. Auf diese Weise werden scheinbar zufällige Ereignisse in Computerspielen erzeugt. Für die Simulation besser geeignet ist ein manuell eingegebener und somit wiederverwendbarer Seed, der eine Reproduktion der erzeugten Zufallszahlen ermöglicht. Durch die Reproduzierbarkeit dieser Zufallszahlen können sie ohne Speicherung für eine Kontrollrechnung wieder erzeugt werden. Durch Verwendung unterschiedlicher Startwerte erhält man jedesmal eine neue Zufallszahlenfolge, somit eine andere Ereignisfolge und daraus resultierend eine andere Zustandsfolge. Ein Experiment unter Zuhilfenahme von Zufallszahlen wird Zufallsexperiment genannt. (vgl. Page (1991), S.102-112)

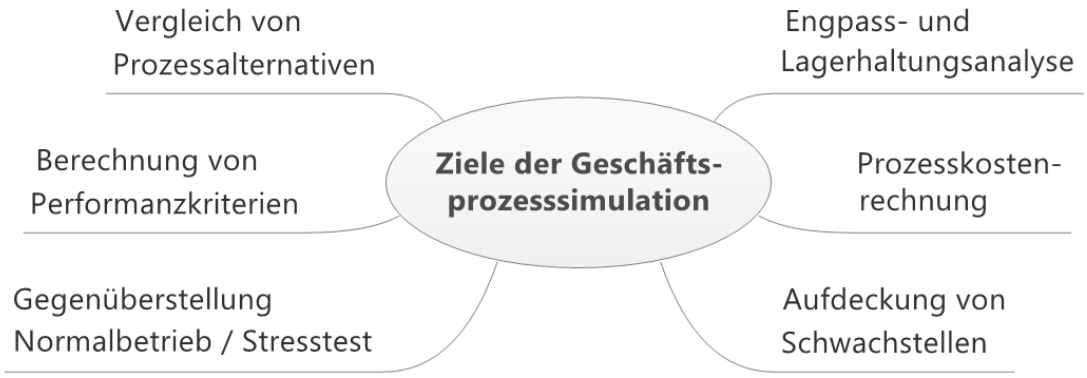


Abbildung 3.1.: Mögliche Ziele der Prozesssimulation

Die Verwendung von Zufallszahlen erfordert jedoch eine hohe Zahl von Experimentläufen. Erst durch häufige Wiederholung eines Zufallsexperimentes mit unterschiedlichen Startwerten, erhält man einen verlässlichen Überblick darüber, was im Originalsystem auch unter ungünstigen Ereignisfolgen passieren kann. Hierzu sollten zu den statistischen Ergebnissen Konfidenzintervalle errechnet werden, um die Präzision der bisher ermittelten Durchschnittswerte bestimmen zu können.

3.1.3. Potential der Simulation

Die Anwendung der Simuationstechnik im Rahmen des BPM bietet hohes Potential, denn wenn sie zielgerichtet durchgeführt wird, können risikoarm und kostengünstig sehr unterschiedliche Aspekte untersucht werden. Allen Untersuchungsgegenständen ist gemein, dass sie auf eine Analyse des dynamischen Laufzeitverhaltens der Prozesse abzielen und häufig im Rahmen von BPI-Vorhaben eingesetzt werden. Abbildung 3.1 zeigt eine Übersicht, über die möglichen Ziele der Geschäftsprozesssimulation, welche im folgenden erläutert werden.

(Rohleder und Silver 1997, S. 150) beschreiben, dass neue Konfigurationen oder *Varianten von Geschäftsprozessen* bereits vor ihrer Einführung mit Hilfe der Simulation verglichen werden können. Die Simulation schlägt dabei nicht selbständig Alternativen vor, so dass „der kreative Teil der Lösung [...] nach wie vor dem Anwender obliegt“ (Stehle 2014, S. 44). Jedoch haben die Prozessverantwortlichen zumeist eine Intuition, an welcher Stelle Verbesserungen vorgenommen werden können (Rohleder und Silver (1997)) oder können sich nach allgemeingültigen Vorschlägen wie in Youngblood (1994) richten. Mit Hilfe der Simulation lässt sich nun vorab abschätzen, ob eine Variante wirklich bessere Kennzahlen erzielen wird (vgl. Hlupic und Robinson (1998)). Es müssen

keine Experimente am laufenden Betrieb unternommen werden, was mit hohen Risiken oder Kosten verbunden sein könnte.

Zur Bewertung der Alternativen können verschiedene *Kennzahlen* herangezogen werden. Am einfachsten zu bestimmen sind die Durchlaufzeiten von Prozessen, aber auch die Prozesseffizienz, die Auslastung und die Wartezeiten innerhalb eines Prozesses sind wichtige Faktoren (vgl. Gladwin und Tumay (1994), Ardhaldjian und Fahner (1994), Tumay (1995)). Als Kennzahlen kommen all diejenigen in Betracht, welche abhängig vom Laufzeitveralten sind und im Zuge eines BPI-Projektes quantifiziert werden sollen (vgl. Abschnitt 2.2.3).

Sehr häufiger Untersuchungsgegenstand in Simulationsstudien sind *Ressourcen- und Engpassanalysen* (Hedstück (2013), S. 75, Page und Kreutzer (2005), S. 83ff, Gladwin und Tumay (1994)). Ressourcen können hier Mitarbeiter, Maschinen oder Werkzeuge sein, die zur Ausführung einer Tätigkeit notwendig sind. Wenn eine Ressource gerade anderweitig genutzt wird, so steht sie nicht für eine neue Tätigkeit zur Verfügung. Die Tätigkeit muss auf die Ressource warten und sich in eine Warteschlange einreihen. Wenn es an einer Stelle zu systemimmanenten Stauungen kommt, so wird von einem Flaschenhals gesprochen. Solche Ressourcenengpässe werden von der Simulation aufgedeckt, in dem sie Werte für die durchschnittliche Länge einer Warteschlange und die durchschnittliche Wartezeit ermitteln. Es ist jedoch nicht zwangsläufig zu erwarten, dass durch Hinzufügen weiterer Ressourcen unbedingt der Gesamtdurchsatz erhöht wird. So kann nun an einer anderen Stelle eine Stauung entstehen, die sich im schlimmsten Fall sogar noch schlechter auf das Gesamtsystem auswirkt. Daher ist es sinnvoll, auch Änderungen an der Ressourcen-Allokation vorab mit Hilfe der Simulation zu testen. Da solche Stauungen nur temporär unter bestimmten Bedingungen auftreten können, wird eine rein analytische Betrachtung der Ressourcenkapazitäten im Vergleich mit der vorhandenen Auftragslast der vorhandenen Komplexität nicht gerecht.

Auch kann eine bestehende Systemkonfiguration einer erhöhten Last ausgesetzt werden, zum Beispiel durch gesteigerte Kundenankunftsraten (vgl. Rücker (2008b)). Seit Stuttgart 21 wird dies im Volksmund als *Stresstest* bezeichnet. So kann überprüft werden, bis zu welcher Last eine Systemkonfiguration und die zur Verfügung stehenden Ressourcen stabil arbeiten. So kann der reale Betrieb vorab angepasst werden, um einer erwarteten Auftragssteigerung gerecht zu werden. In einigen Branchen sind erhebliche saisonale Unterschiede zu bedenken. In der Logistikkette für den Blumenhandel wird ein Großteil des Jahresumsatzes an nur wenigen Tagen im Jahr getätigt. Ist der Logistikdienstleister hierauf nicht angemessen vorbereitet, ist der Jahresgewinn in ernster Gefahr. Eine Vorabanalyse mit Hilfe der Simulation kann rechtzeitig helfen, Engpässe zu beseitigen oder Lieferzeiten zu optimieren.

Die *Gesamtkosten und Kosten pro Vorgang* spielen bei Geschäftsprozessen eine entscheidende Rolle (Bradley et al. (1995), Tumay (1996)). Bei marktwirtschaftlich arbeitenden

Betrieben sind sie der entscheidende Faktor. Mit Hilfe der Simulationstechnik können leistungsmengeninduzierte Kosten für einen Prozess errechnet werden, in dem an einzelnen Aktivitäten Kostenstellen benannt werden. Leistungsmengenneutrale Kosten wie Mitarbeitergehälter oder Mieten können aufwandsgerecht den Produkten und Prozessen zugeordnet werden, auf denen sie arbeiten. Insbesondere die aufwandsgerechte Zuordnung von neutralen Kosten an Prozessdefinitionen ermöglicht eine übersichtliche und leistungsstarke Verteilung auf einzelne Prozesse oder Produkte.

Letztlich eignet sich die Simulation aber auch zur *Absicherung von Modellentwürfen* (vgl. Gladwin und Tumay (1994), Enstone und Clark (2006)). Modellierungsfehler können aufgedeckt, kritische Rückkopplungen erkannt und Verklemmungen eventuell aufgedeckt werden. Es kann untersucht werden, ob einzelne Prozesszweige, Aktivitäten oder Ereignisse überhaupt durchlaufen werden, oder ob hier Fehler vorhanden sind, die bei einer statischen Betrachtung der Prozessmodelle nicht zu erkennen sind. Erst durch Untersuchung des dynamischen Laufzeitverhaltens werden diese ersichtlich. In Enstone und Clark (2006) wird angeführt, dass Verklemmungen in einer Prozessbeschreibung durch die stochastische Simulation grundsätzlich aufgedeckt würden. Dies ist jedoch nicht korrekt, denn mit Hilfe der Simulationstechnik kann zwar das Vorhandensein von Verklemmungen nachgewiesen werden, nicht jedoch die Verklemmungsfreiheit. Da Simulation auf Stochastik basiert, bleibt stets ein Restrisiko, dass eine Verklemmung während eines Simulationslaufs nicht entdeckt wurde. Es kann zwar eine Wahrscheinlichkeit quantifiziert werden, mit der ein Prozess verklemmungsfrei ist, diese kann jedoch niemals bei 100 Prozent liegen.

Gladwin und Tumay (1994) ordnen die möglichen Fragestellungen in zwei Gruppen ein: Die Ermittlung von Laufzeiten, des maximalen Durchsatzes, der benötigten Ressourcen und der Wartezeit des Kunden sind Fragen des Prozessdesigns. Die Zuteilung von Personal und anderweitiger Ressourcen, die Priorisierung von Kundenanfragen und die Planung von Instandhaltungseinsätzen an Maschinen sind Fragen des Prozessmanagements.

Enstone und Clark (2006) identifizieren hingegen drei potentielle Benutzergruppen zur Nutzung der BPMN-Simulation: Der Prozessdesigner kann den Entwurf neu erstellter Prozesse absichern (hier überschneidet er sich mit der Definition von Galdwin). Der Prozess Developer - hier Process Executor genannt- kann die Ausführbarkeit der zu implementierenden Prozesse testen. Der Prozess Controller - hier Process Manage genannt - kann eine Leistungsmessung für bestehende Prozesse vornehmen.

Dies zeigt, dass der mögliche Einsatzbereich der Simulation sehr groß ist. Einzelne Fragestellungen können dabei durchaus auch mit anderen Ansätzen betrachtet werden. So ist zum Beispiel die Zusicherung der Verklemmungsfreiheit von Prozessen durch eine formale, Petri-Netz-basierte Analyse verlässlicher. Für eine umfassende Analyse des Laufzeitverhaltens unter Berücksichtigung verschiedenster zeitlicher Aspekte basierend auf

vorhandenen Prozessdefinitionen ist die Simulationstechnik jedoch ein sehr gut geeignetes Mittel.

3.1.4. Modellbildungszyklus

Bei der Abstraktion von einem System und der Abbildung in einem Modell wird vor der Erstellung eines Simulationsmodells zunächst ein konzeptuelles Modell erstellt, welches die geplanten Abläufe und die für die Simulation relevanten Bestandteile des Systems beinhaltet (vgl. Page und Kreutzer (2005), S. 14). Dieses konzeptuelle Modell muss gemeinsam mit Experten aus der Domäne daraufhin validiert werden, ob es das reale System adäquat beschreibt. Es ist notwendig, dass die Darstellung des konzeptuellen Modells verständlich für diese Domänenexperten ist. Gegebenenfalls müssen Änderungen an dem Modell vorgenommen werden, woraufhin das Modell erneut überprüft wird. An dieser Stelle liegt ein entscheidender Vorteil der Simulation basierend auf Prozessmodellierungssprachen: Als Grundlage für das konzeptuelle Modell können die bereits vorhandenen Prozessbeschreibungen direkt verwendet werden. Unter Umständen sind geringe Anpassungen notwendig, um den zu untersuchenden Gegenstand detaillierter herauszuarbeiten. Die Idee dabei ist, dass diese Modelle für die Prozessverantwortlichen und die Prozessbeteiligten intuitiv verständlich sind oder sie die Modelle bereits aus dem internen Hausgebrauch kennen.

Erst wenn das konzeptuelle Modell eine zufriedenstellende Abbildung ist, wird es in ein Computermodell zur Simulation überführt. Je nach verwendetem Simulationswerkzeug muss zum Beispiel ausführbarer Code erzeugt werden, der durch das Werkzeug interpretierbar ist. Hier liegt ein zweiter Vorteil der Geschäftsprozesssimulation. Das konzeptuelle Modell stellt bereits das ausführbare Modell dar. Es wird keine zusätzliche Modelltransformation benötigt. Die Prozessdefinition muss lediglich mit Simulationsparametern angereichert werden. Durch die Durchführung von ersten Simulationsläufen entstehen Kennzahlen, welche eine Abschätzung des Systemverhaltens erlauben. An dieser Stelle muss verifiziert werden, ob das beobachtete Verhalten der vorgesehenen Modellbeschreibung entspricht, oder Änderungen an dem Modell notwendig werden. Bei dem herkömmlichen Ansatz muss das Verhalten des Computermodells zunächst mit dem konzeptuellen Modell verglichen werden. Anschließend kann es möglich werden, dass auch das konzeptuelle Modell nochmals mit der Realität verglichen werden muss, um hier Anpassungen vorzunehmen. Dieser Schritt der Verifikation wird durch die Zusammenlegung von konzeptuellen Modell und Simulationsmodell wesentlich vereinfacht. Auch müssen Kennzahlen, die aus dem realen System bekannt sind, mit den Ergebnissen der Simulation verglichen werden. Eventuell wird hierbei aufgedeckt, dass einige Parameter angepasst werden müssen.

Sobald das Verhalten des Simulationsmodells als valide Repräsentation der realen Systemdynamik eingestuft wird, kann die eigentliche Simulationsstudie beginnen. Es wird eine große Anzahl von Läufen durchgeführt, deren Ergebnisse Erkenntnisse über das reale System erlauben. Hierbei kann die Notwendigkeit weiterer Experimente entstehen, die eine veränderte Systemkonfiguration untersuchen, zum Beispiel um der Frage nach möglichen Verbesserungsmaßnahmen nachzugehen: Der Modellbildungszyklus beginnt von vorne.

3.2. Historische Entwicklung der Geschäftsprozesssimulation

3.2.1. Simulation mit Flussdiagrammen und DES-Werkzeugen

In den Neunziger Jahren wurde der Simulation als Analysemethode für betriebliche Prozesse zunehmend Aufmerksamkeit gewidmet. Gladwin und Tumay (1994) beschreiben, dass im betrieblichen Kontext in den vorangegangen 15 Jahren der Fokus der Simulation noch auf Produktionsprozessen lag (vgl. Robinson (1994)), sie aber für die nächsten 10 Jahre den Durchbruch der Simulation allgemeiner gefasster Geschäftsprozesse erwarten. Es wurde erkannt, dass die bloße Betrachtung von Prozessdefinitionen nicht ausreicht, um den zu erwartenden Nutzen der Änderung von Prozessdefinitionen zu beurteilen. Grundsätzlich sei die Analyse des dynamischen Laufzeitverhaltens von Prozessen durch Simulation daher ein adäquates Mittel.

In Bradley et al. (1995) werden bereits existierende BPI-Werkzeuge verglichen und die Simulation dabei als eine wünschenswerte Funktion genannt. Zur Bewertung der Werkzeuge wurden Kriterien definiert, welche unter anderem die Detailtiefe der Modellierungswerkzeuge und die Zuordnung von Ressourcen, sowie die untersuchten Aspekte des Simulationswerkzeugs (wie Zeiten und Kosten). Auch die Benutzerfreundlichkeit der Werkzeuge und das notwendige Fachwissen zur Bedienung wurde als Kriterium herangezogen. Letztlich konnten zu diesem Zeitpunkt keine Werkzeuge vollumfänglich überzeugen.

Tumay (1995) benennt die notwendigen Bestandteile eines Simulationsmodells wie folgt: Der Objektfluss durch einen Prozess (Objekte sind hierbei Kunden, Dokumente, Produkte oder Aufträge) muss ebenso beschrieben werden, wie die notwendigen Aktivitäten, die Ressourcen und Warteschlangen. Die Modellierungssprachen für Geschäftsprozesse sind zu diesem Zeitpunkt noch nicht weit entwickelt. Folglich nennt er als mögliche Ansätze zur Modellierung Flussdiagramm-basierte Werkzeuge, die leicht verständlich, aber limitiert bzgl. der Ausdrucksmöglichkeiten seien, sowie DES-Werkzeuge ohne grafische Unterstützung, die sich durch hohe Flexibilität auszeichnen, aber entsprechend schwerer nachzuvollziehen und zu validieren sind.

Ausdrucksstarke, ausreichend formalisierte Geschäftsprozessmodellierungssprachen sind zu diesem Zeitpunkt noch nicht etabliert. Dennoch wurden in den folgenden Jahren mehrere Publikationen veröffentlicht, die der Simulationstechnik in diesem Kontext hohen Nutzen attestierten (z.B. Rohleder und Silver (1997), Hlupic und Robinson (1998)).

1996 betont Tumay, das gerade die Kombination aus Flussdiagramm-Editoren und Simulation eine wertvolle Entscheidungsunterstützung darstellt (Tumay (1996)). Er definiert dabei die Elemente, die das Flussdiagramm besitzen muss (wie z.B. Aktivitäten und Gateways), um eine adäquate Modellierung der Prozesse zu ermöglichen, ohne dabei jedoch auf mittlerweile existente Prozessmodellierungssprachen wie die EPK zurückzugreifen.

Laut einer Benchmarking-Studie, bei der 60 internationale Organisationen befragt wurden, haben im Jahr 1997 bereits zehn Prozent der Firmen die Simulation im Zuge von BPI-Projekten genutzt, alle anderen verwenden nach wie vor Flussdiagramm-Editoren zur Erstellung von Prozessdefinitionen, Excel-Spreadsheets und Textverarbeitungen (vgl. Hiatt (1999)).

In Hlupic und Robinson (1998) wird kritisiert, dass die meisten BPI-Werkzeuge immer noch nicht in der Lage sind, Entscheidungen durch Untersuchung des dynamischen Laufzeitverhaltens mittels Simulation abzusichern. Die meisten Softwareprodukte sind höchstens in der Lage, Prozesslaufzeiten zu errechnen. Die Prozesse werden dabei meist als einfaches Flussdiagramm dargestellt, was er bereits als Entgegenkommen für das Verständnis von Prozessen empfindet. Geschäftsprozessmodellierungssprachen werden zu diesem Zeitpunkt selten genutzt, es sind eher allgemeingehaltene Simulationswerkzeuge, die herangezogen werden.

3.2.2. Verwendung von standardisierten Modellierungssprachen

Mit der Etablierung der EPK als Geschäftsprozessmodellierungssprache kam die BPMS-Software ARIS Toolkit auf den Markt, welche eine Sammlung verschiedener Werkzeuge wie EPK-basierte Modelleditoren darstellt. Ein Werkzeug aus dieser Sammlung ist der ARIS Simulator. Mit Hilfe des grafischen Simulationssystems SIMPLE++ wurden für einige Elemente der EPK enstprechende Simulationsobjekte zur Verfügung gestellt (ARIS-Ereignis, ARIS-Funktion, ARIS-Materialressource, ARIS-Person und ARIS-Organisationseinheit), was eine Simulation von EKP-basierten Prozessmodellen ermöglichte (vgl. Schmauder und Schmidt (1998)). Dies stellte bereits einen großen Schritt zugunsten der Validierbarkeit und der Benutzerergonomie dar.

Kalnins et al. (1998) vergleicht verschiedene auf dem Markt befindliche Werkzeuge zur Simulation von Geschäftsprozessen. Zu diesem Zeitpunkt basieren bereits alle Werkzeuge auf graphischen Notationen. Es gibt zudem Werzeuge wie den ARIS Simulator,

die auf Geschäftsprozessmodellierungssprachen aufsetzen. Jedoch unterstützen diese jeweils nur ein Teilset der in der entsprechenden Sprache vorliegenden Elemente. Es wird bemängelt, dass für die Simulation jeweils eine eigene Semantik geschaffen wurde, die nicht zwangsläufig der von der Sprache vorgesehenen Semantik entspricht. Der ARIS Simulator wird als eindeutig führend auf dem Markt angesehen.

In Cuntz (2004) wird bestätigt, dass verschiedene Werkzeuge zur Simulation von EPK die Modelle unterschiedlich interpretieren und somit Inkonsistenzen in der Interpretation der Semantik vorliegen. Daher wurde bereits in Langner et al. (1998) dem Ansatz gefolgt, EPK-Modelle nach einem selbstgewählten Formalismus in Petri-Netze zu überführen, und die Semantik der EPK als diejenige der resultierenden Petri-Netze zu definieren. Eine abschließende Lösung der Probleme bei der EPK-Simulation ist dies allerdings nicht, da nicht sichergestellt werden kann, ob der Modellierer der EPK-Modelle diese gleichermaßen interpretiert.

„Bei den meisten Simulationstools für ereignisgesteuerte Prozessketten steht nicht die vernetzte Flussdynamik vieler interagierender Prozesse im Vordergrund, sondern es werden viele gleichartige Prozessinstanzen nacheinander in das Modell geschickt, um Kenngrößen wie Auslastung von Ressourcen, Durchsatz oder Durchlaufzeit zu ermitteln oder um überflüssige Modellkomponenten zu identifizieren“ (Hedstück (2013), S. 142). Umfassende Simulationsanalysen einer Prozesslandschaft mit Hilfe von EPK-Modellen scheint also bis heute nicht möglich zu sein.

Aufgrund der Probleme der nicht eindeutigen Ausführungssemantik, wurden in diesen Jahren zur Automatisierung von Workflows noch XML-basierte Sprachen ohne grafische Repräsentation verwendet. Beispiele für diese Sprachen sind die Business Process Execution Language (BPEL), die XML Process Definition Language (XPDL) oder die jPDL Process Definition Langauge (jPDL[4]). Da die Ausführungssemantik dieser Sprachen im Gegensatz zu den existierenden graphischen Notationen eindeutig spezifiziert ist, werden sie in einigen Publikationen und Werkzeugen auch zur Erstellung von Simulationsmodellen verwendet.

In Rücker (2008c) wird ein Werkzeug zusammengestellt, welches die Workflow Engine JBoss jBPMN mit der Simulationsbibliothek DESMO-J verknüpft. Hierfür wurde jBPMN dahingehend erweitert, dass es Aufrufe von Aktivitäten an die Simulationsbibliothek weiterleitet und die Systemzeit von jBPMN und DESMO-J synchronisiert wird (vgl. Rücker (2008a)). Die Prozessmodelle werden jedoch in der XML-basierten Sprache jPDL zur Verfügung gestellt. Auch die Verwendung von XPDL oder

[4]Stellenweise wird das Akronym jPDL mit Java Process Definition Language übersetzt. Der Hersteller JBoss selbst gibt jedoch an, dass es ausgeschrieben „jPDL Process Definition Language“ bedeutet. (http://docs.jboss.com/jbpm/v3/userguide/, zuletzt abgerufen am 25. Februar 2014). Die Verwendung von rekursiven Akronymen wird in der IT-Szene als humorvoll angesehen.

BPEL wird von JBoss untersützt, von Rücker jedoch nicht verwendet. Die Simulationstechnologie wird hier mit einer ausreichend formalisierten Sprache verknüpft, jedoch gehen sämtliche in Kapitel 2 beschriebenen Vorteile verloren, da es sich nicht um grafische Geschäftsprozessmodellierungssprachen handelt. Eine Validierung von XML-Dateien durch Prozessverantwortliche oder -beteiligte erscheint nicht praktikabel. Es gibt Werkzeuge, welche Sprachen wie die BPMN in BPEL überführen, jedoch gehen hierbei einige Informationen verloren, da die Sprachen nicht komplett deckungsgleiche Elemente besitzen.

3.2.3. Simulation von BPMN-Modellen

Ein Jahr nach der Veröffentlichung der BPMN 1.0 Spezifikation präsentierten Enstone und Clark (2006) eine erste Lösung zur Simulation von BPMN-Prozessen. Es wird betont, dass die BPMN sich zum Standard für Prozessmodellierung etablieren wird. Die Verwendung von BPMN-Modellen zur Durchführung von Simulationsstudien hat zum Vorteil, dass es eine Vielzahl guter Modellierungstools gibt, aus denen die Modelle für eine Simulation exportiert werden können. Die bereits für Dokumentationszwecke erstellten Prozessmodelle können dabei wiederverwendet werden. Sollte in einer Unternehmung die BPMN verwendet werden, so finden sich dort Personen, die die Notation beherrschen und Prozessmodelle für die Simulation entwerfen und validieren können.

Auch in Nicolae et al. (2009) wird der Nutzen einer auf BPMN-Modellen basierenden Simulation betont. Es wird festgestellt, dass es immer noch viele Tools gibt, die Prozesssimulation ohne die Verwendung von Geschäftsprozessmodellierungssprachen anbieten, obwohl die Vorteile einer für alle Beteiligten verständlichen, graphischen Notation auf der Hand liegen. Es wird auch betont, dass der Entwicklungsaufwand und dadurch die entstehenden Kosten bei einer auf BPMN basierenden Simulation geringer sind. Es wird jedoch bemängelt, dass einige Aspekte des Laufzeitverhaltens von BPMN 1.0-Modellen nicht ausreichend genau spezifiziert sind, um eine Simulation vollständig umzusetzen.

In Blum (2010) wird ebenfalls eine Bibliothek zur Simulation von BPMN-Prozessen vorgestellt. Das Konzept basiert auf einer Überführung von BPMN 1.0 Modellen in gefärbte Petri-Netze mit Hilfe eines Modellkonvertierers. Hier wird bereits kritisiert, dass bestehende Simulationstools zur BPMN diese häufig nicht vollständig implementieren und es zudem keine Werkzeuge gibt, die erweiterbar sind. Als Beispiel für eine mögliche Erweiterbarkeit wird hier angeführt, dass es in bestehenden Werkzeugen nicht möglich ist, etwa die Berechnung von Steuerraten an einer Aktivität vorzunehmen, hierauf Entscheidungen im Prozess aufbauen zu lassen und diese Berechnungen statistisch auszuwerten und insbesondere keine Werkzeuge Ansätze bieten, um selbsttätig eine solche Erweiterung vornehmen zu können. Als Lösung für dieses Problem wird eine Open-Source-Architektur bereitgestellt, welche es vereinfacht, zusätzliche Funktionalitäten an einzelne

BPMN-Elemente anzuheften. Hierdurch wird die Vielfalt der möglichen zu untersuchenden Fragestellungen bereits beträchtlich erweitert. Es wird jedoch keine Lösung vorgestellt, welche es ermöglicht, die BPMN bereits auf Modellierungsebene um zusätzliche Elemente zu erweitern, um mit ihnen die Interaktion mit Teilmodellen zu realisieren und somit auf bestimmte Domänen zugeschnittene Softwarelösungen bereitzustellen.

Mit der Veröffentlichung der BPMN 2.0 Spezifikation im Jahre 2011 ist das Problem der nicht eindeutigen Ausführungssemantik von graphischen Prozessnotationen endgültig gelöst. Zwar ist die Simulation nicht Bestandteil der Spezifikation (vgl. OMG (2011), S. 22), jedoch besitzt die BPMN 2.0 nun eine wohldefinierte Ausführungssemantik und ermöglicht somit prinzipiell auch die Simulation (vgl. Dijkman und Gorp (2011) oder Jander et al. (2011)). Da die Standardisierung fehlt, ist eine Portierung von Simulationsmodellen zwischen verschiedenen Softwarewerkzeugen leider nicht möglich. Dennoch hat sich die Zahl der BPMS-Tools, welche die BPMN zur Modellierung und die Simulation als Analysemethode anbieten, stark vergrößert. Eine neue Ära der Geschäftsprozesssimulation scheint begonnen zu haben.

Unter anderem werden in Onggo und Karpat (2011), in Guizzardi und Wagner (2011) und in Joschko et al. (2012) Ansätze zur Simulation von BPMN 2.0-Modellen vorgestellt. Die .NET-basierte Lösung aus Joschko et al. (2012) unterstützt die BPMN 2.0 nahezu vollständig und baut auf den Erfahrungen der soeben vorgestellten Publikationen auf. Die Laufzeit von Aktivitäten oder die Zwischenankunftszeit von Ereignissen wird mit Hilfe von stochastischen Verteilungen bestimmt. Die Bibliothek bietet die Möglichkeit, eigene Berechnungen mit Hilfe von Python-Skripten an Aktivitäten zu hinterlegen, ohne dass der Quellcode dafür geändert werden müsste. Diese Bibliothek wurde als Grundlage für die prototypische Umsetzung der Konzepte dieser Arbeit verwendet und dafür mit den entsprechenden Erweiterungsmöglichkeiten für Verknüpfung mit domänenspezifischen Teilmodellen versehen. Die Funktionsweise dieser Bibliothek im Zustand vor Integration der Konzepte dieser Arbeit wird daher in Abschnitt 3.4 detaillierter vorgestellt. Eine BPMS-Software, in der die Bibliothek (bisher ohne Verwendung der in dieser Arbeit vorgenommen Erweiterungen) erfolgreich integriert wurde, wird in Abschnitt 3.3.2 vorgestellt.

3.3. Simulationswerkzeuge

Bei der Betrachtung von Simulationssoftware können zwei Kategorien unterschieden werden: Es gibt Simulationsbibliotheken, welche selbst keine ausführbaren Programme darstellen, sondern die notwendige Kernfunktionalität eines Simulators in Form von Klassen oder Komponenten zur Verfügung stellen (vgl. Göbel et al. (2012)). Hiermit können für den jeweiligen Anwendungssfall speziell zugeschnittene Lösungen entwickelt

werden. Demgegenüber stehen integrierte Entwicklungsumgebungen, welche in der Regel für einen konkreten Anwendungsfall zugeschnitten und out-of-the-box für den Anwender nutzbar sind. Sie unterstützten den gesamten Modellbildungszyklus und stellen somit zum Beispiel Modelleditoren bereit. Die Einarbeitung in eine Simulationsbibliothek erfordert Programmierkenntnisse beim Modellierer, jedoch bietet sie mehr Flexibilität als eine integrierte Entwicklungsumgebung. In der Regel werden Simulationsbibliotheken verwendet, um die Basisfunktionalität für eine integrierte Entwicklungsumgebung, beispielsweise für die Geschäftsprozesssimulation, bereitzustellen.

3.3.1. DESMO-J

DESMO-J ist eine Open-Source-Simultionsbibliothek für DES die in zahlreichen wissenschaftlichen Publikationen und in verschiedenen kommerziellen, integrierten Entwicklungsumgebungen als Simulationskern verwendet wurde (vgl. Göbel et al. (2012) und DESMO-J (2014)). Sie wurde am Fachbereich Informatik der Universität Hamburg entwickelt (vgl. Page et al. (2000)).

Es ermöglicht die gleichzeitige Verwendung der prozessorientierten und ereignisorientierten Paradigmen innerhalb eines Simulationsmodells. Auch gibt es mit FAMOS eine DESMO-J-Erweiterung, die zusätzlich die Integration von agentenbasierten Simulationsmodellen erlaubt (vgl. Knaak et al. (2006)).

DESMO-J ist eine objektorientierte Bibliothek die in Java entwickelt wurde; es liegen aber auch Portierungen für andere Sprachen wie Delphi oder .NET vor. Abbildung 3.2 zeigte eine Übersicht über die wichtigsten Klassen. Bestandteil der Bibliothek sind die für die Simulation benötigten Infrastruktur-Klassen, zum Beispiel eine Simulationsuhr, eine Ereignisliste, eine Experiment-Klasse und Warteschlangen. Zur Erzeugung von Ergebnisberichten werden Statistik-Klassen zur Verfügung gestellt (vgl. Page und Kreutzer (2005)).

Zur Modellerstellung muss der Benutzer ausgewählte abstrakte Klassen für Ereignisse, Prozesse oder Entitäten implementieren und in einer ebenfalls zu implementierenden Modell-Klasse zusammenfassen. DESMO-J unterstützt dabei die Bildung von hierarchischen Modellen. Es können Teilmodelle in ein Simulationsmodell integriert werden, deren Entitäten mit Entitäten aus anderen Teilmodellen interagieren können. Jedes Teilmodell kann eigene Statistiken zur Auswertung des Simulationslaufes führen, diese werden von DESMO-J automatisch in einem Ergebnisbericht für das Gesamtmodell zusammengefasst.

Bei der Entwicklung eines Simulationsmodells wird der Entwickler zunächst die relevanten Entitäten im Untersuchungsgegenstand und deren mögliche Zustandsübergänge identifzieren, sich für eine der Simulationsparadigmen entscheiden und die entsprechenden

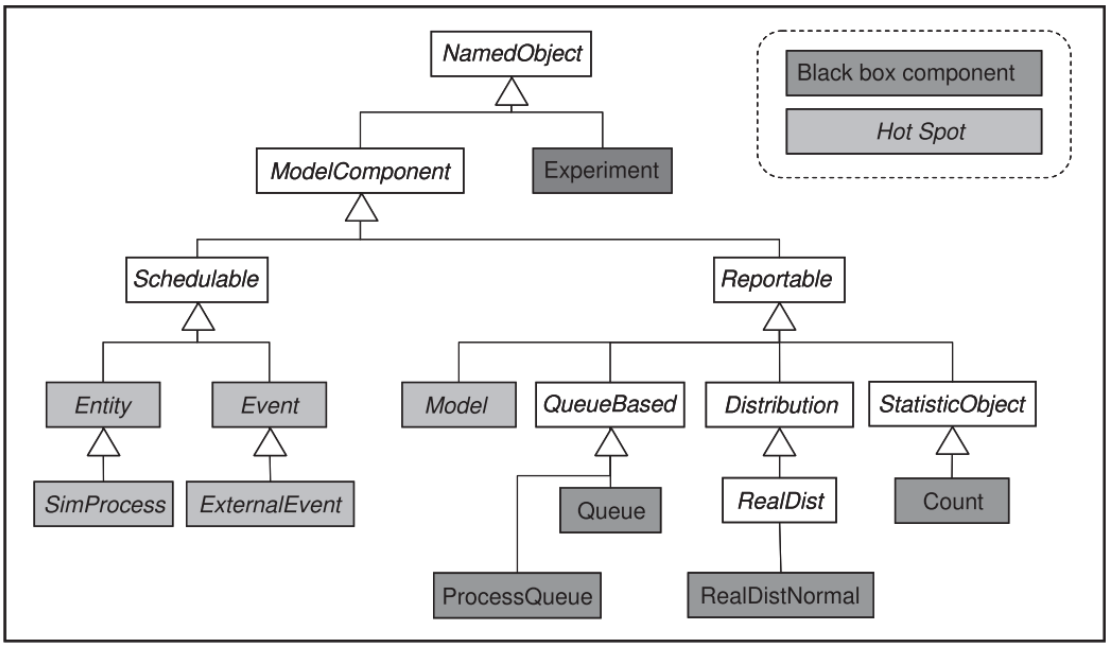


Abbildung 3.2.: Auszug aus der Klassenhierarchie von DESMO-J (Originaltitel: „A snapshot of DESMO-J's class hierarchy"). Abbildung aus: Page und Kreutzer (2005)

Klassen implementieren. Auf diese Weise wurde bereits für die verschiedensten Anwendungsdomänen Erweiterungen für DESMO-J geschaffen, zum Beispiel für die Containerterminalsimulation (vgl. Joschko et al. (2009) oder Bornhöft (2009)), für Transportnetzwerkanalysen (vgl. Goebel (2013)) oder die Geschäftsprozesssimulation (vgl. Abschnitt 3.4).

Sofern die Klassen für eine Anwendungsdomäne einmal implementiert wurden, können hiermit in der Regel Simulationen mit unterschiedlichen Fragestellungen durchgeführt werden. Für verschiedene Anwendungsdomänen, wie zum Beispiel Hafenlogistik, stehen auch vorgefertigte Erweiterungen auf der Webseite von DESMO-J zum Download bereit. Auch existieren Erweiterungen um eine 2d- oder eine 3d-Animation (vgl. Sun (2010)) von Simulationsläufen zu realisieren.

DESMO-J kann auch als Simulationskern verwendet werden, um hierauf aufbauend eine integrierte Entwicklungsumgebung für einen bestimmten Anwendungsbereich zu entwickeln. So wurde in Wohlgemuth (2005) ein DESMO-J-Derivat in Delphi zur Durchführung von Stoffstromanalysen in der Software Milan genutzt und die BPMN-Erweite-

rung von DESMO-J (vgl. Abschnitt 3.4) wurde in die Software IYOPRO integriert (vgl. Abschnitt 3.3.2).

Da für DESMO-J bereits eine BPMN-Erweiterung existiert und zudem in DESMO-J für die verschiedensten Domänen Modelle (oder Teilmodelle) implementiert werden können, dient es auch als Grundlage für den in dieser Arbeit erstellten Prototyp.

3.3.2. IYOPRO

IYOPRO ist eine Business Process Management Suite (BPMS), deren Kernstück ein grafischer Editor für Modelle der BPMN 2.0 ist. IYOPRO bietet Validierungsfunktionen zur syntaktischen Überprüfung der Modelle, beinhaltet Funktionen zur Ausleitung von Prozessdokumentationen im Word- oder HTML-Format und enthält eine vollwertige Workflow Engine inklusive einem Workflow Portal zur Ausführung von Prozessen. Es unterstützt die Modellierung von Organisationsdiagrammen zur Abbildung der Struktur eines Unternehmens und die Modellierung von Prozesslandkarten zur Gestaltung einer Übersicht über die vorhandenen Prozesse. Die Definition interessensbezogener Sichten ist möglich, was die Modellierung eines spezifischen Fokus auf einen Prozess erlaubt. Projektmappen erlauben die Zusammenfassung mehrer Prozessdefinitionen. Eine Animationsfunktion unterstützt die Überprüfung der sequentiellen Abarbeitungsfolge von Aktivitäten.

Die BPMN-Bibliothek aus Joschko et al. (2012) wurde in einer Kooperation zwischen dem Hersteller Intellivate GmbH und dem Technologietransferverein HiTec e.V. des Fachbereichs Informatik der Universität Hamburg in IYOPRO integriert. Hierdurch wird eine Analyse von Prozessdefinitionen gemäß der in Abschnitt 3.1 erläuterten Aspekte möglich. So bietet IYOPRO einen Editor zur Parametrierung der in DESMO-J vorhandenen stochastischen Verteilungen (vgl. Abbildung 3.3) und eine grafische Aufbereitung der Experimentergebnisse mit Hilfe von Diagrammen. Die Organigramme werden hierbei zur Definition von Ressourcen verwendet, mit den Aktivitäten verknüpft und erlauben somit eine Ressourcen- und Engpassanalyse.

Vorzüge von IYOPRO sind seine vielfältigen Funktionen und die ergonomische Bedienbarkeit, die in zahlreichen studentischen Vergleichsstudien bescheinigt wurde. IYOPRO findet vorrangig Verwendung im kommerziellen Umfeld. Auch an verschiedenen Hochschulen im deutschsprachigen Raum wird es in der Lehre zur Ausbildung in der Modellbildung und Simulation genutzt. Hierdurch konnte die Praxistauglichkeit der Simulationskomponente nachhaltig belegt werden. IYOPRO ist eine browserbasierte Webanwendung und kann ohne die Notwendigkeit einer Installation unter IYOPRO (2014) ausprobiert werden. IYOPRO und die Simulationskomponente werden stetig weiterentwickelt.

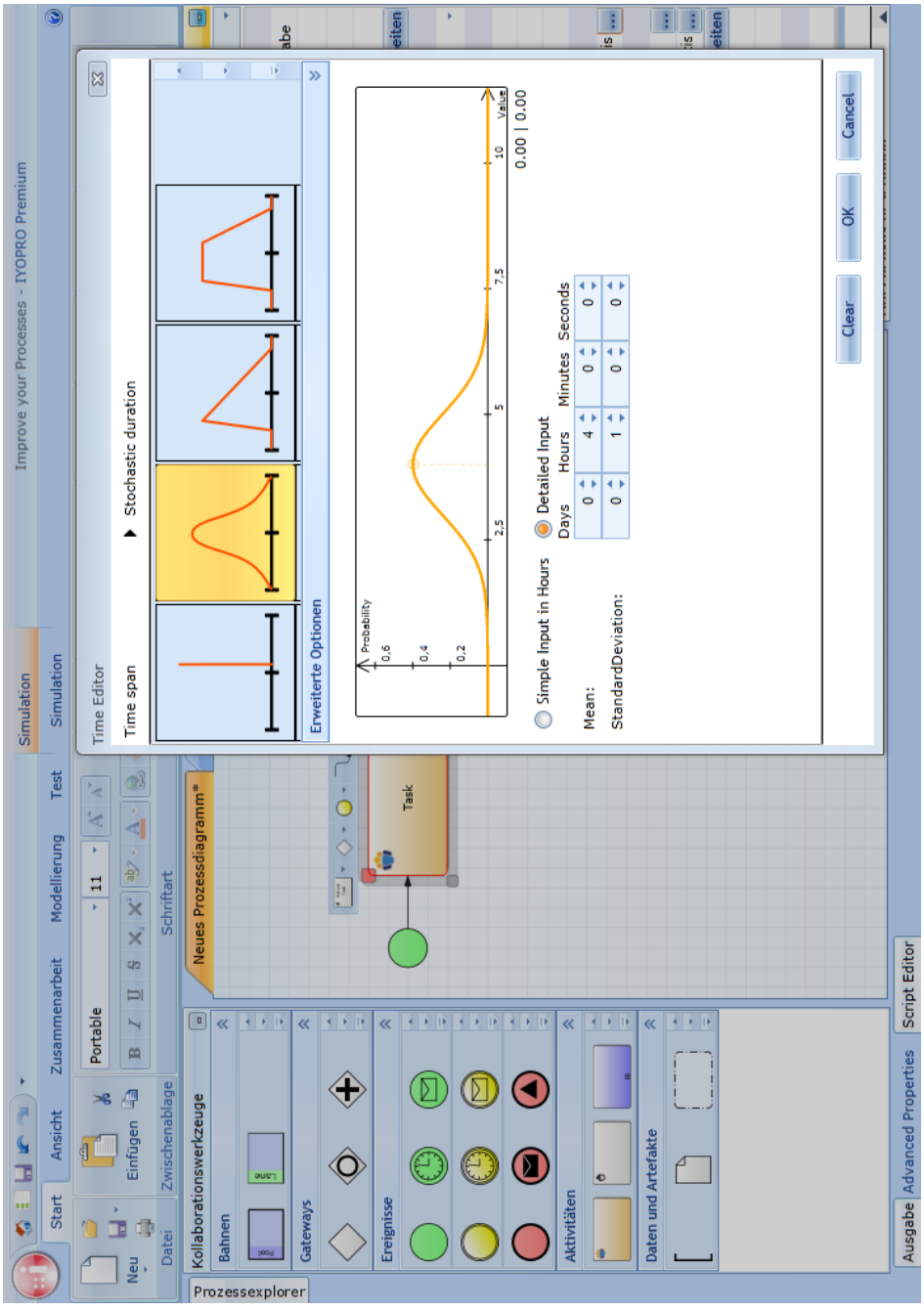


Abbildung 3.3.: Benutzeroberfläche von IYOPRO zur Parametrierung einer stochastischen Verteilung

3.3.3. Weitere Werkzeuge

Es gibt zahlreiche weitere Werkzeuge, welche die Simulation von Geschäftsprozessen unterstützen. Nicht alle Werkzeuge, die in ihrer Produktspezifikation angeben, eine Simulationsanalyse anzubieten, können dies tatsächlich erfüllen. Teilweise wird eine simple Animationsfunktion der sequentiellen Ausführung bereits als Simulation angepriesen. Hierbei werden jedoch weder Statistiken zu Kennzahlen generiert, noch die stochastischen Einflüsse auf Prozesse betrachtet. Auch gibt es Werkzeuge, die zwar die Modellierung in verschiedenen Notationen erlauben, die Simulation jedoch nur für einen Teil dieser Notationen anbieten. Beispiele für Werkzeuge, welche die stochastische Simulation von BPMN 2.0 Prozessen tatsächlich beherrschen, sind:

- IYOPRO Professional Edition
- eClarus Business Process Modeler for SOA Architects
- Bizagi Process Modeller
- Bonita BPM Community
- Logizian Simulacian
- ADONIS Community Edition

In Law et al. (2000) werden verschiedene Eigenschaften identifiziert, anhand derer diese Werkzeuge bewertet werden können. Dies ist die Einfachheit der Benutzung, die Flexibilität bei der Modellierung, der Kundensupport und die Dokumentation, die Zufallsgeneratoren, die stochastischen Verteilungen und die Reproduzierbarkeit der Läufe, die Möglichkeit Anlaufphasen zu identifizieren, den Detailgrad und die Aufbereitung der Ergebnisberichte sowie die Möglichkeit eigene zu messende Kennzahlen zu definieren. In Jansen-Vullers und Netjes (2006) werden verschiedene Werkzeuge untersucht. Sie unterscheiden hierbei Werkzeuge, die speziell auf Geschäftsprozesssimulation zugeschnitten sind sowie sogenannte General Purpose Simulation Tools, welche allgemeinere Ansätze zur Modellbildung und Simulation bieten. Die untersuchten Geschäftprozesssimulationswerkzeuge sind weniger flexibel, da sie auf die Betrachtung der Prozesse beschränkt sind. Die General Purpose Simulations Tools sind komplizierter zu bedienen, unterstützen nicht die Darstellung in Geschäftsprozessmodellierungssprachen, ermöglichen dafür aber die Analyse des Systems über die Prozessgrenzen hinaus. Die vorliegende Arbeit soll diesem Dilemma entgegenwirken, indem es die Modellierung von Geschäftsprozessen ermöglicht, aber zur Betrachtung anderer relevanter Bestandteile des Gesamtsystems unter Verwendung einer General Purpose Bibliothek erweitert werden kann.

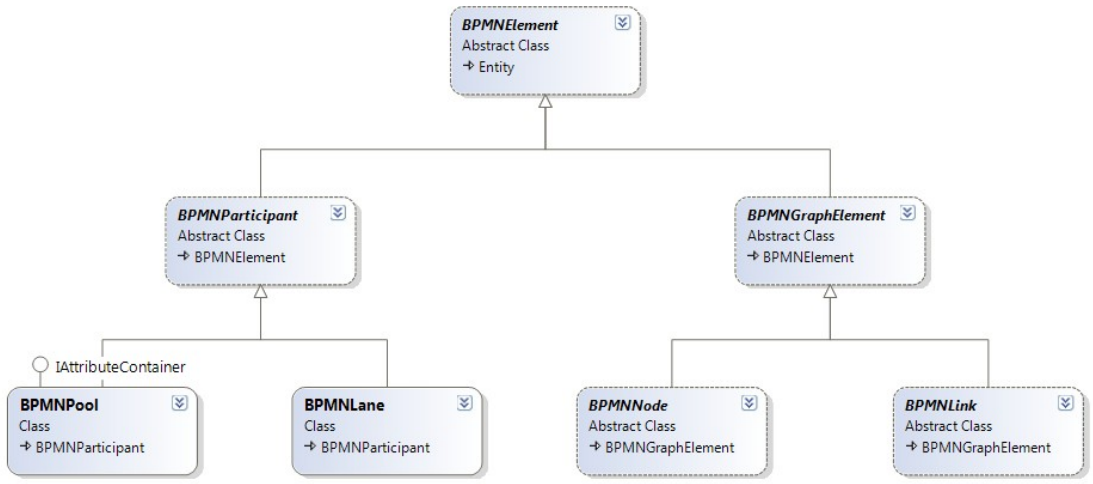


Abbildung 3.4.: Basisklassen der BPMN-Bibliothek von DESMO-J

3.4. BPMN 2.0-Simulation mit DESMO-J

Im folgenden wird die Funktionsweise einer BPMN-Erweiterung von DESMO-J beschrieben, wie sie in Joschko et al. (2012) veröffentlicht wurde. Da es bei weitem nicht die einzige existierende Lösung zur Simulation von BPMN 2.0-Prozessen ist, ist dies nicht der innovative Aspekt der vorliegenden Arbeit. Es handelt sich aber um eine wissenschaftlich fundierte Simulationsbibliothek, welche als Grundlage für die Implementation der Erweiterungskonzepte in den folgenden Kapiteln dient. Die Bibliothek stellt kein selbstständig ausführbares Programm dar und enthält auch keine Benutzeroberfläche, sondern wurde sowohl in die kommerzielle BPMS-Software IYOPRO (vgl. Joschko et al. (2012)) sowie in die prototypische Software DesmoWindparkStudio (vgl. Kapitel 6 und 7)) integriert.

3.4.1. Klassendiagramm der Prozesselemente

Vor dem Starten eines Simulationsexperiments werden von einer *Simulationsfabrik* alle Elemente der Prozessdefinitionen durchiteriert, um für jedes ein entsprechendes Objekt aus der DESMO-J-Bibliothek zu erzeugen. Die Abbildung 3.4 zeigt eine grundlegende Übersicht über die Klassenstruktur dieser DESMO-J-Objekte.

Alle BPMN-Elemente erben von der Klasse `BPMNElement`, welche von der DESMO-J-Klasse Entity abgeleitet ist. Da die BPMN in großen Teilen wie ein mathematisch gerichteter Graph aufgebaut ist, der aus Knoten (`BPMNNode`) und Kanten (`BPMNLink`) besteht,

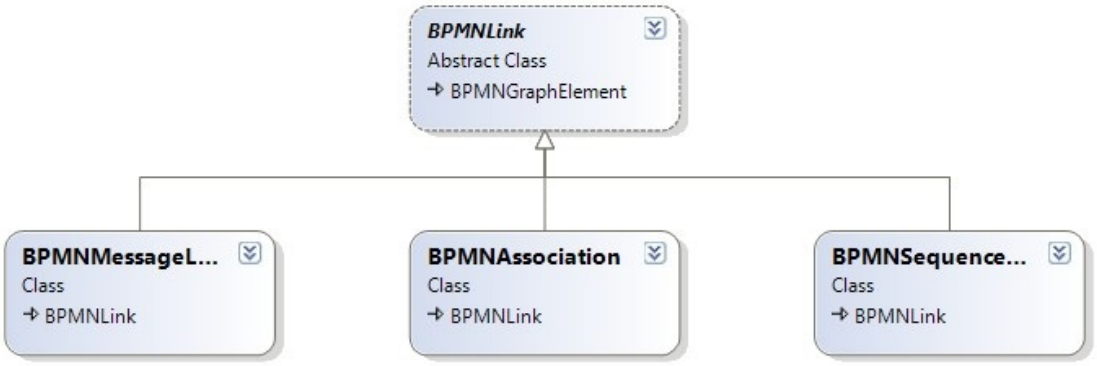


Abbildung 3.5.: Kanten in der BPMN-Bibliothek von DESMO-J

gibt es eine Klasse `BPMNGraphElement`. Dieses besitzt ein Property `Owner`, welches in der Regel die Referenz auf den betreffenden Graphen hält, in einigen Fällen (z.B. angeheftete Ereignisse) aber auch auf ein anderers `BPMNGraphElement` verweisen kann. Lediglich die Pools und Lanes fallen hier aus der Reihe, sie erben direkt von der Klasse `BPMNParticipant`.

Die Abbildung 3.4 zeigt die in der BPMN zur Verfügung stehenden Kanten. Jeder Prozess benötigt Sequenzkanten (`BPMNSequenceLink`). Zur Interaktion mit anderen Prozessen werden Nachrichtenkanten verwendet (`BPMNMessageLink`). Um Artefakte anbinden zu können, werden Assoziationskanten bereitgestellt (`BPMNAssociationLink`). Eine Unterscheidung zwischen DataAssociatonLink und AssociationLink wie in der BPMN-Spezifikation wird hier nicht vorgenommen, da ungerichtete Kanten in der Simulationsbibliothek nicht benötigt werden. Jeder `BPMNLink` hat einen Startknoten und einen Zielknoten.

Der `BPMNNode` hat entsprechend eine Liste von eingehenden und eine Liste von ausgehenden Kanten. Die Abbildung 3.6 zeigt die wichtigsten der in der BPMN zur Verfügung stehenden Knoten. Ereignisse und Aktivitäten haben die Gemeinsamkeit, dass - sobald der Prozess sie erreicht - Aktionen gemäß ihres Typs durchgeführt werden müssen. Eventuell müssen Zustände geändert, Ressourcen belegt oder Nachrichten und Signale empfangen oder gesendet werden. Daher erben sowohl `BPMNActivity` als auch `BPMNEvent` von der Klasse `BPMNActiveNode`, welche die abstrakte Methode `Execute` bereitstellt.

Während der `BPMNGraph` die Prozessdefinition repräsentiert, wird eine Prozessinstanz durch die Klasse `BPMNProcess` implementiert, welche von der DESMO-J-Klasse `SimProcess` erbt. Für jede Prozessinstanz wird zur Laufzeit der Simulation ein neuer `BPMN-`

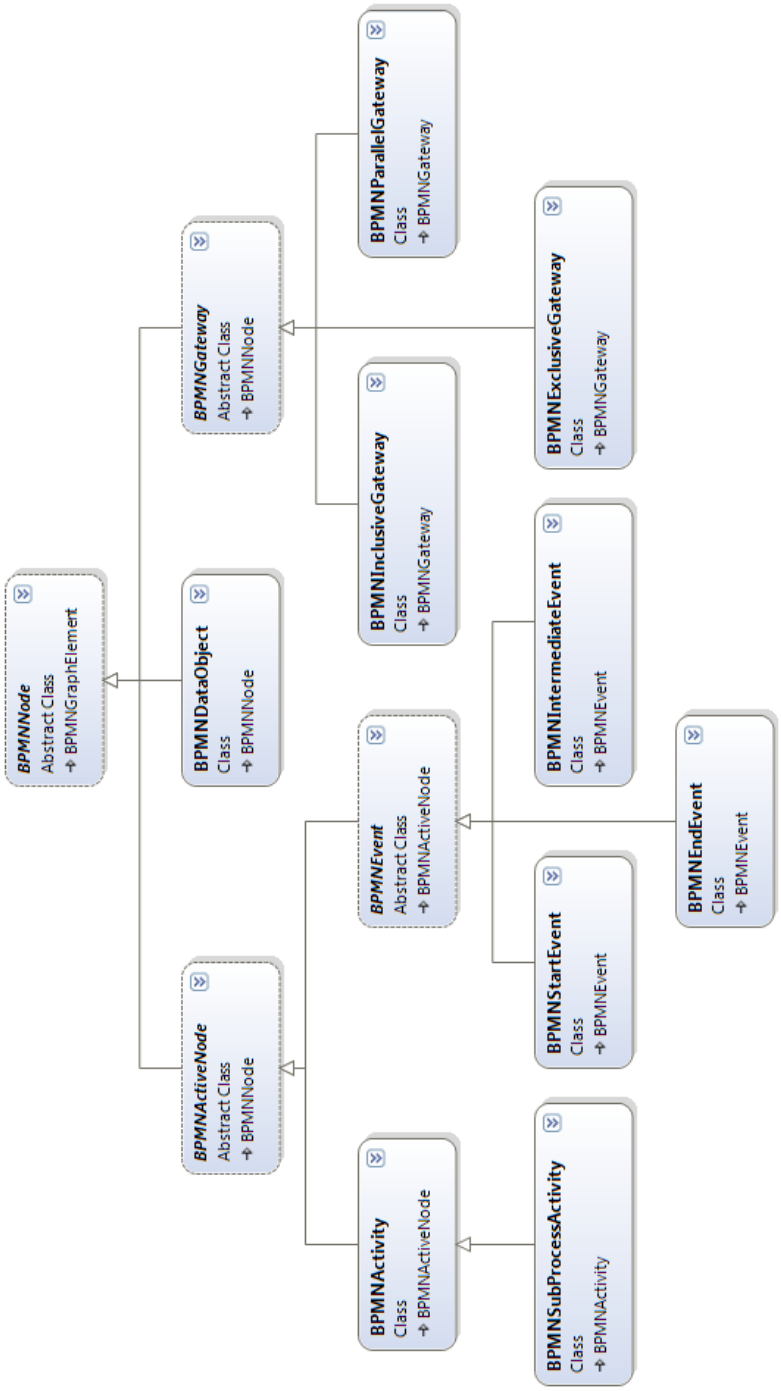


Abbildung 3.6.: Knoten in der BPMN-Bibliothek von DESMO-J

`Process` instanziiert. Die Klasse SimProcess stellt einen eigenen Thread[5] dar, der in seiner Ausführung unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden kann. Der aktuelle Methodenzeiger wird dabei gespeichert, so dass die Unterbrechung an beliebiger Stelle in der Lebenszyklus-Methode des `BPMNProcess` durchgeführt werden kann. Diese Lebenszyklus-Methode iteriert über die Elemente im zugehörigen `BPMNGraph` und führt bei Ereignissen und Aktivitäten die `Execute`-Methode aus.

Die Klasse `BPMNProcess` misst zu Beginn ihres Lebenszyklus die Simulationszeit und berechnet das Delta zur Simulationszeit zum Ende ihres Lebenszyklus, um diese dem Statistikobjekt zur Berechnung von durchschnittlicher Prozesslaufzeit, Minima, Maxima, Lower Median, Upper Median und Modus bekannt zu machen.

3.4.2. Simulationsparameter

Die Simulation erfordert zusätzliche Parameter, die als Erweiterungsattribute an die Klassen der BPMN-Spezifikation gehängt werden. Dieses Vorghen ist gemäß der Spezifikation explizit vorgesehen (vgl.OMG (2011), S. 38). Bei der Generierung eines Simulatonsmodells durch die `SimulationFactory` werden diese Erweiterungsattribute ausgelesen, um die erzeugten DESMO-J-Objekte entsprechend zu parametrieren. Wenn die entsprechenden Attribute nicht vorhanden sind, so wird eine Exception mit einer entsprechenden Fehlermeldung erzeugt und die Ausführung der Simulation unterbunden.

Um für die Dauer von Aktivitäten, die Zwischenankunftszeit von Ereignissen, die Ziehung von Prozessvariablen oder die Entscheidungen, an Gateways entsprechende Zufallszahlen bereitstellen zu können, werden stochastische Verteilungen benötigt. Eine Verteilung besteht aus einem Typ und einer Liste von Parametern. DESMO-J stellt eine große Anzahl von Verteilungen bereit, besitzt jedoch keine Mechanismen zur Persistierung der Parametrierung einer gewählten Verteilung. In der BPMN-Erweiterung wird eine serialisierbare Klasse `DistributionBuilder` bereitgestellt, welche als Erweiterungsattribut in der BPMN eingesetzt werden kann. Diese beinhaltet eine Zeichenkette zur eindeutigen Identifizierung von Namespace und Klasse, sowie eine HashMap bestehend aus Name und Wert für die Parameter der Verteilung. Die Methoden `IsAbleToBuildInstance` und `BuildInstance` werden von der `SimulationFactory` aufgerufen, um die Instanz einer DESMO-J-Verteilung zu erzeugen und diese an die entsprechenden Elemente zu hängen.

[5] Ein Prozess kann aufgeteilt werden in mehrere, nebenläufige Ausführungsstränge. Diese werden als Thread bezeichnet und können auf Multi-Prozessorensystemen auch parallel ausgeführt werden. Die prozessorientierte Simulation beruht darauf, einzelnen Entitäten eigene Threads zuzuordnen. Durch die Verwaltung der Threads entsteht jedoch ein zusätzlicher Rechenaufwand.

3.4.3. Aktivitäten

Die ausschlaggebendste Eigenschaft für die Ergebnisse von Simulationsexperimenten ist die Dauer von Aktivitäten. Gemeinsam mit den Wartezeiten bestimmen sie die Dauer von Prozessen. Die Dauer der Prozesse und der Aktivitäten ist die entscheidende Einflussgröße für fast alle weiteren zu errechnenden Kennzahlen.

Die Klasse `BPMNActivity` wird zur Repräsentation der Aktivitäten der BPMN genutzt. Lediglich für die Unterprozessaktivität gibt es eine abgeleitete Klasse `BPMNSubProcess`, um die Hierarchisierung von Prozessdefinitionen zu ermöglichen. Die Unterscheidung der Spezialisierungen der BPMN-Aktivitäten (wie Benutzeraktivität oder manuelle Aktivität) sind für die Simulation nicht relevant, da sie alle auf dieselbe Art abgebildet werden können.

Die „Manuelle Aktivität“ wird im Kontext der Workflow Automation zwar gemäß der BPMN-Spezifikation von jeder IT-Überwachung ausgeschlossen und somit übergangen. Dennoch verbraucht sie in der Realität Zeit und muss in der Simulation dementsprechend mit einer Dauer versehen werden. Nur für die Subprozess-Aktivität muss keine Dauer angegeben werden, denn ihre Dauer wird von dem betreffenden Unterprozess bestimmt.

Das Property `Duration` ist daher Pflichtfeld für alle Aktivitäten, ausgenommen der Subrozess-Aktivität. Das Property ist vom Typ `Distribution` und wird über den `DistributionBuilder` mit der gewählten DESMO-J-Verteilungen belegt.

Die Klasse `BPMNActivity` implementiert die Methode `Execute` der Klasse `ActiveNode`, welche durch den Lebensyklus eines laufenden `BPMNProcess` angestoßen wird. Aus der `Duration` wird eine Zufallszahl gezogen, welche angibt, für welchen Simulationszeitraum der Thread des `BPMNProcess` mit Hilfe der DESMO-J-Methode `hold` passiviert werden muss. Hierdurch wird die Ausführungskontrolle zurück an den `Scheduler` übergeben und das nächste Ereignis auf der Ereignisliste bearbeitet. Sobald der Simulationszeitpunkt eintritt, zu dem die Aktivität als beendet gilt, wird der Prozess wieder reaktiviert und die Ausführung der `Execute`-Methode der Aktivität fortgesetzt. Die `Execute`-Methode kann zusätzlich verzögert werden, wenn auf das Eintreffen einer Nachricht über eine Nachrichtenkante oder auf die Bereitstellung einer Ressource gewartet werden muss.

Die Methode merkt sich die Simulationszeit zu Beginn der Execute-Methode, berechnet das Delta zum Zeitpunkt vor Beendigung der `Execute`-Methode und führt eine rollierende Statistik zur Berechnung der durchschnittlichen Dauer der Aktivität, sowie von Minima und Maxima aus. Auch wird diese Zeit an den zugehörigen `BPMNProcess` weitergereicht, damit er über den Vergleich der Prozesslaufzeit und der tatsächlichen Bearbeitungszeit erste Anhaltspunkte für die Berechnung der Prozesseffizienz erhält.

3.4.4. Ereignisse

Von der Klasse `BPMNEvent` sind die Start-, End- und Zwischenereignisse abgleitet. Wenn ein Startereigniss ausgelöst wird, so wird hierdurch eine neue Instanz von `BPMNProcess` erzeugt.

Der Typ des Ereignisses wird über eine Listen-Eigenschaft `SpecifiedEvent` bestimmt. Eine Liste ermöglicht, einem Ereignis mehrere Ereignisspezifikationen zukommen zu lassen, wie es für das komplexe Ereignis von der BPMN-Spezifikation gefordert wird. Das Vorgehen entspricht dem Strategie-Pattern aus Gamma (2013): Die `Execute`-Methode der `BPMNEvent`-Instanz sucht nach angehängten `SpecifiedEvents`, um die Ausführungskontrolle an deren `Execute`-Methoden zu übergeben. Die verschiedenen `SpecifiedEvents` erfordern abhängig von ihrer Bedeutung unterschiedliche Implementationen und Simulationsparameter.

An zeitbasierten Ereignissen werden Zwischenankunftszeiten benötigt. Sie werden wie die Dauer von Aktivitäten mit Hilfe von stochastischen Verteilungen beschrieben. Auch generelle Startereignisse benötigen eine solche Zwischenankunftszeit. Beide besitzen also ein Property `InterarrivalTime` vom Typ `Distribution`. In einem Simulationsmodell, das ohne eine Kopplung an externe Modelle auskommen muss, wird an mindestens einem Startereignis eine solche Zwischenankunftszeit benötigt. Dies ist äquivalent zu den in DESMO-J zwingend benötigten initialen Ereignissen, wie in Page et al. (2000) gefordert. Stößt dieser Prozess die anderen Prozesse mit Hilfe von Nachrichten oder Signalen an, so benötigen deren Startereignisse keine Angabe von Zwischenankunftszeiten.

Als Alternative zu stochastischen Zwischenankunftszeiten können auch kalenderbasierte Einträge vorgenommen werden. Dies ist sinnvoll, wenn einige Prozesse zu bestimmten Wochentagen, an einem bestimmten Datum oder zu einer bestimmten Uhrzeit starten sollen. Wo möglich, sollte jedoch Stochastik verwendet werden, um realistische Schwankungen abzubilden.

Empfangende `SpecifiedEvents` müssen den aktuellen `BPMNProcess` passivieren, bis das entsprechende Ereignis eintritt. Dies geschieht mit der DESMO-J-Methode `Passivate`, die sich von der `Hold`-Methode dadurch unterscheidet, dass keine Dauer der Passivierung angegeben wird, sondern die Reaktivierung explizit durch einen Methodenaufruf erfolgen muss.

Außer dem Nachrichtenereignis werden alle sendenden und empfangende Ereignisse gemäß der BPMN-Spezifikation über ihren Namen verknüpft. Die `SimulationFactory` hinterlegt daher bei den sendenden Ereignissen Referenzen auf die zugehörigen Empfangsereignisse. In der `Execute`-Methode der sendenden Ereignisse ist zusätzlich die Entscheidungsfindung implementiert, welche wartenden Prozessinstanzen tatsächlich reaktiviert werden, wobei bei den unterschiedlichen Ereignistypen verschieden Kriterien

angewendet werden müssen. Während zum Beispiel das Signal-Ereignis einen Broadcast darstellt und sämtliche wartenden Prozessinstanzen reaktiviert, wird bei einem Fehlerereignis nur die zugehörige Prozessinstanz des Mutterprozesses reaktiviert.

Nachrichtenereignisse werden mit Hilfe von Nachrichtenkanten verknüpft. Hier entfällt die explizite Zuordnung der sendenden und empfangenden Ereignisse über den Namen. Auch hier muss jedoch das sendende Ereignis entscheiden, welche konkrete Prozessinstanz den Kommunikationspartner darstellt und reaktiviert werden muss. Da vorab nicht feststeht, ob zuerst die Nachricht oder zuerst der empfangende Prozess das Empfangsereignis erreicht, gibt es für beide Fälle eine Warteschlange. Somit wird eine Statistik geführt, wie lange es gedauert hat, bis eine Nachricht entgegen genommen wurde und eine weitere, wie lange der Prozess auf eine Nachricht warten musste.

Nicht zwangsläufig soll der Sender einer Nachricht simuliert werden. Unter Umständen möchte der Modellierer auch hier mit stochastischen Zwischenankunftszeiten arbeiten, um den Nachrichteneingang zu simulieren. Da dennoch aus dem Modell ersichtlich sein soll, dass aus fachlicher Sicht eine Nachricht empfangen werden soll, wäre es der Validierung der Modelle nicht zuträglich, dass der Modellierer nur zum Zwecke der Simulation aus einem Nachrichtenereignis ein zeitbasiertes Ereignis macht. Daher haben empfangende Nachrichtenereignisse ein Property `StochasticEventTrigger`. Wenn dieses gesetzt ist, kann auch hier eine stochastisch errechnete Zwischenankunftszeit hinterlegt werden und es wird nicht auf das tatsächliche Eintreffen einer Nachricht gewartet.

3.4.5. Gateways

Von `BPMNNode` erbt auch die Klasse `BPMNGateway`. Hiervon erben die unterschiedlichen Gatewaytypen `BPMNParallelGateway` (Paralleles Gateway), `BPMNEventBasedGateway` (Ereignisbasiertes Gateway), `BPMNExclusiveGateway` (Exklusives Gateway) und `BPMN-InclusiveGateway` (inklusives Gateway).

Die BPMN-Erweiterung für DESMO-J sieht zwei Mechanismen zur Entscheidungsfindung an exklusiven oder inklusiven Verzweigungen vor.

Im einfachen Fall kann für jede ausgehende Kante eine Wahrscheinlichkeit angegeben werden. Bei exklusiven Gateways muss diese summierte Wahrscheinlichkeit der ausgehenden Kanten genau eins ergeben. Bei der Ausführung wird jeweils eine Zahl zwischen null und eins gezogen, um zu entscheiden, welcher Kante gefolgt wird. Bei inklusiven Gateways sollte die summierte Wahrscheinlichkeit größer als eins sein. Für jede ausgehende Kante wird eine Zahl zwischen null und eins gezogen, um zu entscheiden, ob ihr gefolgt wird oder nicht. Für den Fall, dass für keine der ausgehenden Kanten die Prüfung der

Zufallsbedingung erfolgreich ist, wird für jede der ausgehende Kanten erneut eine Zufallszahl gezogen, bis bei mindestens einer der Kanten die Bedingung erfolgreich ausgewertet wurde.

Wenn bei inklusiven oder parallelen Verzweigungen mehreren Pfaden gefolgt wird, wird für jeden der Pfade ein Teilprozess `BPMNProcess` erzeugt, so dass die verschiedenen Pfade nebenläufig abgearbeitet werden. Bei Zusammenführungen muss abhängig von der Art der Zusammenführung auf die entsprechenden Teilprozesse gewartet werden (vgl. Abschnitt 2.3.4).

Der zweite Mechanimsus zur Entscheidung beruht auf durch den Modellierer formulierbaren Bedingungen und Prozessvariabelen, welche im folgenden Abschnitt beschrieben werden.

3.4.6. Prozessvariablen

Sofern keine Datenobjekte genutzt werden, geht die BPMN davon aus, dass sämtliche Informationen des Prozesskontextes an allen Elementen vorliegen. Dies kann zum Beispiel eine Liste von Auftragspositionen sein, die in einem Prozess abgearbeitet werden muss oder ein konkretes Produkt, welches gefertigt oder versendet werden muss.

Solcherart Informationen werden an dem Objekt `BPMNProcess` in einer HashMap `ProcessVariables` gespeichert. Mit Hilfe von stochastischen Verteilungen können diese über Datenobjekte, welche mit Ereignissen assoziiert wurden oder an Aktivitäten gesetzt werden, angegeben werden. So kann bei der Prozessausführung beim Erreichen des Elementes ein konkreter Wert generiert werden. Hierfür wird in der Prozessdefinition eine HashMap mit einem Variablennamen und einem zugehörigen `DistributionBuilder` gespeichert, welche von der `SimulationFactory` in eine HashMap aus Variablennamen und `Distribution` überführt wird. Die `Execute`-Methoden von Ereignissen und Aktivitäten werten diese aus, ziehen entsprechende Zahlen und hinterlegen die Variablennamen mit zugeordneten Wert in der HashMap des `BPMNProcess`.

An Aktivitäten und Datenobjekten können diese Werte manipuliert werden. Hierfür wird die Skriptprache Python in ihrer Implementation IronPython verwendet. IronPython ermöglicht es, .NET-Objekte aus Python heraus zu verwenden, also Methodenaufrufe durchzuführen und Eigenschaften auszulesen. Zusätzlich zu den durch den Modellierer definierten Variablen werden hier Variablen aus dem Simulationskontext hinzugefügt. Hierzu gehört die aktuelle Simulationszeit, das aktuell bearbeite Produkt, der Startzeitpunkt der aktuellen Prozessinstanz und weitere Informationen. Der Modellierer kann somit komplexe Berechnungen unter Verwendung sämtlicher in Python vorhandenen

höhersprachlichen Programmierkonstrukte vornehmen. In Stehle (2014) wurde der Bibliothek die Möglichkeit hinzugefügt, die verwendeten Variablen statistisch auszuwerten und deren Belegung in den Simulationsbericht zu integrieren.

Als Alternative zur Angabe von Wahrscheinlichkeiten können bei exklusiven und inklusiven Gateways somit auch Bedingungen hinterlegt werden. Hierfür wird vom Modellierer an jeder Kante eine in Python formulierte Bedingung hinterlegt, welche die vorhandenen Prozessvariablen oder die Informationen aus dem Prozesskontext auswertet. In diesem Fall wird nur denjenigen Kanten gefolgt, welche eine im jeweiligen Prozesskontext wahre Bedingung besitzen.

3.4.7. Ressourcen

Die BPMN-Bibliothek enthält auch die Möglichkeit, Ressourcen und Rollen zu modellieren. Hierzu werden in einem separaten Diagramm, welches nicht Bestandteil der BPMN ist, die vorhandenen Ressourcen und die jeweils einnehmbaren Rollen definiert. Innerhalb von BPMN-Diagrammen wird an den Aktivitäten und Swimlanes eine Zuordnung zu den Rollen vorgenommen. Eine Ressource kann mehrere Rollen einnehmen und zu einer Rolle kann es mehrere Ressourcen geben. Benötigt eine Aktivität eine bestimmte Rolle, so wird nach einer freien Ressource gesucht, welche diese Rolle erfüllt. Ist keine Ressource verfügbar, so wird gewartet bis eine frei wird.

Da das Konzept zur Integration von Ressourcen im Zuge dieser Arbeit neu aufgesetzt wurde, wird auf eine nähere Beschreibung der Implementation an dieser Stelle verzichtet und stattdessen auf Abschnitt 7.5.1 verwiesen.

3.4.8. Ergebnisreports

DESMO-J besitzt drei Ausgabekanäle zur Generierung von Simulationsreports: HTML-Reports, XML-Reports und ASCII-Reports. Im Zuge der Integration in eine Anwendungsumgebung für Simulationsstudien können weitere Ausgabekanäle hinzugefügt werden, wie zum Beispel der XAML-Report des DesmoWindparkStudios, welcher die Ergebnisse grafisch aufbereitet.

In Anhang A findet sich ein Beispielprozess mit Auszügen exemplarischer Simulationsergebnisse. Die Abbildung A.1 zeigt die Prozessdefinition eines Kundenprozesses, der den Durchlauf einer Bungee-Jumping-Anlage beschreibt. Die verschiedenen Swimlanes repräsentieren verschiedene Mitarbeiter der Anlage, welche an den Aktivitäten beteiligt sind. Der Kunde beginnt seinen Prozess an der Kasse beim Bezahlen des Sprungs, er wird gewogen, zieht sich ein Sicherungsgeschirr an und wartet darauf, dass er zur Anlage geführt wird. Er erhält eine Sicherheitsunterweisung, bekommt das Seil angelegt, springt

von der Brücke und wird anschließend wieder hochgezogen. Anschließend kann er noch ein Getränk an der zur Anlage zugehörigen Bar einehmen.

In Abbildung A.2 wird ein Auszug aus dem HTML-basierten Ergebnisbericht zu diesem Prozess gezeigt. Im oberen Abschnitt werden Werte dargestellt, aus denen bei einer Visualisierung der Ergebnisse ein Histogramm gezeichnet werden kann. Sie geben an, wieviele Prozesse innerhalb welcher Zeitintervalle beendet werden konnten. Im mittleren Abschnitt findet sich die durchschnittliche Dauer des Prozesses, die minimal und maximal verbrauchte Zeit sowie Lower Quartile, Upper Quartile und Modus. Im unteren Abschnitt wird angegeben, wieviele Prozessinstanzen durchschnittlich, minimal und maximal parallel abliefen, also in diesem Fall, wieviele Kunden sich zeitgleich im System befanden.

Abbildung A.3 enthält Statistiken zur Ressourcenauslastung. Diese werden nicht nach einzelnen Ressourceninstanzen sondern nach Ressourcenpools (also Rollen) aufgeschlüsselt. Genannt werden absolute Zahlen, wieviel die Mitarbeiter der betreffenden Rollen produktiv gearbeitet haben, auf andere Ressourcen warten mussten oder sich ohne Zuteilung einer Aufgabe im System befinden. In der unteren Tabelle werden Statistiken zu den Warteschlangen ausgegeben, die die Wartezeiten auf die betreffenden Ressourcen sowie die durchschnittliche, minimale und maximale Warteschlangenlänge beinhalten.

Abbildung A.4 zeigt, wie häufig die einzelnen Ereignisse aufgetreten sind. Weitere Statistiken, die hier nicht gezeigt werden, geben durchschnittliche Zwischenankunftszeiten zu Ereignissen an, die Dauer der einzelnen Aktivitäten (unterteilt nach produktiver Arbeit und Wartezeiten auf Nachrichten oder Ressourcen), und welchen Pfaden an Gateways wie häufig gefolgt wurde.

3.5. Fazit zur Geschäftsprozesssimulation

In diesem Abschnitt wurde das Potential der Geschäftsprozesssimulation erläutert. Es wurde gezeigt, dass der Nutzen der Geschäftsprozesssimulation für BPI-Projekte bereits in den 90er Jahren erkannt wurde, es jedoch fast 20 Jahre gedauert hat, bis mit der BPMN 2.0 eine Sprache zur Verfügung stand, die die Vereinigung der herkömmlichen Prozessmodellierung aus BPM-Projekten mit der Simulationstechnik auf adäquate Weise ermöglichen kann. Eine Bibliothek zur Simulation von BPMN 2.0-Prozessen wurde vorgestellt. Es gibt mehrere Werkzeuge, welche die Simulation der BPMN 2.0 unterstützen. Es gibt jedoch bisher keine Werkzeuge, welche eine Erweiterung der Notation um domänenspezfisiche Elemente erlauben, um hierüber die Umgebung eines Prozesses einzubinden und in die Simulation integrieren zu können.

4. Ansätze zur Kopplung und Erweiterung von Modellen

Dieses Kapitel erläutert bestehende Ansätze zur Kopplung von Modellen aus dem Bereich der Modellbildung, Simulation und Softwaretechnik. Hieraus werden Anforderungen und Anregungen zur Lösung der gestellten Problematik hergeleitet. Das Potential der komponentenbasierten Softwareentwicklung wird erläutert.

4.1. Kopplung von Prozessmodellen

Die Kopplung von Prozessmodellen mit in anderen Notationen beschriebenen Modellen wurde in diversen Publikationen beschrieben. Diese verfolgen in der Regel die Unterstützung der Anforderungsdefinition für die modellgetriebene Entwicklung von Softwaresystemen. Einige Aspekte lassen sich jedoch auch für die Kopplung an Simulationsmodelle übernehmen.

4.1.1. Beziehungen zwischen Elementen in der UML

Die UML 2.0 definiert eine große Anzahl von Relationen, die zwischen zwei Systemelementen bestehen können. Innerhalb der 14 Diagrammtypen werden 19 Relationstypen benannt. Jedoch fällt bei genauerer Betrachtung auf, dass einige dieser Relationstypen redundant sind, da sie in unterschiedlichen Diagrammtypen zwar verschiedene Namen tragen, jedoch letztlich den gleichen Sachverhalt erläutern. Dies reduziert die Anzahl der tatsächlich vorhandenen Relationstypen erheblich. Es gilt zu untersuchen, mit welchen der verbleibenden Beziehungen Relationen zwischen BPMN-Prozessen und der Prozessumgebung benannt werden können. Prozessinhärent ist der Kontrollfluss, die Assoziation, der Kommunikationsfluss und der Objektfluss, dies überschneidet sich jedoch teilweise mit anderen Relationstypen.

Über einen Kommunikationsfluss kann der Nachrichtenaustausch mit der Systemumgebung dargestellt werden. Hierdurch können Informationen von der Prozessumgebung an den Prozess übermittelt werden, die sich auf den weiteren Prozessverlauf auswirken.

Informationen, die vom Prozess an die Prozessumgebung geliefert werden, können von dieser ausgewertet werden und eine entsprechende Zustandsänderung in der Umgebung verursachen.

Es ist denkbar, dass die Ausführungskontrolle innerhalb eines Prozesses für einen bestimmten Abschnitt an die Systemumgebung übergeht, sofern die Laufzeit und die Handlungen einer Aktivität von der Systemumgebung bestimmt werden. Dies ist jedoch eher vergleichbar mit den UML-Relationen Implementation und Realisation.

Ein Prozess kann mit einem Endereignis schließen, welches eine Nachricht an die Prozessumgebung sendet und dort entsprechende Handlungen anstößt. Zudem kann ein Prozess durch Ereignisse aus der Systemumgebung ausgelöst werden. Beides ist eher der Relation Kommunikationsfluss zuzuordnen, als einer Übergabe des Kontrollflusses, da die Systemumgebung auch während und vollkommen unabhängig von der Prozessausführung ihre eigenen Zustandsänderungen kontrollieren kann.

Der Objektfluss wird in der BPMN implizit über den Kontrollfluss oder über Assoziationen dargestellt. Auch können Elemente aus der Systemumgebung mit Elementen des Prozesses assoziiert sein, zum Beispiel wenn eine Aktivität des Prozesses auf einer bestimmten Entität aus der Systemumgebung arbeitet, was vergleichbar mit der Benutzt-Relation aus der UML ist.

4.1.2. Kopplung von UML-Modellen an Geschäftsprozessmodelle

In Loos und Allweyer (1998) werden EKP-Modelle mit UML-Diagrammen verknüpft. Die Zielsetzung ist dabei nicht die Simulation, sondern die Unterstützung der Beschreibung von Softwarearchitekturen. Während die EPKs die Prozesse beschreiben, welche mit den IT-Systemen in Berührung kommen, werden Klassendiagramme und Zustandsdiagramme der UML genutzt, um die Softwaresysteme und die verwendeten Objekte zu beschreiben. Auch Anwendungsfall- und Aktivitätsdiagramme werden betrachtet. Hier wird betont, dass die Aktivitätsdiagramme der UML selbst nicht geeignet sind, um Geschäftsprozesse adäquat zu beschreiben, was die Notwendigkeit der Integration von EPKs begründet. Die Elemente der EPK-Modelle werden an Klassendiagramme gekoppelt, um zu beschreiben, welche Objekte benötigt, erzeugt oder manipuliert werden. Das Klassendiagramm eines Okjektes wird dabei direkt in die grafische Modelldefinition der EPKs integriert und dabei mit einem Pfeil mit den entsprechenden Tätigkeiten (Funktionen) verknüpft. Auf diese Weise wird dem Betrachter direkt ersichtlich, auf welche Entitäten sich ein Prozess auswirkt. Diese direkte Integration von Entitäten in die Prozessmodelle erscheint auch für die Simulationsmodellierung ein sinnvoller Ansatz, da es die Validierung der Modelle durch die Domänenexperten vereinfacht.

Auch die direkte Verknüpfung mit Methodenaufrufen an den Objekten ist in einer detaillierten Darstellung möglich. Neben der Verknüpfung über gerichtete Kanten, ist dabei auch eine Verknüpfung über die Beschriftung der Funktionen der EPK-Modellen möglich. Hierbei erhält die Funktion den gleichen Namen, wie die aufzurufende Methode an einem Objekt. Dieser Ansatz mag für die modellgetriebene Softwareentwicklung adäquat sein, da die Benennung von Methoden in der Softwareentwicklung meist einheitlich in der englischen Sprache geschieht und die Prozessmodelle eigens für die Softwareentwicklung definiert werden. Für die Simulationsmodellierung birgt dies jedoch den Nachteil, dass auch die zunächst fachlichen Prozessbeschreibungen, welche ursprünglich mit der Zielsetzung der Schaffung einer Kommunikationsgrundlage entworfen wurden, daher in der Muttersprache der Mitarbeiter entworfen sein dürften und für die Simulation wiederverwendet werden sollen, in die englische Sprache übersetzt werden müssen. Bei international agierenden Konzernen liegen Prozessmodelle häufig in verschiedenen Sprachen vor. Sollte die Simulierbarkeit der Modelle abhängig von der Lokalisierung der Prozessbeschreibungen sein, so stellt dies eine Einschränkung dar.

In Riebisch et al. (2011) wird aufgeführt, dass sich mit verschiedenen Diagrammtypen in der Regel verschiedene Aspekte eines Systems jeweils adäquat beschreiben lassen. Für eine vollständige Systembeschreibung müssen die Modelle dabei untereinander verknüpft und deren Abhängigkeiten untereinander beschrieben werden. Anhand des Beispiels der Verknüpfung von BPMN-Modellen mit UML-Klassendiagrammen wird hierfür ein Ansatz präsentiert. Zunächst wird zwischen zwei Arten der Beziehungen von Modellelementen unterschieden. Entweder charakterisieren die Modelleelemente unterschiedliche Aspekte der gleichen Entität, oder zwei unterschiedliche Entitäten, die in einer Relation zueinander stehen. Die Abhängigkeiten können dabei explizit beschrieben oder implizit durch ein Regelwerk erkannt werden.

Die Verknüpfung kann auch hier wie in Loos und Allweyer (1998) über den Namen der Elemente beschrieben, oder explizit mit Hilfe einer separaten XML-Datei gepflegt werden. Hierbei können auch gegenseitige Anforderungen der Modelle aneinander definiert werden, um die Konformität der Modelle sicherzustellen. Zur kollaborativen Arbeit an den Modellen wird ein Versionskontrollsystem empfohlen, welches auf Pflege von Modellen zugeschnitten ist, zum Beispiel EMFStore (ein Modell-Repository des Eclipse Modelling Frameworks, vgl. Bode et al. (2011)). In Riebisch et al. (2011) wird das Risiko von Inkonsistenzen zwischen den Modellbeschreibungen oder in den gegenseitigen Anforderungen betont. Daher müssen die Anforderungen, wie das Vorhandensein bestimmter Elemente oder Eigenschaften, formal beschrieben werden. Für das Versionskontrollsystem ist dabei wichtig, sowohl die Versionierung der Modelle als auch die Versionierung der Verknüpfungsbeschreibungen zu unterstützen. Somit können Inkonsistenzen, wie sich gegenseitig ausschließende Anforderungen, erkannt werden. Ein hier zwar nicht genanntes, aber für die Simulation relevantes Beispiel sind die verwendeten Zeiteinheiten in den Modellen. Es muss sichergestellt werden, dass sämtliche Simulationsmodelle die gleichen

Zeiteinheiten verwenden, um die Interaktion zu bestimmten Zeitpunkten realisieren zu können. Dies stellt nicht nur Anforderungen an die Modelle untereinander, sondern auch an den Simulationskern. DESMO-J beispielsweise kann derartige Inkonsistenzen vermeiden, wenn mit Zeitangaben in einem festgelegten Datumsformat gearbeitet wird (vgl. Klückmann (2009)).

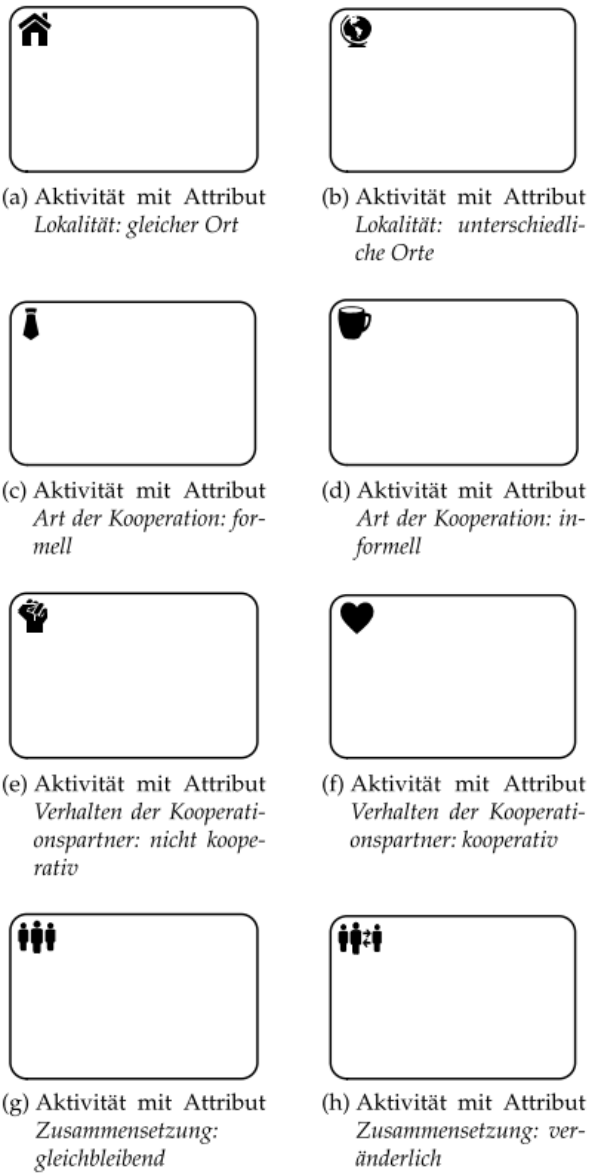
Auch werden mögliche Beziehungen zwischen BPMN-Elementen und UML-Diagrammen beschrieben – ein Aspekt der besonders interessant für diese Arbeit ist. Bei Nachrichtenkanten und sendenden Aktivitäten können Klassendiagramme genutzt werden, um die Struktur der Nachricht zu beschreiben. Datenobjekte oder Datenbanken können Objektinstanzen repräsentieren, welche in einem Klassendiagramm definiert werden. Für Aktivitäten wird eine besonders große Anzahl von möglichen Relationen angegeben (zum Beispiel „Ruft auf“, „Substituiert“ oder „Überprüft“), mit der die vielfältig denkbare Wirkung einer Aktivität auf die Elemente des verknüpften Modells beschrieben werden.

Der Ansatz, BPMN-Artefakte wie Datenobjekte oder Datenbanken als direkte Repräsentation einer Entität aus einem anderen Modell zu verwenden, ist für die Simulation sehr sinnvoll. Hierdurch kann beschrieben werden, auf welche Entitäten sich ein Prozess oder eine Aktivität auswirkt beziehungsweise welche Entitäten genutzt werden. Die Art der Auswirkung von Aktivitäten auf diese Entitäten ist dabei sehr vielfältig und sollte entsprechend flexibel in dem zu entwickelnden Konzept ausgearbeitet werden, um eine möglichst große Anzahl von Anwendungen zu ermöglichen.

4.1.3. Erweiterung von BPMN-Modellen

Verschiedene Arbeiten haben sich mit der Erweiterung von BPMN-Modellen beschäftigt. Nicht alle halten sich dabei an die von der Spezifikation vorgesehenen Erweiterungsmöglichkeiten. Dies ist allerdings mit der Zielsetzung der Austauschbarkeit der Editorkomponenten dringend anzuraten. Allen Erweiterungen ist gemein, dass sie weitere Aspekte bei der Modellierung von Geschäftsprozessen einbringen, welche für bestimmte Anwendungsfälle notwendig sind.

Schäfer (2012) erkennt, dass die Kollaborationsdiagramme der BPMN 2.0 trotz ihres vielversprechenden Namens Schwächen bei der Beschreibung von kollaborativen Tätigkeiten haben. Die Bezeichnung Kollaborationsdiagramm rührt aus der Möglichkeit, Interaktionen zwischen zwei Prozessen mit Hilfe von Nachrichtenkanten zu beschreiben. Die gemeinsame Ausführung einer Aktivität wird jedoch laut der Spezifikation besser mit Choreographiediagrammen beschrieben. Schäfer (2012) bietet einen Ansatz, um zumindest die Art von kollaborativen Aktivitäten, wenn auch nicht die beteiligten Akteure, im Kollaborationsdiagramm zu beschreiben. Hierfür führt er spezielle Symbole zur Markierung



(a) Aktivität mit Attribut *Lokalität: gleicher Ort*

(b) Aktivität mit Attribut *Lokalität: unterschiedliche Orte*

(c) Aktivität mit Attribut *Art der Kooperation: formell*

(d) Aktivität mit Attribut *Art der Kooperation: informell*

(e) Aktivität mit Attribut *Verhalten der Kooperationspartner: nicht kooperativ*

(f) Aktivität mit Attribut *Verhalten der Kooperationspartner: kooperativ*

(g) Aktivität mit Attribut *Zusammensetzung: gleichbleibend*

(h) Aktivität mit Attribut *Zusammensetzung: veränderlich*

Abbildung 4.1.: "Grafische Attribute zu ausgesuchten Merkmalen kooperationsintensiver Tätigkeiten". Abbildung aus: Schäfer (2012)

von Aktivitäten ein (vgl. Abbildung 4.1). Die Symbole werden dabei so gewählt, dass die Art der Kollaboration möglichst leicht verständlich sein soll. Die BPMN-Spezifikation erlaubt dieses Vorgehen explizit, sofern nur Erweiterungsattribute genutzt werden, welche die Art der Kollaboration beschreiben. Unter diesen Umständen kann auch ein anderes Softwarewerkzeug die Prozesse weiterhin anzeigen, auch wenn es die speziellen Symbole selbst nicht darstellen kann.

In Friedenstab et al. (2012) wird die BPMN erweitert, um die Kopplung an Monitoring-Systeme zu beschreiben. Monitoring-Systeme sind natürlicher Bestandteil von Workflow-Engines; durch eine solche Beschreibung kann das Verhalten des Workflow-Systems näher spezifiziert werden. Hierfür wird ein Meta-Modell geschaffen, welches neben den BPMN-Elementen zusätzliche Knoten und Kanten zur Beschreibung des Monitoring-Systems beinhaltet. Das Modell des zu koppelnden Systems wird also komplett als zusätzlicher Graph in die grafische Darstellung des Prozessmodells integriert. Die Einführung von neuen Kantentypen ist allerdings ein Verstoß gegen die BPMN-Spezifikation. Als Artefakt dürfen zwar neue Knoten eingeführt werden, dabei gilt es aber zu bedenken, dass das Artefakt in einer anderen Software nur als Annotation dargestellt werden kann, während die unbekannten Kantentypen gar nicht dargestellt werden können. Diese Modelle können nicht im Standard-XML-Format der BPMN gespeichert werden und sind daher nicht zwischen verschiedenen Werkzeugen übertragbar. Für Prozessmodelle, welche in ein Workflow-System überführt werden sollen, ist dies ein eklatanter Nachteil. Es handelt sich hier allerdings nur um den Entwurf eines Metamodels, welcher mit Hilfe von Visio Stencils umgesetzt wurde. Derartige Vorschläge können als Anregungen für künftige Versionen der BPMN dienen. Die Übertragung in eine standardkonforme Modellierungs- oder gar Workflowsoftware ist aber derzeit nicht möglich, daher ist dies ein Beispiel für einen im Kontext dieser Arbeit nicht nachzuahmenden Ansatz. Eine behutsamere Vorgehensweise bei der Erweiterung – wie in Schäfer (2012) – ist anzuraten.

4.2. Modellkopplung in der Simulation

Neben der Erweiterung und der Kopplung auf Modellierungsebene, spielen die Möglichkeiten zur Kopplung von Simulationsmodellen zur Laufzeit eine entscheidende Rolle für diese Arbeit. Viele Arbeiten beschäftigen sich mit der Modellkopplung mit der Zielsetzung der verteilten Ausführung von Simulationsexperimenten. Dies ist nicht das Vorhaben dieser Arbeit, dennoch lassen sich Anregungen übernehmen.

4.2.1. High Level Architecture

Die Partitionierung von Simulationsmodellen für deren verteilte Ausführung ist ein weit verbreiteter Anwendungsfall. Mit der *High Level Architecture* steht daher eine Architektur zur integrierten und verteilten Simulation zur Verfügung, welche im Jahr 2000 von der IEEE zum Standard erhoben wurde (vgl. IEEE-Standard 1516). Hierbei wird ein Gesamtmodell (Föderation) in mehrere Teilmodelle (Föderaten) partitioniert. Die einzige Interaktionsbeziehung ist die Kommunikation, bei der Daten über ein Netzwerk ausgetauscht werden. Diese Kommunikationsbeziehungen werden über eine zentrale Komponente (Run-Time-Infrastructure) verwaltet, welche die Verteilung der Daten über Netzwerkprotokolle wie TCP oder UDP vornimmt. Die Daten können entweder Informationen beinhalten, welche von dem empfangenden Föderat ausgewertet werden können, oder Ereignisse, welche auf den Föderaten wirken.

Um die Interaktion zu ermöglichen, müssen die Föderaten erstens Schnittstellen erfüllen, die von der Run-Time-Infrastructure vorgegeben werden, und sich zweitens an ein Object Model Template halten, über welches definiert wird, welche Datenstrukturen die Föderaten untereinander austauschen können. Hierbei werden Objekte über ihre Attribute und Ereignisse über ihre Parameter beschrieben.

Zudem werden bestimmte Regeln vorgegeben, die die Teilmodelle erfüllen müssen, um als standardkonform zu gelten. In dieser Arbeit wird keine verteilte Simulation vorgesehen. Dennoch bietet die High Level Architecture hier Anregungen, die auch für die vorliegende Arbeit gelten können. Hierzu gehört, dass der gesamte Datenaustausch gemäß der vorgegeben Schnittstellen und der vorgegeben Datenstruktur erfolgen muss. Da diese Arbeit auf die Kopplung von Teilmodellen an BPMN-Prozesse abzielt, müssen die Schnittstellen von der BPMN-Bibliothek vorgegeben werden. Der Informationsaustausch zwischen den domänenspezifischen Modellen ist zwar möglich, wird in dieser Arbeit aber nicht genauer strukturiert. Alle Objektinstanzen müssen einem Föderat zugeordnet sein. Jede Objektinstanz darf zu einem Zeitpunkt nur unter der Kontrolle maximal eines Teilmodells stehen. Der Simulatorkern muss also darauf achten, dass nur ein Teilmodell zur Zeit Auswirkungen auf ein Objekt haben kann, um nichtdeteterministisches Verhalten zu vermeiden. Durch die Verwendung von DESMO-J ist gewährleistet, dass zu einem gegebenen Zeitpunkt nur ein Teilmodell die Ausführungskontrolle inne hat. Der gleichzeitige Zugriff auf ein Objekt durch verschiedene Teilmodelle ist nicht möglich.

4.2.2. Hardware in the Loop und Software in the Loop

Simulationstechnik kann zur Absicherung bei der Entwicklung von Hardware, zum Beispiel eingebettete Systemen oder Steuerungssysteme verwendet werden. Wenn ein Modell

die Systemumgbeung einer real existierende Hardware simuliert, um das korrekte Verhalten der Hardware zu untersuchen, zu testen oder abzusichern, so wird dies als Hardware in the Loop bezeichet. Ein Beispiel hierfür ist die Funktionsprüfung von Steuerungssystemen, wie sie zum Beispiel im Automobilbau angewendet wird. Wenn nicht die Hardware selbst, sondern ein Simulationsmodell dieser Hardware für die Tests verwendetet wird, so spricht man von Software in the Loop. Dies kann vor der Implementation einer realen Hardware die Anforderungsanalyse an diese Hardware unterstützen.

In beiden Fällen müssen die relevanten Zustandsänderungen der Systemumgebung und die Schnittstelle zum zu testenden System in dem Simulationsmodell abgebildet werden. Die Schnittstelle beschreibt den möglichen Nachrichtenaustausch zwischen Hardware und Umgebung. Hierfür existiert in der Regel ein für die Hardware verständliches Protokoll, über welches Signale versandt und entgegengenommen werden. Ein Beispiel zum Testen eines Containerterminalsteuerrungssystems mit Hilfe von DESMO-J wird in Joschko et al. (2009) beschrieben. Hier wurde detailliert das kinematische Verhalten von Containerbrücken und die Schnittstelle zu dem Steuerungssystem abgebildet. Das Steuerungssystem fordert die Containerbrücke über Nachrichten zur Durchführung von Aktivitäten auf und erhält eine Rückmeldung, sobald diese abgeschlossen sind. Auf diese Weise kann die Implementation des Steuerungssystems vor der Integration in den realen Terminalbetrieb getestet werden.

Auch bei diesem Ansatz wird die komplette Interaktion zwischen einem System und seiner Systemumgebung durch Nachrichtenaustausch realisiert. Dieser Form von Interaktion kommt auch bei der Abbildung einer Prozessumgebung eine gewichtige Rolle zu, jedoch sind die Möglichkeiten hiermit noch nicht ausgeschöpft, wie in Abschnitt 4.1 dargestellt wurde.

4.2.3. Softwaretechnische Ansätze zur Interaktion

Liebig et al. (2001) beschreibt die Notwendigkeit, Modelle verschiedener Systeme in einem Simulationsmodell zu koppeln. Er nennt als Beispiel ein Simulationsmodell eines Schienenfahrzeugs mit aktiver hydraulischer Neigetechnik, für welches mechanische und hydraulische Modelle gekoppelt werden müssen. Er betont die Notwendigkeit passender Editoren für die Teilmodelle, welche auf die jeweiligen Ingenieursdisziplinen zugeschnitten sein müssen. Diesem Ansatz der Integration geeigneter Editoren soll auch in dieser Arbeit gefolgt werden, um die Validierung und Pflege der domänenspezifischen Teilmodelle durch fachkundige Mitarbeiter des Unternehmens zu ermöglichen. Die Art der Kopplung wird anhand des Ortes der Berechnung unterschieden. Geschieht die Kopplung auf Modellebene und wird durch den selben Simulatorkern ausgeführt, so ist der Kopplungsaufwand zwar geringer, aber die Simulationskomponente muss die verschiedenen Modelle interpretieren können (One-Solver-Solution). Werden für die Teilmodelle

verschiedene Simulationskomponenten genutzt (Multi-Solver-Solution), so müssen die Teilmodelle zu festgelegten Zeitpunkten Nachrichten austauschen. In dieser Arbeit wird dem One-Solver-Solution-Ansatz gefolgt. Die Modelldefinitionen werden in Simulationsmodelle überführt, welche vom gleichen Simulationskern ausführbar sind, um den Kopplungsaufwand geringer zu halten.

Grimm und Schroll (2007) benennen als mögliche Mittel der Kommunikation zwischen Teilmodellen zum einen den gegenseitigen Methodenaufruf oder gemeinsam benutzte Variablen auf Anwendungsebene und zum anderen die Verwendung von Mitteln wie Shared Memory oder Semaphoren auf Betriebssystemebene. Da bei der vorliegenden Arbeit alle Teilmodelle der Simulation in der selben Anwendung laufen, können der Methodenaufruf oder gemeinsam genutzte Variablen verwendet werden. Grimm und Schroll (2007) betonen, dass neben dem Code zur Simulatorkopplung auch Code benötigt wird, der die Wertetypen konvertiert, sofern in verschiedenen Teilmodellen verschiedene Typen oder Einheiten verwendet werden. Dieser Code kann direkt in die Modelle beziehungsweise die Methoden zum Nachrichtenaustausch integriert werden. Typisch hierfür seien „Wrappermodelle“, welche bestehende Teilmodelle umschließen, um die Interaktion mit den Schnittstellen der vorhandenen Modelle zu ermöglichen.

Die Simulationsbibliothek DESMO-J bietet die Möglichkeit, ein Simulationsmodell in mehrere Teilmodelle zu partitionieren. Jedes Teilmodell beinhaltet seine eigenen Entitäten, Statistiken, Ereignisse, Prozesse und Warteschlangen. Ereignisse oder Prozesse können auf die Warteschlagen anderer Teilmodelle zugreifen und auf diesem Weg an Referenzen auf Entitäten aus diesen Teilmodellen gelangen. Somit können Entitäten aus anderen Teilmodellen aktiviert und passiviert werden oder es können Zustandsänderungen an den Entitäten ausgelöst werden. Hierfür muss jedoch die Referenz auf die entsprechende Warteschlange in dem jeweiligen Teilmodell bekannt sein. Diese Referenzen können bei der Initialisierung der Teilmodelle verteilt oder explizit bei den Teilmodellen erfragt werden, sofern die Referenzen auf die Teilmodelle bekannt sind. Eine flexiblere Möglichkeit der Interaktion bieten die sogenannten Message-Cross-Bars, über welche über Modellgrenzen hinweg Nachrichten verschickt werden können. Entitäten können sich an dieser Message-Cross-Bar anmelden, um bekanntzugeben, welche Nachrichten sie empfangen wollen. Äquivalent zu der Simulation von nur einem Modell wird auch bei der Simulation mit mehreren Teilmodellen die Ausführungskontrolle immer nur an einen Prozess zur Zeit übergeben.

4.2.4. Parallele Simulation

Bei der parallelen Simulation werden die Teilmodelle auf verschiedenen Prozessorkernen oder verschiedenen Rechnern in einem Netzwerk berechnet. Ein mögliches Ziel ist, einen Simulationslauf zu beschleunigen, was unter bestimmten Umständen möglich ist. Auch

ist es denkbar, dass einzelne Teilmodelle nur auf einer bestimmten Art von Hardware laufen und deshalb die parallele Berechnung notwendig wird. Dabei gibt es zwei verschiedene Ansätze: Die optimistisch-parallele und die konservativ-parallele Simulation.

Bei der konservativ-parallelen Simulation muss die Berechnung der Teilmodelle synchron erfolgen. Jeder Simulatorkern darf nur bis zu demjenigen Zeitpunkt in der Simulationszeit fortschreiten, zu dem eine mögliche Zustandsänderung im eigenen Teilmodell durch eines der anderen Teilmodelle noch möglich ist. Bei der optimistisch-parallelen Simulation berechnet der Simulator hingegen ohne Rücksicht auf die Simulationszeit der anderen Modelle die im eigenen Modell anstehenden Zustandsänderungen. Wenn durch ein anderes Teilmodell eine Zustandsänderung zu einem Zeitpunkt erfolgt, welcher gemäß dem Stand der eigenen Simulationsuhr bereits in der Vergangenheit liegt, so werden sämtliche Berechnungen seit diesem Zeitpunkt hinfällig. Sie müssen verworfen werden und der Zustand, der zu diesem Zeitpunkt gültig war, muss wiederhergestellt werden. Dies erfordert eine genaue Protokollierung der Modellzustände. Erst wenn alle anderen Teilmodelle über einen bestimmten Zeitpunkt hinaus gerechnet haben, dürfen die Zustände vor diesem Zeitpunkt verworfen werden, um Speicher frei zu geben. Ein Ansatz zur optimistisch-parallelen Simulation wird in Kunert (2010) beschrieben.

Da die Teilmodelle in der vorliegenden Arbeit auf dem selben Simulatorkern ausgeführt werden sollen, werden alle Teilmodelle vollkommen synchron berechnet und die Notwendigkeit der Speicherung von Zustandsfolgen, um bei Kausalitätsfehlern zu einem Zeitpunkt, zu dem der Fehler noch nicht vorlag, zurückspringen zu können, entfällt. Die in Kunert (2010) oder ähnlichen Arbeiten beschriebenen Ansätze könnten in zukünftigen Arbeiten übernommen werden. Er beschreibt jedoch, dass sämtliche Zustandsänderungen, die an einem anderen Teilmodell durchgeführt werden sollen, durch Nachrichten angestoßen, aber letztlich durch das entsprechende Teilmodell selbst durchgeführt werden sollen. Dies ist notwendig, um Inkonsistenzen zu vermeiden. Die weiteren in Grimm und Schroll (2007) beschriebenen Möglichkeiten der Interaktion, durch gemeinsamen Zugriff auf Variablen, Semaphore oder Speicherbereiche zu realisieren, wäre demzufolge ein weniger zu empfehlender Ansatz.

4.3. Plug-in-basierte Entwicklung als softwaretechnische Grundlage

Die komponentenbasierte Entwicklung, insbesondere die Plug-in-basierte Entwicklung, ist ein wichtiger Ansatz zur Kopplung von Software-Artefakten und somit Grundlage für die Konzepte dieser Arbeit. Die hier entstehenden Vorteile werden im folgenden erläutert. Zwei Beispiele für erweiterbare Simulationswerkzeuge werden vorgestellt.

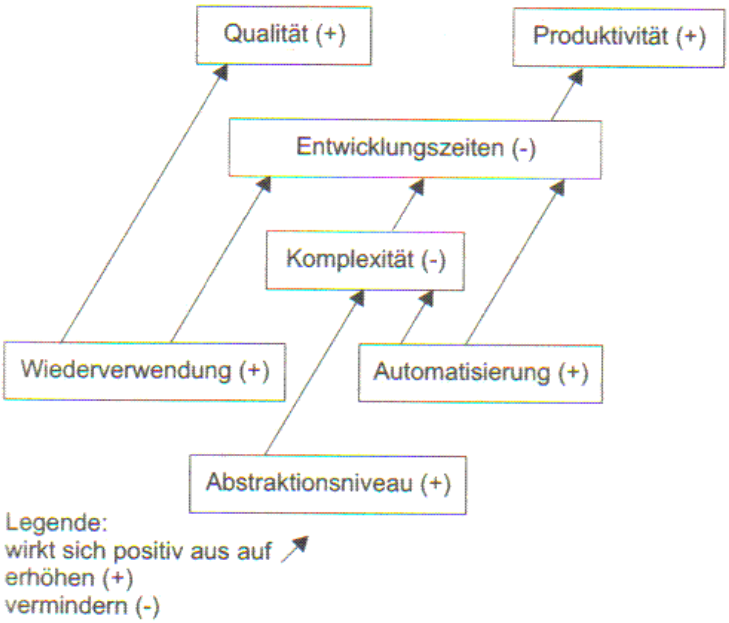


Abbildung 4.2.: „Ziele der komponentenbasierten Softwareentwicklung“ aus Zwintzscher (2005), S.15

4.3.1. Zielsetzung

Der Ansatz eine Software so zu entwerfen, dass sie aus wiederverwendbaren, kombinierbaren Elementen besteht, wurde schon 1968 von McIlroy postuliert, als er forderte „Software aus in Massen gefertigten Softwarekomponenten“ zusammenzusetzen (vgl. McIlroy (1968), S.138). Vergleichbar mit der modularisierten Bauweise komplexer Produkte in der Industrie, soll dies die Entwicklung von Softwareprodukten vorantreiben (vgl. Floyd und Züllighoven (2002), S.781). Seit den ersten höheren Programmiersprachen werden Programme in Prozeduren gegliedert, damit bereits implementierte Algorithmen bei der Softwareentwicklung wiederverwendet werden können. Die objektorientierten Sprachen führten die Kapselung von Datenstrukturen mit dazugehörigen Prozeduren zu Klassen ein. Hier wird eine öffentliche Schnittstelle spezifiziert, über die einzelne Instanzen miteinander kommunizieren können. Die Kenntnis dieser Schnittstelle ist notwendige Voraussetzung für die Kommunikation. Bei der komponentenbasierten Softwareentwicklung wird die Granularität wiederum vergröbert. Sie kapselt mehrere kollaborierende Klassen und Ressourcen zu einem semantisch in sich geschlossen Softwarebaustein, welcher seine Funktionalität anderen Komponenten über Schnittstellen zur Verfügung stellt.

Die Abbildung 4.2 verdeutlicht den Zusammenhang der Vorteile dieses Ansatzes. Software-Komponenten sind wiederverwendbar. Dies spart Entwicklungszeit ein, da bei der Entwicklung eines neues Software-Produktes auf bestehende Komponenten zurückgegriffen werden kann. Wenn beispielsweise ein domänenspezifischer Simulator entworfen wird, können bestehende Komponenten der Benutzeroberfläche, des Simulators oder zur Definition von Teilmodellen wiederverwendet werden. Nur die domänenspezifischen Funktionen der gewünschten Anwendung müssen neu entwickelt werden. Dies erhöht auch die Qualität der Softwareanwendung, da einzelne Komponenten von einer größeren Benutzergruppe getestet werden.

Durch die gröbere Granularität wird das Abstraktionsniveau erhöht. Für den Entwickler erscheint die Software übersichtlicher und weniger komplex, denn er muss sich nicht mit den detaillierten Interna einer bereits fertigen Komponente beschäftigen, sondern er benutzt lediglich deren Schnittstellen. Verteilte Entwicklung an der Software wird vereinfacht, da die Softwarekomponenten voneinander unabhängig gehalten werden. Durch die Verringerung der Komplexität wird zusätzlich Entwicklungszeit eingespart. Die beteiligten Personen können produktiver arbeiten, da sie nur noch Expertenwissen über ihre Komponenten benötigen.

4.3.2. Komponenten

Der Begriff *Komponente* ist nicht einheitlich definiert, da er in vielen verschieden Kontexten verwendet wird. Auch innerhalb der Softwaretechnik gibt es sehr allgemeine Definitionen, die es erlauben, nahezu jedes wiederverwendbare Softwareartefakt als Komponente zu bezeichnen (vgl. Booch et al. (1998), S. 345). Auf der anderen Seite gibt es sehr stark eingrenzende Definitionen, welche zum Beispiel Komponenten und Klassenbibliotheken aufgrund fehlender Komponentenschnittstellen sauber voneinander trennen (vgl. Zwintzscher (2005), S.23). Szyperski (1997) (S. 34) beschreibt eine Komponente als eine Kompositionseinheit, welche ihre Schnittstellen vertraglich spezifiziert. Diese Festlegung ist zur Kollaboration zwischen den Komponenten und einer Rahmenanwendung oder den Komponenten untereinander vonnöten.

Durch die Betrachtung existierender Kompononentenmodelle wie JavaBeans, EJB, COM, CORBA oder .NET werden in Zwintzscher (2005) (S. 79-95) noch einige weitere, technische Aspekte von Komponenten herausgearbeitet werden:

- Eine Komponente wird als binärer Softwarebaustein an den Kunden geliefert, zum Beispiel als CAB-, JAR- oder DLL-Datei.
- Eine Komponente enthält Metadaten um ihren eigenen Aufbau zu beschreiben.
- Die Schnittstellen beschreiben Methoden, Eigenschaften und Ereignisse.

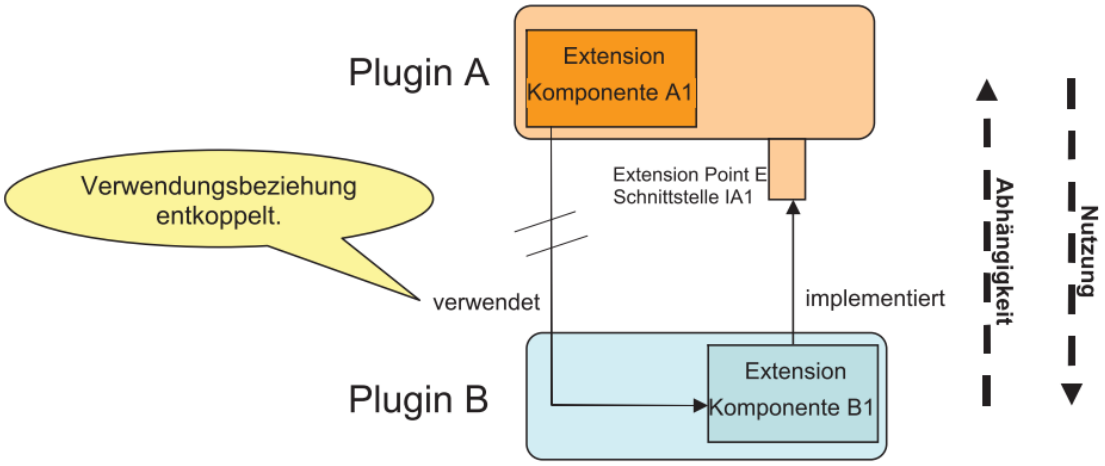


Abbildung 4.3.: Kommunikation zwischen zwei Plug-in-Komponenten. Abbildung aus: Schnackenbeck et al. (2007)
)

Der Rahmen, in dem Komponenten agieren, kann hinsichtlich der Kopplung mit der Komponente stark differenzieren. Eine *statische Kopplung* von Komponenten meint die feste Einbindung einer Komponente in eine Anwendung zu einem monolithischen Ganzen, ohne deren Vorhandensein die Anwendung nicht funktioniert. Demgegenüber steht die *dynamische Kopplung*, bei der Komponenten an- und abgemeldet werden können. Bei ihnen kann der Benutzer auch nach der Installation der Software Komponenten hinzufügen, entfernen oder austauschen (vgl. Griffel (1998), S.161f).

4.3.3. Rahmenanwendung und Plug-ins

Wenn dynamisch koppelbare Komponenten für eine konkrete Rahmenanwendung entworfen werden, werden sie auch als Plug-in-Komponenten bezeichnet. Die Rahmenanwendung oder andere Komponenten geben vertragliche Schnittstellen vor, die von der Plug-in-Komponente implementiert werden können. Beispiele für solche Plug-in-Rahmenwerke sind die Eclipse IDE in der Java-Welt oder Empinia für .NET-basierte Anwendungen. Die Schnittstelle wird als *Erweiterungspunkt* bezeichnet und die Klassen, welche die Schnittstellen erfüllen, sind die dazu passenden *Erweiterungen*. Die Rahmenanwendung oder eine Komponente A kann die von Plug-In-Komponente B bereitgestellte Erweiterung verwenden, ohne deren interne Implementation kennen zu müssen (vgl. Abbildung 4.3).

Dies ist nicht zu verwechseln mit Konzepten aus der objektorientierten Entwicklung wie dem Observer-Pattern[6]. Auch hier verwendet zwar ein Objekt A ein Objekt B gemäß der durch A geforderten Schnittstelle, jedoch muss B die hierfür erforderliche Referenz auf sich selbst aktiv bei A bekannt geben. Dazu muss bereits eine Instanz von B existieren, welche die Ausführungskontrolle inne hat, was bei dem Plug-in-basierten Ansatz nicht der Fall sein kann. Hier wird daher das Inversion-of-Control-Prinzip (IoC) angewandt, welches auch „Hollywood-Prinzip“ genannt wird: „Don’t call us, we call you!“. Damit die Rahmenanwendung die notwendigen Objekte aus Plug-in B gemäß dem IoC-Prinzip instanziieren kann, benötigt es Kenntnis über diejenigen Klassen, welche die Erweiterungen bereitstellen. Diese werden über eine Manifest-Datei bekanntgegeben, welche zum Beispiel im XML-Format der Komponenten beigelegt wird. Hier sind der Name und der Namensraum der implementierenden Klasse, sowie der durch sie erfüllte Erweiterungspunkt angegeben.

Die Rahmenanwendung durchsucht beim Start der Anwendung ihr Applikationsverzeichnis nach Programmbibliotheken im Format der jeweiligen Plattform (zum Beispiel JAR oder DLL). Alle gefundenen Bibliotheken werden geöffnet und nach einer Manifest-Datei durchsucht. Wenn diese vorhanden ist, so wird die Bibliothek als Plug-in erkannt. Die in ihr definierten Erweiterungen und Erweiterungspunkte werden ausgelesen und die entsprechenden Objekte zur gegebenen Zeit per Reflection-Mechanismen instanziiert.

Dieses Konzept ermöglicht, dass ein Plug-in unabhängig von der Rahmenanwendung von Dritten entworfen werden kann. Abhängig vom zugrundeliegenden Komponentenmodell muss dies auch nicht zwangsläufig in der gleichen Programmiersprache erfolgen. In Gegensatz zu Klassen stellt eine Komponente somit eine Auslieferungseinheit dar und wird als Baustein direkt an den Kunden geliefert (vgl. Zwintzscher (2005), S.20).

Auf diesem Prinzip wird ein BPMN-Simulator entworfen, der Erweiterungspunkte für domänenspezifische Teilmodelle bietet. Der saubere Entwurf der Schnittstellen ist dabei von wesentlicher Bedeutung, denn eine spätere Änderung der Schnittstellen zieht in der Regel eine Änderung von allen bereits erstellten Komponenten nach sich. Ein sauberer Entwurf bedeutet hierbei eine fundierte Analyse der Anforderungen und die Anwendung von möglichst bewährten Entwurfsmustern (vgl. Griffel (1998), S.162f).

4.3.4. Komponentenbasierte Simulation

Die komponentenbasierte Entwicklung wurde bereits in anderen Arbeiten zur Implementation von erweiterbaren Simulationswerkzeugen genutzt. In Wohlgemuth (2005) wird

[6] Das Observer-Pattern ist ein Entwurfsmuster aus der objektorientierten Programmierung (vgl. Gamma (2013)).

einen plug-in basierter Stoffstromsimulator vorgestellt, in welchem betriebliche Prozesse mit Hilfe von FlowCharts modelliert werden. Die Knoten der Flussdiagramme repräsentieren Blöcke gemäß der transaktionsorientierten Modellierung.

Blöcke sind statische Systemkomponenten, welche permanent im System vorhanden sind und einen Zustand besitzen. Dies können zum Beispiel sein:

- Bedienstationen unterschiedlichster Art, zum Beispiel Maschinen, Maschinenkomplexe oder Arbeitstische
- Speicher, welche ein Lager mit begrenzter, vorgegebener Kapazität darstellen
- Leitstellen, welche die Transaktionen gemäßbestimmter Bedingungen steuern

(vgl. Page (1991), S.32)

Transaktionen sind temporäre, dynamische Systemelemente (zum Beispiel ein zu fertigendes Produkt). Sie betreten das System über einen Systemeingangsblock und verlassen das System über einen Systemausgangsblock. Durch die Leitstellen gesteuert, werden sie von Block zu Block durch das System gereicht und lösen dort Zustandsänderungen aus. So kann zum Beispiel ein Maschinenblock belegt oder freigegeben, ein Lagerblock gefüllt oder eine Warteschlange geleert werden. Alternativ können mehrere Bestandteile eines Werkstücks an einem Block zusammengesetzt werden, Werkstücke zu Batches zusammengefasst und wieder auseinandergenommen werden.

Welche Art von Blöcken und Transaktionen benötigt werden, ist dabei abhängig von der Domäne, in welcher der Simulator eingesetzt wird. Sie werden durch entsprechende Plug-in-Komponenten bereitgestellt, der Simulator ist somit adaptierbar für verschiedene Branchen. Für den Stoffstromsimulator gibt es beispielsweise ein Plug-in mit Blöcken für allgemeine Produktionsprozesse und eines für die Fabrikation von Halbleitern (vgl. Vacek et al. (2007)). Die Blöcke werden in einer Symbolleiste angemeldet, so dass der Benutzer diese per Drag & Drop in den passenden graphbasierten Editor ziehen kann. Abbildung 4.4 verdeutlich die Software-Architektur des Simulators, in der verschiedene Plug-ins an- und abgemeldet werden können. Hierzu zählen Plug-ins mit Modellkomponenten (Transaktionen und Blöcke), mit Editoren zur Manipulation der Modelle und Komponenten zur Auswertung der Ergebnisse.

Der Fokus dieses Simulators liegt auf der Kombination der ökonomischen und ökologischen Sichtweise auf einen Produktionsprozess. Neben den monetären Kosten wird der Stoffdurchsatz errechnet, was die benötigten Roh- und Hilfsstoffe sowie die erzeugten Emissionen und Abfallstoffe beinhaltet. Hierfür werden weitere Plug-ins zur Verwaltung der Materialen und der entsprechenden ökologischen Kennzahlen benötigt.

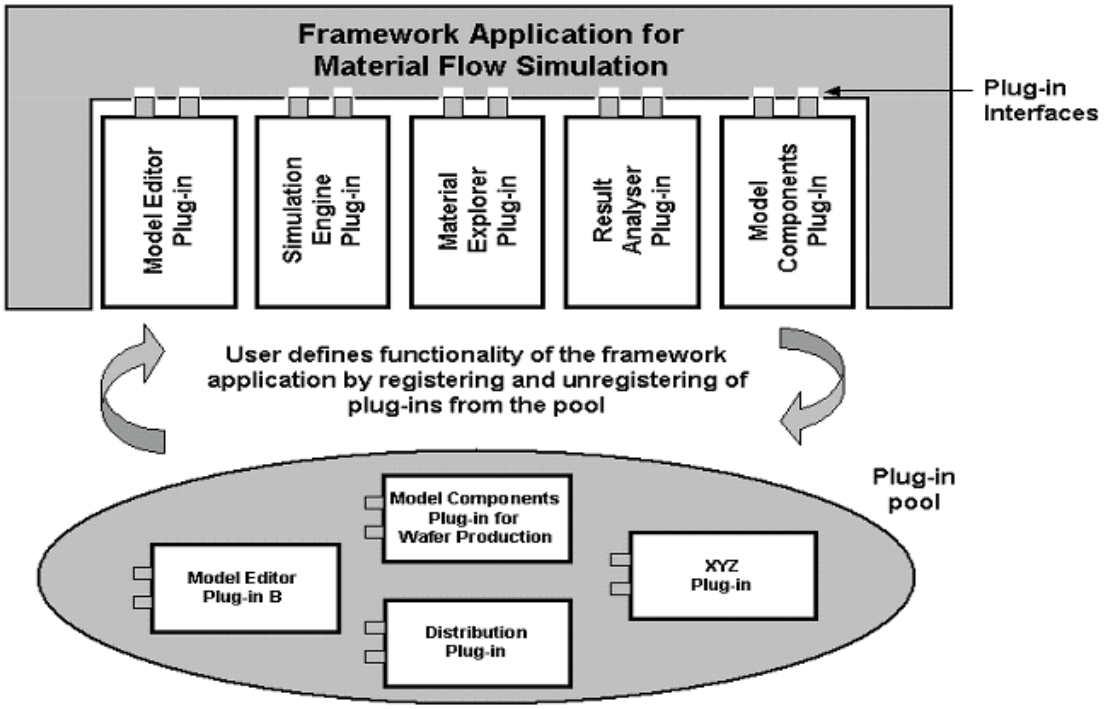


Abbildung 4.4.: Simulationsrahmenwerk zusammengesetzt aus Software-Komponenten. Originaltitel: „Building an Application Framework for Material Flow Simulations from Component-Based Plug-Ins“ Abbildung aus: Wohlgemuth et al. (2006)

4.3.5. Simulation mit Empinia

In Jahr et al. (2009) wird eine Reimplementation des Stoffstromsimulators aufbauend auf dem Empinia-Rahmenwerk beschrieben. Während die Version aus Wohlgemuth (2005) noch auf einem eigens entwickelten Rahmenwerk in der Programmiersprache Delphi aufsetzte und zur Interaktion der Komponenten das Common Object Model (COM) von Microfft nutze, wird in dieser Version nun das aktuellere Komponentenmodell des .NET Frameworks genutzt, welche die direkte Referenzierung von Objektinstanzen aus unterschiedlichen Komponenten erlaubt.

Empinia ist ein Open-Source-Projekt und stellt ein Rahmenenwerk zur komponentenbasierten Softwareentwicklung in .NET bereit[7]. Es ist vergleichbar mit dem in der Java-Welt weit verbreiteten Eclipse-Rahmenwerk. Es bietet einen Kollaborationsrahmen für beliebige Anwendungenskomponenten und beinhaltet diverse nützliche, wiederverwendbare Basiskomponenten. Die flexible Kombination dieser Basiskomponenten und die vorgehaltenen Konzepte für selbst zu entwickelnde anwendungsspezifische Komponenten ermöglichen die kostengünstige Implementation individuell zugeschnittener, erweiterbarer Fachanwendungen. Erweiterungen können durch bloßes Hinzufügen einer DLL-Datei in das Applikationsverzeichnis integriert werden. Empinia durchsucht beim Start das Applikationsverzeichnis nach DLL-Dateien, öffnet diese und überprüft, ob eine Manifest-Datei namens „Bundle.xml“ enthalten ist. In dieser Datei werden die zur Verfügung gestellten Erweiterungen beschrieben, so dass Empinia diese bei Bedarf instanziieren kann. Für einen umfassenden Überblick über die Architektur von Empinia sei auf Schnackenbeck et al. (2007) und Panic et al. (2008) verwiesen. Empinia wird auch für den in dieser Arbeit entwickelten Prototypen verwendet. Die hierbei genutzten Funktionen werden in Abschnitt 6.1 beschrieben.

Typischerweise sind die mit Hilfe von Empinia entwickelten Anwendungen im Bereich der Betrieblichen Umweltinformationssysteme (BUIS), der Modellbildung und der Simulation anzusiedeln. Empinia wurde von der HTW Berlin, der ifu Hamburg GmbH, der Leuphania Universität Lüneburg und der Universität Hamburg entwickelt.

In Jahr et al. (2009) werden Komponenten zur Realisierung des Stoffstromsimulators vorgestellt, welche selbst Erweiterungspunkte zur Anmeldung von Modellkomponenten mitbringen. Hierzu gehört erneut die Simulationsinfrastruktur, eine Komponente zum Materialmanagement und eine Analysekomponente zur Ergebnisauswertung. Der komponentenbasierte Ansatz wurde hier jedoch gegenüber Wohlgemuth et al. (2006) verfeinert. So stehen nun Erweiterungspunkte für verschiedene Modellelemente wie Arbeitsstationen, Transportsysteme, Puffer, Systemeingänge und -ausgänge zu Verfügung.

[7] Der Source Code kann unter Empinia (2014) heruntergeladen werden.

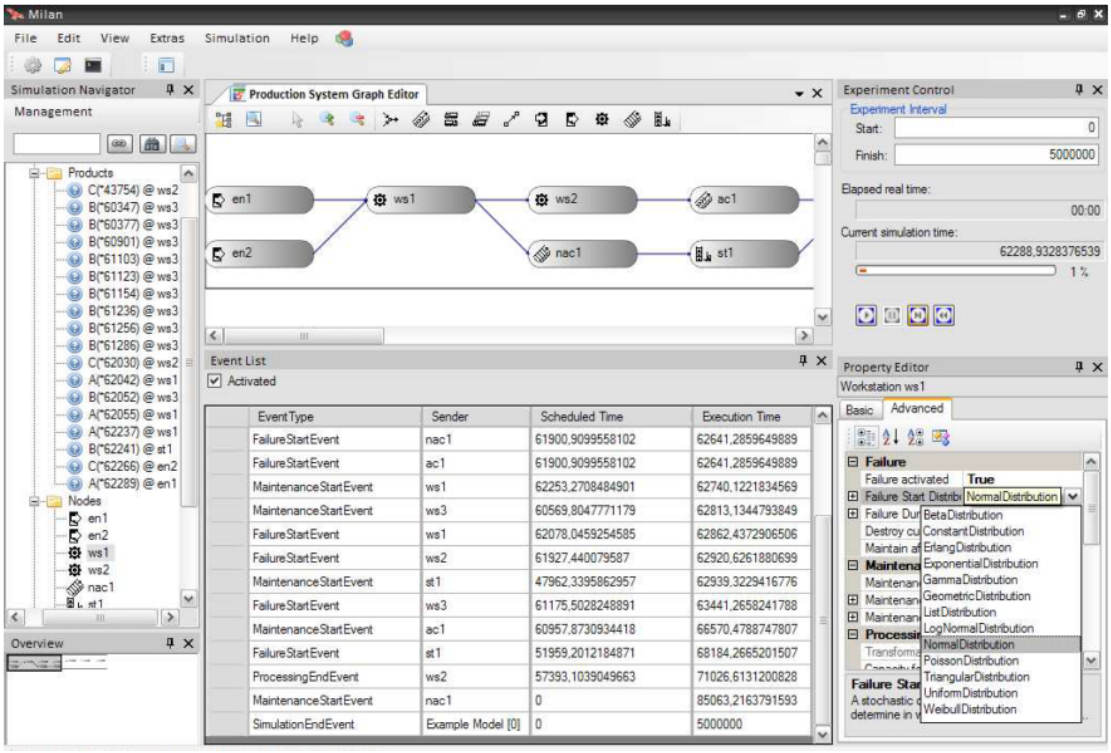


Abbildung 4.5.: Screenshot der Plug-in-basierten Simulationssoftwrae MILAN. Originaltitel: „Screenshot of MILAN application based on EMPINIA“. Abbildung aus: Jahr et al. (2009)

Durch eine Plug-in-Komponente können konkrete Implementationen für diese Modellelemente bereitgestellt werden. Die verwendeten Modellelemente werden dadurch austauschbar, was eine leichte Modifikation der Modelle für verschiedene Einsatzszenarien ermöglicht. Auch können innerhalb eines Simulationsmodells nun Modellelemente aus verschiedenen Softwarekomponenten verwendet werden.

Abbildung 4.5 zeigt die Benutzeroberfläche des Simulationswerkzeuges basierend auf Empinia. Auf der linken Seite ist eine Toolbox mit Modellelementen zu sehen, welche in den Dokumentbereich oben in der Mitte gezogen werden können. Auf der rechten Seite können zu den einzelnen Modellelementen weitere Einstellungen vorgenommen werden. Die Oberfläche des in dieser Arbeit entwickelten Prototypen wird hieran angelehnt. Jedoch wird nicht die Simulationskomponente aus Jahr et al. (2009) verwendet, sondern die in Abschnitt 3.4 vorgestellte Bibliothek zur BPMN-Simulation.

5. Konzeption eines erweiterbaren BPMN-Simulators

In diesem Kapitel werden Konzepte für einen erweiterbaren BPMN-Simulator entwickelt. Zunächst wird beschrieben, wie Prozessdefinitionen auf Modellierungsebene um domänenspezifische Symbole erweitert werden können. Die Verknüpfungsmöglichkeiten dieser Symbole mit domänenspezifischen Modelldefinitionen und deren Entitäten werden dargestellt. Hierdurch werden die Voraussetzungen für die Kopplung der Simulationsmodelle für eine Interaktion zur Laufzeit geschaffen. Es wird beschrieben, welche Komponenten der Entwickler einer Erweiterung bereitstellen muss und wie diese Komponenten mit der Rahmenanwendung interagieren. Die im Einzelnen notwendigen Schritte werden aufgezeigt und Rollen mit jeweils erforderlichem Expertenwissen zugeordnet. Eine prototypische Implementation dieser Konzepte wird anschließend in Kapitel 6 vorgestellt.

5.1. Domänenspezifische Erweiterungen für die BPMN 2.0

Zunächst wird beschrieben, welche Erweiterungen der BPMN 2.0 auf Modellierungsebene vorgenommen werden können. Für jeden Elementtyp wird begründet, ob hier eine domänenspezifische Erweiterung sinnvoll ist und welche Art von Interaktionsbeziehung damit beschrieben werden kann.

5.1.1. Wahl der Symbole

Wie in Kapitel 2.3 beschrieben, ist eines der Ziele der BPMN 2.0, eine möglichst breite Zielgruppe von Mitarbeitern in einem Betrieb zu erreichen. Die in der Notation verwendeten Symbole wurden daher großteils so gewählt, dass sie möglichst intuitiv verständlich sind. Diese intuitive Verständlichkeit soll auch bei der Verknüpfung von Prozessen mit domänenspezifischen Teilmodellen ermöglicht werden. Die Verwendung von in der Domäne bekannten Symbolen, oder die Verwendung von Symbolen, die eindeutig auf die entsprechende Domäne verweisen, ist ratsam. Im Rahmen dieser Arbeit werden daher keine Symbole vorgegeben; vielmehr sollen für die entsprechende Erweiterung passende Symbole frei gewählt werden können. Jedoch muss festgelegt werden,

an welchen Stellen innerhalb der BPMN 2.0 überhaupt sinnvolle Erweiterungen für die Simulation vorgenommen werden können, um Konzepte bereit zu stellen, wie die Erweiterung implementiert werden kann und welche Auswirkungen diese auf die Simulation haben.

Wie in Abschnitt 2.3.8 beschrieben, erlaubt die BPMN die Verwendung selbstdefinierter Symbole nur an bestimmten Stellen. So dürfen keine neuen Flusselemente eingeführt werden. Stattdessen dürfen die bestehenden Elemente meist lediglich mit Erweiterungsattributen versehen werden, welche mit eigenen Symbolen visualisiert werden dürfen. Dabei muss das grundsätzliche Erscheinungsbild von Ereignissen und Aktivitäten beibehalten werden. Die grundsätzliche Bedeutung von Ereignissen und Aktivitäten darf nicht verändert werden.

Für die im folgenden beschriebenen Erweiterungsoptionen sollen also keine neuen BPMN-Klassen implementiert werden, sondern lediglich die bestehenden Klassen mit Attributen versehen werden, die auf die jeweilige Domäne verweisen. Anhand der Belegung dieser Attribute wird erkannt, um welches Domänenelement es sich handelt, so dass dieses entsprechend visualisiert werden kann.

5.1.2. Ereignisse

Es gibt in der BPMN 2.0 empfangende und sendende Ereignisse. Sendende Ereignisse werden durch schwarz gefärbte Symbole ausgedrückt, empfangende durch weiß gefärbte Symbole. Durch Verwendung eines entsprechenden Symbols kann also dargestellt werden, dass ein Signal an einen bestimmten Modelltyp gesendet oder von einem bestimmten Modelltyp empfangen wird.

In der Regel wird eine zielgerichtete Kommunikation mit bestimmten Modellinstanzen oder Entitäten aus einem Modell benötigt, die bloße Verknüpfung mit einem Modelltyp reicht nicht aus. In der BPMN wird die Zuordnung zu einem Empfänger mit Nachrichtenereignissen ausgedrückt. Diese sind jedoch der Verwendung zwischen zwei Prozessdefinitionen vorbehalten. Für die Verknüpfung mit Teilmodellen werden andere Ansätze benötigt, dies wird in Abschnitt 5.2 beschrieben.

Beim Austausch von Nachrichten können jeweils Informationen übermittelt werden. So kann das Teilmodell Informationen durch den Versand von Nachrichten an eine Prozessinstanz übermitteln. Der Prozess wiederum kann Daten an das Teilmodell schicken, die entsprechend ausgewertet werden können.

Die BPMN-Spezifikation gibt vor, dass Startereignisse nur in der empfangenden Variante, Endereignisse nur in der sendenden Variante und Zwischenereignisse in beiden Varianten

existieren. Da dies der Ausführungslogik eines Geschäftsprozesses entspricht, darf diese Vorgabe im Rahmen von Erweiterungen nicht verletzt werden.

Bei der herkömmlichen Prozesssimulation können neue Prozessinstanzen nur durch stochastische Ereignisse oder durch andere, bereits existierende Prozessinstanzen erzeugt werden. Die Besonderheit eines empfangenden, domänenspezifischen Startereignisses ist hierbei, dass nun zusätzlich durch das betreffende Teilmodell eine neue Geschäftsprozessinstanz gestartet werden kann.

Verschiedene Typen von Signalen oder Nachrichten können durch verschiedene Symbole dargestellt werden. Hierbei sollte auf Verständlichkeit der verwendeten Symbole geachtet werden. Die Kombination von bestehenden Symbolen der BPMN mit domänenspezifischen Symbolen kann genutzt werden, um die Wiedererkennbarkeit zu erhöhen. So kann zum Beispiel das Fehlersymbol oder das Zeitgebersymbol integriert werden, um zu konkretisieren, um was für eine Art von Nachricht es sich handelt.

5.1.3. Aktivitäten

Aktivitäten beschreiben durchzuführende Arbeiten innerhalb eines Prozesses. Die BPMN 2.0 sieht an dieser Stelle bereits verschiedene Symbole für verschiedene Arten von Arbeiten vor (vgl. Abschnitt 2.3.2). Durch Verwendung eines domänenspezifischen Symbols kann hierbei ausgedrückt werden, dass es sich um Arbeiten handelt, die das entsprechende System aus der Prozessumgebung betreffen.

Die Dauer dieser Arbeiten kann somit von dem entsprechenden Teilmodell definiert werden. Auch kann sich sowohl der Beginn, als auch das Ende der Arbeiten auf den Zustand des jeweiligen Teilmodells manipulativ auswirken. Letztlich kann die gesamte Aktivität durch das Teilmodell implementiert werden.

Wie bei den Ereignissen, ist durch die Verwendung eines Symbols noch nicht konkret festgelegt, im Rahmen welcher Modellinstanz oder Entität die Arbeiten durchgeführt werden. An dieser Stelle ist lediglich der zu verknüpfende Modelltyp festgelegt. Durch die Verwendung verschiedener Symbole können auch hier verschiedene Arbeiten dargestellt werden, die zwar denselben Modelltyp betreffen, aber unterschiedlich implementiert werden, und somit unterschiedliche Auswirkung auf das Laufzeitverhalten sowie die Zustände von Prozess- und Teilmodellen haben.

Bei der Durchführung von Aktivitäten können wie beim Empfang von Nachrichten Informationen generiert werden, die durch das jeweilige Teilmodell übermittelt werden und anschließend im Kontext der Prozessinstanz zu Verfügung stehen.

5.1.4. Kanten

Es ist weder erlaubt, noch wäre es für die Simulation zielführend, eigene Kantentypen einzuführen.

An der grundlegenden, durch Sequenzkanten beschriebenen Ausführungssemantik eines Prozesses soll nichts verändert werden. Nachrichtenkanten sind der Interaktion zwischen Geschäftsprozessen vorbehalten. Sie dürfen lediglich Elemente aus verschiedenen Pools verknüpfen. Diese Syntax soll nicht verändert werden.

Lediglich die Assoziationskanten ermöglichen es, für die Kopplung von Simulationsmodellen gewinnbringende Informationen zu modellieren, da mit ihnen die Assoziation eines Ereignisses oder einer Aktivität mit einem beliebigen Artefakt ermöglicht wird. Hier wird jedoch keine Modifikation der Kanten, sondern eine Modifikation der Artefakte benötigt, wie im nächsten Abschnitt beschrieben.

5.1.5. Artefakte

Artefakte sind die einzigen Elemente in der BPMN, welche mit einer völlig frei wählbaren Symbolik belegt werden dürfen. Lediglich eine Überschneidung mit bestehenden Symbolen ist nicht erlaubt. Artefakte beschreiben in der BPMN bestimmte Entitäten aus dem Prozesskontext, zum Beispiel Datenbanken oder Datenobjekte. Ob diese Entität nur im Prozesskontext oder global bekannt ist, hängt dabei von dem verwendeten Artefakt ab, beide Prinzipien werden in der BPMN bereits verwendet Das Datenobjekt ist nur innerhalb einer Prozessinstanz bekannt. Die Datenbank ist global verfügbar[8]. Im Rahmen einer Erweiterung der BPMN 2.0 sollen Artefakte Entitäten aus dem jeweiligen Teilmodell oder bestimmte Teilmodellinstanzen referenzieren.

Es sollen Artefakte ermöglicht werden, welche Referenzen auf ein bestimmtes Teilmodell oder eine Entität aus diesem Teilmodell beschreiben. Mit Hilfe von Assoziationskanten werden diese, wie für Artefakte vorgesehen, mit Ereignissen oder Aktivitäten verknüpft. Die Richtung der Assoziationskante beschreibt dabei, ob in einer laufenden Prozessinstanz eine Referenz auf solch eine Entität hinterlegt wird, oder ob eine Referenz vom Artefakt zur Verwendung im Prozesskontext geholt wird.

Ein empfangendes Ereignis kann hierdurch die Referenz auf den Sender der Nachricht in einem domänenspezifischen Artefakt speichern. So kann sich eine Prozessinstanz merken, mit welcher Entität sie interagiert. Eine sendendes Ereignis kann sich diese Referenz

[8] Streng genommen zählen diese beiden Objekte zwar nicht zu den Artefakten, werden aber ebenso wie Artefakte mit Assoziationskanten verknüpft.

holen, um eine zielgerichtete Nachricht an genau diese Entität zu senden. Auch Aktivitäten können sich diese Referenz holen, um konkret den Zustand dieser Entität zu manipulieren, oder die Dauer der Ausführung von dieser Entität bestimmen zu lassen.

5.1.6. Gateways

Gateways dienen in einer Prozessdefinition der Wahl von möglichen Ausführungsvarianten eines Prozesses. Die verschiedenen Arten von Gateways geben dabei an, ob die Verzweigung exklusiv, parallel oder inklusiv zu verstehen ist. Das Spektrum der Möglichkeiten aus den gegebenen Pfaden zu wählen, wird durch die insgesamt sieben in der BPMN 2.0 zur Verfügung stehenden Gateways umfassend ausgeschöpft, weitere Gateways werden nicht benötigt.

Die Entscheidung für einen Pfad kann sich zwar auf Informationen beziehen, die einem der Teilmodelle entnommen wurden, die zugrundeliegende Entscheidung bei Verzweigungen findet allerdings in einer Aktivität vor dem Gateway statt. So muss auch im Falle domänenbezogener Entscheidungen die benötigte Information zur Entscheidungsfindung einem entsprechendem Empfangsereignis oder einer Aktivität in dem Pfad vor dem Gateway entnommen und in den Prozesskontext übertragen werden. An dem Gateway wird diese Information aus dem Prozesskontext lediglich ausgewertet. Diese besteht aus einem Variablennamen sowie einem Wert und unterscheidet sich strukturell nicht von Daten, die nicht einem domänenspezifischen Teilmodell entnommen wurden. Die Erweiterung der Notation um domänenspezifische Gateways bringt daher keinen Mehrwert.

5.1.7. Beispiele für die ausgewählten Elemente

Abbildung 5.1 zeigt exemplarisch mögliche domänenspezifische BPMN-Elemente aus der Domäne der Offshore Windparks, welche in Kapitel 7 als Fallstudie dienen wird. Zu sehen sind zwei Ereignisse, welche den Empfang eines Signals von einer Windparkanlage (oben links) sowie den Wechsel von Wetterbedingungen repräsentieren (unten rechts). Wie für Empfangsereignisse üblich, sind diese schwarz umrandet und weiß gefüllt. Die dargestellte Aktivität repräsentiert Arbeiten auf einer Windpark-Anlage (unten links). Das Artefakt (oben rechts) wird mittels Assoziationskanten mit anderen Ereignissen oder Aktivitäten verbunden und repräsentiert die Referenz auf eine Windparkanlage.

5.1.8. Prozessdefinition

Mit Hilfe der vier genannten Erweiterungsmöglichkeiten (sendende Ereignisse, empfangende Ereignisse, Aktivitäten und Artefakte) kann auf Modellierungsebene bereits die

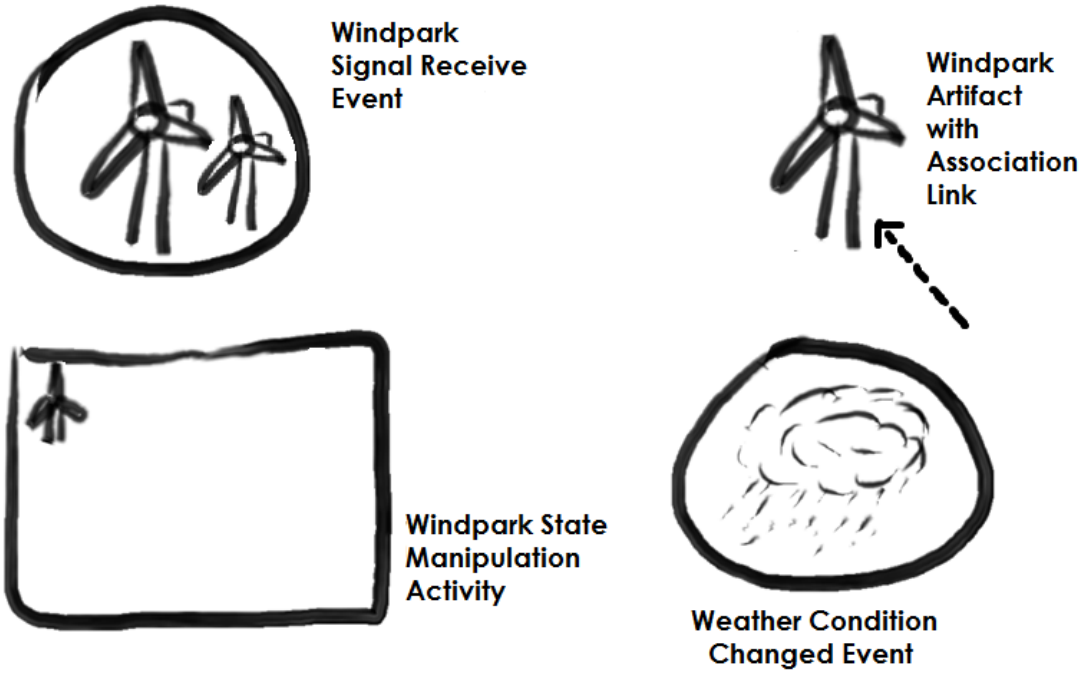


Abbildung 5.1.: Beispiele für die Verwendung von domänenspezifischen Symbolen in BPMN-Elementen

Interaktion mit Teilmodellen vermerkt werden. Für eine vollständige Prozessdefinition werden diese Elemente mit den Standardelementen der BPMN 2.0 kombiniert. Ereignisse und Aktivitäten werden über Sequenzflüsse in die Prozessdefinition eingebaut. Beide können mit Hilfe von Assoziationskanten auf konkrete Teilmodelle oder Entitäten zugreifen. Herkömmliche Ereignisse und Aktivitäten können ebenfalls in die Prozessdefinition integriert werden, um Aktivitäten oder Ereignisse zu beschreiben, die nicht mit domänenspezifischen Modellen, sondern mit anderen Prozessen interagieren oder komplett ohne eine Interaktion auskommen.

Hierdurch wird die Modellierung von Geschäftsprozessen ermöglicht, die punktuell mit anderen Modellen interagieren. Für eine vollständige Definition der Verknüpfung mit anschließender Simulation sind diese Prozessdefinitionen jedoch noch nicht ausreichend. Die zusätzlich notwendigen Konzepte zur konkreten Verknüpfung werden im folgenden beschrieben.

5.2. Verknüpfung der BPMN-Elemente

Durch die Verwendung domänenspezifischer Elemente bei der Modellierung ist lediglich die Assoziation des Prozessschritts mit einem bestimmten Modelltyp gewährleistet. Im folgenden wird beschrieben, wie die Verknüpfung genauer spezifiziert und eine Kopplung zur Simulationszeit realisiert wird. Die Simulationsbibliothek wird um Elemente erweitert, welche die Verknüpfung zu den Teilmodellen realisieren. Zur genaueren Spezifikation der Verknüpfungsbeziehung werden in der Prozessdefinition die entsprechenden BPMN-Modellelemente mit Erweiterungsattributen versehen.

Bei der Erzeugung eines Simulationsmodells werden bei der Generierung von Prozessmodellen diese Informationen im ersten Schritt ignoriert. Für Ereignisse und Aktivitäten werden daher zunächst die herkömmlichen Simulationsobjekte zur Repräsentation der generellen Elemente erzeugt. Diese werden erst im zweiten Schritt, sofern die entsprechenden Teilmodelle in die Simulation einbezogen werden, mit Referenzen auf zusätzliche, domänenspezifische Objekte angereichert.

5.2.1. Statische Verknüpfung von Modellinstanzen

Jedes domänenspezifiische Element erhält ein Attribut, über welches eine Modellinstanz angegeben werden kann, aber nicht muss. Die Modellinstanz muss von dem Typ sein, der zu dem domänenspezifischen Element passt. Dies geschieht technisch über eine ID, die jedes geladene Teilmodell erhält. Eine Objektreferenz ist nicht sinnvoll, da diese nach einer Serialisierung und Deserialisierung der Modelle nicht mehr gültig ist. Auf der Benutzeroberfläche kann der Anwender aus den geladenen Modellen des entsprechenden Typs auswählen, die ID wird automatisch zugeordnet.

Hierdurch wird eine statische Kopplung definiert, die sich zur Simulationszeit nicht verändert. Das entsprechende Element kann daraufhin nur mit der entsprechenden Modellinstanz interagieren.

5.2.2. Dynamische Verknüpfung von Entitäten über Artefakte

In bestimmten Fällen ist eine dynamische Kopplungen gewünscht, die sich zur Laufzeit verändern kann.

So kann bei einer Prozessdefinition erwünscht sein, dass deren Instanzen mit verschiedenen Teilmodellinstanzen interagieren. Auch ist es in der Simulation notwendig, dass eine Verknüpfung nicht nur zu Modellinstanzen, sondern auch zu konkreten Entitäten aus den Modellinstanzen hergestellt wird. Entitäten selbst können jedoch dynamisch zur

Laufzeit erzeugt werden. Vor dem Simulationslauf ist also nicht zwangsläufig bekannt, welche Entitäten zur Laufzeit entstehen werden. Für beide Fälle wird der im folgenden beschriebene Mechanismus zur dynamischen Kopplung verwendet.

Domänenspezifische Artefakte werden über Assoziationskanten mit den entsprechenden Modellelementen verknüpft. Artefakte halten dabei Referenzen auf Entitäten oder Modelle, auch mehrere Referenzen können gehalten werden. Eine Verknüpfung mit dem Pool kann dabei bedeuten, dass sämtliche Elemente in dem Pool mit der entsprechenden Entität assoziiert sind. Die Modellierung der Verknüpfung kann also grafisch über den BPMN-Editor erfolgen.

Führt die Assoziationskante vom Element zum Artefakt, so wird an dieser Stelle in dem Artefakt eine Referenz hinterlegt, auf die später zugegriffen werden kann. Die Kopplung wird zur Laufzeit von dem verknüpften domänenspezifischen Element erzeugt, sobald eine Prozessinstanz das entsprechende Element erreicht.

Führt die Assoziationskante vom Artefakt zum Element, so wird hiermit beschrieben, dass sich ein Element auf die entsprechende Entität oder das entsprechende Modell bezieht, um zum Beispiel zielgerichtet an die betreffende Entität eine Nachricht zu übermitteln.

Es soll möglich sein, in einem Artefakt mehrere Referenzen auf Entitäten zu hinterlegen. Auch kann der Inhalt eines Artefaktes entweder global oder nur in der entsprechenden Prozessinstanz bekannt sein (vgl. Abschnitt 2.3.6). Daher kann der Entwickler einer entsprechenden Erweiterungsbibliothek eigene Implementationen von Artefakten, die eine bestimmte Schnittstelle erfüllen müssen, vornehmen. Diese Schnittstelle erfordert Methoden zum Hinterlegen und zum Abholen von Referenzen.

Dadurch, dass der Entwickler diese Implementation vornimmt, kann er entscheiden, ob Referenzen, die von einem Element abgeholt wurden, anschließend gelöscht oder beibehalten werden. So kann er mit Hilfe von Artefakten die in der Simulation typischen Warteschlangen realisieren und dabei selbst entscheiden, welche Art von Warteschlange verwendet wird (z.B. FIFO oder LIFO). Auch kann er Artefakte bereitstellen, die eine global verfügbare Warteschlange repräsentieren, oder Artefakte, die jeweils nur für eine bestimmte Prozessinstanz gültige Referenzen beinhalten. Das konkrete Verhalten eines Artefaktes muss für den späteren Modellierer sauber dokumentiert werden.

5.2.3. Austausch von Informationen

Wie in Abschnitt 3.4 beschrieben, werden im Kontext einer Prozessinstanz Informationen mit Hilfe von Variablen gehalten. Diese Informationen werden dazu verwendet, an Verzweigungen Bedingungen mit Hilfe einer Skriptsprache auswerten zu können oder die

Häufigkeit der Ausführung von Mehrfachaktivitäten zu beschreiben. Je nach verwendeter Skriptssprache können primitive Datentypen oder auch komplexe Objekte genutzt werden, um die Daten vorzuhalten.

In der BPMN ist vorgesehen, dass Nachrichtenkanten solcherart Daten in eine andere Prozessinstanz transportieren können. Es ist sinnvoll, dass auch Teilmodelle Informationen übermitteln oder empfangen können. Hierfür kann über Attribute von sendenden Ereignissen und Aktivitäten angegeben werden, welche Variablen an das Teilmodell übermittelt werden sollen. Empfangende Ereignisse und Aktivitäten können Informationen aus dem Teilmodell übernehmen und in den entsprechenden Prozesskontext schreiben.

Auf diese Weise können sich Entscheidungen an Verzweigungen auf Daten aus einem Teilmodell beziehen und das Teilmodell kann auf übermittelte Parameter reagieren. In Abschnitt 5.2.4 wird ein weiterer Anwendungsfall zur Nutzung dieser Informationen zur Beschreibung von dynamischen Verknüpfungsbeziehungen erläutert.

5.2.4. Verknüpfung von empfangenden Ereignissen mit Teilmodellen

Empfangende Ereignisse werden durch das jeweils korrespondierende Teilmodell ausgelöst. Bei Erreichen eines solchen Ereignisses wird die Prozessinstanz zunächst passiviert und wartet darauf, durch Auslösen des Ereignisses reaktiviert zu werden. Das Teilmodell muss daher eine Referenz auf das Ereignis kennen, um es auslösen und den Prozess damit weiterlaufen lassen zu können. Dabei sollen jedoch nicht alle Ereignisse dieses Typs in allen Prozessinstanzen ausgelöst werden. Eine Konkretisierung der Verknüpfungsbeziehung ist notwendig.

Konkretisierung der Verknüpfung

Hierfür kann zunächst festgelegt werden, von welcher Modellinstanz das Ereignis überhaupt Nachrichten entgegegen nehmen darf. Der Modellierer hinterlegt statisch die ID der gewünschten Modellinstanz in einem entsprechendem Attribut (vgl. Abschnitt 5.2.1). Wird diese ID nicht gesetzt, so darf das Ereignis durch alle Modellinstanzen des betreffenden Typs ausgelöst werden.

Mit Hilfe der in der Simulationsbibliothek verwendeten Skriptsprache kann der Benutzer zusätzliche Bedingungen hinterlegen, welche die von einem Ereignis übermittelten Informationen auswerten. Das Ereignis wird nur ausgelöst, wenn die Bedingung wahr ist. Dies eignet sich zunächst zur Prüfung, ob eine bestimmte Entität der Sender ist und damit einerseits zur statischen Verknüpfung mit Entitäten, welche bei der Modellerstellung bereits bekannt sind. Es erlaubt andererseits auch zu prüfen, ob die sendende

Entität bestimmte Eigenschaften erfüllt. Da sich die Zustände der Entitäten zur Laufzeit verändern können, kann sich die Menge der Entitäten, von denen empfangen werden kann, dadurch zur Laufzeit ändern. In diesem Fall handelt es sich um eine dynamische Verknüpfung.

Desweiteren können in der Bedingung Informationen ausgewertet werden, die von dem betreffenden Sender übermittelt wurden. So wird eingeschränkt, dass das Ereignis nur durch bestimmte Nachrichteninhalte ausgelöst wird. Da bei der Bedingung auch Variablen aus dem jeweiligen Prozesskontext genutzt werden können, kann die Prüfung der Bedingung für verschiedene Instanzen der gleichen Prozessdefinition unterschiedlich Ergebnisse liefern.

Über Assoziationskanten mit Artefakten kann zusätzlich eine dynamische Verknüpfung modelliert werden. So kann die dynamische Verknüpfung erzeugt werden, in dem eine Referenz auf den Sender im entsprechenden Artefakt hinterlegt wird (vgl. Abschnitt 5.2.2). Eine bestehende Referenz kann aber auch genutzt werden, um einzuschränken, dass nur von dieser Entität gesendete Nachrichten empfangen werden.

Anforderungen an die Implementation

Zur Implementation von domänenspezifischen Ereignistypen, die zur Simulationslaufzeit genutzt werden können, soll eine abstrakte Klasse implementiert werden. Diejenigen Methoden, die Prozesse in der Warteschlange einreihen und passivieren, sind bereits durch die abstrakte Klasse vorgegeben. Eine Methode zum Auslösen des Ereignisses gemäß dem oben beschriebenen Regelwerk wird ebenso vorgegeben, kann jedoch vom Entwickler überschrieben werden.

Die Referenz auf das Ereignis-Objekt wird dem bereits erzeugten, generellen BPMN-Objekt nachträglich bei der Verknüpfung der Modelle bekannt gemacht. Dies entspricht dem Strategie-Pattern aus Gamma (2013). Zusätzlich wird allen betreffenden Teilmodellen eine Referenz auf dieses Objekt übermittelt, so dass diese bei internem Auftreten des Ereignisses, das BPMN-Ereignis auslösen können. Die Verknüpfung von Prozess- und und Teilmodell wird hierdurch hergestellt.

Die Methode zur Reaktivierung von Prozessinstanzen kann vom Entwickler der Erweiterungsbibliothek implementiert werden. Dies ermöglicht die Flexibilität, die Auswahl des zu reaktivierenden Prozesses nach eigenen Regeln zu treffen. So wird die Implementation eines Nachrichtenempfangsereignisses, welches zielgerichtet eine bestimmte Prozessinstanz reaktiviert, ebenso möglich wie die Implementation eines Signalemfpangsereignisses, welches alle wartenden Prozessinstanzen reaktiviert. Auch die Wahl der Art der

Warteschlange für passivierte Prozessinstanzen obliegt dem Programmierer der Erweiterungsbibliothek. So kann er wahlweise FIFO-, LIFO- oder priorisierende Warteschlangen verwenden.

5.2.5. Verknüpfung von sendende Ereignissen mit Teilmodellen

Konkretisierung der Verknüpfung

Auch hier ist sowohl eine statische als auch eine dynamische Verknüpfung möglich. Durch eine statische Verknüpfung wird sendenden Signalereignissen ermöglicht, Informationen an das entsprechende Teilmodell weiterzureichen. Die Auswertung obliegt der entsprechenden Modellinstanz.

Die Realisierung der Verknüpfung von sendenden Nachrichtenereignissen ist durch eine statische Verknüpfung nicht möglich. Wie in Abschnitt 2.3 dargestellt, unterscheiden sich Signalereignisse und Nachrichtenereignisse dadurch, dass Nachrichtenereignisse zielgerichtet mit bestimmten Entitäten kommunizieren. Das Senden einer Information an ein komplettes Teilmodell ist hingegen als nicht zielgerichteter Broadcast und somit als Signal zu verstehen.

Zum Senden von Nachrichten wird die dynamische Verknüpfung mit einem Artefakt benötigt. Das Ereignis sendet also an diejenige Entität, welche durch die im Artefakt hinterlegte Referenz angegeben ist.

Anforderungen an die Implementation

Sendende Ereignisse werden durch Ableitung einer abstrakten Klasse implementiert, wobei diejenige Methode implementiert werden muss, welche die Nachrichten übermittelt.

Bei der Verknüpfung der Modelle wird das Objekt erzeugt und an das bestehende, generelle BPMN-Ereignis angeheftet, so dass die sendende Methode ausgelöst wird, sobald das Element im Prozessfluss erreicht wird (Strategie-Pattern). Falls eine dynamische Verknüpfung assoziiert ist, kann in der sendenden Methode auf die entsprechende Entität zugegriffen werden. Falls keine dynamische Verknüpfung assoziiert ist, jedoch eine eine statische Verknüpfung angegeben wurde, wird die Referenz auf das entsprechende Teilmodell dem Ereignis bekannt gemacht. Falls auch keine statische Verknüpfung angegeben ist, werden alle Referenzen aller Teilmodelle des entsprechenden Typs übergeben, so dass an alle Teilmodellinstanzen ein Signal übermittelt werden kann.

Es liegt in der Verantwortung des Entwicklers der Erweiterungsbibliothek, die Auswirkungen der gesendeten Nachricht auf die entsprechenden Modellinstanzen oder Entitäten zu beschreiben (vgl. Abschnitt 5.3).

5.2.6. Verknüpfung von Aktivitäten mit Teilmodellen

Aktivitäten können durch konkrete Teilmodelle implementiert werden, wodurch sowohl ihre Laufzeit bestimmt, als auch Manipulationen des Teilmodells möglich werden kann.

Aktivitäten werden ausgeführt, sobald sie im Sequenzfluss erreicht werden. Domänenspezifische Aktivitäten können sich auf den Zustand des entsprechenden Teilmodells auswirken. Das Teilmodell hingegen kann die Dauer einer Aktivität bestimmen und Informationen in den Prozesskontext hineingeben.

Konkretisierung der Verknüpfung

Sowohl die statische als auch die dynamische Form der Verknüpfung kann gewählt werden. Mindestens eine davon muss gewählt werden, da sonst nicht definiert wird, an welchem Objekt (Modell oder Entität) Manipulationen vorgenommen werden sollen.

Die Aktivität kann durch eine statische ID an eine konkrete Modellinstanz geknüpft werden. Die Entscheidung, an welchen Entitäten der Zustand manipuliert, oder wodurch die Dauer der Aktivität bestimmt wird, obliegt in diesem Fall komplett der Implementation des Teilmodells.

Die Aktivität kann aber auch über Assoziationskanten mit einem Artefakt verbunden werden und erhält von dem Artefakt somit dynamisch die Referenz auf eine Modellinstanz oder -entität. Hierdurch können verschiedene Prozessinstanzen mit unterschiedlichen Teilmodellen oder -entitäten interagieren.

Auch kann die Aktivität eine Referenz auf eine Entität in demselben oder einem anderen Artefakt hinterlegen.

Anforderungen an die Implementation

Domänenspezifische Aktivitäten werden durch Implementation einer abstrakten Klasse erstellt. Diese kann zunächst vier EventHandler anmelden:

- Start einer Mehrfachaktivität
- Start einer Aktivität (mehrfach ausgelöst bei Mehrfachaktivitäten)

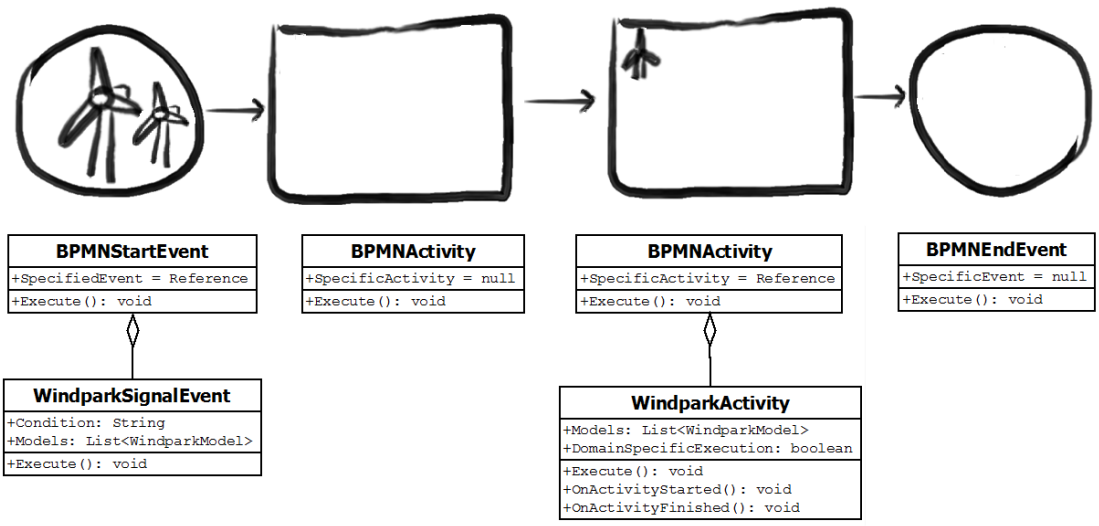


Abbildung 5.2.: Domänenspezifische Elemente einer Prozessdefinition und zugehöriges Objektdiagramm

- Ende einer Aktivität (mehrfach ausgelöst bei Mehrfachaktivitäten)
- Ende einer Mehrfachaktivität

Desweiteren setzt sie ein Flag, welches angibt, ob die Dauer der Aktivität durch das Teilmodell bestimmt werden soll. Für diesen Fall muss zusätzlich eine Methode überschrieben werden, welche den Prozess für die gewünschte Zeit passiviert und anschließend reaktiviert.

Bei der Verknüpfung der Modelle wird dieses Objekt der entsprechenden generellen BPMN-Aktivität bekannt gemacht, so dass die entsprechenden Methoden bei Ausführung angestossen werden können.

Wird eine Aktivität durchgeführt, so werden die entsprechenden EventHandler ausgelöst, in denen entsprechende Zustandsveränderungen am Teilmodell durchgeführt werden können. Desweiteren wird überprüft, ob die Dauer auf die herkömmliche Art durch eine stochastische Verteilung oder durch einen von der Erweiterungsbibliothek bereitgestelltem Algorithmus bestimmt wird.

5.2.7. Integration in Simulationsmodell

Die Prozessdefinition enthält ausschließlich Klassen, welche von der BPMN 2.0-Spezifikation vorgesehen sind. Diese werden mit Erweiterungsattributen versehen, deren Werte durch Zeichenketten ausgedrückt werden. Daher können Prozessmodelle, in denen bereits eine domänenspezifische Verknüpfung integriert wurde, immer noch mit anderen Softwarewerkzeugen bearbeitet werden. Die Aktivitäten und Ereignisse werden dort als unbestimmte bzw. generelle Elemente und Artefakte als Annotationen angezeigt. Sollte der Editor eine Textdarstellung von Erweiterungsattributen unterstützen, so bleibt sogar erkennbar, um welches domänenspezifische Element es sich konkret handelt.

Wird aus der Prozessdefintion ein Simulationsmodell erzeugt, so werden im ersten Schritt die domänenspezfisichen Erweiterungsattribute ignoriert. Es werden zunächst nur diejenigen Simulationselemente erzeugt, welche von der BPMN-Simulationsbibliothek vorgesehen sind (vgl. Abschnitt 3.4), da die Simulationsfabrik für BPMN-Prozesse die jeweilige domänenspezifischen Ausprägung, die zugehörige Klasse und die hierfür notwendigen Verknüpfung nicht kennen kann.

An diese Elemente werden in einem späteren Schritt (vgl. Abschnitt 5.3.7) die jeweils geforderten domänenspezifischen Elemente über das Strategie-Entwurfsmuster (vgl. Gamma (2013)) angeheftet (vgl. Abbildung 5.2). Das heißt, dem generellen BPMN-Element wird eine Referenz auf das zu erzeugende domänenspezifische Element und somit die konkret zu verwendende Strategie übergeben. Zur Laufzeit prüfen die Simulationselemente, ob eine spezialisierte Strategie vorliegt. Ist dies wie bei dem Startereignis in der Abbildung der Fall, wird bei Ereignissen nicht die eigene Methode zur Ausführung angewendet, sondern die Execute-Methode des spezialisierten Elements. Das gleiche Prinzip wird in der BPMN-Bibliothek auch für die herkömmlichen, von der Spezifikation vorgesehenen Ereignistypen (Fehlerereignisse, Signalereignisse etc.) angewendet. Liegt wie bei dem Endereignis in der Abbildung kein spezielles Ereignis vor, so wird die Execute-Methode der eigenen Objektinstanz aufgerufen.

Bei Aktivitäten werden beim Start der Ausführung der Aktivität und bei Beendigung der Aktivität die entsprechenden EventHandler der spezialisierenden Klasse aufgerufen. Zusätzlich wird nach Überprüfen des Flags DomainspecificExecution entweder die ursprüngliche Execute-Methode oder die Execute-Methode der domänenspezifischen Aktivität aufgerufen. Bei der ersten Aktivität in der Abbildung liegt keine domänenspezifische Ausprägung, bei der zweiten die Ausprägung einer Windpark-Aktivität vor.

5.3. Domänenspezifische Teilmodelle

5.3.1. Übersicht über die Erweiterungskomponente

Wie in den Abschnitten 1.2 und 1.3 beschrieben können diejenigen Teilmodelle, die die Systemumgebung eines Prozesses repräsentieren, sehr unterschiedlich strukturiert sein. Um eine domänenspezifische Anpassung des BPMN-Simulators vorzunehmen, müssen verschiedene Erweiterungen für vorgegebene Erweiterungspunkte bereitgestellt werden.

So soll in dieser Arbeit kein generischer Ansatz entwickelt werden, der es ermöglicht unterschiedliche Teilmodelle mit ein und derselben Komponente zu beschreiben. Das wäre für bestimmte Gruppen von Modellen zwar möglich, zum Beispiel mit Hilfe einer Notation wie UML oder DEVS, ist aber nicht Gegenstand dieser Arbeit. Stattdessen soll ermöglicht werden, unterschiedliche, auf die jeweilige Domäne zugeschnittene Editoren einzubinden. Hierfür wird ein Erweiterungspunkt benötigt, der durch die Erweiterungskomponente erfüllt werden muss. Desweiteren werden Erweiterungen benötigt, welche die Modelldefinitionen für einen Simulationslauf parametrieren, ausführbare Simulationsmodelle generieren und letztlich die Verknüpfung der Teilmodelle mit Prozessmodellen ermöglichen. Um dies zu ermöglichen, werden ebenfalls jeweils Erweiterungspunkte bereitgestellt.

Um diese — in den folgenden Abschnitten näher beschriebenen — Erweiterungspunkte nutzen zu können, werden zudem verschiedene Meta-Informationen zur Konfiguration eines Modelltyps benötigt. Diese Meta-Informationen werden einmalig von der Erweiterungskomponente definiert, und dann auf alle Modellinstanzen dieses Typs angewendet.

Abbildung 5.3 zeigt eine Übersicht über die vom Entwickler der Erweiterungen bereitzustellenden Klassen. Diese lassen sich grob in Erweiterungen unterteilen, welche die Benutzeroberfläche betreffen (in der Abbildung das obere Paket `DomainSpecificLibary.UI`), und in Erweiterungen, mit welchen der Benutzer nicht selbst in Berührung kommt, da sie nur intern durch das Rahmenwerk genutzt werden (`DomainSpecificLibary.API`).

Zu den Erweiterungen an der Benutzeroberfläche gehört die Definition domänenspezifischer BPMN-Elemente (vgl. Abschnitt 5.1 und 5.2), sowie die Bereitstellung eines Editors für die domänenspezifischen Teilmodelle (siehe Abschnitt 5.3.4).

Für die interne Verwendung muss dem Rahmenwerk eine Modellkonfiguration bekannt gegeben werden, die verschiedene Meta-Informationen enthält (siehe Abschnitt 5.3). Beim Starten eines Simulationsexperimentes werden vom Rahmenwerk die in der jeweiligen Modellkonfiguration angegebenen Klassen zur Parametrierung der Modelldefinitionen (siehe Abschnitt 5.3.5), für die Generierung von Simulationsmodellen (siehe

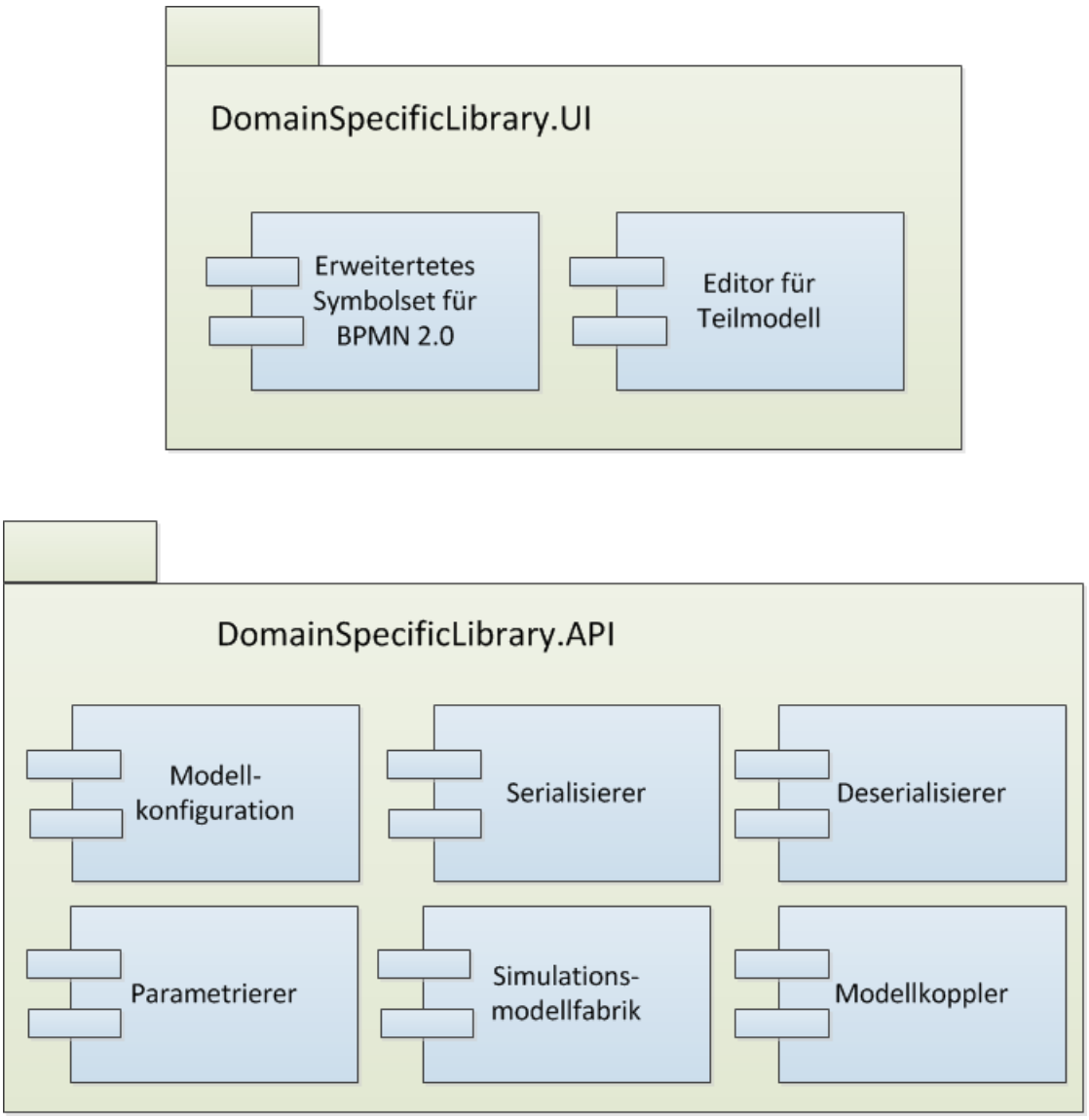


Abbildung 5.3.: Bestandteile der bereitzustellenden Bibliotheken zur Erweiterung der Rahmenanwendung

Abschnitt 5.3.6) und für die Verknüpfung der Modellinstanzen (siehe Abschnitt 5.3.7) gesucht.

Wenn eine solche Erweiterung für eine bestimmte Domäne einmalig entwickelt und bereitgestellt wurde, so ist der BPMN-Simulator für die entsprechende Domäne einsatzbereit. Prozessverantwortliche und Simulationsexperten können die Software nun zur Durchführung von domänenspezifischen Simulationsstudien nutzen.

In den folgenden Abschnitten werden die Anforderungen an diese Erweiterungen detaillierter beschrieben.

5.3.2. Identifikation der Modelle

Zunächst muss die Identifikation eines Modelltyps ermöglicht werden, um die bereitgestellten Meta-Informationen zuordnen zu können. Die Identifikation könnte zum Beispiel über die Dateiendung des Modelltyps geschehen. Das schränkt allerdings ein, dass verschiedene Modelltypen auch unterschiedliche Dateiendungen benötigen würden. Da der Entwickler einer Bibliothek jedoch nicht weiss, welche anderen Bibliotheken in die Rahmenanwendung integriert wurden, können hier leicht Namenskollisionen auftreten. Eine bessere Möglichkeit wäre, jede Modelldatei zu öffnen und deren Inhalt zu identifizieren. Sofern eingeschränkt wird, dass alle Modelle in einem XML-Format vorliegen, wäre eine Identifikation über den Namen des äußeren XML-Elementes sinnvoll.

Um auch Modelldefinitionen in einem anderen Format (z.B. CSV, YML oder gar binär codiert) zu ermöglichen, müsste zunächst das verwendete Format erkannt werden. Hierfür muss von der entsprechenden Komponente jeweils eine Klasse bereit gestellt werden. Diese prüft eine Modelldatei darauf, ob es sich um den in der Erweiterung beschriebenen Modelltyp handelt. Sollte eine Erweiterung mehrere Modelltypen bereitstellen, so muss für jeden Modelltyp eine identifizierende Klasse bereit gestellt werden. Diese Klasse muss eine Schnittstelle mit einer Methode implementieren, welche als Parameter den Pfad zu einer Modelldatei entgegennimmt und als Rückgabe einen Wahrheitswert liefert, der angibt, ob die Datei den entsprechenden Modelltyp beinhaltet.

5.3.3. Serialisierung und Deserialisierung

Ist der Modelltyp bestimmt, so kann die Modelldatei mit einer geeigneten Klasse deserialisiert werden. Auch dieser *Deserialisier* muss eine vorgegebene Schnittselle erfüllen, so dass er eine Methode implementiert, welche aus einem übergebenen Pfad eine Datei lädt und ein Objektmodell erzeugt. Das erzeugte Objektmodell kann beliebig viele Objektinstanzen enthalten, benötigt aber einen Wurzelknoten, von dem ausgehend alle Objekte des Modells über kaskadierende Referenzen erreichbar sind. Die einzelnen

Objektinstanzen können zum Beispiel in dem Modell enthaltene Entitäten beinhalten, während der Wurzelknoten das Modell selbst repräsentiert. Dieser Wurzelknoten ist der Rückgabewert der Methode. Dessen Klassentyp muss ebenfalls in der Konfiguration des Modelltyps angegeben werden, um zu bereits geladenden Modellen ebenfalls die entsprechende Konfiguration zuordnen zu können.

Außerdem wird eine Klasse *Serialisier* benötigt, welche ein im Arbeitsspeicher befindliche Modelldefinition persistiert und in einer Datei im Dateisystem ablegt.

5.3.4. Editorkomponente

Für die nun geladene Modelldefinition muss ein *Editor* bereitgestellt werden, mit dessen Hilfe das Modell angezeigt und manipuliert werden kann. Um solche beliebigen Editor-Komponenten einbinden zu können, wird ein Erweiterungspunkt für Modelleditoren benötigt. Die jeweilige Erweiterung muss — wie alle anderen in diesem Abschnitt beschriebenen Erweiterungen — über die Modellkonfiguration mit dem zugehörigen Modelltyp verknüpft werden, so dass die Software zu verschiedenen Modellen den geeigneten Editor laden kann.

Es kann sich um eine bereits existierende Komponente, die in der Domäne genutzt wird, oder um speziell für die Simulationsanwendung zugeschnittene Modelleditoren handeln. In jedem Fall ist es ratsam, hier eine Benutzeroberfläche zur Verfügung zu stellen, mit welcher die Anwender aus der jeweiligen Domäne leicht umgehen können.

In einigen Fällen bietet es sich an, bestehende Softwarekomponenten wiederzuverwenden. So könnten räumliche Daten zum Beispiel mit Hilfe eines Geoinformationssystems bearbeitet werden. Für bereits existierende Komponente wird eine Wrapper-Klasse benötigt, welche die Komponente kapselt und dabei die vom Rahmenwerk geforderte Schnittstelle erfüllt.

Wird eine neue Editorkomponente für den Anwendungsfall entwickelt, so sind je nach Anwendungsfall grafische Editoren, formularbasierte oder textbasierte Editoren geeignet. Der Modellierer soll hier nicht die Struktur des jeweiligen Domänenmodells beschreiben, da diese durch die Domäne fest vorgegeben ist. Er soll stattdessen die benötigten Parameter und die genaue Ausprägung seiner Modellinstanz definieren. Auch der Import von bestehenden Datensätzen aus betrieblichen Datenbanken ist unter Umständen möglich und ratsam. Gegenenfalls können vom Rahmenwerk spezialisierbare Editorkomponenten vorgegeben werden, welche für den entsprechenden Anwendungsfall angepasst werden, zum Beispiel graphbasierte Editoren für Modelle mit einer Graphenstruktur.

5.3.5. Parametrierung von Modellen

Für die Durchführung von Simulationsstudien ist es in der Regel erforderlich, verschiedene Parameterkonfigurationen experimentell zu untersuchen (vgl. Abschnitt 3.1). Die hierfür notwendige Definition von zu untersuchenden Parameterräumen kann vom jeweiligen Editor ermöglicht werden. Aus den für die verschiedenen Modelle definierten Parameterräumen ergibt sich ein Kreuzprodukt, dessen Elemente jeweils eine Konkatenation von Parametern beinhalten. Dadurch ergibt sich eine bestimmte Mindestanzahl von Simulationsläufen. Aufgabe der Rahmenanwendung ist es, die erforderliche Anzahl von Simulationsläufen zu errechnen, wofür allerdings außer der Größe des Parameterraums auch noch andere Faktoren, wie eine gewünschte Anzahl von Läufen pro Parameterkombination, hinzugezogen werden können. Für jeden Simulationslauf wird ein eigens parametriertes Teilmodell benötigt.

Dafür muss eine weitere Klasse *Parametrierer* bereitgestellt werden, welche eine Modelldefinition kopiert und mit den jeweils vorgesehenen Parametern versieht. Der *Parametrierer* muss also wissen, welche Werte in der Modelldefinition überhaupt Parameter darstellen. Er muss überprüfen können, ob hierfür vom Modellierer auch ein Parameterraum vorgesehen wurde und diese Werte der Rahmenanwendung zurückmelden, so dass diese die entsprechende Anzahl von Simulationsläufen generieren kann.

Eine weitere Methode des *Parametrierers* nimmt eine Modelldefinition und von der Rahmenanwendung ausgewählte Parameter entgegen und ersetzt die Parameterräume in der Modelldefinition durch die konkreten, ausgewählten Parameter. So kann die Rahmenanwendung verschiedene Algorithmen vorsehen, mit denen sie die im nächsten Lauf zu verwendenden Parameter bestimmt. Die Erweiterung der Rahmenanwendung um Optimierungsstrategien (vgl. Czogalla et al. (2006)) wird durch dieses Konzept ermöglicht, im Rahmen dieser Arbeit aber nicht weiter verfolgt.

5.3.6. Generierung von Simulationsmodellen

Aus der parametrierten Modelldefinition muss ein Simulationsmodell generiert werden, welches als Teilmodell innerhalb des Simulationsexperimentes verwendet werden kann. Dessen Bestandteile müssen mit den entsprechenden Elementen der Prozessdefinitionen verknüpft werden (vgl. Abschnitt 5.3.7). In der Zielsetzung der Arbeit (vgl. Abschnitt 1.3) wird davon ausgegangen, dass alle generierten Simulationsmodelle sich einen Simulationskern teilen und somit Zugriff auf eine gemeinsame Ereignisliste haben.

Im Rahmen dieser Arbeit wird die Simulationsbibliothek DESMO-J (vgl. Abschnitt 3.3.1) verwendet, da diese die Implementation von beliebigen zeitdiskreten Modellen ermöglicht und in diversen Projekten die Anwendbarkeit in verschiedenen Domänen

unter Beweis gestellt hat. Basierend auf DESMO-J wurde vom Autor eine BPMN-Bibliothek entwickelt (vgl. Abschnitt 3.4), welche ihm Rahmen dieser Arbeit um Schnittstellen erweitert wird, die die Kopplung mit den Teilmodellen ermöglichen. Durch die Möglichkeit der Verwendung verschiedener Simulationsparadigmen innerhalb eines Modells, wie ereignisorientierter, prozessorienter oder agentenbasierter Simulation, wird es zudem vereinfacht, unterschiedliche Arten von Teilmodellen innerhalb eines Simulationsexperimentes zusammenzufassen. Für den in Abschnitt 6 verwendeten Prototypen muss dementsprechend ein DESMO-J-Modell aus der Modelldefinition erzeugt werden.

Unabhängig von der Wahl des Simulationskerns wird, um die Simulation verschiedener Modelldefinition auf einem Simulationskern zu ermöglichen, eine Klasse *Simulationsfabrik* benötigt, welche aus der Modelldefinition ein zum Simulationskern passendes, ausführbares Simulationsmodell erzeugt. Die *Simulationsfabrik* muss für jeden Modelltyp bereit gestellt werden. Sie implementiert die vom Simulationskern vorgesehenen Klassen, wie zum Beispiel Ereignisse, Entitäten oder Prozesse. Ereignisse und Entitäten können wie vorab beschrieben in Abschnitt (vgl. 5.2) genutzt werden, um die Verknüpfung zu Prozessdefinitionen herzustellen (vgl. Abschnitt 5.3.7).

Außerdem müssen die gewünschten Statistikklassen eingebunden werden. Diese sind zum einen notwendig, um das Verhalten des Teilmodells nachvollziehen und das Gesamtmodell verifizieren zu können. Unter Umständen sind durch das Teilmodell generierte Kennzahlen Teil des Untersuchungsgegenstandes. Hier können spezielle Reportkomponenten bereitgestellt und eingebunden werden. DESMO-J fasst die Statistiken einzelner Teilmodelle in einem Ergebnisbericht zusammen.

Die *Simulationsfabrik* muss eine vorgegebene Schnittstelle erfüllen und im Rahmen der Modellkonfiguration dem Rahmenwerk bekannt gemacht werden. Sie nimmt eine Modelldefinition und ein vom Experimentplaner erzeugtes Hauptmodell entgegen, erzeugt ein das Teilmodell repräsentierendes Submodell und fügt die Referenz hierauf in das Hauptmodell ein. Anforderung an das erzeugte Simulationsmodell ist dabei, dass es die gleichen Zeiteinheiten verwendet wie die BPMN-Bibliothek, damit es zu keiner Fehlerinterpretation der Einträge auf der Ereignisliste kommt.

5.3.7. Verknüpfung der Simulationsmodelle

Nachdem für jeden BPMN-Prozess und für jedes domänenspezifische Teilmodell ein Simulationsmodell erzeugt wurde, können diese Simulationsmodelle miteinander verknüpft werden. Für jeden Modelltyp wird dafür eine Klasse *Modellkoppler* bereitgestellt, welche eine vorgegebene Schnittstelle implementiert.

Für jeden erweiterbaren BPMN-Elementtyp ist eine Methode vorgesehen, welche das jeweilige domänenspezifische BPMN-Element erzeugt und die notwendigen Referenzen

setzt. Bei sendenden Ereignissen, Aktivitäten und Artefakten müssen an dem BPMN-Element Referenzen auf das Teilmodell, bei empfangenden Ereignisse an den Teilmodellen Referenzen auf das BPMN-Element hinterlegt werden.

Das Rahmenwerk durchsucht dafür sämtliche BPMN-Prozessdefinitionen nach Elementen, die durch den entsprechenden Modelltyp spezifiziert werden. Für jedes gefundene Element wird die entsprechende Methode des *Modellkopplers* aufgerufen. Als Parameter werden das Simulationsobjekt, welches Bestandteil der DESMO-J-BPMN-Bibliothek ist und noch keine domänenspezifische Ausprägung besitzt, und das logische Element aus der Prozessdefiniton, welches ein Element aus dem Klassenmodell der BPMN-Spezifikation ist, übergeben. Das logische Element ist notwendig, denn es enthält die vom Benutzer angegebenen Erweiterungsattribute, wie den Typ des geforderten Domänenelements, ggf. die statische Verknüpfung zu einer Modellinstanz oder eine in einer Skriptsprache formulierte Bedingung.

Die Methode erzeugt eine Simulationsinstanz des entsprechenden domänenspezifischen BPMN-Elements (vgl. Abschnitt 5.2), dessen Referenz an das bestehende, noch domänenlose BPMN-Element übergeben wird (vgl. Abbildung 5.2 in Abschnitt 5.2.7). Somit können während eines Simulationslaufs jeweils die entsprechenden Methoden oder Event-Handler des domänenspezifischen Objektes aufgerufen werden, wenn der Prozessfluss dieses Element erreicht.

Aufgabe des Modellkopplers ist es außerdem, die in Abschnitt 5.2 beschriebenen Einschränkungen zur Verknüpfung vorzunehmen. Die sendenen Ereignisse, die Aktivitäten und die Artefakte benötigen Referenzen auf Teilmodelle, mit denen sie verknüpft werden sollen. Dies können entweder alle Teilmodelle des entsprechenden Typs sein, oder nur diejenigen deren statische ID als Erweiterungsattribut im logischen Element hinterlegt wurden. Wenn ein empfangendes Ereigniss erstellt wurde, so muss dessen Referenz den entsprechenden Teilmodellen bekannt gemacht werden. Alle weiteren Erweiterungsattribute, zum Beispiel die in einer Skriptsprache hinterlegten Bedingungen, werden an das entsprechende BPMN-Element übergeben.

Aufbauend auf den in den vorangegangenen Abschnitten beschriebenen Erweiterungspunkten, wird durch diesen Schritt konzeptuell die Zielsetzung aus Abschnitt 1.3 erreicht: Es liegen nun mit Prozessmodellen verknüpfte, domänenspezifische Teilmodelle vor, die in einem gemeinsamen Simulationslauf miteinander interagieren werden.

Um auch die Verknüpfung zwischen verschiedenen domänenspezifischen Teilmodellen zu ermöglichen, gibt ein überschreibbares Property des *Modellkopplers* an, welche Domänenmodelle zusätzlich benötigt werden. Die Default-Einstellung dieses Propertys ist, dass nur BPMN-Modelle angefordert werden. Sollte diese Liste noch weitere Typen von zuätzlichen Domänenmodellen enthalten, so wird zusätzlich eine Methode aufgerufen, welche eine Liste mit sämtlichen Instanzen dieser Domänentypen als Parameter erhält.

Dieses Vorgehen setzt voraus, dass der Entwickler die entsprechenden anderen Domänentypen und deren Klassennamen kennt, und überlasst zudem das Beschreiben und Auswerten sämtlicher notwendiger Verknüpfungen ebenfalls dem Entwickler. Es ist nicht Gegenstand dieser Arbeit, geordnete Konzepte zur Verknüpfung von verschiedenen Domänenmodellen vorzugeben. Dieser Punkt wird jedoch in Abschnitt 8.2 thematisiert.

5.3.8. Durchführung eines Simulationsexperimentes

Abbildung 5.4 zeigt eine zeitliche Übersicht über die Verwendung der in den Abschnitten 5.3.5 bis 5.3.7 vorgestellten Konzepte.

Der Anwender startet ein Simulationsexperiment über eine dafür vorgesehene Benutzeroberfläche, auf der die benötigten Experimentparameter eingestellt werden (vgl. Czogalla et al. (2006)). Hierzu gehören in jedem Fall der Start- und Endzeitpunkt für einen Simulationslauf, die Anzahl der Läufe pro Parameterkombination und der Startwert für die stochastischen Verteilungen (Seed). Weitere Experimentparameter sind abhängig von dem verwendeten Simulationskern denkbar, zum Beispiel verschiedene Ausgabekanäle für die Ergebnisberichte.

Nachdem der Anwender die Experimentplanung abgeschlossen hat, wird ein Experimentplan generiert. Hierfür muss das Rahmenwerk die Anzahl der vorliegenden Parameterkombinationen von den Parametrierern der einzelnen Teilmodelle erfragen. Anschließend muss für jeden Simulationslauf eine entsprechend parametrierte Modelldefinition erzeugt werden. Die hierfür notwendige Erweiterung `Parametrierer` wird vom oberen grüner Kreis repräsentiert (vgl. Abschnitt 5.3.5)).

Aus den parametrierten Modelldefinitionen werden ausführbare Simulationsmodelle generiert. Diese Simulationsmodelle müssen kompatibel mit dem verwendeten Simulationskern sein und in ein vom Rahmenwerk bereitgestelltes Hauptmodell als Submodell eingefügt werden. Für BPMN-Prozesse bringt das Rahmenwerk die `Simualionsfabrik` bereits mit, jedoch werden hier noch Elemente ohne domänenspezifische Ausprägung erzeugt. Zur Generierung der Simulationsmodelle der domänenspezifischen Teilmodelle ist eine weitere Erweiterung notwendig, repräsentiert durch den mittleren grünen Kreis (vgl. Abschnitt 5.3.6).

Im nächsten Schritt werden sämtliche Teilmodelle miteinander verbunden. Die domänenspezifischen BPMN-Elemente müssen erzeugt und mit den Teilmodellen durch Bekanntmachung der Objektreferenzen verknüpft werden. Diese Erweiterung wird durch den unteren grünen Kreis repräsentiert (vgl. Aschnitt 5.3.7).

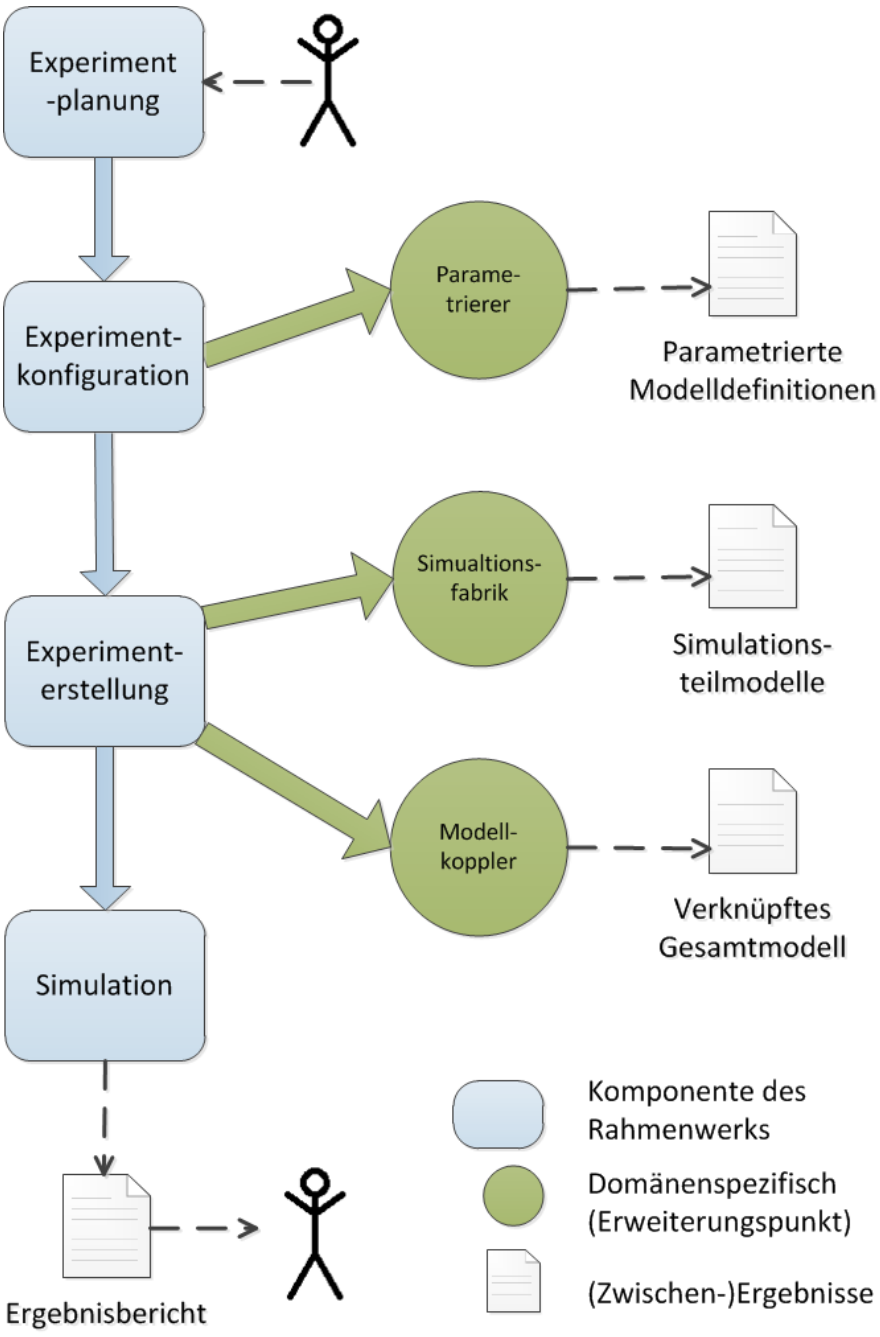


Abbildung 5.4.: Zeitliche Abfolge der Vorbereitung eines Simulationsexperiments

Das Experiment kann anschließend durchgeführt und der Ergebnisbericht im Anschluss vom Anwender ausgewertet werden. Falls die Experimentplanungskomponente dies unterstützt, können die Eregbnisreports der einzelnen Läufe aggregiert und zu einem Simulationsreport zusammengefasst werden.

5.4. Bereitstellung einer Erweiterungskomponente in der Praxis

Zur Durchführung von domänenspezifischen Simulationsstudien sind Arbeitsschritte notwendig, die von Experten in verschiedenen Bereichen durchgeführt werden müssen. Abbildung 5.5 zeigt die benötigten Rollen, die zu vollziehenden Aktivitäten und die jeweils dabei entstehenden Artefakte, die in den folgenden beiden Abschnitten beschrieben werden.

5.4.1. Beteiligte Rollen

In Abschnitt 2.2.4 wurde beschrieben, dass zur Durchführung von BPM-Projekten verschiedene Rollen benötigt werden. Zwei dieser Rollen finden sich auch bei der Durchführung von Simulationsvorhaben wieder; eine weitere Rolle wird nun benötigt.

Ein Prozessverantwortlicher bringt die Domäne betreffendes Fachwissen mit, kennt die zugrundeliegenden Prozesse und hat die Fähigkeit, diese Prozesse mit Hilfe der BPMN 2.0 zu beschreiben. Er ist der maßgebliche Projektleiter und wird die Teilergebnisse der anderen Beteiligten absegnen.

Ein Softwareentwickler beherrscht die komponentenbasierte Softwareentwicklung und kennt die softwaretechnischen Konzepte, die in dieser Arbeit entwickelt wurden. Er ist dafür verantwortlich, die Simulationssoftware durch Bereitstellung einer entsprechenden Erweiterung für das vorliegende Projekt anzupassen. Sofern die Organisation Process Developer (vgl. Abschitt 2.2.4) beschäftigen, kann diese Rolle von diesen übernommen werden, sofern die Kenntnisse der Softwareentwicklung ausreichend sind.

Ein Simulationsexperte hat Erfahrung in der korrekten Durchführung von Simulationsstudien, beherrscht stochastisches Fachwissen und kann Simulationsergebnisse interpretieren und aufbereiten. Er berät den Prozessverantwortlichen und den Softwareentwickler, führt die eigentliche Simulationsstudie durch und präsentiert dem Prozessverantwortlichen die Ergebnisse.

5.4.2. Notwendige Arbeitsschritte

Die Arbeitsschritte in der Abbildung 5.5 werden durch Pfeile repräsentiert. Bis Schritt 5 ist den Pfeilen entgegen, anschließend mit dem Uhrzeigersinn. Die verschiedenen Pfeiltypen repräsentieren Kommunikationsbeziehungen, die Erzeugung von Artefakten und die Nutzung von Artefakten.

Am Anfang einer Simulationsstudie steht grundsätzlich Entwicklung einer exakten Fragestellung. Hierfür wird eine genaue Beschreibung der Problemstellung benötigt, welche vom Prozessverantworlichen an den Simulationsexperten kommuniziert wird. Beide entwickeln dabei kooperativ die Fragestellung (Schritt 1). Der Prozessverantwortliche kennt die Problemfelder aus seiner Domäne und liefert die Idee, welche Aspekte einer Untersuchung mit Hilfe der Simulation unterzogen werden sollen. Der Simulationsexperte kennt die Möglichkeiten und Grenzen der Simulation, und steht bei der Entwicklung der Fragestellung somit beratend zur Seite. Der Prozessverantwortliche ist als Auftraggeber zu verstehen, der Simulationsexperte als Auftragnehmer.

Im nächsten Schritt muss der Simulationsexperte dem Softwareentwickler eine konkrete Liste von Anforderungen für die domänenspezifische Komponente übergeben (Schritt 2). Diese enthält eine Beschreibung der Domäne inklusive der sich aus dem Fachgebiet der Simulationstechnik ergebenden Anforderungen an das Teilmodell, sowie die gewünschten Kopplungsmöglichkeiten an die BPMN-Prozesse. Mit diesem Wissen kann der Softwareentwickler eine geeignete Benutzeroberfläche zur Definition der Struktur und der Parameter der Teilmodelle bereitstellen. Er muss fachliche Symbole mit entsprechendem Wiedererkennungswert innerhalb der Branche zur Verwendung innerhalb von BPMN-Prozessen bereitstellen. Er muss zudem diejenigen Komponenten entwickeln, welche die Modelldefinition in ein ausführbares, zeitdiskretes und verknüpfbares Simulationsmodell überführen (Schritt 3). Das Ergebnis ist eine Erweiterungsbibliothek, welche das Rahmenwerk für die vorliegende Domäne anpasst.

Diese Komponenten nutzt der Prozessverantwortliche (Schritt 4), um die konkrete Ausprägung der Teilmodelle zu spezifizieren. Er modelliert mit dem BPMN-Editor Prozessdefinitionen und nutzt hierbei das für den Anwendungsfall erweiterte Symbolset (Schritt 5). Alternativ kann er bestehende Prozessdefinitionen importieren und entsprechende Elemente durch die nun zur Verfügung stehenden speziellen Elemente ersetzen. Falls der Prozessverantwortliche die Details der notwendigen Prozesse nicht kennt, muss er diese mit den Prozessbeteiligten aus seiner Organisation besprechen. Da die BPMN zu einem Großteil intuitiv verständliche Symbole enthält und diese im Zuge der Erweiterung nur um ebenfalls leicht verständliche Symbole erweitert werden sollen, ist die Validierung der Prozessdefinitonen mit dem entsprechenden Fachpersonal möglich, auch wenn dies kein Expertenwissen über Geschäftsprozessmodellierung oder Simulation besitzen.

Das Ergebnis dieser Schritte sind Prozessdefinitionen und Definitionen der benötigten Teilmodelle.

Diese Modelldefinitionen erweitert der Simulationsexperte nun um die in der Simulation benötigten Parameter (Schritt 6). Es müssen sowohl die fixen, in allen Simulationsläufen wiederzuverwendenden Parameter angegeben werden, als auch die möglichen Parameterräume abgesteckt werden, in deren Rahmen Experimente durchgeführt werden sollen. Dies kann sowohl Parameter der Prozessdefinitionen, als auch Parameter der Teilmodelle betreffen und geschieht in enger Absprache mit dem Prozessverantwortlichen. Im Regelfall ist dieser Punkt der Datenerhebung der kritischste Abschnitt bei der Durchführung von Simulationsstudien. Liegen nicht alle benötigten Parameter vor, kann mit Hilfe von Sensibilitätstests überprüft werden, ob eine ungefähre Schätzung der Parameter ausreichend ist.

Nach der Durchführung erster Simulationsläufe (Schritt 7) muss die Plausibilität des dynamischen Laufzeitverhaltens und somit die Validität der Modelle und Parameter gemeinsam mit dem Prozessverantwortlichen überprüft werden (Schritt 8). Gegebenenfalls müssen die Prozessdefinitionen, die Teilmodelle oder die verwendeten Parameter in Zuge dieses Schrittes korrigiert werden (Wiederholung der Schritte 6 bis 8). Der Prozessverantwortliche und der Simulationsexperte durchlaufen an dieser Stelle gemeinsam den Modellbildungszyklus, bis im Ergebnis plausible, parametrierte Modelle zur Durchführung der eigentlichen Simulationsstudie zur Verfügung stehen.

Mit Hilfe dieser Modelle führt der Simulationsexperte nun das eigentliche Simulationsexperiment durch (Wiederholung Schritt 8). Hierfür muss er feststellen, wie viele Läufe zur Errechnung von verlässlichen Werten benötigt werden, wie lang diese Läufe sein müssen und ab welchem Punkt die Anlaufphase einer Simulationslaufes erreicht ist (vgl. Kapitel 3.1). Die hierbei entstehenden Ergebnisse müssen nun aufbereitet werden und dem Prozessverantwortlichen zur Verfügung gestellt werden, damit dieser die entsprechenden Rückschlüsse gemäß der anfangs erfolgten Fragestellung ziehen kann (Schritt 8).

5.4.3. Durchführung von Folgestudien

Gegebenenfalls wird der Prozessverantwortliche seine Prozesse nun an die neuen Erkenntnisse anpassen (vgl. Business Process Improvement). In diesem Fall wird er nun die ursprüngliche Fragestellung unter der alternativen Prozesskonfiguration erneut untersuchen lassen (Wiederholung Schritte 5-8). Da die domänenspezifische Softwarekomponente sowie die bereits parametrierten Teilmodelle hierfür wiederverwendet werden können, ist der Aufwand zur Durchführung von Folgestudien erheblich geringer, sobald die domänenspezifischen Komponenten einmal bereitstehen.

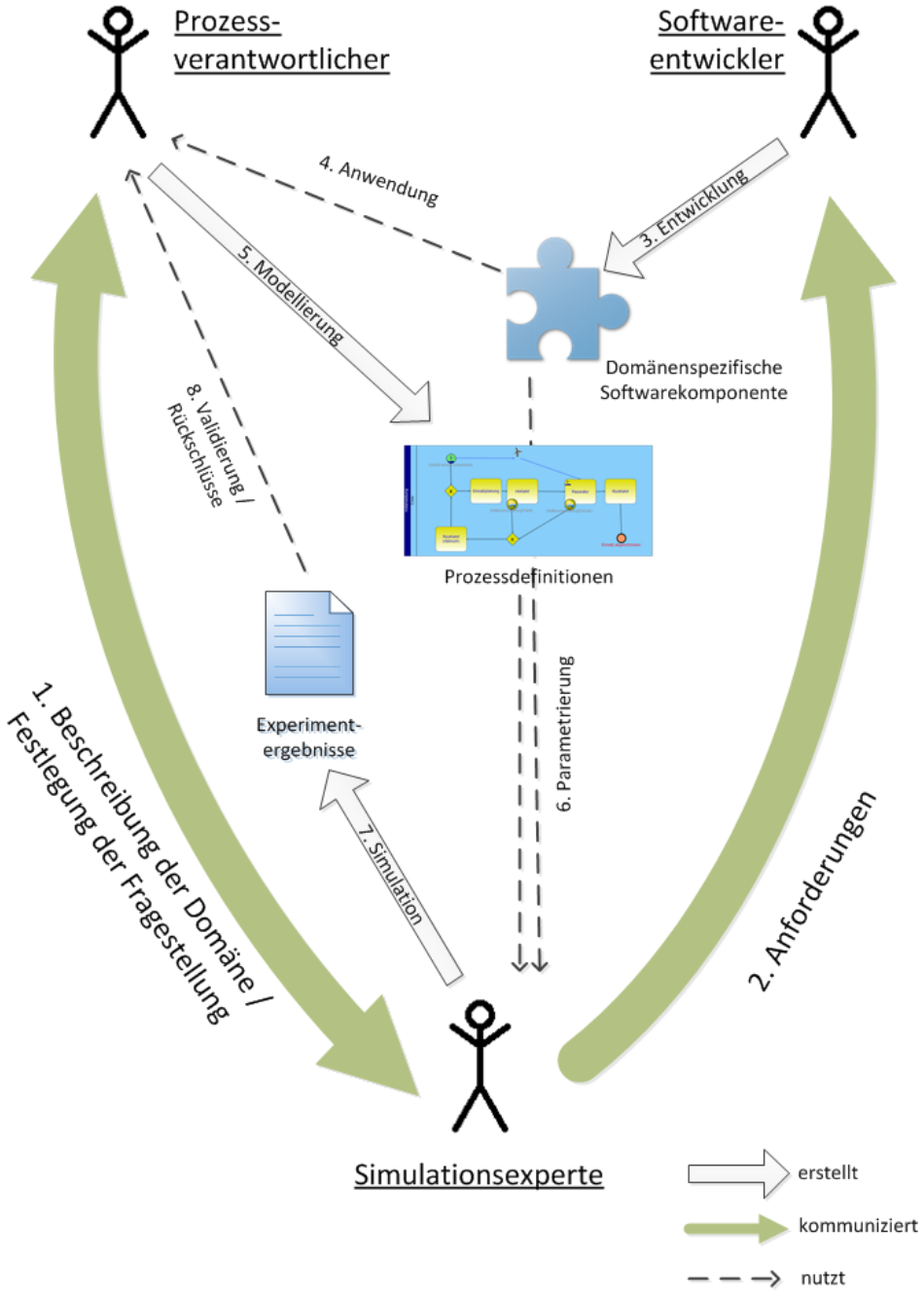


Abbildung 5.5.: Rollen und Aktivitäten zur Durchführung von domänenspezifischen Simulationsstudien

Sollten die Teilmodelle die dafür erforderlichen Aspekte bereits beinhalten, können auch andere Fragestellung mit entsprechend geringerem Aufwand untersucht werden (Wiederholung Schritt 1, sowie 5-8). Ob die domänenspezifische Komponenten die Anforderungen an die neue Fragestellung vollständig erfüllen, sollte vom Simulationsexperten vorab beurteilt werden.

6. Implementation eines komponentenbasierten Prototyps

In diesem Kapitel wird die prototypische Umsetzung der Konzepte aus Kapitel 5 beschrieben. Zu den relevanten Komponenten werden exemplarisch die Implementationen erläutert, die Schnittstellen beschrieben und Klassendiagramme gezeigt. Die Umsetzung geschieht unter .NET mit C-Sharp, die Vorgehensweise ist jedoch übertragbar auf andere Plattformen, welche Konzepte zur komponentenorientierten Entwicklung anbieten. Der Prototyp wird genutzt, um die Umsetzbarkeit der Konzepte zu beweisen und anschließend in Kapitel 7 die Anwendbarkeit in der Praxis zu untersuchen.

6.1. Komponentenbasierte Entwicklung

Zur Implementation des Prototypen wurde das Plug-in-Rahmenwerk Empinia verwendet, welches bereits in Abschnitt 4.3.5 vorgestellt wurde. Die in dieser Arbeit verwendeten Konzepte des Rahmenwerks werden nun zum besseren Verständnis kurz erläutert. Für einen umfassenderen Überblick über die Funktionen wird auf die Arbeiten Schnackenbeck et al. (2007) oder Panic et al. (2008) verwiesen.

6.1.1. Dienste und Erweiterungspunkte

Empinia bietet mehrere Konzepte zur Einbindung von Erweiterungskomponenten. Die zwei folgenden werden in der vorliegenden Arbeit verwendet.

Dienste stellen Methoden und Properties bereit, welche von anderen Komponenten verwendet werden können. Ein Dienst muss die von Empinia bereitgestellte Schnittstelle `IService` implementieren und über die in die DLL zu integrierende „bundel.xml“ angemeldet werden. Innerhalb einer Anwendung darf nur eine Klasse diese Schnittstelle implementieren. Diese wird von Empinia als Singleton instanziiert. Der Zugriff auf diesen Singleton wird per Dependency Injection realisiert. Das bedeutet, der Konstruktor der Klasse, welche den Dienst nutzen soll, erhält die Schnittstelle als Konstruktorparameter (Constructor Injection) oder es wird Property vom Typ der Schnittstelle definiert

(Property Injection). Empinia sucht bei der Instanziierung einer solchen Klasse oder bei dem Zugriff auf ein solches Property nach der passenden Service-Instanz. Der Nutzer des Dienstes benötigt keine Kenntnisse darüber, durch welche Erweiterung der Dienst implementiert ist.

Ein Beispiel für einen in Empinia mitgelieferte Dienst ist der `SelectionService`. Bei ihm können selektierte Objekte bekannt gemacht und die aktuell selektierten Objekte erfragt werden. Objekte werden typischerweise durch anwendungsspezifische GUI-Komponenten wie Editoren selektiert, damit andere Komponenten auf diese Selektion reagieren können. Beispielsweise wird von Empinia eine generische GUI-Komponente `PropertyEditor` (Eigenschaftseditor) bereitgestellt, welche die per Reflection-Mechanismen erfassten Eigenschaften der selektierten Objekte zur weiteren Manipulation anzeigen können.

Erweiterungspunkte sind Schnittstellen, welche im Gegensatz zu Diensten von beliebig vielen Komponenten realisiert werden können. Auch hier wird die geforderte Schnittstelle (abgeleitet von `IExtensionPoint`) öffentlich bekannt gemacht. Empinia bringt Erweiterungspunkte für einige Anwendungsfälle mit (vgl. Abschnitte 6.1.2 und 6.1.3); es können jedoch auch eigene Erweiterungspunkte definiert werden, um Komponenten bereitzustellen, welche selbst erweiterbar sind. Von Empinia wird der Dienst `ExtensionRegistry` bereitgestellt, über welchen alle angemeldeten Erweiterungen zu einem Erweiterungspunkt erfragt werden können. Durch Testen der Eigenschaften der angemeldeten Erweiterungen kann eine Komponente diejenige bereitgestellte Erweiterung auswählen, welche für den vorliegenden Anwendungsfall am besten geeignet ist. Ist die ID einer konkreten Erweiterung bekannt, so kann deren von Empinia erzeugte Instanz auch direkt aufgerufen werden.

6.1.2. Kommandos und Aktionen

Zwei von Empinia bereitgestellte Erweiterungspunkte sind `Command` und `Action`.

Ein `Command` stellt genau eine Methode bereit. Typischerweise werden dieser Methode die hierfür benötigten Dienste per Dependency Injection übergeben. Da die Schnittstelle der auszuführenden Methode durch `ICommand` vorgegeben ist, müssen die für den Methodenaufruf benötigten Parameter in einer Liste `IExecutionArguments` übergeben werden. Der Rückgabewert `IExecutionResult` gibt den Erfolg oder Misserfolg der Ausführung zurück. Der Zugriff auf Kommandos geschieht über deren ID. Für den Aufruf des Kommandos muss die ID daher entweder hardcodiert werden, oder kann von einer angemeldeten Erweiterung erfragt werden.

Mit Hilfe einer `Action` kann ein Kommando auf der Benutzeroberfläche zugänglich gemacht werden. Bei der Definition einer solchen Aktion ist die ID des entsprechenden Kommandos und die Position auf der Benutzeroberfläche anzugeben. Dies kann zum

Beispiel eine Position in einer Symbolleiste oder in einem Menü sein. Es ist nicht notwendig, dass die Aktion und das Kommando Bestandteile der gleichen Komponente sind. So können bestehende Kommandos auch in die Benutzeroberfläche von neu zu entwerfenden GUI-Komponenten integriert werden.

6.1.3. Benutzeroberfläche

Empinia bietet zwei Erweiterungspunkte zur Anmeldung von anwendungsspezifischen Benutzeroberflächen. Zugrunde liegt eine MDI-Oberfläche (Multiple Document Interface), in deren Hauptfenster mehrere Unterfenster angezeigt werden können, welche als `View` bezeichnet werden. So können zum Beispiel mehrere Instanzen einer Editorkomponente parallel in verschiedenen `Views` unterschiedliche Modelle anzeigen. Auch können verschiedenartige Anzeigewerkzeuge parallel zueinander auf demselben Model arbeiten. Eine `View` erbt von der Schnittstelle `IView`. Der Zugriff auf eine View geschieht über die ID, welche wie bei Kommandos entweder hardcodiert oder von einer Erweiterung abgefragt werden kann.

Innerhalb des Hauptfenster können die `Views` beliebig angeordnet werden. Zur Entwicklungszeit kann die Position von `Views` mit Hilfe von Perspektiven vordefiniert werden. Diese stellen einen weiteren vordefinierten Erweiterungspunkt dar. Mit einer Perspektive können alle `Views`, zu denen die ID bekannt ist, an relativen oder absoluten Positionen angeordnet werden. Zur Laufzeit kann der Benutzer die Views beliebig neu anordnen

Symbolleisten können am oberen Rand einer View platziert werden und enthalten Knöpfe, mit denen der Benutzer Aktionen auslösen kann. Die Symbolleiste einer View wird über eine ID bekannt gemacht. Andere Komponenten können Aktionen in ihr ablegen und die Symbolleiste einer bestehenden View nachträglich erweitern.

Ein von Empinia bereitgestellter `Command` ist der `ShowViewCommand`. Ihm wird als Parameter die ID einer View übergeben. Er erzeugt eine neue Instanz der `View` und zeigt diese auf der Benutzeroberfläche an. Abbildung 6.1 zeigt das Multiple Document Interface von Empinia mit drei geöffneten `Views`. Auf der linken Seite ist ein Editor für Petri-Netze (Joschko (2008)) und auf der rechten Seite der Eigenschaftseditor zu sehen. Der Eigenschaftseditor zeigt die Eigenschaften des über den `SelectionService` bekannt gemachten, aktuell selektierten Graphknotens an. Der Benutzer kann die Anordnung einer Perspektive zur Laufzeit verändern, in dem er eine `View` mit der Maus greift, und an vordefinierten Ankerpunkten anderer `Views` andockt.

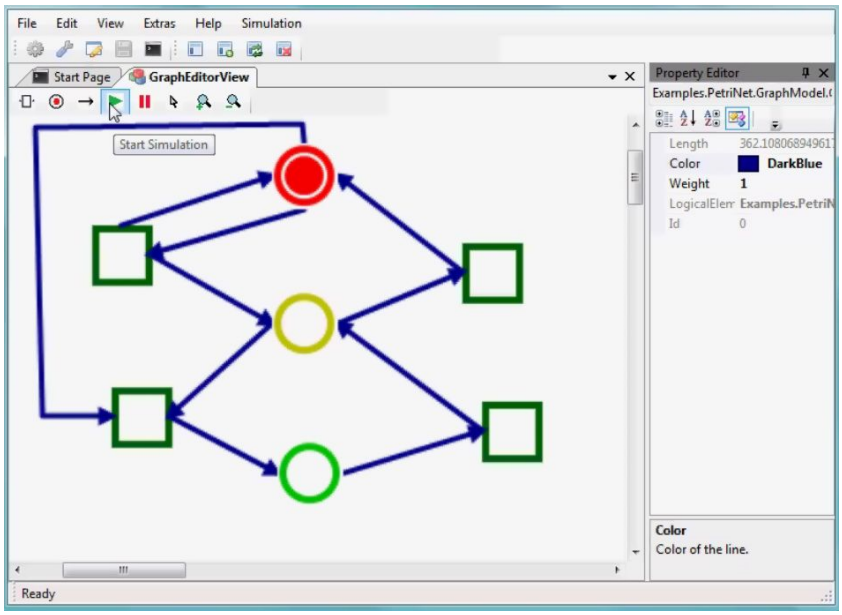


Abbildung 6.1.: Screenshot des Empinia-Rahmenwerkes mit zwei geöffneten Views (aus Joschko (2008))

6.2. Projektmappen

Aufbauend auf den gerade beschriebenen softwaretechnischen Fähigkeiten wurden die Konzepte aus Kapitel 5 umgesetzt. Die folgenden Abschnitte beschreiben diejenigen Punkte, welche im Rahmen dieser Arbeit umgesetzt werden mussten.

6.2.1. Implementation der Projektmappe

Zunächst wird eine Umgebung benötigt, in der die Prozesse und Teilmodelle zur Verfügung gestellt werden. Als Container für alle geladenen Modelle dient die Klasse `Solution`, welche zur besseren Lesbarkeit als Projektmappe bezeichnet wird. Es handelt sich hierbei nicht um eine Komponente im Sinne der komponentenorientierten Entwicklung, sondern lediglich um eine Klasse im herkömmlichen Sinn der objektorientierten Programmierung. Der Zugriff auf diese Klasse wird in Abschnitt 6.2.2 beschrieben. Abbildung 6.2 zeigt als Übersicht ein Klassendiagramm, welches im folgenden erläutert wird.[9]

[9] In der letztendlichen Implementation des Prototypen wurde die Klasse um weitere Methoden erweitert, diese zu erläutern ist für das Verständnis der in dieser Arbeit beschriebenen Konzepte jedoch nicht notwendig

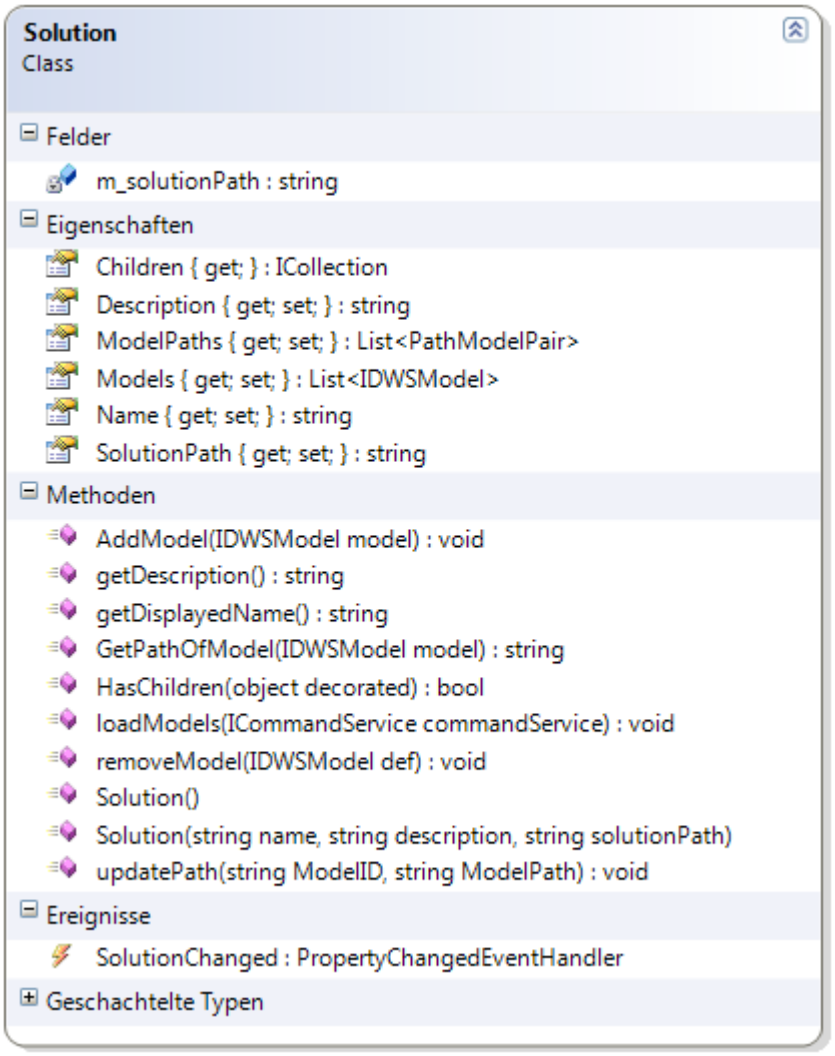


Abbildung 6.2.: Klassendiagramm von `Solution` mit Eigenschaften, Methoden und Ereignissen

Die Projektmappeneigenschaft `Models` enthält Referenzen auf alle enthaltenen Modelldefinitionen. Hierbei wird vorausgesetzt, dass jede Modelldefinition entweder direkt die Schnittstelle `IDWSModel` erfüllt oder (in den meisten Fällen) eine Wrapper-Klasse zur Verfügung steht, die die Schnittstelle erfüllt. `IDWSModel` erfordert lediglich einen Namen, eine ID und die Referenz auf den Wurzelknoten des Models. Die Eigenschaft `ModelPaths` der Projektmappe enthält die Informationen über den Speicherort - also Pfad und Dateinamen zu jeder Modelldefinition. Über die Methoden `AddModel` und `RemoveModel` werden Modelldefinitionen hinzugefügt oder entfernt.

Die Eigenschaften `SolutionPath` und `Name` geben den Speicherort und den Dateinamen der Projektmappe an (zur Persistierung siehe Abschnitt 6.2.4). Die Eigenschaft `Description` enthält eine Kurzbeschreibung, in der der Anwender den Inhalt der Solution beschreiben soll.

Das Ereignis `SolutionChanged` wird ausgelöst, wenn der Projektmappe Modelle hinzugefügt oder aus ihr entfernt werden. Komponenten, die mit Projektmappen arbeiten, können hier einen `EventHandler` anmelden, um auf entsprechende Ereignisse reagieren zu können.

6.2.2. Solution Service

Eine zentrale Komponente der Anwendung ist der Dienst `SolutionService`, über welchen die aktuelle Projektmappe erfragt werden kann. Hierfür wird der Dienst denjenigen Klassen, welche mit der Projektmappe arbeiten müssen, per Dependency Injection übergeben. Die Schnittstelle `ISolutionService` benennt die geforderten Methoden des Dienstes. Abbildung 6.3 zeigt eine Übersicht über die Schnittstelle, im Codebeispiel B.1 im Anhang findet sich der zugehörige Quellcode.

Das Property `CurrentSolution` liefert eine Referenz auf die aktuelle Projektmappe. Im vorliegenden Prototyp kann immer nur eine Projektmappe zur Zeit geladen werden, was für einen Prototyp gemäß der hier beschriebenen Vorgehensweise ausreichend ist.

Das Property `SelectedModel` gibt das aktuell selektierte Modell zurück. Das Property besitzt neben der Get-Methode auch eine Set-Methode, über welche zum Beispiel GUI-Komponenten ein durch den Benutzer selektiertes Modell bekannt machen können. Der Dienst hat zusätzlich Kenntnis über den Empinia-Dienst `SelectionService`, über welchen eigenständig erkannt werden kann, wenn ein Modell selektiert wurde.

Der `SolutionService` hat noch eine weitere wichtige Aufgabe: Er liefert zu gegebenen Modelldefinitionen eine `ModelTypeConfiguration`, welche für die Handhabung heterogener Modelldefinitionen benötigt wird (siehe Abschnitt 6.3). Die Methode `GetConfiguration(Type)` liefert die Konfiguration für einen übergebenen Modelltyp, die Methode

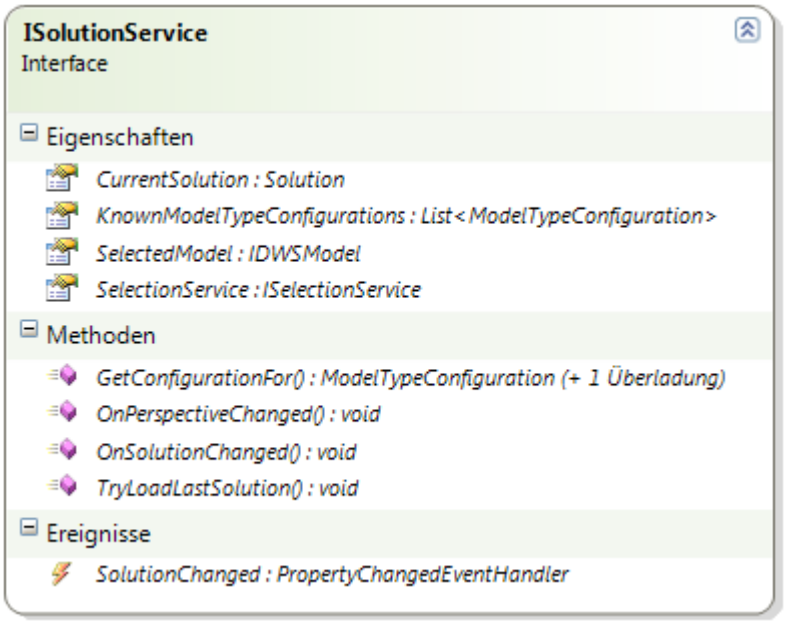


Abbildung 6.3.: Schnittstelle des SolutionServices

`GetConfiguration(IDWSModel)` für ein instanziiertes Modell und das Property `KnownModelTypeConfigurations` liefert eine Liste aller angemeldeten Modellkonfigurationen. Um diese `ModelTypeConfigurations` finden zu können, hat der Dienst Kenntnis über den Empinia-Dienst `ExtensionRegistry`.

6.2.3. Benutzeroberfläche zur Projektmappe

Der Dienst `SolutionService` ermöglich die Implementation einer View `ProjectExplorer`. Diese View zeigt dem Benutzer eine Übersicht über die Modelle der aktuell geladenen Projektmappe. Die Referenz auf den `SolutionService` wird über Constructor Injection übergeben, so dass über dessen Property `CurrentSolution` die in der aktuellen Projektmappe referenzierten Modelle erfragt werden können. Abbildung 6.4 zeigt die View mit einer geladenen Projektmappe. In der Abbildung werden BPMN-Modelle, Windparkmodelle, Wettermodelle und ein Ressourcenmodell angezeigt.

Die in der Projektmappe referenzierten Modelle werden in einer `TreeView` angezeigt. Zu einem Modelltyp kann ein `LabelDecorator` angegeben werden, welcher ein Symbol bereitstellt, um dem Benutzer die Identifikation des Typs zu ermöglichen.

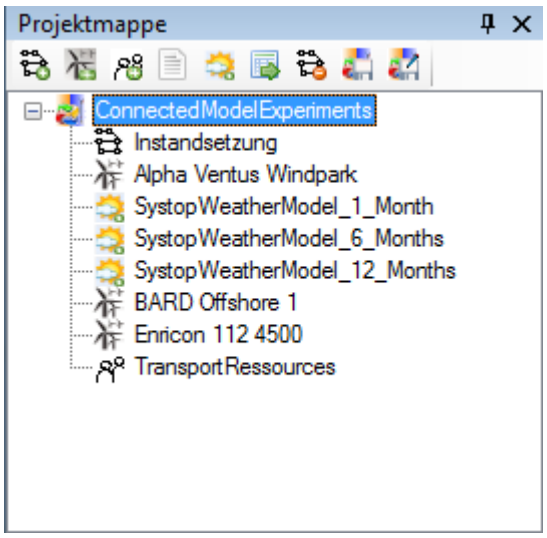


Abbildung 6.4.: Projektmappe mit geladenen Modellen

Der `ProjectExplorer` enthält am oberen Rand eine Symbolleiste, in welcher die Erweiterungskomponenten Aktionen ablegen können. In dem gezeigten Beispiel wurden Aktionen zum Erzeugen neuer Modelldefinitionen platziert. Vorgebenen sind einen Knopf zum Speichern der Projektmappe (siehe Abschnitt 6.2.4), ein Knopf zum Laden von Modelldefinitionen (siehe Abschnitt 6.3.2) sowie ein Knopf zum Entfernen von Modellreferenzen aus der Projektmappe. Ein Mausklick auf ein Modell öffnet dieses mit einem geeigneten Editor (siehe Abschnitt 6.3.2).

Über einen an das Ereignis `SolutionChanged` der Klasse `Solution` angemeldeten `EventHandler` reagiert die Projekmappe auf Änderungen in der Projektmappe, so dass neue Modelle angezeigt oder entfernte Modelle aus der Anzeige entnommen werden.

6.2.4. Projektmappen persistieren

Ein Objekt der Klasse `Solution` wird als XML-Datei persistiert. Hierfür werden die zu persistierenden Properties mit Attributen versehen, welchen den Namen des XML-Elementes angeben. Dies ermöglicht die Serialisierung und Deserialisierung eines Objektes unter Verwendung der Standard-.NET-Serialisierungsmethoden. Da hier nur primitive Datentypen, Listen und Objekte, die wiederum aus primitiven Datentypen bestehen, persistiert werden müssen, muss kein spezieller Serialisierer implementiert werden.

Persistiert werden die Eigenschaften `ModelPaths`, `Name` und `Description`. Die Eigenschaft `Models` wird nicht serialisiert, da sie Objektreferenzen enthält, die erst entstehen,

wenn mit Hilfe der Eigensschaft `ModelPaths` alle Modelle geladen wurden. Auch die Eigenschaft `SolutionPath` wird nicht persistiert, denn ihr Wert ergibt sich aus dem Ort, von dem die `Solution` geladen wurde.

Zu Deserialisierung muss eine spezielle Klasse implementiert werden. Sie lädt die XML-Datei zunächst mit Hilfe des Standard-XML-Deserialisierers und kann dabei diejenigen Eigenschaften mit Werten befüllen, zu denen die entsprechenden XML-Elemente über ein Attribut definiert wurden. Anschließend wird die Eigenschaft `SolutionPath` gesetzt und alle Modelle aus der Liste `ModelPaths` geladen (vgl. Abschnitt 6.3.2). Die dabei entstehenden Referenzen werden in der Liste `Models` abgelegt.

6.3. Konfiguration von Modelltypen

Zu jedem Modelltyp werden Meta-Informationen benötigt, welche den korrekten Umgang mit dem Modells innerhalb der laufende Software ermöglichen. Die Erweiterungsbibliothek, die einen neuen Modelltyp anmeldet, muss diese Informationen bereitstellen. Hierfür wurde der Erweiterungspunkt `ModelTypeConfiguration` entworfen. Die Eigenschaften der geforderten Erweiterungen zeigt Abbildung 6.5 in der Übersicht. Das Code-Beispiel B.2 im Anhang zeigt die hierfür implementierte Klasse. Der Erweiterungspunkt wird am Rahmenwerk über einen Eintrag in eine Datei Bundle.xml angemeldet (vgl. B.3 im Anhang). Eine Erweiterungskomponente muss sich bei der Anmeldung einer `ModelTypeConfiguration` auf die dort angegebene ID beziehen. In Abschnitt 7 wird die beispielhafte Anwendung gezeigt.

6.3.1. Identifikation von Modellen

Sowohl der Benutzer als auch die Rahmenanwendung sollen einer gegebenen Modelldefinition den jeweiligen Modelltyp zuordnen können.

Die Eigenschaften `ModelType` und `ModelDescription` liefern Texte, welcher der Identifikation des Modelltyps durch den Benutzer dient. Dieser Text kann an verschiedenen Stellen auf der Benutzeroberfläche wiedergegeben werden, zum Beispiel beim Laden von Modellen oder bei der Auswahl von Modellen für Simulationsexperimente. Während der `ModelType` einen Kurztext ausgeben soll, eignet sich die `ModelDescription` für eine detailliertere Erläuterung.

Die Eigenschaft `PostFix` benennt die Dateiendung des Modelltyps. Im entwickelten Prototyp werden hierüber Modelldefinitionen bereits vor dem Laden identifiziert, so dass die zugehörige `ModelTypeConfiguration` ausgewählt werden kann. Dies schränkt wie in Abschnitt 5.3 beschrieben ein, dass verschiedene Modelltypen auch unterschiedliche

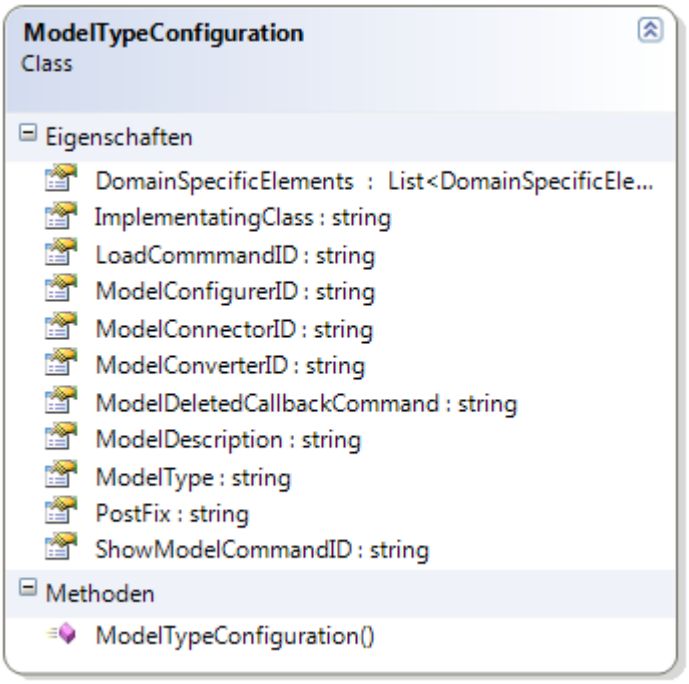


Abbildung 6.5.: Eigenschaften der Klasse ModelTypeConfiguration

Dateiendungen benötigen. Im Konzept wird daher eine Klasse gefordert, welche eine Modelldatei daraufhin überprüft, ob sie den entsprechenden Modelltyp enthält. Dies erfordert für den hier beschriebenen Prototypen einen nicht notwendigerweise zu erbringenden zusätzlichen Implementationsaufwand, da die grundsätzliche Funktionsweise der hier beschriebenen Konzepte auch bewiesen werden kann, wenn in diesem Fall der etwas ungünstigere Weg gewählt wird. Zielsetzung ist es an dieser Stelle lediglich, die grundsätzliche Identifizierbarkeit von Modelltypen aufzuzeigen, um hierauf aufbauend Entscheidungen für den weiteren Umgang mit verschiedenen Modelltypen treffen zu können.

Die Eigenschaft `ImplementingClass` dient der Identifikation von bereits in den Arbeitsspeicher geladenen Modellen, so dass die entsprechende `ModelTypeConfiguration` zugeordnet werden kann. Sie gibt den Klassennamen der Wurzel des Objektbaumes an, der eine Modelldefinition beschreibt. Die Eigenschaft `DomainSpecificElements` gibt eine Liste mit den Typbezeichnung von denjenigen domänenspezfiischen BPMN-Elementen an, welche mit dem Teilmodell verknüpfbar sind.

6.3.2. Notwendige Kommandos

Die Eigenschaft `LoadCommandID` gibt die ID des Kommandos an, welches zur Deserialisierung der jeweiligen Modelldefinition benötigt wird. Das Kommando nimmt als Parameter den Dateinamen und -pfad entgegen. Es lädt die serialisierten Informationen aus einer Datei und instanziiert ein entsprechendes Objektmodell. Das erzeugte Objektmodell benötigt einen Wurzelknoten, von dem ausgehend alle Objekte des Modells über kaskadierende Referenzen erreichbar sind. Die Referenz auf diesen Wurzelknoten wird dem `SolutionService` übergeben, dessen Referenz dem Kommando per Constructor Injection bekannt gemacht wurde.

Die Eigenschaft `ShowModelCommandID` gibt die ID des Kommandos an, welches zur Anzeige von Modellen genutzt wird. Als Parameter wird die Objektreferenz zur geladenen Modelldefinition übergeben. Bei Ausführung erzeugt das Kommando die Instanz einer `View`, welche die übergebene Modelldefinition anzeigt und dem Benutzer Methoden zur Manipulation der Definition verfügbar macht. Per Constructor Injection wird bereits bei Instanziierung des Kommandos der `PageService` übergeben, damit die erzeugte Instanz der `View` in die bestehende Benutzeroberfläche eingebunden werden kann.

Die Eigenschaft `ModelDeledCallbackID` gibt die ID des Kommandos an, welches ausgeführt wird, wenn ein Modell aus der Projektmappe entfernt wird. Es bekommt per Constructor Injection den `SolutionService` übergeben, um ggf. andere Modelle nach zu löschenden Referenzen durchsuchen zu können.

6.4. Domänenspezifische BPMN-Elemente

Eine Erweiterungskomponente zur Integration von Teilmodellen stellt domänenspezifische BPMN-Elemente zur Verfügung. Hierfür musste die Simulationskomponente angepasst werden, es mussten Klassen und Schnittstellen bereitgestellt werden, welche die Implementation solcher Elemente ermöglichen und es musste der grafische BPMN-Editor dahingehend erweitert werden, dass domänenspezifische Elemente in Prozessdefinitionen integriert und im Editor angezeigt werden können.

6.4.1. Erweiterbarkeit der Simulationskomponente

Um die Integration domänenspezifischer Elemente zur Simulationslaufzeit zu ermöglichen, benötigt die in Abschnitt 3.4 beschriebene Simulationskomponente nur geringfügige Anpassungen. Über das Strategie-Pattern sollen domänenspezifische Ereignisse und Aktivitäten an die generellen BPMN-Elemente angehängt werden können. Ein domänenspezifisches Element aus der Prozessdefinition wird also durch die Kombination von zwei

Objekten repräsentiert: Dem generellen BPMN-Element mit erweiterten Eigenschaften und einem domänenspezifischen Element.

Ereignisse

Die Ereignisse besitzen eine Liste `SpecifiedEvent`, um die in der BPMN vorgesehenen Ereignistypen (z.B. Nachricht, Fehler, Eskalation) hinterlegen zu können. Die `Execute`-Methode des generellen Ereignissen iteriert über diese Liste und führt bei allen hinterlegten Ereignisspezifikationen nebenläufig die `SpecifiedEventExecute`-Methode aus. Erst wenn in allen Ereignistypen aus dieser Liste dieser Methodenaufruf abgeschlossen wurde, kann die `Execute`-Methode des generellen Ereignisses terminieren. Dieses Prinzip kann auch für die vorliegenden Konzepte verwendet werden, es sind daher keine Änderungen am generellen `BPMNEvent` der DESMO-J-Erweiterung notwendig. Der Modellkoppler muss lediglich das domänenspezifische Ereignis ebenfalls in diese Liste eintragen.

Zur Implementation von domänenspezifischen Empfangsereignissen werden allerdings zwei weitere Klassen benötigt:

Die erste repräsentiert das domänenspezifische BPMN-Element und ist dasjenige Objekt, welches in die gerade beschriebene Liste eingetragen wird. Hierfür wird eine allgemeingültige Klasse `DomainSpecificCatchEvent` bereitgestellt, welche für alle domänenspezifischen Empfangsereignisse verwendet werden kann. Diese Klasse implementiert die Methode `SpecifiedEventExecute`, welche den ankommenden Prozess passiviert und diesen in eine Liste einträgt. Außerdem wird eine Methode `Release` bereitgestellt, welche den Prozess wieder aktiviert und aus der Liste entfernt. Eine dritte Methode `TriggerEvent` wertet die durch den Modellierer hinterlegten Konkretisierungen zur Kopplung der Modelle aus: Es wird überprüft, ob das sendende Teilmodell dasjenige ist, zu dem eine statische ID hinterlegt wurde. Es wird überprüft, ob eine Bedingung erfüllt ist, welche über die Python-Enginge ausgewertet werden kann. Assozierte Artefakte mit Kanten vom Artefakt zum Element werden dahingehend überprüft, ob dort eine Referenz enthalten ist, die der des Senders der Nachricht entspricht. Sind diese Prüfungen erfolgreich, so wird die `Release`-Methode aufgerufen, was das tatsächliche Auslösen der Ereignisses innerhalb einer Prozessinstanz bedeutet. Sind diese Prüfungen nicht erfolgreich, so wird das Ereignis in der entsprechenden Prozessinstanz ignoriert. Wenn ein Artefakt mit einer Assoziationskante vom Element zum Artefakt hinterlegt wurde, wird dort die Referenz auf den Sender der Nachricht hinterlegt. Bei Bedarf können diese drei Methoden durch Überschreiben angepasst werden, zu diesem Zweck sind die entsprechenden Methoden als `virtual` gekennzeichnet. Dadurch kann auf das Prüfen bestimmter Bedingungen verzichtet werden, wenn diese für das Ereignis nicht vorgesehen sind. Alertnativ kann der Umgang mit den assoziierten Artefakten für die jeweiligen Bedürfnisse

angepasst werden, so dass zum Beispiel anstatt Referenzen auf sendende Entitäten mit den Referenzen der sendenden Teilmodelle gearbeitet wird.

Die zweite benötigte Klasse muss vom Entwickler der Erweiterungsbibliothek implementiert werden. Sie sollte von der abstrakten Klasse `Desmoj.Core.Simulator.Event` erben und die Methode `EventRoutine` so implementieren, dass es die `TriggerEvent`-Methode der zugehörigen BPMN-Elemente auslöst. Zusätzlich können Manipulationen an dem entsprechenden Teilmodell vorgenommen werden. Es wird von dem domänenspezifischen Teilmodell instanziiert und für einen definierten Zeitpunkt auf der Ereignisliste eingetragen. Alternativ kann die `TriggerEvent`-Methode auch aus dem `LifeCycle` einer von `SimProcess` erbenden Klasse angestoßen werden. Der Entwickler hat in jedem Fall dafür zu Sorge zu tragen, dass aus dem Teilmodell heraus das oben beschriebene `DomainSpecificCatchEvent` über die `TriggerEvent`-Methode angestoßen wird.

Zur Implementation von sendenden Ereignissen muss vom Entwickler eine Klasse bereitgestellt werden, die von `SpecifiedEvent` erbt. Hier muss die Methode `SpecifiedEvent-Execute` implementiert werden, so dass entsprechende Methoden an dem Teilmodell oder an Entitäten des Teilmodells aufgerufen werden, welche den Empfang einer Nachricht behandeln. Um die korrekten Teilmodelle oder die korrekten Entitäten auswählen zu können, kann der Entwickler hier entsprechende Prüfungen vornehmen, welche sich auf assoziierte Artefakte oder auf die statische Modell-ID beziehen. Die Referenzen auf die Teilmodelle müssen vom Modellkoppler an das sendende Ereignis übergeben werden (vgl. Abschnitt 6.5.3).

Aktivitäten

Um domänenspezifische Aktivitäten zu ermöglichen, waren Änderungen an der bestehenden Bibliothek notwendig. Da in der Simulation bisher verschiedene Arten von Aktivitäten nicht unterschieden werden mussten, war ein den Ereignissen vergleichbares Prinzip bisher nicht implementiert. Im Konzept sind zwei Möglichkeiten beschrieben, das Verhalten der Aktivitäten durch domänenspezifische Elemente zu modifizieren: Die `Execute`-Methode der Aktivität wird komplett durch das domänenspezifische Element übernommen, oder es werden EventHandler zu Beginn und zum Ende einer Aktivität aufgerufen. Während die `Execute`-Methode nur von einer domänenspezifischen Aktivität übernommen werden kann, können sich an den EventHandlern potentiell mehrere Aktivitäten anmelden.

Hierfür wurden der bestehenden `BPMNActivity` vier Events und entsprechende Delegates hinzugefügt (vgl. das Codebeispiel B.4 im Anhang):

- `ActivityStartEventHandler`

- `ActivityFinishEventHandler`
- `MultipleActivityStartEventHandler`
- `MultipleActivityFinishEventHandler`

Der Entwickler der Erweiterungsbibliothek muss eine Klasse implementieren, die von `DomainSpecificActivity` erbt (vgl. den Quellcode B.5 im Anhang). In ihr sind Methoden gemäß dieser Delegates enthalten, die beim Konstruktoraufruf automatisch an den entsprechenden Events der `BPMNActivity` angemeldet werden. Standardmäßig ist der Methodenrumpf dieser EventHandler leer. Wenn ein domänenspezifisches Verhalten erwünscht wird, können diese Methode überschrieben werden. Die angemeldeten EventHandler werden von der `BPMNActivity` synchron ausgeführt, so dass die Ausführung der Aktivität durch den Aufruf der EventHandler verzögert werden kann.

Außerdem hält die `BPMNActivity` eine Referenz auf maximal eine angemeldete `DomainSpecificActivity`, welche ein Bool-Property `DomainSpecificExecute` beinhaltet. Dieses wird zu Beginn der `Execute`-Methode der BPMN-Aktivität geprüft - bei Erfolg wird die Execute-Methode der `DomainSpecificActivity` aufgerufen und der Rest der Execute-Methode der `BPMNActivity` übersprungen. Hierdurch kann die Ausführungsdauer anstatt über stochastische Verteilungen, die an der BPMN-Aktivität hinterlegt wurden, durch das domänenspezifische Teilmodell bestimmt werden.

Artefakte

Für das domänenspezifische Artefakt wird eine abstrakte Klasse `DomainSpecificArtifact` bereitgestellt (vgl. Quellcode B.6 im Anhang). Sie beinhaltet ein Property vom Typ `Desmoj.Core.Simulator.Queue`, für den ein Warteschlangentyp angegeben werden muss, in welchen Entitäten aus dem entsprechenden Teilmodell eingereiht werden können. Zwei Methoden `PutEntity` und `GetEntity` erlauben es, Referenzen zu hinterlegen oder abzurufen. Ob tatsächlich mehr als eine Entität in der Warteschlange liegen darf, obliegt dem Entwickler. Es ist auch denkbar, das Artefakt so zu implementieren, dass es nur genau eine Referenz hält. In diesem Fall sollte die Entität von der Methode `GetEntity` nicht aus der Warteschlange entfernt werden, so dass sie eine für die Lebenszeit der Prozessinstanz dauerhafte Referenz darstellt. Auch muss in der Methode `GetEntity` entschieden werden, ob die Warteschlange global anzusehen ist, oder nur innerhalb einer Prozessinstanz bekannt ist. Im letzteren Fall kann das Property `Queue` so implementiert werden, dass es auf eine HashMap zugreift, welche für jede Prozessinstanz eine Warteschlange bereithält. Da die aktuelle Prozessinstanz über die DESMO-J-Methode `CurrentProcess` global zugreifbar ist, muss sie nicht als Parameter der `GetEntity`-Methode übergeben werden.

6.4.2. Bereitstellung der grafischen Repräsentation

Für die Simulationszeit stehen nun bereits Klassen zur Verfügung. Auch im BPMN-Editor müssen die domänenspezifischen Elemente bereitgestellt werden, um dem Modellierer deren Integration in eine Prozessdefinition zu erlauben. Für den verwendeten Prototypen steht bereits ein Editor zur Manipulation von BPMN-Prozessen zur Verfügung (vgl. Oeser (2013)). Dieser Editor setzt auf einer Empinia-Editorkomponente für spezialisierte Graphen auf (vgl. Joschko (2008)).

Diese Editorkomponente erlaubt es, zu den Elementen eines logischen Objektmodells, welches einen Graphen repräsentiert, Visualisierungsklassen anzumelden, die in einer Editoroberfläche angezeigt werden. Wenn der Editorkomponente ein Objektmodell übergeben wird, sucht diese nach passenden Visualisierern und führt deren `Draw`-Methode aus, so dass eine Repräsentation auf der Zeichenfläche erstellt wird. Wenn in dieser Zeichenfläche ein Element selektiert wird, wird das zugehörige logische Element an den `SelectionService` übergeben, so dass die Empinia Komponente `PropertyEditor` über den Reflection-Mechanismus deren Properties identifziert, in einer seperaten View anzeigt und deren Manipulation erlaubt. Die Elemente können vom Benutzer verschoben werden; es ist die Aufgabe der visualiserenden Klassen, die Veränderung der Koordinaten an das entsprechende logische Element zu melden. Auf diese Weise können für beliebige graphbasierte Modelle Erweiterungen bereitgestellt werden, um deren Manipulation über den grafischen Editor zu ermöglichen. Über eine Toolbox können diejenigen Elemente bereitgestellt werden, die von dem Benutzer per Drag & Drop in dem Modell platziert werden dürfen. Welche Toolbox geladen wird, ist dabei abhängig von dem Wurzelknoten des übergebenen Modells. (vgl. Joschko (2008))

Die visualisierenden Klassen werden über die Bundle.xml einer Erweiterungskomponente angemeldet. Hierdurch kann die Editorkomponente eine Vorentscheidung treffen, welche Visualisier potentiell für ein logisches Objekt in Frage kommen. An diesen Visualisierern wird die Methode `CanVisualise` aufgerufen. Ihr wird das logische Element als Parameter übergeben, so dass der Visualisier anhand der Eigenschaften entscheiden kann, ob er tatsächlich für dieses Element zuständig ist.

In Oeser (2013) wurde aufbauend auf dieser Editorkomponente aus Joschko (2008) eine Erweiterung für BPMN-Modelle geschaffen. Diese enthält Visualisierungsklassen für alle in der BPMN-Spezifikation vorgesehenen Elemente. Sie ist somit in der Lage, das Objektmodell der BPMN, welches durch die OMG vorgesehenen ist, zu visualisieren. Aus der von der OMG bereitgestellten XSD-Datei (OMG (2011)) wurde mit dem XML Schema Definition Tool von Microsoft ein .NET-Klassenmodell generiert (vgl. Haan (2012)). Die Einhaltung der BPMN-Spezifikation wird dadurch sichergestellt. So können auch mit anderen BPMN-Editoren, welche die Spezifikation der OMG erfüllen, Prozesse modelliert und anschließend in den vorliegenden Prototyp zur weiteren Manipulation im-

portiert werden. Auch ein Export der modifizierten Modelle und eine Weiterbearbeitung in Editoren von anderen Herstellern ist möglich. Geringfügige Änderungen an dem Objektmodell waren notwendig, um die Definition von domänenspezifischen Elementen zu erlauben. Die Erweiterungsattribute `IsDomainSpecificElement`, `DomainSpecificType`, `DomainSpecificModelID` und `DomainSpecificCondition` wurden hinzugefügt. Alle haben den Typ `String`. Da es diese an der von der OMG vorgesehen Position für Erweiterungsattribute eingefügt wurden, ist die Austauschbarkeit der Editoren weiterhin gewährleistet.

Zur Implementation der Konzepte dieser Arbeit waren zudem geringfügige Modifikationen an der Editorkomponente aus Oeser (2013) und dem Objektmodell aus Haan (2012) nötig. Während bisher alle Elemente eines spezialisierten Graphen innerhalb derselben Bundle.xml angemeldet werden mussten, ist es nun möglich, die Anmeldungen der Visualisierer über verschiedene Komponenten zu verteilen. Die `CanVisualise`-Methode der bestehenden BPMN-Visualisierer wurde dahingehend geändert, dass sie prüft, ob das Erweiterungsattribut `IsDomainSpecificElement` auf `false` gesetzt ist. Ist es `true`, so wird durch die ursprünglich vorgesehenen Visualisierer verweigert.

Um die Visualisierung von domänenspezifischen Elementen zu erlauben, müssen von der Erweiterungskomponente eigene Visualisierer bereitgestellt werden. Hierfür werden abstrakte Klassen bereitgestellt:

- BPMN.UI.Visualisations.Activity
- BPMN.UI.Visualisations.StartEvent
- BPMN.UI.Visualisations.IntermediateEvent
- BPMN.UI.Visualisations.EndEvent
- BPMN.UI.Visualisations.Artifact

Da die BPMN für Start-, Zwischen- und Endereignisse verschiedene Umrandungen vorsieht, werden hier verschiedene Visualisierer benötigt. Bei allen fünf Klassen müssen zwei Methoden überschrieben werden. Die Methode `DrawSymbol` lädt eine als Ressource hinterlegte Grafikdatei und gibt diese als Rückgabewert wieder. Hier soll eine Grafik gewählt werden, welche möglichst intuitiv verständlich auf das Teilmodell und die Art der Interaktion verweist. Dabei gilt zu bedenken, dass schwarze Symbole für sendende Ereignisse stehen, und lediglich schwarz umrandete aber weiß ausgefüllte Symbole für empfangende Ereignisse. Außerdem muss die Methode `CanVisualise` so überschrieben werden, dass sie das Proptery `DomainSpecificType` des logischen Elementes überprüft und `true` zurückgibt, wenn dies der für die Visualisierung vorgesehene Typ ist.

6.5. Generierung und Verknüpfung von Modellen

Zur Erzeugung von Simulationsexperimenten müssen der Parametrierer, die Simulationsfabrik und der Modellkoppler in der `ModelTypeConfiguration` angemeldet werden. Zur Implementation werden jeweils abstrakte Klassen oder Schnittstellen zur Verfügung gestellt, an denen es einzelne Methoden zu implementieren gilt.

6.5.1. Parametrierer

Die erforderliche Schnittstelle für den Parametrierer (in der prototypischen Implementation als `ModelConfigurer` bezeichnet) findet sich in der linken Hälfte von Abbildung 6.6 und in dem Codebeispiel B.7 im Anhang. Das Konzept für den Parametrierer von BPMN-Modellen wurde in Stehle (2013) implementiert und im Rahmen dieser Arbeit auf die Parametrierung von Teilmodellen ausgeweitet. Zwei Methoden müssen bereitgestellt werden. Der Methode `GetParameterRange` wird eine Modelldefinition übergeben. Der Parametrierer durchsucht diese nach Stellen, an denen anstatt eines einzelnen Parameters ein Parameterraum angegeben wurde und gibt eine Liste der möglichen Parameter zurück. Der Typ `ParameterValueDescription` enthält dabei den Namen eines Parameters und einen zugehörigen Wert. Wenn die domänenspezifische Erweiterung keine Definition von Parameterräumen unterstützt, kann hier eine leere Liste zurückgegeben werden.

Der Experimentplaner sammelt auf diese Weise aus allen Teilmodellen eine Liste der möglichen Parameter und kann darauf basierend eine bestimmte Anzahl von Simulationsläufen vorsehen. Da der mögliche Parameterraum sehr groß werden kann, so dass eine effiziente Durchführung der Simulationsexperimente nicht mehr möglich ist, ist es sinnvoll, hier Optimierungsstrategien wie zum Beispiel in der in Czogalla et al. (2006) gezeigten Experimentierumgebung zu hinterlegen. Der entworfene Prototyp unterstützt jedoch keine Optimierungsstrategien, sondern führt für jede Parameterkombination die angegeben Anzahl von Simulationsläufen durch.

Hierfür muss der Parametrierer eine zweite Methode `GetConfiguredModel` bereitstellen. Ihr wird erneut eine Modelldefinition übergeben, sowie eine Liste mit den zu verwendenden Parametern. Die Methode soll einen Klon der Modelldefinition erzeugen und die angegeben Parameterräume durch die nun konkret zu verwendenden Paramter ersetzen. Hierdurch entsteht eine neue, nur temporär vorhandene Modelldefinition, welche im nächsten Schritt an die Simulationsfabrik übergeben wird.

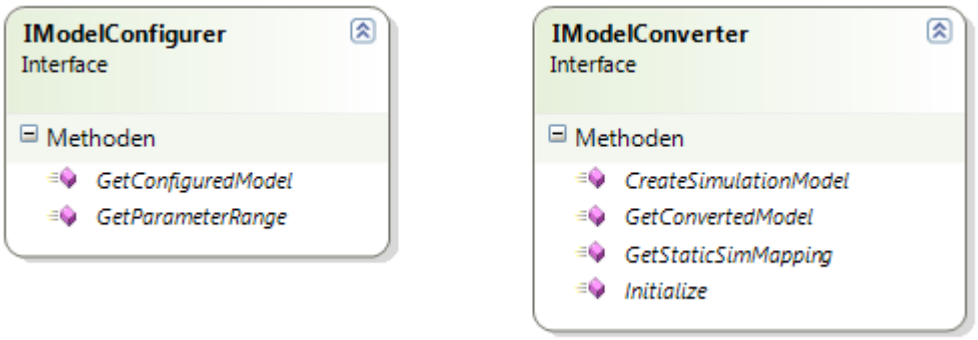


Abbildung 6.6.: Klassendiagramm Parametrierer und Simulationsfabrik

6.5.2. Simulationsfabrik

Die Simulationsfabrik erzeugt aus einer Modelldefinition ein Simulationsmodell. Der Entwurf dieser Schnittstelle ist abhängig von dem verwendeten Simulationskern - in diesem Fall wird ein DESMO-J-Modell erzeugt. Die erforderliche Schnittstelle, welche hier als `IModelConverter` bezeichnet wird, findet sich in der rechten Hälfte von Abbildung 6.6 und in dem Codebeispiel B.8 im Anhang.

In DESMO-J müssen alle Teilmodelle als Submodell eines Hauptmodells implementiert werden. In diesem Fall dient das Hauptmodell lediglich als Containerklasse, da sämtliche für die Simulation relevanten Bestandteile in den Submodellen beschrieben werden. Einem Submodell muss bereits in dem Konstruktoraufruf das entsprechende Hauptmodell übergeben werden.

Die Methode `Initialize` nimmt das durch den Experimentplaner erzeugte Hauptmodell entgegen, um den Konstruktoraufruf des Teilmodells zu ermöglichen. Außerdem wird in dem Hauptmodell eine Referenz auf das Submodell hinterlegt. Der zweite Parameter nimmt einen `DistributionManager` entgegen. Dieser ist in DESMO-J dafür verantworlich, Seeds für die einzelnen stochastischen Verteilungen, basierend auf einem globalem Seed, zur Verfügung zu stellen. Er wird zur Instanziierung von stochastischen Verteilungen innerhalb der Teilmodelle benötigt.

Die Methode `CreateSimulationModel` muss die erforderlichen Prozesse, Ereignisse und Entitäten für das jeweilige Teilmodell implementieren und dabei auf die von DESMO-J bereitgestellten Klassen `SimProcess`, `Event` und `Entity` zurückgreifen. Der Methode wird als Parameter die Modelldefinition übergeben, der Rückgabewert ist ein DESMO-J-Modell.

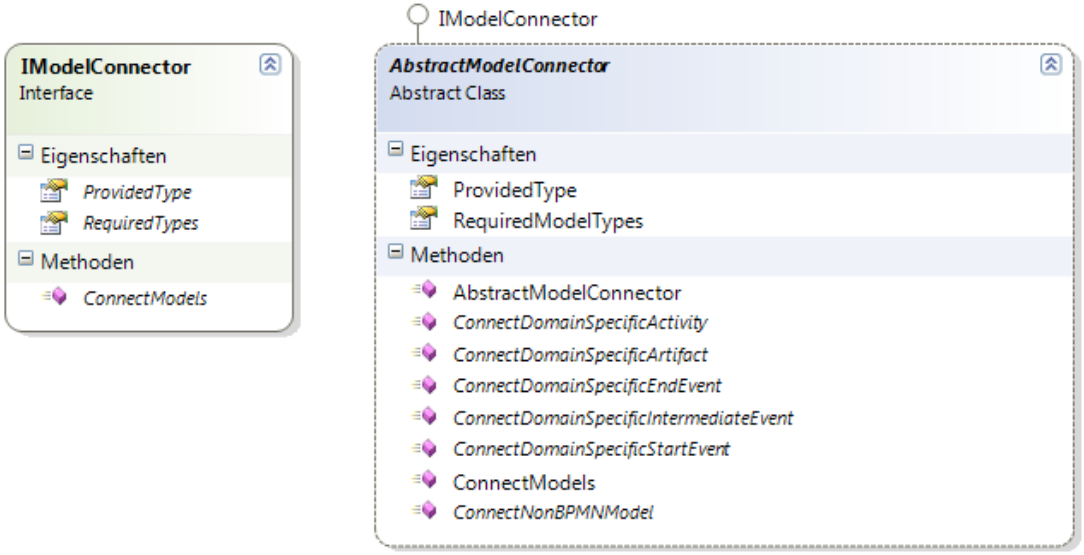


Abbildung 6.7.: Klassendiagramm Modellkoppler - Schnittstelle und Abstrakte Klasse

Bei der Generierung der Simulationsobjekte soll ein `Dictionary` geführt werden, welches eine Zuordnung von logischen Objekten aus der Modelldefinition zu den jeweils erzeugten Simulationsobjekten führt. Dieses `Dictionary` kann über die Methode `GetStaticSimMapping` abgerufen werden. Der Simulationskoppler kann hieraus im nächsten Schritt die notwendigen Informationen ziehen, um die Verknüpfung der Teilmodelle zu realisieren.

6.5.3. Modellkoppler

Für den Modellkoppler wird ein Interface `IModelConnector` und eine abstrakte Klasse `AbstractModelConnector` bereitgestellt. Es ist nur verpflichtend das Interface zu erfüllen, jedoch wird durch Ableitung von der abstrakten Klasse die Implementation des Modellkopplers stark vereinfacht. Auch zwingt der `AbstractModelConnector` den Entwickler dazu, die Konzepte dieser Arbeit einzuhalten, da nur diejenigen BPMN-Elemente mit domänenspezifischen Erweiterungen versehen werden können, die dafür vorgesehen sind.

Die Schnittstelle schreibt lediglich die Implementation von zwei Properties und einer Methoden vor (vgl. Abbildung 6.7 und Codebeispiel B.9 im Anhang). Von dem nur lesbaren Property `RequiredTypes` müssen diejenigen Modelltypen zurückgegeben werden, mit denen eine Kopplung möglich ist. Hier sollte mindestens der Typ `BPMNSimulationModel`

angegeben werden, es können bei Bedarf auch die Typen von anderen Erweiterungen hinterlegt werden. Das Property `ProvidedType` gibt den Modelltyp an, für welchen der Modellkoppler zuständig ist. Die Methode `ConnectModels` soll die eigentliche Kopplung implementieren. Sie nimmt eine Liste von Modellen, die vom `ProvidedType` sind, sowie eine Liste von Modellen, die den `RequiredTypes` entsprechen, entgegen. Außerdem bekommt sie die `Dictionary`-Objekte übergeben, die von der Simulationsfabrik erzeugt wurden und die Zuordnung von Objekten aus der Modelldefinition zu Objekten aus dem generierten Simulationsmodell enthalten.

Diese Schnittstelle bietet wenig Unterstützung bei der Implementation eines Modellkopplers. Einfacher wird es für den Entwickler der Erweiterungsbibliothek durch Ableiten von der abstrakten Klasse `AbstractModelConnector` (vgl. Codebeispiel B.10 im Anhang). Das Property `RequiredTypes` ist hier bereits implementiert und liefert `BPMNSimulationModel` als Rückgabewert - jedoch kann dieses Property bei Bedarf überschrieben werden. Die Methode `ConnectModels` ist ebenfalls bereits implementiert. Sie durchsucht die übergebenen BPMN-Prozesse nach Elementen, für welche domänenspezifische Erweiterungen angegeben wurden. Für jedes dieser Elemente wird eine abstrakte Methode aufgerufen, die vom Entwickler implementiert werden muss. Es gibt drei Methoden für die Ereignisse (`ConnectDomainSpecificStartEvent`, `ConnectDomainSpecificInermediateEvent` und `ConnectDomainSpecificEndEvent`), eine Methode für Aktivitäten (`ConnectDomainSpecificActivity`) und eine für Artefakte (`ConnectDomainSpecificArtifact`). An diese Methoden wird jeweils das erzeugte, generelle DESMO-J-Objekt und das zugehörige Objekt aus der Prozessdefinition als Parameter übergeben. Desweiteren wird eine Liste mit denjenigen Teilmodellen übergeben, für die der Modellkoppler zuständig ist, damit dort entsprechende Referenzen gesetzt werden können. Innerhalb der Methoden muss nun das domänenspezifische Element erzeugt werden, bei Bedarf mit Referenzen auf das Teilmodell versehen werden und über das Strategie-Pattern an das bereitgestellte generelle BPMN-Element angehängt werden.

Eine weitere Methode `ConnectNonBPMNModel` kann implementiert werden, um Teilmodelle untereinander zu verknüpfen. Ihr werden alle Teilmodelle aus den `RequiredTypes` übergeben, welche nicht BPMN-Modelle sind.

6.6. Benutzeroberfläche

Für den Prototypen wurde aufbauend auf den Mechanismen des Empinia-Rahmenwerks eine Benutzeroberfäche bereitgestellt. Für die drei Phasen der Simulationsmodellierung – Modellierung, Experimentplanung und Ergebnisanalyse – wurden dabei verschiedene Perspektiven (vgl. Abschnitt 6.1.3) entworfen.

6.6.1. Modellierungsumgebung

Abbildung C.1 im Anhang zeigt die Perspektive zur Modellierung von BPMN-Prozessen. Links oben ist die Toolbox zu sehen, über welche BPMN-Elemente in den Diagrammbereich in der Mitte gezogen werden können. In dieser Toolbox finden sich nach einer Erweiterung auch die domänenspezifischen Elemente. Unterhalb der Toolbox befindet sich die Projektmappen-Ansicht, über welche das aktuell zu bearbeitende Modell ausgewählt werden kann. Auf der rechten Seite befindet sich ein Eigenschaftseditor. Er beinhaltet verschiedene Sichten auf die Eigenschaften eines ausgewählten Elements. In der Abbildung ist ein Editor zur Auswahl und Parametrierung von stochastischen Verteilungen für die Dauer einer Aktivität zu sehen. Eine andere Sicht zeigt sämtliche Eigenschaften des Elements in der Übersicht.

Bei der Auswahl eines domänenspezifischen Teilmodells über die Projektmappe schließen sich die Toolbox und der Property Editor. Anstatt des BPMN-Editors wird im mittleren Bereich die View der bereitgestellten Editor-Komponente angezeigt.

6.6.2. Experimentierumgebung

Abbildung C.2 zeigt die Perspektive zur Planung von Experimentreihen. Auf der linken Seite werden Experimentparameter angezeigt, welche vor einem Simulationslauf einzustellen sind. Hierzu gehört die Auswahl derjenigen DESMO-J-Klasse, welche für die Darstellung der Ergebnisreports verwendet werden sollen. Zur Verfügung stehen neben einer XML-, einer ASCII- und einer HTML-basierten Sicht, auch eine in Stehle (2014) entwickelte Sicht, welche auf der Windows Presentation Foundation beruht und durch Diagramme angereichert ist. Sollte die HTML-, die XML- oder die ASCI-basierte Reportdarstellung gewählt werden, so kann ein Verzeichnispfad angegeben werden, in dem die Ergebnisse gespeichert werden.

Außerdem muss die gewünschte Anzahl der Simulationsläufe eingegeben werden, welche für jede Parameterkombination durchgeführt wird. In einer weiteren - in der Abbildung nicht zu sehenden - View kann für jeden der Simulationsläufe ein Seed eingestellt werden. Es werden automatisch Seeds generiert, welche von dem Benutzer verändert werden können. Um eine Vergleichbarkeit der Ergebnisse bei verschiedenen Parametern zu gewährleisten, werden diese Seeds bei der Wiederholung von Simulationsläufen mit anderen Parameterkombinationen wiederverwendet.

Desweiteren kann unter den Experimentparametern angegeben werden, ob ein Trace-Output, welcher sämtliche Vorkommnisse auf der Ereignisliste protokolliert, generiert werden soll. Dies ist hilfreich, um das Verhalten eines Simulationslaufs im Detail nachvollziehen zu können. Bei langen Simulationsläufen wird dieser Bericht allerdings sehr

groß. Daher kann ein bestimmter Zeitausschnitt konfiguriert werden, in welchem die Protokollierung stattfinden soll. Ein noch detailliertes Protokoll bietet der Debug-Report, für welchen ein seperater Zeitausschnitt angegeben werden kann.

Unterhalb der View zur Experimentparametereingabe findet sich eine modifizierte Variante der Projektmappe. Hier kann ausgewählt werden, welche der geladenen Prozess- und Teilmodelle in das Experiment einbezogen werden sollen. Noch bei der Planung der Experimentreihen können für die BPMN-Prozesse Parameterräume angegeben werden. Daher findet sich in der Mitte der Perspektive erneut ein Dokumentbereich zur Darstellung der Prozesse und am rechten Rand der Eigenschaftseditor.

6.6.3. Ergebnisberichte

Abbildung C.3 zeigt die Perspektive zur Darstellung von Ergebnisberichten. Auf der linken Seite befindet sich eine View, mit der die Berichte für die einzelnen Simulationsläufe betrachtet werden können. Neben der Ausgabe der Statistiken finden sich dort ggf. auch die Debug- und Trace-Reports. Desweiteren beitet DESMO-J einen Error-Report, welcher mögliche Fehler innerhalb eines Simulationslaufs beinhaltet.

Für jede Parameterkombination werden die Ergebnisse in einem Batch-Report zusammgefasst (vgl. Stehle (2013)). Dort werden die durchschnittlichen Werte, sowie Maximal- und Minimalwerte zu jeder einzelnen Statistik aufgeführt.

In der Mitte der Abbildung ist die Darstellung eines HTML-basierten Reports zu sehen. Dies ist die Standardausgabe von DESMO-J, an dieser Stelle können jedoch auch die anderen möglichen Ausgabekanäle dargestellt werden. Durch einen Mausklick auf die einzelnen Report-Einträge am linken Rand wird der entsprechende Report in der Mitte dargestellt.

7. Fallstudie des Einsatzes im Bereich Offshore-Windkraftanlagen

In diesem Kapitel wird mit der Offshore-Windbranche ein Fallbeispiel vorgestellt, bei welchem zur Leistungsmessung der Prozesse mit Hilfe der Simulation domänenspezifische Simulationsmodelle benötigt werden. Die Anforderungen wurden dem Forschungsprojekt „System Optimierung Offshore Wind“ entnommen. Die Entwicklung von passenden Erweiterungskomponenten wird vorgestellt und deren Verwendung anhand eines kleinen Beispiels demonstriert. Die Anwendbarkeit der Konzepte in der Praxis kann somit überprüft werden.

7.1. Motivation des Fallbeispiels

In diesem Abschnitt wird das Projekt „System Optimierung Offshore Wind“ motiviert und die sich hieraus ergebenden Anforderungen an eine spezifische Simulationssoftware abgeleitet.

7.1.1. Offshore Wind Energie

Der Übergang der Energieversorgung der BRD hin zu erneuerbaren Energien wird mittlerweile „durch einen breiten politischen und gesellschaftlichen Konsens gestützt“ (Klinke und Klarmann (2012), S. II). Zielsetzung ist – wie im Kyoto-Protokoll und den Nachfolgeverträgen vereinbart – eine Verminderung des Schadstoffaustoßes, um der drohenden globalen Erwärmung entgegen zu wirken. In den letzten Jahren wurden insbesondere die regenerativen Energieformen Windkraft und Photovoltaik stetig ausgebaut (siehe Abbildung 7.1). Die Bundesregierung sieht vor, den Anteil der erneuerbaren Energien an der Stromerzeugung von derzeit 25 Prozent (Nickel (2013), S. 12) auf mindestens 35 Prozent im Jahre 2020 zu erhöhen (Ziesing (2013), S. 38).

Der Windenergie kommt hier eine tragende Rolle zu. Laut dem BMU könnten 25 Prozent des deutschen Strombedarfs durch Windenergie und hiervon 60 Prozent durch Windkraftanlagen in Nord- und Ostsee gedeckt werden (Falk (2007), S. 32). Daher geht der

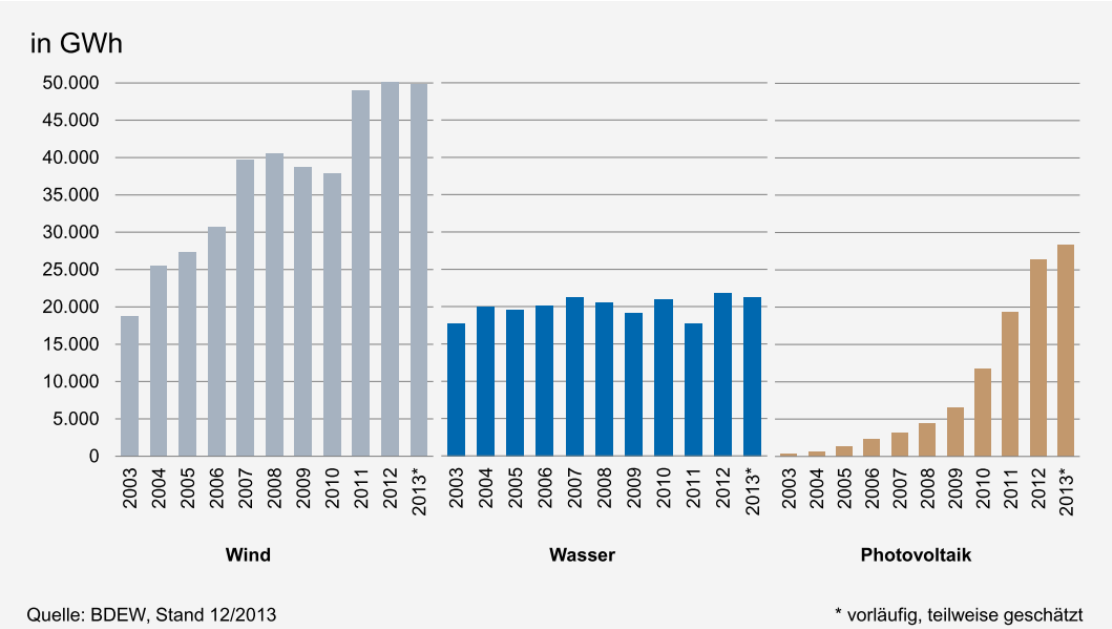


Abbildung 7.1.: „Entwicklung der Stromerzeugung aus einzelnen erneuerbaren Energieträgern“. Abbildung aus: Nickel (2013)

Trend in Deutschland zu immer größeren und leistungsstärkeren Windkraftanlagen. Besonders leistungsstarke Offshore-Anlagen können in Nord- und Ostsee platziert werden, denn hier herrschen starke und gleichmäßige Windverhältnisse. Im Jahr 2012 wurden bereits 31.308MW durch Windkraftanlagen bereitgestellt, davon jedoch nur 280 MW durch Offshore-Anlagen (Ziesing (2013), S. 27). Die Bundesregierung plant bis zum Jahr 2030 20.000 MW bis 25.000 MW Nennleistung durch Offshore-Energie installieren zu lassen (Bundesregierung (2002), S. 7). Diese Pläne wurde in jüngster Zeit erneut bestätigt (Deutscher Bundestag (2013), S. 2).

Hier ist ein völlig neuer Industriezweig in der Entstehung. In Deutschland werden zur Zeit erst die ersten Erfahrungen mit Offshore-Windparks (OWP) gesammelt. Seit 2010 existiert mit Alpha Ventus ein Forschungswindpark zur Erprobung der Offshore-Energieerzeugung. Mittlerweile sind mit Bard Offshore 1 und Riffgat in der Nordsee, sowie Baltic I in der Ostsee auch drei kommerzielle Windparks im Betrieb. Offshore-Windparks anderer Länder stehen erheblich näher an der Küste und somit in seichteren Gewässern als dies in Deutschland aufgrund des Wattenmeerschutzes möglich ist. Außerhalb der 12-Seemeilen-Zone herrschen jedoch Betriebsbedingungen, welche Betriebs- und Instandhaltungsprozesse erfordern, die erst noch zu etablieren sind.

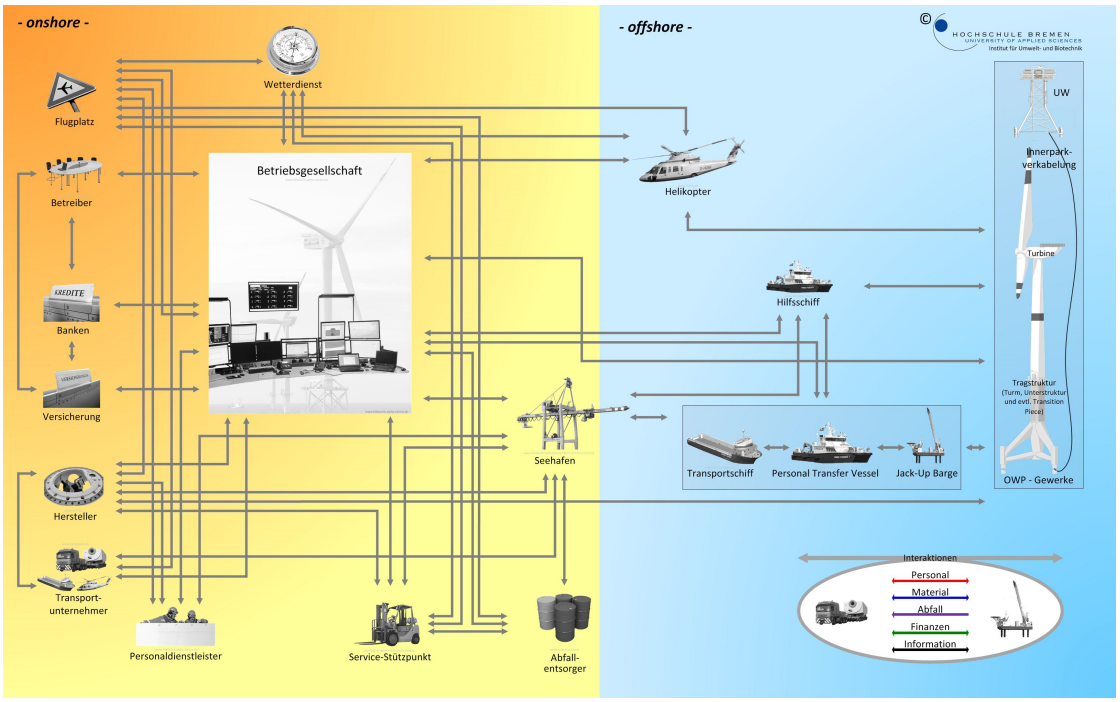


Abbildung 7.2.: Auszug aus dem Leistungssystem Offshore-Windpark, Originaltitel „Extract of the system offshore wind farm in repair status “. Abbildung aus: Greiner et al. (2014)

7.1.2. Verbundprojekt SystOp Offshore Wind

Die bisherige Erfahrung zeigt, dass sich der effiziente und optimale Betrieb zu Wasser wesentlich komplexer darstellt als zu Lande. Neue und teilweise unangepasste Technologien und Prozesse, sowie unvollständige Betriebs- und Instandhaltungskonzepte sind maßgebliche Faktoren, die die wirtschaftliche Tragfähigkeit eines Windparks beeinträchtigen können. Circa 30 Prozent der Stromgestehungskosten von Offshore-Windparks entfallen auf die Betriebsphase (vgl. Weber (2013)). Durch eine effizientere Gestaltung der Abläufe können die Kosten noch erheblich gesenkt werden (vgl. Hobohm et al. (2013)). Das effiziente Management von betrieblichen Prozessen ist in Hochtechnologiebereichen seit langem etabliert. In der Offshore-Branche müssen diese Prozesse jedoch erst noch erprobt werden. Es fehlt an Erfahrung und eingespielten Strukturen. Es existieren keine allgemeingültigen, bewerteten Prozessmodelle, welche die Grundlage für die Durchführung eines BPI-Vorhabens bilden könnten.

Diese Lücke soll durch das vom BMU geförderte Forschungsprojekt „Systemoptimierung Offshore Wind“ (SystOp) geschlossen werden. In diesem Projekt befassen sich die Hochschule Bremen, die Universität Hamburg, die BTC AG und die IZP AG mit der Erfassung, der Analyse und der Bewertung der Betriebsphase von Offshore-Anlagen (vgl. Projekthomepage unter SystOp (2014)). Wie in Abbildung 7.2 zu sehen, ist die Anzahl der beteiligten Organisationen und der Verflechtung der Interaktionsbeziehungen in diesem „Leistungssystem“ sehr hoch, was einen eklatanten Unterschied zwischen Offshore- und Onshore-Anlagen darstellt. Zwei Dutzend Industriepartner stehen daher zur Verfügung, um Einblick in ihre Prozesse und Daten zu gewähren.

Ein wesentlicher Bestandteil der Betriebsphase sind die Instandhaltungsprozesse. Diese setzen sich zusammen aus in regelmäßigen Abständen stattfindenden Wartungsprozessen (z.B. Öl- und Filterwechsel) und Instandsetzungsprozessen zur Behebung von auftretenden Störungen. Diese Instandhaltungsprozesse werden im SystOp-Projekt mit Hilfe der BPMN 2.0 modelliert. Die BPMN 2.0 ist hier besonders geeignet, da sie im Gegensatz zu anderen Geschäftsprozessmodellierungssprachen durch die Unterscheidung von Sequenz- und Nachrichtenfluss die einzelnen Prozesse der verschiedenen Beteiligten und deren Interaktion adäquat darstellen kann. Im bisherigen Projektverlauf hat sich zudem gezeigt, dass die Prozessmodelle leicht verständlich für die beteiligten Industriepartner sind, was eine Validerung der Modelle vereinfacht.

Im Zuge des Projektes entstehen Referenzprozessmodelle, welche künftig als Kommunikationsgrundlage für die Branche, für Behörden und für die Wissenschaft dienen werden. Eine erste Version dieses Referenzprozessmodells ist bereits öffentlich verfügbar (Klinke und Klarmann (2012)). Im weiteren Projektverlauf werden diese Modelle detaillierter ausgearbeitet, gemeinsam mit Industriepartnern validiert und mit Hilfe von Simulationstechnik mit Kennzahlen versehen.

Die Durchführung der Simulationsexperimente liegt im Aufgabenbereich der Universität Hamburg. Durch die Verwendung der BPMN als Modellierungssprache ist eine direkte Überführung der Prozessmodelle in Simulationsmodelle möglich, dies wurde bereits in Mengel (2013) gezeigt. Ergebnisse aus den Simulationsexperimenten können direkt in die Erstellung der Geschäftsprozessmodelle zurückfließen und somit helfen, die Prozessmodelle auszuarbeiten. Durch die Simulation können im ersten Schritt Fehler in der Modellbeschreibung und kritische Prozesspfade identifiziert werden. Die erzeugten Kennzahlen helfen bei der Validierung der Prozessmodelle gemeinsam mit den Industriepartnern. Als Projektergebnis werden diese Kennzahlen in die Prozessdokumentation einfließen und somit eine Ausgangsbasis für künftige BPI-Vorhaben bilden.

7.1.3. Besondere Anforderungen im Bereich Offshore-Windparks

Die Instandhaltungsprozesse von Offshore-Anlagen sind im besonderen Maße durch ihre Systemumgebung beeinflusst. Typisch für die Instandsetzungsprozesse ist zum Beispiel, dass sie durch Fehler an den technischen Systemen ausgelöst werden. Der Zeitpunkt der Instandsetzungstätigkeiten und somit der korrigierenden Zustandsänderung an der technische Anlage beeinflusst den Zeitpunkt der nächsten Störung, und somit den Zeitpunkt des nächsten notwendigen Instandsetzungsprozesses. Eine kybernetische Kopplung zwischen den Prozessen und den technischen Systemen liegt vor.

Die Durchführbarkeit der Instandhaltungsarbeiten ist zudem abhängig von den aktuellen und prognostizierten Wetterbedingungen. Je nach Zugangssystem zu der Anlage und den verwendeten Transportfahrzeugen kann nur bis zu bestimmten Wellenhöhen an den Anlagen angelegt werden, um Mitarbeiter auf die Anlagen zu überführen. Zu starke Windgeschwindigkeiten oder eine zu geringe Sichtweite verhindert ebenfalls ein Anlegen und somit ein Übersetzen an der Anlage. Auch der Zugang mit Hilfe von Helikoptern ist dann nicht mehr möglich.

In den Wintermonaten herrschen besonders kräftige Windverhältnisse. Diese Monate sind daher einerseits besonders ertragreich, jedoch ist unter Umständen über mehrere Wochen hinweg kein Zugang zu den Anlagen möglich. Die Sommermonate sind potentiell weniger windig und bieten somit zwar weniger Ertrag, aber auch größere Zeitfenster für zu vollziehende Wartungs- und Instandsetzungsarbeiten. Eine Durchführung der benötigten Arbeiten in den Sommermonaten ist daher zu bevorzugen.

Das Beispiel ist daher sehr gut geeignet, um die Notwendigkeit und die Durchführbarkeit der Kopplung von domänenspezifischen Teilmodellen an Prozessmodelle der BPMN aufzuzeigen. Die maßgeblichen Faktoren für die Bewertung der Instandhaltungsprozesse sind dabei die Energieerzeugung, die Kosten für die Prozesse und die Risiken, die bei den Prozessen auftreten. Die Energieerzeugung ist dabei abhängig von den Windgeschwindigkeiten und der Funktionstüchtigkeit der Anlage.

7.1.4. Konzeption einer geeigneten Simulationssoftware

Die Zielsetzung der Simulationsexperimente ist, Kennzahlen zu den Instandhaltungsprozessen zu errechnen, welche dabei helfen, die Prozesse zu bewerten. Diese sollen im ersten Schritt dabei helfen, die Prozessmodelle des Leistungssystems in Gesprächen mit Industriepartnern zu validieren. Zum Abschluß des Projektes werden die Prozessmodelle in Form einer Dokumentation veröffentlich; die sich aus der Simulation ergebenden Kennzahlen werden dort integriert. Dies dient der Branche künftig als Kommunikationsgrundlage. Desweiteren sollen potentielle Fehler in den Modellbeschreibungen entdeckt

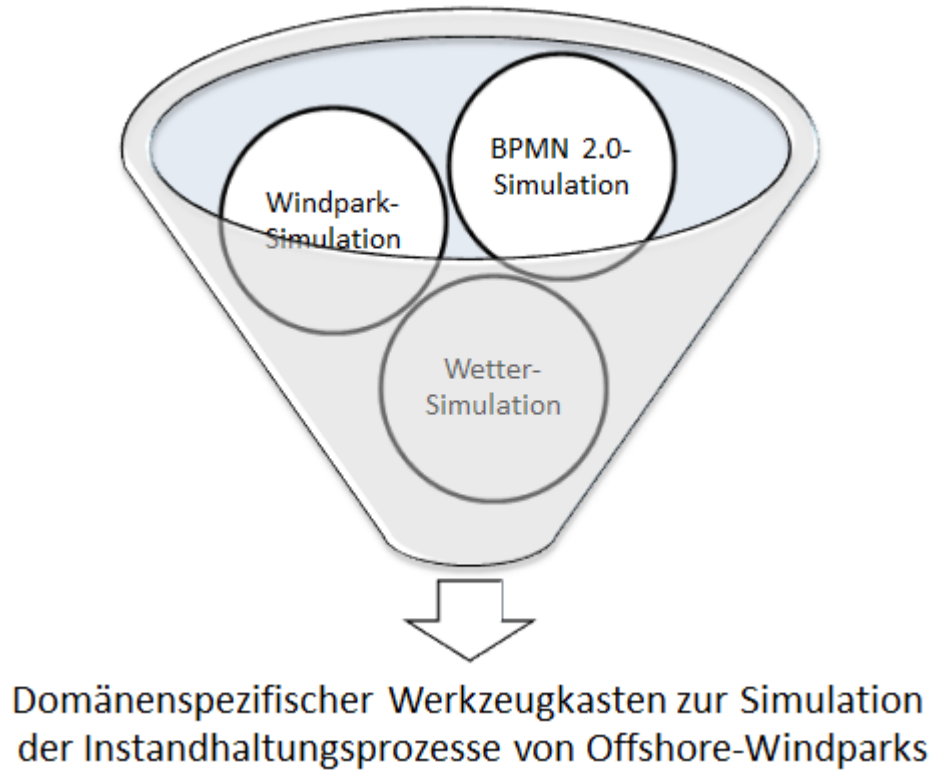


Abbildung 7.3.: Benötigte Komponenten zur Simulation der Instandhaltungsprozesse von Offshore-Windparkanlagen

werden, wie Prozesspfade die nicht oder im Vergleich zur Realität zu selten durchlaufen werden. Es ist zu betonen, dass es sich hier nicht um ein Workflow-System handelt, sondern um manuell durchgeführte Prozesse. Die garantierte Aufdeckung auch von besonders seltenen Deadlock-Situationen spielt eine untergeordnete Rolle, da das menschliche Handeln in solch einem Fall notfalls von gegebenen Prozessdefinitionen abweichen kann. Daher ist die Simulation zur Absicherung der Modellentwürfe trotz ihrer Schwächen beim Aufdecken von seltenen Deadlocks ein geeignetes Mittel. Die Dokumentation soll in späteren Projekten als Entscheidungshilfe für Optimierungsmaßnahmen und zur Findung geeigneter Instandhaltungsstrategien dienen. Die entworfenen Prozessalternativen sollen in diesem Fall ebenfalls mit Kennzahlen, die durch Simulation errechnet werden, bewertet werden können.

Um für solche Simulationsexperimente die Systemumgebung adäquat abbilden zu können, müssen geeignete Editoren für die Windparkanlagen, entsprechende Simulationskompo-

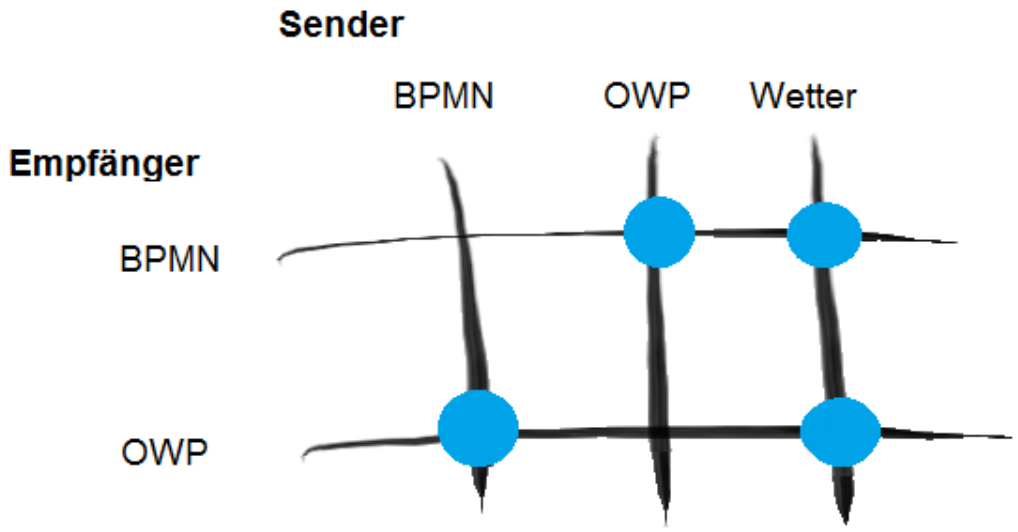


Abbildung 7.4.: Interaktionsbeziehungen zwischen Prozess-, Windpark-, Wetter- und Logistikmodellen

nenten und geeignete BPMN-Elemente zur Darstellung der Interaktion zwischen Prozessen und Windparkanlagen geschaffen werden. Auch muss der Wettereinfluss abgebildet werden. Hierfür werden ebenfalls ein Simulationsmodell und angepasste BPMN-Elemente benötigt. Wie in Abbildung 7.3 illustriert, müssen zwei Erweiterungen für das vorgestellte Rahmenwerk entwickelt werden, um eine domänenspezifische Software zur Simulation der Instandhaltungsprozesse zu schaffen. Diese Herangehensweise wurde bereits in Joschko et al. (2013b) und Joschko et al. (2013a) beschrieben. In den folgenden Abschnitten wird die Entwicklung dieser beiden Erweiterungskomponenten beschrieben.

Die Abbildung 7.4 zeigt eine Message-Cross-Bar und verdeutlicht die Richtung der Interaktionsbeziehungen zwischen den Prozessmodellen (BPMN), den Windparkmodellen (OWP) und dem Wettermodell. Zwischen Windparkmodellen und Instandhaltungsprozessen besteht eine bidirektionale Beziehung, welche entscheidend für den Zustand der Anlagen ist. Zwischen den Wettermodellen und den Instandhaltungsprozessen besteht nur eine unidirektionale Beziehung; einige Prozesse können nur ausgeführt werden, wenn die Wetterbedingungen dies erlauben. Auch zwischen Wettermodellen und Windparkmodellen besteht eine unidirektionale Beziehung; diese ist entscheidend für die erzeugte Strommenge in einem gegebenen Zeitintervall, sofern der Zustand der Anlage in diesem Intervall eine Stromerzeugung erlaubt. Das Wettermodell empfängt keine Nachrichten von den anderen Modellen.

7.2. Stochastischer Wettergenerator

Der folgende Abschnitt beschreibt die Entwicklung der Wetterkomponente. Die Anforderungen werden beschrieben; die Implementation des Modells und die dafür notwendige Datenbasis werden kurz erläutert. Im Zuge dieser Arbeit besonders relevant ist die Beschreibung der vorgenommen Erweiterungen an der BPMN-Bibliothek, der notwendigen Simulationsfabrik und des passenden Modellkopplers.

7.2.1. Anforderungen an die Wetterkomponente

Bei der Durchführung der Prozesse spielen verschiedene Wettereinflüsse eine Rolle. Unter einer bestimmten Temperaturgrenze sind Arbeitseinsätze an den Anlagen nicht mehr möglich. Die Feuchtigkeit auf den Rotorblättern gefriert zu Eis; durch die Rotation der Rotorblätter können diese Ablagerungen zu tödlichen Geschossen werden. Eine Annäherung an die Anlage ist unter diesen Umständen lebensgefährlich.

Derzeit werden von den Herstellern noch verschiedene Zugangssysteme zu den Anlagen erprobt. Erfolgt der Zugang per Schiff, so ist der Zugang nur bis zu bestimmten, vom Zugangssystem abhängigen Wellenhöhen möglich. Einige Anlagen bieten auch Helikopterlandeplattformen. Hier kann der Zugang durch zu starke Windgeschwindigkeiten gefährlich werden. Durch Nebel kann die Sicht so stark eingeschränkt werden, dass ebenfalls ein Andocken an den Anlagen zu gefährlich wäre.

Sofern während eines laufenden Arbeitseinsatzes die Wetterbedingungen schlechter werden, müssen die Arbeiter schnellstmöglich von der Anlage zurück auf das Transportmittel übersteigen. Wenn die Bedingungen sich zu schnell ändern, und kein gefahrloser Überstieg mehr möglich ist, muss der Kapitän die Arbeiter auf der Anlage zurück lassen. Auf allen Anlagen lagern daher Schlafsäcke und für mehrere Tage Proviant als Notfallsversorgung.

Die folgenden Wetterdaten sind daher besonders relevant für die Instandhaltungsprozesse an den Anlagen:

- Wellenhöhe
- Temperatur
- Sichtweite
- Windgeschwindgkeit

Es werden zwei Möglichkeiten vorgesehen, um zur Simulationszeit aus einem laufenden Prozess auf Wetterdaten zuzugreifen. Über eine BPMN-Aktivität können aktiv Wetterdaten abgefragt werden, zum Beispiel um abzubilden, dass beim Wetteramt die aktuellen Daten erfragt werden. Über ein Zwischenereignis kann ein Wetterumschwung abgebildet werden. So kann das Zwischenereignis beispielsweise als angeheftetes Ereignis an eine Aktivität geheftet werden, welche wiederum den Einsatz an der Anlage abbildet. Über eine Python-Bedingung wird beschrieben, unter welchen Wetterbedingungen das Ereignis auslöst. Dies kann beispielsweise abhängig vom verwendeten Transportmittel oder dem Zugangssystem der Anlage sein. Löst das Ereignis aus, so muss die Aktivität abgebrochen und dem Prozesspfad hinter dem angehefteten Ereignis gefolgt werden.

Sofern die Simulationsmethode zur Planung von Einsätzen in der operativen Phase genutzt wird, ist hierfür eine Anbindung an kurzfristige, meteorologische Wetterprognosemodelle notwendig. Bei einer strategischen Analyse der Prozessmodelle, wie sie in SystOp erfolgen soll, sollen mehrere Betriebsjahre am Stück untersucht werden. Prognosemodelle über einen solch langen Zeitraum sind nicht möglich. Daher wird ein stochastischer Wettergenerator benötigt, welcher realistische Wetterdaten für einen langen Zeitraum generiert. Hierdurch können Experimente getätigt werden, die die gesamte Lebensphase eines Windparks von insgesamt 20 Jahren abbilden.

7.2.2. Funktionsweise von stochastischen Wettergeneratoren

Stochastische Wettergeneratoren generieren realistische Wetterdatensätze auf der Grundlage von historischen Daten. Diese synthetischen Zeitreihen können benutzt werden, um das Verhalten eines Modells unter verschiedenen Wetterbedingungen zu erproben. Ein seit längerer Zeit etabliertes Anwendungsgebiet hierfür ist die Agrarindustrie (vgl. Racsko, P.; Semenow (1989)), weitere sind das Umweltmanagement oder Hydrologie (vgl. Res et al. (1998)). Auch im Rahmen der langfristigen Prognose zur Windenergieproduktion werden solche Modelle mittlerweile benutzt (vgl. Feijóo et al. (2011)).

Die generierten Daten sind den historischen Daten ähnlich und daher näherungsweise realistisch. In der Regel kann konfiguriert werden, wie stark die Daten von den historischen Daten abweichen dürfen. Je größer der Zeitraum ist, für den die historischen Daten vorliegen, um so bessere Werte können generiert werden, ohne dabei den historischen Daten zu ähnlich zu sein.

Das Bundesamt für Umwelt-, Naturschutz- und Reaktorsicherheit (BMU) fördert drei Messplattformen in Nord- und Ostsee, um Forschungen über die Auswirkungen der Offshore-Industrie auf die maritime Umwelt und die Weiterenwticklng von Offshore-Technologien zu unterstützen. Das Projekt „Forschungsplattformen in Nord- und Ostsee“

(FINO) wurde im Jahr 2002 gestartet. Seitdem werden umfangreiche physikalische physikalische, hydrologische, chemische, biologische, meteorologische und ozeanographische Daten erfasst (BSH (2014)).

Die Verbundparter des SystOp-Projekts haben Zugriff auf die FINO-Daten erhalten. Es konnten historisce Daten im Format einer CVS-Datei (Comma Seperated Values) zur Temperatur, zur Wellenhöhe und zur Windgeschwindigkeit aus der Datenbank extrahiert werden, welche als Grundlage für die Implementation eines stochastischen Wettergenerators dienten. Leider liegen keine Daten zur Sichtweite vor, daher muss auf die Betrachtung der Sichtweite im Rahmen des SystOp-Projektes verzichtet werden.

7.2.3. Integration eines stochastischen Wettergenerators

Als Grundlage für die Entwicklung wurde die OpenSource-Implementation „CLIMA Weather Generator“ des Research Centre for Industrial Crops - Agriculture Research Council genutzt (CRA-CIN (2014)). Es handelt sich um eine .NET-Bibliothek, die im Rahmen des SystOp-Projektes mit einer Benutzeroberfläche versehen und in das Empinia-Rahmenwerk integriert werden konnte. Diese Bibliothek implementiert verschiedene wissenschaftliche Strategien, um auf der Basis von historischen Daten Wetterdaten zu generieren. Die verschiedenen Berechnungs-Strategien erfordern verschiedene Eingabeparameter, so dass in Abhängigkeit vom zur Verfügung stehenden Datenmaterial entschieden werden kann.

Da diese Strategien für die Agrarindustrie entwickelt wurden, können sie zwar Wind-, Temperatur- und Niederschlagsdaten generieren, jedoch keine Wellendaten. Hier wurden die historischen Daten dahingehend analysiert, welche der vorhandenen Berechnungsstrategien am ehesten passend ist. Die durchschnittliche Wellenhöhe beträgt 1,43m, die maximale Wellenhöhe im betrachteten Zeitraum lag bei 9,77m und die minimale Wellenhöhe 0,14m. In der Visualisierung der Daten ist deutlich zu erkennen, dass die Wellenhöhe mit der Windgeschwindigkeit korrespondiert und bei gleichbleibend starken Wind zunehmend größer wird. Bisher konnte für dieses Problem jedoch noch keine zufriedenstellende Lösung gefunden werden. Für die Vorstellung des prototypischen Fallbeispiels tut dies jedoch keinen Abbruch. Im folgenden werden für die Wellenhöhe historische Daten verwendet.

7.2.4. Benutzeroberfläche zur Generierung von Wetterdaten

Die Benutzeroberfläche zur Generierung von Wetterdaten beinhaltet ein Assistenten-System, welches den Anwender durch die erforderlichen Schritte leitet. Der obere Teil der Abbildung D.1 auf S. 196 im Anhang zeigt das Assistentensystem. Hier werden

der nächste erforderliche Schritt benannt und nähere Informationen zur Erläuterung bereitgestellt. Die Schritte werden vom Benutzer in dem Abschnitt unter dem Assistentensystem ausgeführt. Dasjenige Steuerelement, an dem die nächste Eingabe erfolgen soll, wird mit einem roten Kreis und der Nummer des Schrittes markiert.

Im ersten Schritt muss ausgewählt werden, welche Art von Wetterdaten generiert werden sollen (Wellenhöhe, Windgeschwindigkeit, Temperatur). Im unteren Teil der Benutzeroberfläche werden zu dem aktuellen Schritt Zusatzinformationen angezeigt. Im ersten Schritt erscheinen Informationen, in welchen Einheiten die jeweiligen Daten vorzuliegen haben.

Im zweiten Schritt (siehe Abbildung D.2 auf S. 197) müssen der Zeitraum und die Frequenz ausgewählt werden, für die die Daten generiert werden sollen. Zur Simulation der kompletten Lebensphase eines Offshore Windparks wird ein Zeitraum von 20 Jahren benötigt. Eine Frequenz von einem Datensatz pro Stunde ist angebracht, um die wechselnden Wetterbedingungen im Laufe eines Tages abbilden zu können.

Als drittes muss eine Berechnungsstrategie ausgewählt werden (siehe Abbildung D.3 auf S. 198). Im unteren Teil der Abbildung ist ein Infofeld zu sehen, in dem zu jeder Berechnungsstrategie nähere Informationen angezeigt werden. Dies ist in der Regel ein Verweis auf die wissenschaftliche Publikation, in der die entsprechende Strategie vorgestellt wurde. Für die Generierung von Wind- und Temperaturdaten wurde die HWMitchel-Strategie als passend angenommen, welche in Mitchel et al. (2000) beschrieben wird.

Nachdem diese Schritte durchlaufen wurden, können entsprechende Datensätze generiert werden. Abbildung D.4 auf S. 199 zeigt eine Übersicht über die generierten Werte für ein Windjahr. Diese Ansicht unterstützt den Benutzer bei der Auswahl einer geeigneten Berechnungsstrategie. So können die Daten mit den historischen Daten oberflächlich verglichen werden, bis eine Strategie gefunden wurde, welche zufriedenstellende Werte liefert. Desweiteren entsteht ein erster Eindruck, welche Zeiträume bei der Durchführung von Instandhaltungseinsätzen kritisch sein könnten. So ist in der Abbildung zu sehen, dass in einem bestimmten Intervall die Wind- und Wellenstärke besonders hoch ist; hierbei handelt es sich um die bereits erwähnten ertragreichen, aber für die Instandhaltung kritischen Wintermonate.

7.2.5. Implementation und Anmeldung der Erweiterungsbibliothek

In diesem Abschnitt wird exemplarisch die Implementation des Wetterereignisses aufgezeigt. Die Implementation einer domänenspezifischen Aktivität wird in Abschnitt 7.3.4 anhand der Windparkkomponente beschrieben. Neben dem logischen BPMN-Element muss auch die grafische Visualisierung im BPMN-Editor konfiguriert werden. Außerdem

muss eine Modellkonfiguration, eine Simulationsfabrik und ein Modellkoppler bereitgestellt werden, deren Implementation in diesem Abschnitt beschrieben wird.

Der Quellcode E.1 auf S. 201 zeigt exemplarisch die Implementation des domänenspezifischen Zwischenereignisses `BPMNWeatherConditionEvent`. Es erbt von der in Abschnitt 6.4.1 vorgestellten Klasse `DomainSpecificIntermediateCatchEvent`. Bereitzustellen ist die Methode `TriggerEvent`, welche eine Zeichenkette entgegen nimmt (zum Beispiel `'wind=12; wave=5'`), die die Variablennamen und Werte zu den aktuellen Wetterkonditionen beinhaltet. Im Prozessmodell wird bei Verwendung des Elementes eine Bedingung angegeben (zum Beispiel `'wind>12 OR wave=5'`). Über die Python-Engine wird die Bedingung in der `TriggerEvent`-Methode überprüft, nur im Erfolgsfall wird die Methode `Release` aufgerufen und der wartende Teilprozess reaktiviert.

Um die `TriggerEvent`-Methode anstoßen zu können, wird als sendendes Gegenstück ein DESMO-J-Ereignis als Bestandteil des Wettermodells benötigt (vgl. Klasse `WeatherConditionChangedEvent` im Quellcode E.2 auf S. 202). Dieses schlägt in einer Liste des Wettermodells nach, welche `BPMNWeatherConditionEvent` vorhanden sind und ausgelöst werden müssen. Das Ereignis wird zu denjenigen diskreten Zeitpunkten aufgerufen, zu denen es Änderungen in der Wetterlage gibt. Im Wettermodell wird zu diesem Zweck hinterlegt, für welche Intervalle die Daten vorliegen. Die `EventRoutine` des Ereignisses nimmt dabei als Entitäten die Zeitreihen zu Windgeschwindigkeit, Wellenhöhe und Temperatur entgegen und erzeugt über die Methode `CurrentValueToExpresion` die an das `WeatherConditionEvent` zu übergebende Zeichenkette. Mit jedem Auslösen des Ereignisses wird das nächste Auftreten über den Aufruf `ScheduleNextEvent` in den DESMO-J-Scheduler eingetragen.

Damit das Ereignis im BPMN-Editor verfügbar wird, muss ein Visualisier bereitgestellt werden (vgl. Quellcode E.3 auf S. 203). Diese Klasse `WeatherEventVisualiser` erbt von den für Ereignisse bereits zur Verfügung stehenden Visualisier `BPMN.UI.Visualisation-.Event`. Sie überschreibt die Methoden `CanVisualize` und `CanInitialize`, damit der Visualisierer vom BPMN-Editor nur für das vorgesehene Wetterereignis verwendet wird. Die Methode `Clone` muss mit einem Konstruktoraufruf auf diese Klasse zur Verfügung gestellt werden, um die Bereitstellung des Elements in der Toolbox des Editors zu ermöglichen. Durch die Methode `SetSymbolForEvent` wird ein passendes Symbol zur Verfügung gestellt.

Diese Visualisierungsklasse muss über die Bundle.xml-Datei der Komponente an dem BPMN-Editor angemeldet werden. Der hierfür notwendige Code ist in Quellcode 204 zu sehen. Zunächst wird die visualisierende Klasse und das hierdurch visualisierte logische Element bestimmt. Anschließend wird eine Konfiguration bereitgestellt, über welche Eigenschaften der Visualisierung wie Hintergrundfarbe und Linienstärke angegeben werden können.

Zur Anmeldung der restlichen Bestandteile der Erweiterungsbibliothek wird die in Abschnitt 6.3 eingeführte `ModelTypeConfiguration` zur Verfüggung gestellt (vgl. Quellcode E.5 auf S. 205). Hierzu gehören die Kommandos zum Laden und zur Anzeige von Modellen, die Dateiendung sowie die Simulationsfabrik und der Modellkoppler. Die Quellcodes dieser Kommandos sind sehr trivial, aber werden aufgrund der von Empinia abhängigen Plattformspezifität nicht näher erläutert. Relevanter sind die Simulationsfabrik und der Modellkoppler.

Die Simulationsfabrik (vgl. Quellcode E.6 auf S. 207) nimmt die für das Simulationsexperiment ausgewählten Wetterdaten entgegen und generiert in der Methode `CreateSimulationModel` ein DESMO-J-Modell. Für jede der Datenreihen (Wind, Wellen, Temperatur) wird dafür eine DESMO-J-Entität erzeugt. Das Ereignis `WeatherConditionEvent` wird initial zum Startzeitpunkt der Simulation vorgemerkt. Da das Ereignis sämtliche Folgeereignisse anstößt, ist die Abarbeitung des Wettermodells somit in Gang gesetzt.

Der Modellkoppler muss das Wetterereignis in die bestehenden Prozessmodelle integrieren (vgl. Quellcode E.7 auf S. 209). Die Simulationsfabrik für die BPMN-Prozesse hatte die domänenspezifische Ausprägung noch ignoriert. Der Experimentplaner jedoch erkennt, dass eigentlich eine domänenspezifische Ausprägung vorliegt, und ruft die entsprechenden Methoden des Modellkopplers der Wetterkomponente auf. Dieser `ModelConnector` erbt vom `AbstractModelConnector`. Da hier lediglich ein Zwischenereignis zur Verfügung gestellt und verknüpft werden muss, sind die Methodenrümpfe aller Methoden außer `ConnectDomainSpecificIntermediateEvent` leer. In `ConnectDomainSpecificIntermediateEvent` wird das `BPMNWeatherConditionEvent` instanziiert und an das bereits vorhandene Zwischenereignis angeheftet. Dem Wettermodell wird eine Referenz auf das `BPMNWeatherConditionEvent` übergeben, so dass dieses eine Liste bereithält, auf welches das `WeatherConditionChangedEvent` zur Laufzeit zugreifen kann.

7.3. Windparkerweiterung

Im folgenden Abschnitt wird die Entwicklung der Windparkerweiterung beschrieben. Erneut werden die Anforderungen an die Modelldefinition, eine Benutzeroberfläche zur Pflege dieser Modelldefinitionen, die möglichen Interaktionsbeziehungen mit den Prozessmodellen und die daraus resultierenden BPMN-Elemente sowie die erstellte Simultionsfabrik und der Modellkoppler vorgestellt.

Als Vorlage für das Beispielmodell dient der Forschungswindpark Alpha Ventus, da hierzu besonders viele Daten öffentlich zugänglich sind. Alpha Ventus wird von der Deutschen Offshore- Testfeld und Infrastuktur GmbH (DOTI) betrieben, welche von der EWE AG, der E.ON Energy Projects GmbH und Vattenfall Europe gemeinsam gehalten wird.

7.3.1. Interaktionsbeziehungen und Anforderungsanalyse

Im Rahmen der Instandhaltung finden Wartungs- und Instandsetzungsprozesse statt. Die Häufigkeit der regulären Wartungseinsätze wird von den Herstellern der Anlagen vorgegeben und kann sich somit von Anlage zu Anlage unterscheiden. Diese Wartungspläne müssen innerhalb des Windparkmodells angelegt werden. Zu den vorgesehenen Zeitpunkten werden hierdurch Prozessinstanzen gestartet, welche je nach Detaillevel der Prozessmodelle zunächst einen Planungsprozess oder direkt den Wartungseinsatz beschreiben. Hierfür muss ein domänenspezifisches Startereignis bereitgestellt werden, welches zugleich Informationen übermittelt, an welchen Anlagen welche Arbeiten durchgeführt werden müssen.

Instandsetzungseinsätze werden durch Fehler an den Anlagen ausgelöst. Hierfür müssen an den Anlagen Zwischenankunftszeiten für das Auftreten von Fehlern hinterlegt werden. Die Anlage sendet in solch einem Fall ein Fehlersignal an den Leitstand und löst somit einen Prozess aus. Auch hier wird ein domänenspezifisches Startereignis benötigt.

Die Prozessbeschreibungen aus dem SystOp-Projekt umfassen die Planung, die Vorbereitung, die Hin- und Rückfahrt, sowie den eigentlichen Einsatz an der Anlage. Während einer Wartung muss die Anlage vorübergehend deaktiviert werden, bei Instandsetzungseinsätzen wird in Abhängigkeit von der Schwere des Fehlers bereits beim Auftreten des Fehlers oder nur während der Durchführung der Arbeiten die Anlage deaktiviert. Die Aktivitäten der Prozessmodelle wirken sich somit auf den Zustand einer konkreten Anlage aus. Es wird eine Aktivität zur Deaktivierung der Anlage benötigt, eine Aktivität zur Behebung des Fehlers und eine Aktivität zur Reaktivierung der Anlage.

Der Wartungsbedarf und die Fehlersignale gehen von konkreten Anlagen oder Anlagenkomponenten aus. Diese Entitäten treten als Sender zu den domänenspezifischen, empfangenden Startereignissen auf. Damit an späterer Stelle in dem Prozess die domänenspezifischen Aktivitäten sich auf genau diese Entitäten beziehen können, wird ein Artefakt benötigt, in welchem die Referenzen auf den Sender gespeichert werden. Die domänenspezifischen Startereignisse und die domänenspezifischen Aktivitäten werden zu diesem Zweck mit diesen Artefakten assoziiert.

Hieraus ergeben sich folgende benötigte, domänenspezifische BPMN-Elemente:

- WindparkMaintenanceStartEvent
- WindparkFailureSignakStartEvent
- WindparkStateManipulationActivity
- WindparkSwitchOffActivity
- WindparkSwitchOnActivity

Die Energieproduktion der Windparkanlagen ist abhängig von dem Zustand der Anlage und der aktuellen Windstärke. Daher sollen die Informationen über die aktuelle Windgeschwindigkeit auch an die Windparkanlagen übertragen werden können - ohne das hierfür Geschäftsprozessmodelle definiert werden müssen.

7.3.2. Benutzeroberfläche zur Definition von Windparkmodellen

Zur Definition der Windparks, der Windanlagen und der Anlagenkomponenten werden mehrere Views zur Eingabe der notwendigen Daten angeboten.

Die Abbildung F.1 auf S. 212 im Anhang zeigt eine Eingabemaske zur Definition eines Windparks. Hier können zunächst einige Meta-Daten wie der Name des Parks, die geografische Position und eine Kurzbeschreibung eingegeben werden. Diese Daten sind lediglich Zusatzinformationen, welche dem Benutzer bei der Identifikation der Modelle helfen sollen.

Für die Simution ist der untere Teil der View relevant. Hier können verschiedene Turbinentypen ausgewählt und deren Anzahl zugeordnet werden. Alpha Ventus besteht aus sechs Anlagen vom Typ Repower 5M und aus sechs Anlagen vom Typ AREVA Wind M5000; beides sind Anlagen mit 5 MW Nennleistung. Mit nur zwölf Anlagen ist dieser Forschungswindpark im Vergleich zu kommerziellen Windparks, welche gewöhnlich aus 80 Anlagen bestehen, sehr klein. Jedoch können über die Benutzeroberfläche beliebige Windparks konfiguriert werden.

Die Anlagentypen die einem Windpark zugeordnet sind, werden über zwei weitere Views definiert. Falls für andere Windparkmodelle bereits Anlagen angelegt wurden, können die Anlagen importiert werden. Ansonsten können über die in Abbbildung F.2 auf S. 213 dargestellte Benutzeroberfläche Meta-Daten zu den Anlagentypen, zum Beispiel Name, Hersteller und Beschreibung angegeben werden. Eine wesentliche Kennzahl zur Bewertung der Instandhaltungsprozesse ist die über ein Windjahr hinweg erzeugte Energiemenge. Hierfür muss für eine gegebene Windgeschwindigkeit in einem bestimmten Intervall die jeweils erzeugte Energie errechnet werden können.

Diese ist abhängig von einigen technischen Daten zur Anlage, die vom Benutzer ebenfalls über diese Benutzeroberfläche bereitgestellt werden müssen. In Byon et al. (2011) werden eine passende Formel und die dafür benötigten Parameter vorgestellt. Die erzeugte Energie P errechnet sich dabei aus der Nennleistung P_r, der Einschaltgeschwindigkeit V_{ci}, der Nenngeschwindigkeit V_r, der Abschaltgeschwindigkeit V_{co} und der aktuellen Windgeschwindigkeit V wie folgt:

$$P = \begin{cases} 0, & \text{falls } 0 \leq V < V_{ci} \\ P_r(a + b \cdot V + c \cdot V^2) & \text{falls } V_{ci} \leq V < V_r \\ P_r, & \text{falls } V_r \leq V < V_{co} \\ 0, & \text{falls } V_{co} \leq V \end{cases}$$

Byon et al. (2011)

Die Parameter a und b, die zur Errechnung der Energieerzeugung bei Windgeschwindigkeiten zwischen Einschalt- und Nenngeschwindigkeit benötigt werden, ergeben sich dabei aus eben dieser Einschalt- und Nenngeschwindigkeit:

$$a = \frac{1}{(V_{ci} - V_r)^2} \left[V_{ci}(V_{ci} + V_r) - 4V_{ci}V_r \left(\frac{V_{ci} + V_r}{2V_r} \right)^3 \right]$$

$$b = \frac{1}{(V_{ci} - V_r)^2} \left[4(V_{ci} + V_r) \left(\frac{V_{ci} + V_r}{2V_r} \right)^3 - (3V_{ci} + V_r) \right]$$

Byon et al. (2011)

Die Parameter P_r, V_{ci}, V_r und V_{co} müssen daher für jede verwendete Turbinenart vom Benutzer eingegeben werden. Diese Parameter unterscheiden sich bei verschiedenen Turbinen und sind in der Regel öffentlich zugänglichen Datenblättern der Anlagenhersteller zu entnehmen. Eine Eingabemaske hierfür wurde in die Erweiterungskomponente integriert. Der Benutzer erhält nach Eingabe der Parameter eine Visualisierung der Energieerzeugungskurve, wie in Abbildung 7.5 zu sehen.

Um die Ausfallwahrscheinlichkeiten von Komponenten angeben zu könne, welche gleichzusetzen sind mit den Zwischenankunftszeiten der Startereignisse der Instandhaltungsprozesse, wird eine Oberfläche zur Definition der Anlagenkomponenten bereitgestellt (siehe Abbildung F.3 auf S. 214 im Anhang). Hier kann eine beliebig detaillierte Komponentenstruktur für jeden Anlagentyp hinterlegt werden. Die Hauptkomponenten einer Windkraftanlage (die Rotorblätter, das Pitch System, das Rotorlager, das Getriebe, der Generator, die elektronischen Kontrollsysteme und das Hydrauliksystem) werden als Standardkomponenten vorgegeben, der Benutzer kann jedoch weitere Komponenten hinzufügen.

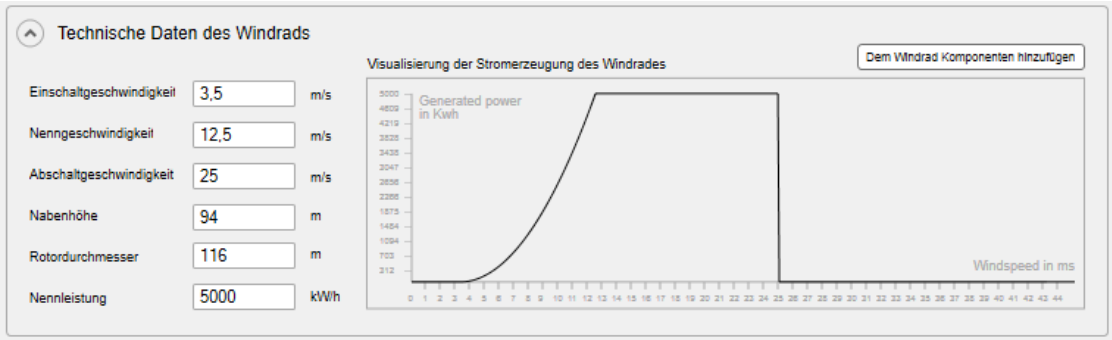


Abbildung 7.5.: Parametereingabe zur Energieerzeugung

Zu jeder Komponente können die Stückkosten für einen Austausch und die Zwischenankunftszeit eines Ausfalls über eine stochastische Verteilung angegeben werden. Die Ausfallwahrscheinlichkeit von technischen Anlagen kann näherungsweise über eine Weibullverteilung beschrieben werden. Da für Offshore-Anlagen noch nicht ausreichend Daten vorliegen, musste hier auf Daten von Onshore-Anlagen zurückgegriffen werden, welche aufgrund ihrer Größe und der durch die geografische Lage gegebenen Umwelteinflüsse am ehesten vergleichbar mit Offshore-Anlagen sind. Diese Daten wurden von Industriepartnern aus dem SystOp-Projekt ausgewählt und bereitgestellt. Es handelt sich also nicht um die tatsächlichen Ausfallswahrscheinlichkeiten der abgebildeten Anlagen, da diese Daten erst in ein paar Jahren zur Verfügung stehen werden.

Desweiteren muss ein Wartungsplan für die Anlage hinterlegt werden. Zur Wartung gehören regelmäßig wiederkehrende Arbeiten wie der Austausch von Schmiermitteln oder Filtern. Typischerweise finden diese in halbjährlichen Abständen statt, wobei abwechselnd kleinere und größere Wartungen durchgeführt werden müssen. Auch kann es sein, dass alle vier Jahre ein noch größerer Wartungsaufwand angesetzt wird. Zu diesem Zweck können verschiedene Wartungsintervalle angelegt werden (rechter Teil der Abbildung F.4 im Anhang). Jedem der Wartungsintervalle wird ein Zeitaufwand zugeordnet. Eine Zuordnung der Komponenten zu den Wartungsintervallen ist bereits in der Benutzeroberfläche vorgesehen (linker Teil der Abbildung), spielt zum gegenwärtigen Zeitpunkt im SystOp-Projekt aber noch keine Rolle.

7.3.3. Domänenmodell und Persistierung

Die über die Modelleditoren getätigten Angaben werden in einer serialisierbaren Objektstruktur gehalten und als XML-Datei persistiert. Wie in Abbildung 7.6 zu sehen, werden hierfür vier Klassen bereitgestellt. Das `WindparkModel` hält die Metainformationen wie Name und Beschreibung zum Modell. Es hält eine generische Liste mit Objekten

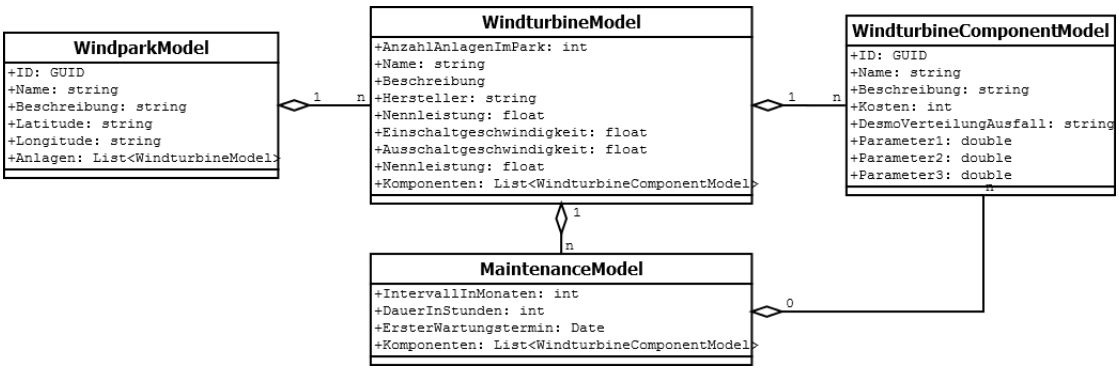


Abbildung 7.6.: Klassendiagramm der Modelldefinition eines Windparks

vom Typ `WindturbineModel`. Hier werden ebenfalls Metainformationen wie Name und Hersteller hinterlegt, sowie die Anzahl der Anlagen von diesem Typ, die sich in dem entsprechenden Windpark befinden. Die Anlagen halten eine generische Liste mit Objekten vom Typ `WindturbineComponentModel`. Hier wird der Name der Komponente und die Zwischenankunftszeit für Fehlerereignisse als stochastische Verteilung hinterlegt. Zu dieser Ausfallwahrscheinlichkeit gehört der Name der Verteilung sowie eine Liste von Parametern. Desweiteren hält jeder Anlagentyp eine Liste mit Wartungsplänen, welche vom Typ `MaintenanceModel` sind. Hier wird ein Intervall angegeben, in welchen Abständen die Wartungseinsätze erfolgen müssen. Die erforderliche Anzahl von Arbeitsstunden für den Wartungseinsatz wird ebenfalls hinterlegt. Die Verknüpfung des Wartungsplans mit den Anlagenkomponenten ist optional.

7.3.4. Implementation und Anmeldung der Erweiterungen

Da die Implementation der Wartungsmeldungen äquivalent und somit redundant zu den Fehlermeldungen ist, wird im folgenden nur der Entwicklungsaufwand für die Fehlerereignisse, das Windparkartefakt und die Windparkaktivität beschrieben.

Der Quellcode G.1 auf S. 217 zeigt die Implementation eines Windpark-Artefaktes. Dieses `BPMNWindparkArtefakt` wird abgeleitet von `DomainSpecificArtefact`. Der Entwickler muss im Konstruktor lediglich die Art der Warteschlange bestimmen, in der die Windparkentitäten (Anlagen oder Anlagenkomponenten) eingereiht werden können. Hier wurde eine FIFO-Queue gewählt.

Das `BPMNWindparkFailureReceiveEvent` (vgl. Quellcode G.2 auf S. 217) ist ähnlich dem bereits gezeigten Wetterereigniss. Jedoch handelt es sich hierbei um ein Start-, nicht um ein Zwischenereignis, daher erbt es vom `DomainSpecificStartEvent`. Auch hier muss die

`TriggerEvent`-Methode bereitgestellt werden. Hier wird ohne vorherige Prüfungen von Bedingungen die `Release`-Methode aufgerufen, welche den Prozess startet. Außerdem wird die Referenz des Senders der Nachricht in den assoziierten Artefakten abgelegt, sofern denn welche vorhanden sind.

Die Implementation der `WindparkStateManipulationActivity` wird in Quellcode G.3 auf S. 218 gezeigt. Diese erbt von der Klasse `DomainSpecificActivity` und muss verschiedene EventHandler überschreiben. In diesem Fall soll erst zum Abschluss der Aktivität diejenige Komponente, welche als defekt gemeldet wurde und somit über das `BPMNWindparkArtifact` zugreifbar ist, wieder in ihren Ursprungszustand zurückversetzt werden. Dies geschieht über den Methodenaufruf in Zeile 21. Die aufgerufene Methode wird den nächsten Ausfallzeitpunkt der Komponente festlegen und prüfen, ob nun alle Komponenten intakt sind und die Anlage wieder in den stromproduzierenden Zustand wechseln kann.

Als sendendes Gegenstück zum `BPMNWindparkFailureReceiveEvent` muss ein DESMO-J-Ereignis `WindparkFailureEvent` in das Windparkmodell integriert werden (vgl. Quellcode G.4 auf S. 219). Die `EventRoutine` holt sich vom Windparkmodell alle angemeldeten `BPMNWindparkFailureReceiveEvent`, um an ihnen die `TriggerEvent`-Methode aufzurufen.

Für die drei BPMN-Elemente müssen Visualisierungsklassen bereitgestellt werden. Diese erben von den Klassen `BPMN.UI.Visualisation.Event`, `BPMN.UI.Visualisation-.Activity` und `BPMN.UI.Visualisation.Artifact`. Über die Methode `SetSymbolFor-Event` muss eine passende Ressource als Symbol zur Verfügung gestellt werden. Diese Visualisierungsklassen müssen an der Editorkomponente für BPMN-Modelle angemeldet werden (vgl. Quellcode G.5 auf S. 220). Hier wird eine Zuordnung der Visualisierungsklassen zu den entsprechenden BPMN-Elementen hinterlegt. Da hier nur die Klassennamen der BPMN-Elemente angegeben werden können, diese an dieser Stelle jedoch nicht über die Eigenschaft `DomainSpecificType` zusätzlich eingeschränkt werden können, müssen an den Visualisierungsklassen jeweils die Methoden `CanVisualize` und `CanInitialize` so überschrieben werden, dass sie `true` nur für diejenigen BPMN-Elemente zurückgeben, wenn das logische Element über einen passenden Eintrag im Erweiterungsattribut `DomainSpecificType` verfügt.

Über eine `ModelTypeConfiguration` werden erneut die Kommandos zum Laden und zum Öffnen des in Abschnitt 7.3.2 vorgestellten Editors angemeldet (vgl. Quellcode G.6 auf S. 222). Außerdem werden die Simulationsfabrik und der Modellkoppler bekannt gemacht.

Die Simulationsfabrik erzeugt aus der Modelldefinition aus Abschnitt 7.3.3 ein DESMO-J-Simulationsmodell. Zu jeder Klasse aus der Modelldefinition wird hierfür eine passende DESMO-J-Entität erzeugt. Da die Relationen zwischen diesen Entitäten äquivalent zu

den in Abbildung 7.6 gezeigten Relationen sind, wird auf eine erneute Abbildung des Objektmodells an dieser Stelle verzichtet. Bei den Anlagenkomponenten müssen die Ausfallwahrscheinlichkeiten hinterlegt werden. Dafür muss der in der Prozessdefinition als Zeichenkette angegebene Verteilungstyp und seine Parameter ausgelesen werden, so dass eine Verteilung aus dem DESMO-J-Rahmenwerk instanziiert werden kann. Die Objekte, welche die Anlagenkomponenten im Simulationsmodell repräsentieren, werden bei der Initialisierung aus der ihr zugehörigen Verteilung den ersten Ausfallzeitpunkt bestimmen und zu diesem Zeitpunkt ein `WindparkFailureEvent` auf der Ereignisliste vormerken.

Der Modellkoppler aus Quellcode G.8 auf S. 227 erbt von `AbstractModelConnector` und muss Methoden bereitstellen, um die domänenspezifischen BPMN-Elemente zu instanziieren und mit den Windparkmodellen zu verknüpfen. Die Methode `ConnectDomainSpecificStartEvent` erzeugt das `BPMNWindparkFailureReceiveEvent` und hinterlegt eine Referenz in den Windparkmodellen, so dass das `WindparkFailureEvent` hierauf zugreifen kann. Die Methode `ConnectDomainSpecificActivity` erzeugt ein `WindparkStateManipulationActivity`. Da die Referenz auf die Windparkentitäten erst zur Laufzeit aus den Windparkartefakten geholt wird, ist hier keine weitere Referenzierung notwendig. Die Methode `ConnectDomainSpecificArtefact` erzeugt dementsprechend ein `BPMNWindparkArtefakt`. Auch hier ist keine weitere Referenzierung notwendig, da erst zur Laufzeit Referenzen von dem `BPMNWindparkFailureReceiveEvent` hinterlegt werden.

Desweiteren muss der Modellkoppler das Windparkmodell mit dem Wettermodell verknüpfen, um die Berechnung der Energieerzeugung in Abhängigkeit zu der Windstärke vorzunehmen. Dies geschieht in der Methode `ConnectDomainSpecificActivity`. Zu diesem Zweck wurde das Property `RequiredModelTypes` überschrieben, so dass der Experimentplaner nicht nur die Prozessmodelle, sondern auch die Wettermodelle an den Modellkoppler übermittelt.

7.4. Anwendung der Komponenten

Die Anwendung der Erweiterungskomponenten wird an einem Beispiel demonstriert, mit welchem die Funktionsweise nachvollzogen werden kann. Es wird gezeigt, dass die Wetter- und Windparkereignisse sich auf die Prozessinstanzen und die Aktivität sich auf den Zustand der Anlagen auswirken. Die mit Hilfe eines Simulationsexperimentes errechneten Kennzahlen werden vorgestellt und erläutert.

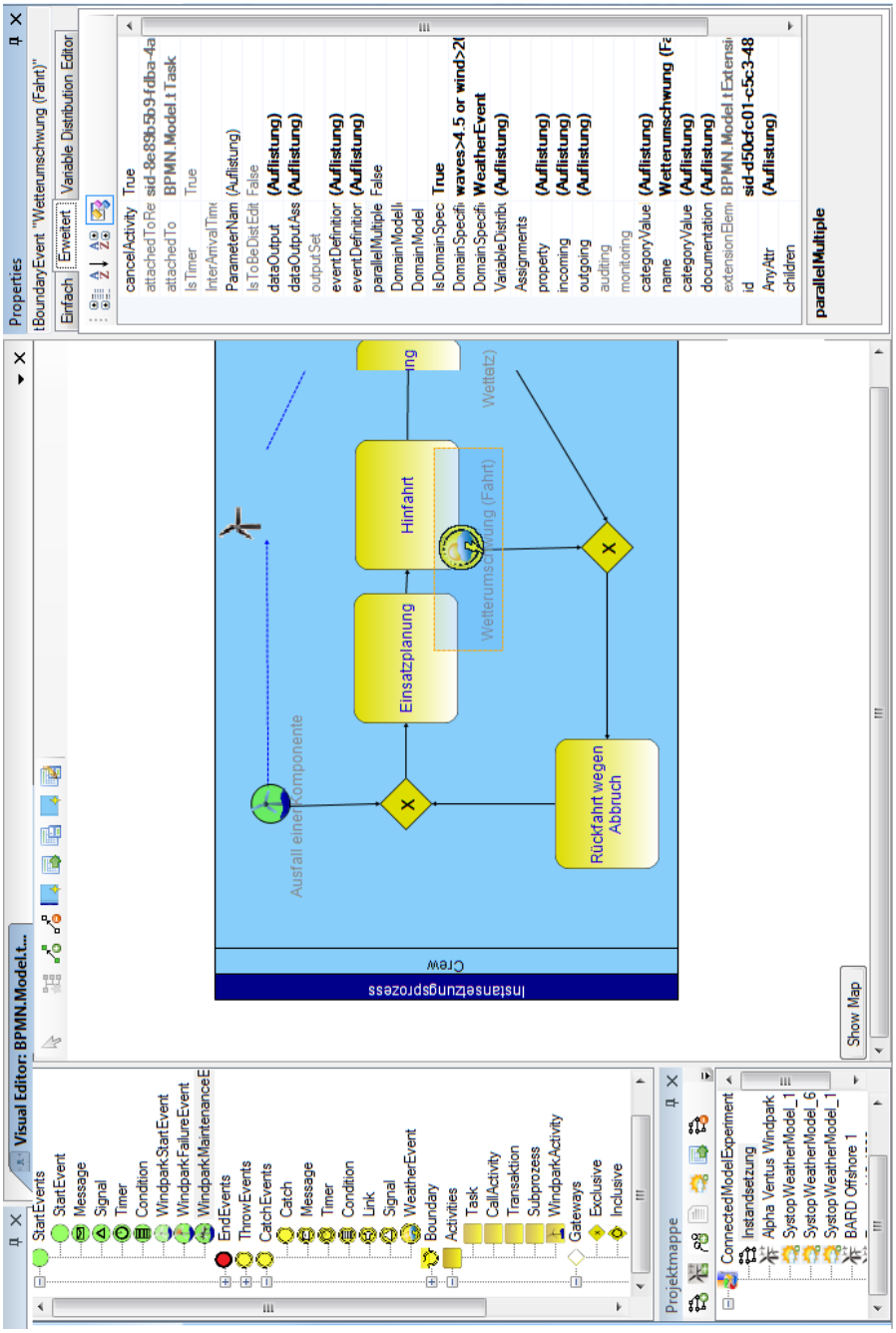


Abbildung 7.7.: Benutzeroberfläche des Prototypen mit angemeldeter Windpark- und Wetterkomponente

7.4.1. BPMN-Editor mit angemeldeten Erweiterungen

Nachdem die Erweiterungsbibliotheken im Applikationsverzeichnis von Empinia hinterlegt wurden, stehen die vorgenommenen Erweiterungen zur Verfügung. Der Benutzer bemerkt dies an der nun veränderten Benutzeroberfläche des BPMN-Editors (vgl. Abbildung 7.7). In der Symbolleiste des Projektmappenexplorers wurden zusätzliche Aktionen angemeldet (unten links). Hierüber ist es möglich, neue Windpark- oder Wettermodelle zu erstellen. Wenn der Knopf zum Laden von Modellen betätigt wird, schlägt das sich öffnende Dateiauswahlfenster nun auch die zu den Windpark- oder Wettermodellen gehörenden Dateiendungen vor. In der Projektmappe werden die verschiedenen Modelltypen mit entsprechenden Icons kenntlich gemacht. Hierfür wurde sogenannte Label Decorator angemeldet.

In der Toolbox mit BPMN-Elementen (oben links) finden sich nun auch die angemeldeten domänenspezifischen Elemente „WindparkFailureEvent“, „WindparkStartEvent“, „WeatherEvent“ und „WindparkActivity“. Das Dokument in der Mitte der Abbildung zeigt bereits einen Ausschnitt des Beispielprozesses, in welchem der Benutzer diese Elemente verwendet hat. Mit einer Wetterwolke wird das Wetterereignis symbolisiert, mit einer Windkraftanlage die Windparkelemente.

7.4.2. Modellbeschreibung

Eine umfassende Darstellung der in SystOp erfassten Prozesse ist für ein Fallstudie im Rahmen dieser Arbeit aufgrund der Komplexität der Prozesse ungeeignet. Um dennoch die Funktionsweise der Erweiterungsbibliotheken zu Wetter und Windparkmodellen aufzeigen zu können, wurde der simplifizierte Instandhaltungsprozess aus Abbildung 7.8 entworfen. Er beginnt oben links mit einem domänenspezifischen Startereigniss „Ausfall einer Komponente“. Als erste Aktivität wird eine „Einsatzplanung“ durchgeführt. Diese Aktivität ist in den SystOp-Modellen sehr viel detaillierter beschrieben, unter anderem müssen in der Realität ein passendes Zeitfenster ausgewählt und die notwendigen Ressourcen gebucht werden. An dieser Stelle wird von den tatsächlich notwendigen Arbeitsschritten abstrahiert. Nach der Planung folgt die Hinfahrt zu der Anlage. Dort ist ein Ereignis „Wetterumschwung (Hinfahrt)“ angeheftet, welches zu einem Abbruch der Aktivität führen kann. Sollte das Ereignis eintreten, so wird dem Pfad hinter dem Ereignis gefolgt und die „Rückfahrt wegen Abbruch“ angetreten. Über ein zusammenführendes exklusives Gateway wird eine Schleife im Prozess definiert und erneut die „Einsatzplanung“ durchgeführt. Sollte die „Hinfahrt“ erfolgreich abgeschlossen werden, folgt hierauf die domänenspezifische Aktivität „Instandsetzung“ an der Anlage. Auch diese Aktivität kann durch ein Ereignis „Wetterumschwung (Einsatz)“ abgebrochen werden, welches ebenfalls über die Aktivität „Rückfahrt wegen Abbruch“ zu der

„Einsatzplanung“ zurückführt. Wenn die „Instandsetzung“ abgeschlossen werden konnte, wird die „Rückfahrt“ angetreten. Anschließend endet der Prozess mit dem generellen Endereignis „Einsatz abgeschlossen“.

Um Referenzen auf diejenigen Komponenten hinterlegen zu können, welche den Fehler gesendet haben, wurde ein „Windpark-Artefakt“ platziert. Es ist mit dem Startereignis und der Aktivität „Instandsetzung“ assoziiert. Das Startereignis hinterlegt eine Referenz der fehlersendenden Anlage in der durch das Artefakt realisierten Warteschlange. Bei erfolgreichem Abschluß der „Instandsetzung“ wird diese wieder aus der Warteschlange entfernt. Über die übergebene Referenz kann die domänenspezifische Aktivität den Zustand der betreffenden Komponente zurücksetzen. Die Komponente meldet diesen Vorgang an die Anlage, welche nun überprüft, ob sie wieder in den stromproduzierenden Zustand schalten kann. Eine dynamische Kopplung der Prozessinstanzen an einzelne Entitäten aus dem domänenspezifischen Modell ist hiermit realisiert.

An den Wettereignissen wurden Bedingungen hinterlegt, die beschreiben, unter welchen Wetterbedingungen das Ereignis auslöst. In diesem Beispielmodell wurde von verwendeten Transportmittel und Zugangssystemen abstrahiert und angenommen, dass ein Einsatz bis zu einer Wellenhöhe von 4 Metern und einer Windgeschwindigkeit von 20 Metern pro Sekunde möglich ist. In den SystOp-Prozessmodellen werden die Wetterdaten nicht mit fixen Werten, sondern mit Eigenschaften der Transportmittel und Zugangssysteme verglichen.

Für die einzelnen Aktivitäten wurden stochastische Verteilungen zur Beschreibung der Dauer hinterlegt. An allen Aktivitäten wurde die Normalverteilung gewählt. Die Parameter aus Tabelle 7.1 sind für dieses Beispielmodell mit einer groben Annäherung an Erfahrungen aus dem SystOp-Projekt gewählt. Die Einsatzplanung ist mit ca. vier Tagen die längste Aktivität, hat jedoch eine große Varianz. Für die Hinfahrt wurden vier Stunden angenommen, für die Durchführung des Einsatzes ein Arbeitstag.

Title	Type	Mean	Std. Dev
Instandsetzung	Gaussian	8	2
Rückfahrt	Gaussian	4	1
Hinfahrt	Gaussian	4	1
Einsatzplanung	Gaussian	96	36
Rückfahrt wegen Abbruch	Gaussian	3	1

Tabelle 7.1.: Parameter der Dauer der Aktivitäten im Beispielmodell

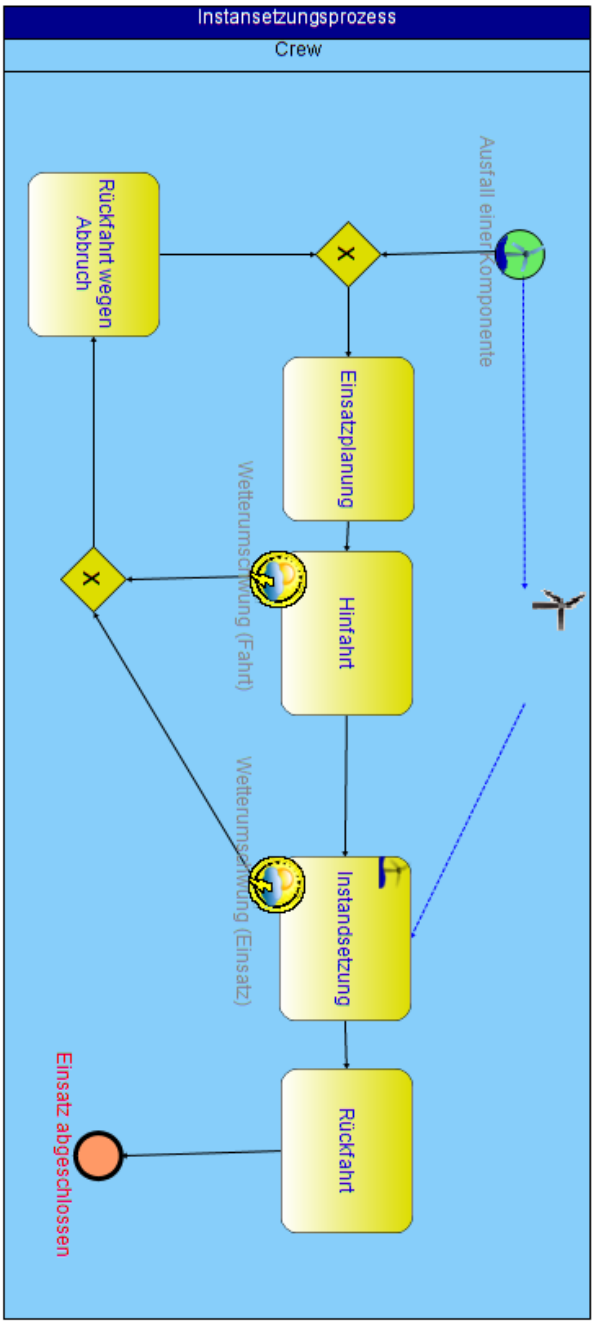


Abbildung 7.8.: Beispielmodell für einen trivialen Instandhaltungsprozess

7.4.3. Durchführung einer Experimentreihe

Zur Durchführung eines Simulationsexperimentes wird in die Experimentplanungsperspektive gewechselt. Über den Projektmappenexplorer werden diejenigen Modelle ausgewählt, die in das Experiment einbezogen werden sollen. Es wird der Instandhaltungsprozess, das Alpha Ventus Windparkmodell und ein Wettermodell mit Daten zu 20 Windjahren ausgewählt. Da von den domänenspezifischen Teilmodellen nur jeweils eines einbezogen wurde, müssen an den BPMN-Elementen keine weiteren statischen Verknüpfungen vorgenommen werden. Das Experiment kann jedoch auch mit den ebenfalls erstellten Modellen von Bard Offshore I oder Riffgat durchgeführt werden, dafür muss lediglich das enstprechende Windparkmodell ebenfalls selektiert werden.

Die Dauer eines Simulationslaufes wird auf 20 Jahre festgelegt. Es sollen zehn Simulationsexperimente durchgeführt werden. Hierbei soll überpüft werden, wie häufig ein Abbruch des Prozesses aufgrund der Wetterbedingungen zu erwarten ist, wie groß der Zeitraum zwischen Fehlersignal und Instandsetzung der Anlage ist und wieviel Strom die einzelnen Anlagen produzieren konnten.

7.4.4. Erläuterung der Ergebnisse

Im folgenden werden die Statistiken erläutert, welche bei der Durchführung eines Simulationsexperimentes entstanden sind. Da es sich um ein stark simplifiziertes Prozessmodell mit teilweise geschätzen Parametern handelt, stellen diese Zahlen keine tatsächliche Bewertung der Instandhaltuntsprozesse von Alpha Ventus dar, sondern demonstrieren lediglich die Anwendungsmöglichkeiten der entwickelten Software.

In der Ergebnistabelle 7.2 wird quantifiziert, wie häufig die einzelnen Ereignisse des Prozesses aufgetreten sind. Es gab in dem simulierten Zeitraum 256 Fehlersignale und gleichviel abgeschlossene Einsatze. In 81 Fällen ist schon während der Hinfahrt und in 33 Fällen während der eigentlichen Instandhaltung ein Wetterumschwung eingetreten, welcher zu einem vorläufigen Abbruch des Einsatzes führte. Die Höhe dieser Zahlen verdeutlicht bereits die Relevanz einer sorgfältigen Auswahl eines geeigneten Zeitfensters für den Einsatz schon bei der Planung, da ein abgebrochener Einsatz mit hohen Kosten verbunden ist.

Tabelle 7.3 gibt Kennzahlen zu den Laufzeiten der Prozessinstanzen wieder. Der Mittelwert liegt bei 157, der Median bei 125 Stunden. Das obere und das untere Quartil liegen mit 94 und 171 Stunden in der gleichen Größenordnung. Dass das Maximum jedoch bei 699 Stunden und damit fast bei einem Monat liegt, zeigt, dass es teilweise starke Ausreisser bei den Prozesslaufzeiten gibt. Eine naheliegende Ursache hierfür dürften die aufgrund der Wettereinflüsse abgebrochenen Einsätze sein. Sollte es tatsächlich im

Title	Type	Value
Triggered Event „Ausfall einer Komponente“	Count	256
Triggered Event „Wetterumschwung (Fahrt)“	Count	81
Triggered Event „Wetterumschwung (Einsatz)“	Count	33
Triggered Event „Einsatz abgeschlossen“	Count	256

Tabelle 7.2.: Anzahl der aufgetrenen Ereignisse

laufenden Betrieb zu einem einmonatigen Produktionsausfall an einer Anlage kommen, dürfte sich dies sehr negativ auf die Rentabilität des Windparks auswirken. Abbildung 7.9 zeigt diese Prozesslaufzeiten als Histogramm. Hier ist abzulesen, dass insgesamt 31 Einsätze länger als 250 Stunden gedauert haben.

Mean	Std. Dev	Min	Max	Median	Lower Quartile	Upper Quartile
157,3	108,4	21,6	699,7	125,0	94,7	171,7

Tabelle 7.3.: Laufzeiten der Instandhaltungsprozesse in Stunden

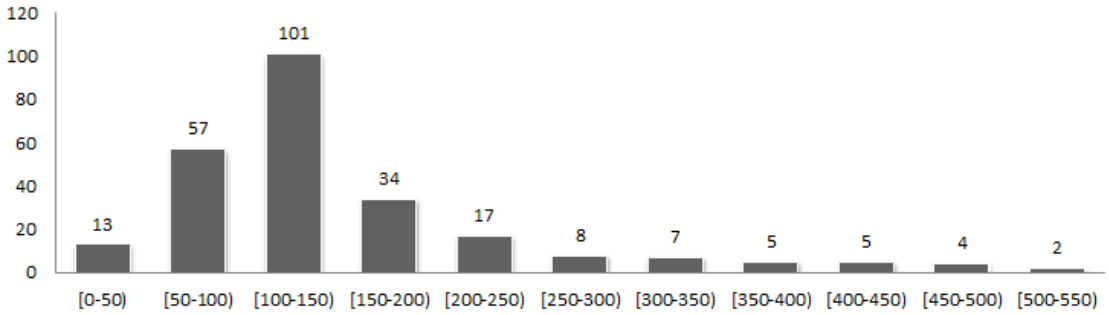


Abbildung 7.9.: Laufzeiten der Instandhaltungsprozesse in Stunden als Histogramm

Die gleiche Art von Kennzahlen wird zu den einzelnen Aktivitäten geliefert (vgl. Tabelle 7.4). Die Mittelwerte und Standardabweichungen zu den Aktivitäten „Einsatzplanung“, „Rückfahrt“ und „Rückfahrt wegen Abbruch“ entsprechen den vorgegeben Modellparametern. Dass die Mittelwerte zu „Hinfahrt“ und „Instandsetzung“ unter dem als Eingangsparameter vorgegeben Mittelwert liegen, ist durch den häufigen Abbruch dieser Aktivitäten begründet.

Eine Ergebnisstatistik, die auch ohne explizite Modellierung von Ressourcen erste Rückschlüsse auf einen Ressourcenbedarf erlaubt, weist Tabelle 7.5 auf. Der Spalte „Mean“

Title	Mean	Std. Dev	Min	Max	Median	LQ	UQ
Instandsetzung	7,3	2,6	0,1	13,5	7,5	5,7	9,0
Rückfahrt	4,0	1,0	1,0	6,9	4,0	3,3	4,8
Hinfahrt	3,3	1,5	0,1	6,2	3,5	2,6	4,4
Einsatzplanung	96,1	34,1	5,0	198,0	96,6	72,2	119,6
Rückfahrt (Abbruch)	3,0	0,9	1,1	5,4	2,4	3,0	3,6

Tabelle 7.4.: Laufzeiten der Aktivitäten in Stunden

zu den Aktivität gibt an, wieviele dieser Aktivitäten im Durchschnitt über den Gesamtzeitraum betrachtet aktiv waren. Ein Wert um 0,01 bedeutet, dass lediglich in einem Prozent des betrachteten Zeitraums diese Aktivität ausgeführt wurde. Ressourcen, die ausschließlich für diese Aktivitäten verwendet werden würden, wären also zu 99 Prozent ungenutzt. Allerdings traten alle Aktivitäten auch parallel auf. Mit der Einsatzplanung waren bis zu vier Prozessinstanzen gleichzeitig beschäftigt. Wenn eine Ressourcenanalyse in die Modelle integriert werden würde, würden sich die Prozesslaufzeiten an dieser Stelle verzögern, sofern nicht mehrere Ressourcen zur Verfügung stehen. Die Integration von Ressourcenmodellen dürfte bei größeren Windparkmodellen mit 80 statt 12 Anlagen und bei der Betrachtung von mehreren Windparks in einem Simulationsexperiment aufschlussreiche Kennzahlen zum Ressourcenbedarf und zu entstehenden Wartezeiten liefern. Die in diesem Experiment gelieferten Zahlen sind allerdings ein Indiz für eine Problematik der Offshore-Industrie: Obwohl Ressourcen zu einem Großteil der Zeit ungenutzt sind, werden diese regelmäßig zur gleichen Zeit an verschiedenen Stellen benötigt. Die Bereitstellung von ausreichend Transportkapazitäten wäre einerseits notwendig, um mehrere Arbeiten parallel ausführen zu können, ist aber andererseits zum gegenwärtigen Zeitpunkt aufgrund der geringen Auslastung noch unrentabel.

Title	Mean	Std. Dev	Min	Max
Instandsetzung	0,012	0,11	0	2
Rückfahrt	0,006	0,0	0	2
Hinfahrt	0,007	0,08	0	2
Einsatzplanung	0,207	0,5	0	4
Rückfahrt wegen Abbruch	0,002	0,05	0	2

Tabelle 7.5.: Paralles Auftreten der Aktivitäten

Eine besonders wichtige Kennzahl zur Bewertung von Instandhaltungsstrategien in dieser Domäne ist die durch den Windpark erzeugte Energiemenge. Eine Statistik hierfür wird durch das Windparkmodell bereitgestellt und beruht auf der durch das Wettermo-

dell gegebenen Windstärken und dem Zustand der Anlagen. Nur eine funktionstüchtige Anlage kann Energie liefern. Ausgegeben werden die in der Gesamtlebenszeit von Alpha Ventus insgesamt erzeugte Energie in MWh sowie die Energieproduktion der einzelnen Anlagen. Dass die Repower-Anlagen geringfügig weniger Strom liefern als die AREVA-Anlagen, liegt an der für die Erbringung der Nennleistung notwendige, geringfügig höhere Windstärke. Da jedoch als Ausfallwahrscheinlichkeiten und für den Wartungsaufwand der beiden Anlagentypen exakt die gleichen Parameter angenommen wurden, sind diese Zahlen nicht als tatsächlicher Wirtschaftlichkeitsvergleich der Anlagentypen zu verstehen. Wenn jedoch verschiedene Instandhaltungsstrategien verglichen werden sollen und dabei exakte Zahlen zum Instandhaltungsaufwand der einzelnen Anlagen vorliegen, liefert diese Statistik eine sehr relevante Bewertungsgrundlage.

Title	Value
Alpha Ventus Windpark	3987283 MWh
AREVA Wind M5000 #1	343495 MWh
AREVA Wind M5000 #2	341956 MWh
AREVA Wind M5000 #3	344013 MWh
AREVA Wind M5000 #4	341710 MWh
AREVA Wind M5000 #5	341932 MWh
AREVA Wind M5000 #6	341932 MWh
Repower 5M #1	322519 MWh
Repower 5M #2	324856 MWh
Repower 5M #3	322009 MWh
Repower 5M #4	320977 MWh
Repower 5M #5	320977 MWh
Repower 5M #6	321813 MWh

Tabelle 7.6.: Erzeugte Strommengen der einzelnen Anlagen und des Windparks im Beispielexperiment

Typisch für die Bewertung von Simulationsexperimenten sind Statistiken zu Warteschlangen. Diese können implizit bei der Modellierung von Ressourcen oder Nachrichtenaustausch entstehen oder durch die Verwendung domänenspezifischer Artefakte explizit modelliert werden. Die Tabelle 7.7 zeigt die Warteschlangenstatistik zum Windparkartefakt und gibt somit die durchschnittliche und maximale Zeit wieder, die zwischen Fehlersignal und Instandsetzung einer Anlage verstrichen. Auch ist hier abzulesen, dass sich bis zu vier Anlagen zur gleichen Zeit im Störungszustand befanden.

Eine weitere domänenspezifische Kennzahl sind die Lauf- und Stillstandszeiten der einzelnen Anlagen. In Tabelle 7.8 ist abzulesen, wie lange eine Anlage mindestens, durchschnittlich und maximal bis zum nächsten Auftreten eines Fehlers funktionstüchtig war,

Qmax	Qnow	Qavg.	max. Wait	avg.Wait
4	0	0,22	695,5 h	153,27 h

Tabelle 7.7.: Warteschlangenstatistik des Windparkartefakts

Title	Mean	Std.Sv	Min	Max
Mean Break Down AREVA #1	153,4	95,5	37,8	405,3
Mean Running AREVA #1	8213,86	10577,5	152,1	45156
Mean Break Down AREVA #2	167,4	103,3	62,5	396,5
Mean Running AREVA #2	7803,8	8013,5	112,0	25170,6
Mean Break Down AREVA #3	173,8	119,0	18,1	494,4
Mean Running Down AREVA #3	7133,43	13636,17	147,86	68941,18

Tabelle 7.8.: Lauf- und Stillstandszeiten der Anlagen in Stunden

bzw. wie lange es mindestens, durchschnittlich und maximal gedauert hat, bis eine Störung behoben wurde. Um hier nicht den falschen Eindruck zu erwecken, diese Arbeit behandele den Vergleich von Repower und AREVA Anlagen, wurden die Statistiken zu den Repower-Anlagen nicht und zu den AREVA-Anlagen nur drei der sechs existierenden Anlagen abgebildet. Die Ausfallwahrscheinlichkeiten sind völlig anderen Anlagentypen entnommen und sind explizit keine realistische Abbildung der tatsächlich zu erwartenden Ausfallzeiten einer AREVA- oder einer Repower-Anlage.

Die Abbildung dieser Statistiken verdeutlicht, welche Zahlen theoretisch mit Hilfe der um domänenspezifische Modelle angereicherten Prozesssimulation als Kennzahlen zur Verfügung gestellt werden können. Die Abbildung der domänenspezifischen Umwelt bietet somit einen erheblichen Mehrwert zur reinen Prozesssimulation.

7.4.5. Anwendung in der Praxis

Im SystOp-Projekt werden die Instandhaltungsprozesse sehr viel detaillierter als in diesem Beispiel beschrieben. Neben der Instandsetzung spielt auch die Wartung eine entscheidende Rolle. Durch die Modellierung der Interaktion zwischen verschiedenen Prozessbeteiligten entsteht ein zusätzlicher Einfluss auf das dynamische Laufzeitverhalten, welches unter anderen durch weitere Warteschlangenstatistiken ausgewertet werden kann. Die Integration von Ressourcenmodellen, welche die tatsächlich zur Verfügung stehenden Transportmittel abbilden, und die Anwendung auf andere Windparkmodelle, erlaubt es, kritische Prozessbestandteile zu identifizieren, welche oftmals nur verzögert

ausgeführt werden können. Eine rein analytischer Vergleich der anfallenden Arbeitslast mit den zur Verfügung stehenden Ressourcenkapazitäten erlaubt im Gegensatz zur Simulationstechnik nicht, kritische Zeiträume, in denen sich Aufträge anstauen, zu identifizieren.

Durch die hier vorgestellten Komponenten können Kennzahlen, wie die eben aufgezeigten, zu den im SystOp-Projekt erfassten Prozessen bereitgestellt werden. Sollten die Instandhaltungsprozesse einzelner Windparks im Rahmen von künftigen BPI-Prozessen Verbesserungen unterzogen werden, liefern diese Prozessmodelle und dieser Software-Prototyp eine Unterstützung bei der Bewertung der Prozessalternativen und somit eine wertvolle Entscheidungsunterstützung.

7.5. Weitere Erweiterungskomponenten

Aufbauend auf den Konzepten dieser Arbeit wurden zwei weitere Komponenten entwickelt. Beide stellen keine domänenspezifischen BPMN-Elemente für die Benutzeroberfläche bereit, nutzen aber das Konzept, das Verhalten von Aktivitäten durch Hinterlegung von Ausführungsstrategien zu beeinflussen. Diese Ausführungsstrategien haben Einfluss bzw. sind beeinflusst von entsprechenden Teilmodellen. Auf diese Weise wird es möglich, das Verhalten sämtlicher Aktivitäten zu verändern, sofern die entsprechenden Erweiterungsattribute hinterlegt sind. Diese beiden Komponenten werden hier kurz erläutert, um die Breite des möglichen Anwendungsspektrums der Konzepte dieser Arbeit zu unterstreichen.

7.5.1. Ressourcen-Modelle

Die Integration von Ressourcen-Modellen ist im Zuge der Geschäftsprozesssimulation besonders relevant. Erst hierdurch wird eine fundierte Engpassanaylse möglich. Zudem beeinflussen sich verschiedene Prozessinstanzen durch die Konkurrenz um Ressourcen durch eine indirekte Interaktion. In einigen Branchen kann die zeitnahe Abarbeitung und somit der Bedarf an ausreichenden Ressourcenkapazitäten eine kritische Rolle spielen. Eine Prozesssimulationssoftware sollte daher unbedingt eine Komponente zur Definition von Ressourcenmodellen bereitstellen. Auch die hier verwendete DESMO-J-Bibliothek zur BPMN-Simulation bietet diese Funktion. Dieses Konzept wurde nach Bereitstellung der Erweiterungsmechanismen jedoch überarbeitet, um künftig eine Austauschbarkeit der Ressourcenkonzepte zu ermöglichen.

Hierfür wurde keine Erweiterung der BPMN-Notation um zusätzliche Elemente vorgenommen. Stattdessen werden vom Modellkoppler der Ressourcen-Komponente die herkömmlichen BPMN-Aktivitäten daraufhin überprüft, ob das in der BPMN-Spezifikation

bereits vorgesehene Feld „Participant“ ausgefüllt wurde. Ist dies der Fall, so wird gemäß der in Abschnitt 5.2 beschriebenen Konzepte eine Ressourcen-Aktivität an die BPMN-Aktivität angeheftet, bei welcher die EventHandler `OnMultipleActivityStarted` und `OnMultipleActivityFinished` überschrieben wurden. Dadurch wird die Ausführung der Aktivität solange verzögert, bis die benötigten Ressourcen frei werden.

Das Ressourcen-Modell selbst wird in einer seperaten XML-Datei definiert, für welche ein graphbasierter Editor über die vorgegebenen Erweiterungsmechanismen eingebunden wurde. Dieser unterscheidet die beiden Knotentypen Ressourcenpool und Ressourceninstanz. Eine Ressourceninstanz kann mehreren Ressourcenpools zugeordnet sein und somit mehrere Rollen erfüllen. Als Participant wird an den BPMN-Aktivitäten der Name eines Ressourcenpools angegeben. Wird die Aktivität ausgeführt, so wird aus dem Pool eine freie Ressource entnommen. Wenn diese zugleich mit anderen Ressourcenpools verknüpft ist, so wird sie auch aus diesen Pools entnommen. Wenn keine Ressource frei ist wird darauf gewartet, bis eine frei wird. Wenn die Aktivität abgearbeitet wurde, wird die Ressourceninstanz in all ihre zugehörigen Pools zurückgelegt.

Das Ressourcen-Konzept dieser Lösung ist leicht austauschbar, denn alternative Implementationen können ohne Änderung der Code-Basis der BPMN-Bibliothek bereitgestellt werden. Es wäre denkbar, dass Ressourcen in diesem Zuge mit weiteren Eigenschaften versehen werden, zum Beispiel Rüst- und Nachbereitungszeiten für Maschinen oder Schichtpläne für menschliche Ressourcen. Auch wäre denkbar, ein Ressourcenmodell so zu implementieren, dass die Ressource selbst die Dauer der Aktivität bestimmt. Hierfür müsste das Property `DomainspecificExecution` der Ressourcen-Aktivität auf `true` gesetzt werden und eine `Execute`-Methode bereitgestellt werden, welche die Laufzeit bestimmt. So könnte ein erfahrener Mitarbeiter eine Aufgabe schneller bearbeiten als ein noch anzulernender Mitarbeiter. Bei der Abbildung von Transportaktivitäten wird die Dauer des Transports abhängig von dem gewählten Transportmittel sein. Diese Möglichkeiten sind derzeit zwar noch nicht in den Prototoypen integriert, mit den vorgegebenen Konzepten jedoch leicht umsetzbar.

Auch ist eine Konkurrenz der veschiedenen domänenspezifischen Teilmodelle um die zur Verfügung stehenden Ressourcen prinzipiell möglich, wodurch die Möglichkeit der indirekten Interaktion zwischen den Teilmodellen entsteht (vgl. Abschnitt 8.2).

7.5.2. Kostenrechnung

Eine Erweiterung zur Prozesskostenrechnung wurde ebenfalls aufbauend auf Teilen der Konzepte dieser Arbeit entwickelt. Zur Analyse der Kosten einzelner Prozesse oder Produkte werden zwei verschiedene Kostenarten herangezogen. Die Höhe von leistungsmengeninduzierten Kosten (lmi-Kosten) ist abhängig von der Durchführung von Aktivitäten.

Die Höhe von leistungsmengenneutralen Kosten (lmn-Kosten) ist unabhängig vom Auftreten der Aktivitäten. Beiden Kostenarten werden Kostenstellen zugeordnet. Typische Kostenstellen für lmn-Kosten sind Gebäudemieten oder Mitarbeitergehälter; für lmi-Kosten sind dies beispielsweise Materialeinzelkosten oder Fertigungseinzelkosten. In der klassischen Prozesskostenrechnung werden die lmn-Kosten über einen Zuschlagssatz auf die Prozesse oder Produkte verteilt.

Die vorgenommene Erweiterung bietet einen Editor zur Definition der Kostenstellen. Hier wird für jede Kostenstelle hinterlegt, ob sie den lmn- oder den lmi-Kosten zuzurechnen ist, und wie hoch die Kosten jeweils anzusetzen sind. Für die Berechnung der lmi-Kosten stehen drei Buchungsindikatoren zur Verfügung: Die Kosten können abhängig von der Dauer der Aktivität, abhängig von der Häufigkeit der Aktivität oder abhängig von einer Prozessvariablen wie zum Beispiel die Stückzahl eines Auftrages sein. Hierdurch wird eine Betrachtung von monetären Kennzahlen im Rahmen der Simulation grundsätzlich möglich.

Die lmn-Kosten können wie in der herkömmlichen Prozesskostenrechnung über einen Zuschlagssatz auf die Prozesse verteilt werden. Da diese Kosten jedoch häufig bestimmten Ressourcen zugehörig sind, zum Beispiel die Fixgehälter eines Mitarbeiters, kann die Simulation hier einen besseren Ansatz bieten. Es werden bereits Kennzahlen generiert, die angeben, auf welchen Prozessen oder Produkten ein Mitarbeiter arbeitet. Daher können diese lmn-Kosten anstatt über einen Zuschlagssatz gemäß dem tatsächlichen Gebrauch aufwandsbezogen verteilt werden. Dies ermöglicht eine realisitischere Abschätzung der Verteilung von Prozesskosten als in der herkömmlichen Prozesskostenrechnung ohne Simulation.

Um die Berechnung dieser Kennzahlen zu realisieren, wird über ein Erweiterungsattribut an der BPMN-Aktivität angegeben, welche im Kostenmodell definierte Kostenstelle betroffen ist. Es wird eine Kosten-Aktivität bereitgstellt, welche durch den Modellkoppler der Kostenerweiterung an der BPMN-Aktivität als Strategie hinterlegt wird. Diese stellt einen `OnActivityFinished`-EventHandler bereit, in welchem die Buchungen an der betreffenden Kostenstelle vorgenommen werden. Da an einer BPMN-Aktivität mehrere spezifische Aktivitäten hinterlegt werden können, welche jeweils eigene EventHandler bereitstellen, ist die Kombination von domänenspezifischen Aktivitäten (zum Beispiel der Windpark-Aktivität) mit der Ressourcen-Aktivität und der Kosten-Aktivität möglich. Es darf allerdings nur einer der Aktivitäten eine neue `Execute`-Methode bereitstellen und dadurch die Laufzeit der Aktivität bestimmen.

8. Diskussion und Bewertung

8.1. Innovationsbeitrag der Arbeit

Der Innovationsbeitrag der Arbeit ist die Konzeption eines erweiterbaren, auf Prozessmodellen der BPMN 2.0 basierendes Kollaborationsrahmenwerk, welches die Durchführung von Simulationsstudien unter Einbeziehung domänenspezifischer Teilmodelle ermöglicht. Hierdurch werden heterogene Branchenlösungen möglich, welche eine simulationsbasierte Leistungsmessung für bestehende Prozessdefinitionen unter Einbeziehung der jeweils relevanten Prozessumgebung erlauben. Die entscheidenden Aspekte dieser Arbeit sind im Überblick:

- Integration der domänenspezifischen Prozessumgebung bei der Prozesssimulation
- Ermöglichung der Analyse von domänenspezifischen Auswirkungen auf Prozesse
- Ermöglichung der Bewertung von Prozessen durch domänenspezifische Kennzahlen
- Dynamische und statische Verknüpfung von Prozessen mit Modellentitäten
- Explizite Modellierung von Warteschlangen für domänenspezifische Modellentitäten
- Flexible Implementation von domänenspezifischen Prozessaktivitäten
- Hinterlegung von Bedingungen zur Spezialisierung von Empfangsereignissen
- Standardkonforme Verknüpfung der Modelle
- Sicherstellung der Standardkonformität der Prozessmodelle mittels strikter Vorgaben zur Erweiterung
- Lösungen für verschiedene Domänen durch generellen Ansatz möglich
- Integration aufgabenangemessener Editoren für domänenspezifische Modelle möglich
- Komponentenbasierte Erweiterungen ohne Änderung der bestehenden Code-Basis
- Grundlage für branchenspezifische Softwarelösungen zur Prozesssimulation

Bisherige Lösungen zur Simulation von Geschäftsprozessen (Enstone und Clark (2006), Rücker (2008c)) beschränkten sich auf die Simulation der Prozesse ohne eine Betrachtung der Systemumgebung. Die Schnittstellen zu anderen Teilsystemen konnte bisher nicht in die Prozessdefinitionen integriert werden, obwohl diese für eine fundierte Berechnung der relevanten Kennzahlen ausschlaggebend sein können. Sofern die Systemumgebung sich unidirektional auf die BPMN-Modelle auswirkt, ohne dass eine Rückkopplung auf die Umgebung existiert, konnte diese noch durch eine Abbildung von Zeitreihen in bestehenden Werkzeugen integriert werden. Wenn jedoch eine bidirektionale, kybernetische Kopplung zwischen den Modellen existiert, so war eine aufgabenangemessene Simulation bisher nicht möglich. Diese Lücke wurde im Rahmen dieser Arbeit geschlossen.

Die Verknüpfung mit den Teilmodellen wird dabei über standardkonforme Erweiterungen der BPMN 2.0 realisiert. Jede domänenspezifische Erweiterung bringt neben einem Editor für das betreffende Teilmodell auch eine Menge von domänenspezifischen BPMN-Elementen mit. Die Definition von Erweiterungselementen der BPMN wurde bereits in anderen Arbeiten thematisiert (z.B. Schäfer (2012)), jedoch gibt es bisher keine domänenspezifischen Erweiterungen mit der Zielsetzung der Simulation. Für die Simulation werden sendende und empfangende Ereignisse, Aktivitäten und Artefakte benötigt. Durch Ereignisse wird die Interaktion mittels Nachrichtenaustausch ermöglicht. Durch Aktivitäten wird einerseits die Manipulation der Teilmodelle ermöglicht, zum anderen aber auch die Implementation und damit die Definition der Ausführungsdauer einer Aktivität durch das Teilmodell selbst.

Um eine zielgerichtete Interaktion mit konkreten Entitäten und die Realisierung von in der Simulation typischen Warteschlangen zu ermöglichen, wurden domänenspezifische Artefakte eingeführt. Hierdurch lassen sich sowohl globale als auch nur im Prozesskontext verfügbare Referenzen auf Entitäten hinterlegen und abrufen. Die Verwendung von Warteschlangen ist optional; die Wahl des Warteschlangentyps ist dem Entwickler der Erweiterungsbibliothek überlassen. Während bei der herkömmlichen Prozesssimulation nur implizite Warteschlangen für Nachrichten und Ressourcen existieren, wird hier erstmals die explizite Modellierung von Warteschlangen für Entitäten ermöglicht. Dies stellt einen Mechanismus zur dynamischen Kopplung dar, welcher es erlaubt, dass verschiedene Instanzen derselben Prozessdefinitionen mit unterschiedlichen Entitäten interagieren. Jedoch ist hier in der Regel der Empfang einer Nachricht von der entsprechenden Entität vonnöten, um die Referenz über das Artefakt speichern zu können. Alternativ kann bei der Implementation des Artefaktes ein anderer Mechanismus zur Auswahl und Referenzierung von Entitäten bereitgestellt werden, dem Entwickler der Erweiterungsbibliothek werden hier keine Grenzen gesetzt.

Empfangende Ereignisse können zudem mit Bedingungen versehen werden, welche den Prozesskontext sowie die vom Teilmodell übermittelten Informationen auswerten. Hierdurch wird ein weiterer dynamischer Mechanismus bereitgestellt, welcher die Auswahl

von relevanten Nachrichten unterstützt, ohne dass die entsprechenden Entitäten sich der Prozessinstanz durch Nachrichtenversand bereits bekannt gemacht haben müssen. Zusätzlich können an allen Elementen statische Vorgaben gemacht werden, auf welche Teilmodellinstanz sie sich beziehen. Hierdurch werden insgesamt drei Kopplungsmechanismen bereitgestellt, welche der Anwendung im Rahmen der Prozesssimulation gerecht werden.

Durch die konsequente Verwendung von Konzepten der komponentenbasierten Entwicklung sind Erweiterungen ohne eine Änderung der bestehenden Code-Basis möglich. Es werden Erweiterungspunkte zur Verfügung gestellt, unter deren Verwendung die Konfiguration und Integration von domänenspezifischen Teilmodellen durch bloßes Hinzufügen einer Bibliothek in das Applikationsverzeichnis möglich ist. Hierbei müssen die domänenspezifischen BPMN-Elemente, sowie vier Klassen zur Integration der Teilmodelle in ein zusammengesetztes Simulationsmodell bereitgestellt werden. Ein Editor zur Manipulation der Teilmodelle wird benötigt. Ein Parametrierer ermöglicht die Festlegung von Variablen im Zuge von mehrläufigen Simulationsexperimenten unter verschiedenen Parameterkombinationen. Eine Simulationsfabrik übersetzt die Modelldefinition in ein zum verwendeten Simulationskern passendes Simulationsteilmodell. Der Modellkoppler übernimmt die Verknüpfung der domänenspezifischen BPMN-Elemente mit den Objekten der entsprechenden Teilmodelle. Hierfür werden abstrakte Klassen vorgegeben, welche bei der Erstellung der Klassen unterstützen, dem Anwendungsentwickler jedoch maximale Flexibilität zur Generierung und Verknüpfung der Teilmodelle ermöglichen. Der Nutzen von komponentenbasierten Simulationsumgebungen wurde bereits in anderen Arbeiten gezeigt (Panic et al. (2008)), diese sind jedoch auf die Stoffstromanalyse und nicht auf die Geschäftsprozessmodellierung und -simulation zugeschnitten.

Durch die engen Vorgaben zur Erweiterung ist der Entwickler einer Erweiterungsbibliothek gezwungen, sich an die Spezifikation der BPMN 2.0 zu halten. So erlaubt die Erweiterungsmöglichkeit des graphischen Editors lediglich, die Symbole von Ereignissen und Aktivitäten an den vorgesehenen Stellen auszutauschen. Das Laufzeitverhalten der Elemente während der Simulation kann durch die vorgegebenen, zu implementierenden Methoden nicht so verändert werden, dass es der ursprünglichen Spezifikation entspricht. Lediglich die Bedeutung von Artefakten kann sehr frei implementiert werden, dies ist von der Spezifikation genau so vorgesehen.

Dennoch wurde darauf geachtet, dem Entwickler der Anwendungsbibliothek einen großen Freiraum bei der konkreten Implementation der Erweiterung zu lassen, um die potentiellen Anwendungsgebiete nicht unnötig einzuschränken. So können beliebige Editoren zur Manipulation der Teilmodelle eingesetzt werden. Die Teilmodelle können nach beliebigen ereignisdiskreten Paradigmen in Simulationsmodelle überführt werden. Bei der Implementation der domänenspezifischen Elemente kann der Entwickler die vorgegeben

Methoden und EventHandler mit beliebig komplexen Algorithmen ausstatten, welche für die Interaktion mit dem jeweiligen Teilmodell notwendig sind.

Die Verwendung einer etablierten Prozessmodellierungssprache ermöglicht die Wiederverwendung der im Rahmen von BPM-Projekten geschaffenen Prozessdefinitionen. In einer Unternehmung, welche die BPMN erfolgreich nutzt, sind Mitarbeiter verfügbar, die diese Prozessdefinitionen zum Zwecke der Simulation anpassen können. Durch die Verwendung von selbst wählbaren, domänenspezifischen Symbolen bei der Verknüpfung mit Teilmodellen wird eine leichte Verständlichkeit der Prozessmodelle ermöglicht. Hierbei Symbole zu wählen, die tatsächlich für den Betrachter intuitiv verständlich sind, liegt jedoch im Aufgabenbereich des Entwicklers der entsprechenden Erweiterungsbibliothek und konnte dementsprechend im Rahmen dieser Arbeit nicht evaluiert werden.

Es wurde eine fundiertes, flexibles und und standardkonformes Konzept geschaffen, welches für verschiedene Branchen adaptierbare Softwarelösungen ermöglicht. Diese Erweiterbarkeit vergrößert dabei das mögliche Einsatzfeld der Prozesssimulation im erheblichen Maße.

8.2. Kritische Bewertung und Grenzen der Arbeit

Nicht alle im Rahmen der domänenspezifischen Geschäftsprozesssimulation auftretenden Fragen konnten im Rahmen dieser Arbeit geklärt werden. Einige Kritikpunkte an dem Konzept lassen sich in künftigen Arbeiten lösen. Für andere gibt es keine direkte technische Lösung, stattdessen muss bei den Anwendern das Bewusstsein über die vorhandenen Komplikationen geschaffen werden, so dass ihnen durch entsprechende Maßnahmen entgegengewirkt werden kann.

Der Aufwand zur Implementation einer Erweiterungskomponente ist trotz der komponentenorientierten Ansätze relativ hoch. Der Entwickler der Schnittstelle muss softwaretechnisch sehr fundiertes Wissen haben und benötigt zugleich Kenntnisse in der BPMN-Simulation. Die Dokumentation des verwendeten BPMN-Rahmenwerks und der erwarteten Schnittstellen muss sehr detailliert ausfallen, um Erweiterungen zu ermöglichen. Ein umfassendes Lastenheft über potentielle Fragestellungen, die im Rahmen der Simulationsexperimente geklärt werden sollen, muss schon vor der Entwicklung der Erweiterungsbibliothek erstellt werden. Eine intensive Absprache zwischen Domänenexperten, Prozessverantwortlichen, Softwareentwicklern und Simulationsexperten ist notwendig.

Auch die entwickelten Erweiterungen müssen sehr detailliert dokumentiert werden, damit keine Missverständnisse beim Erstellen der Modelldefinitionen anfallen. Der Modellierer muss klar nachvollziehen können, welche Bedeutung die Erweiterungselemente haben, damit er diese korrekt benutzen kann. Der Ansatz, das entsprechende Teilmodell

als Black-Box anzusehen, ist in diesem Sinne kritisch, da die interne Funktionsweise des Simulationsteilmodells für den Modellierer nicht überprüft werden kann, sondern er vollständig auf die Korrektheit der Dokumentation angewiesen ist. Jedoch ist dies ein grundsätzliches Problem integrierter Entwicklungsumgebungen für Simulationsstudien.

Die Verwendung von domänenspezifischen Symbolen innerhalb der Prozessdefinitionen soll den Wiedererkennunswert und die Verständlichkeit des Bezugs zu den entsprechenden Teilmodellen stärken. Ob durch diesen Ansatz tatsächlich die Verständlichkeit der Modelle erhöht wird, wird stark von der Wahl der Symbole und der Erwartungshaltung der Anwender abhängen. Da dies von Anwendungsfall zu Anwendungsfall variieren kann, war eine grundsätzliche Evaluation dieser Vermutung im Rahmen dieser Arbeit nicht möglich, sondern muss für jede geschaffene Erweiterung separat erfolgen.

Da der Modellierer außer einer Dokumentation keine Informationen über die interne Struktur der Teilmodelle hat, ist eine Überprüfung der Validität der Modelle nicht möglich. Der Nachweis von Eigenschaften wie Verklemmungsfreiheit oder Lebendigkeit ist daher nicht leistbar. Zwar kann ein Deadlock potentiell mit der Simulation aufgedeckt und somit nachgewiesen werden, es bleibt jedoch stets ein Restrisiko, dass eine mögliche Verklemmung in den durchgeführten Simulationsexperimenten nicht entdeckt wurde. Der Nachweis von Lebendigkeit ist mit Hilfe der Simulation grundsätzlich nicht möglich. Die Konformatitätsprüfung von Modellen wurde im Rahmen dieser Arbeit ebenfalls nicht behandelt. So obliegt es zum Beispiel dem Entwickler der Erweiterungsbibliothek dafür Sorge zu tragen, dass innerhalb des Teilmodells die gleichen Zeiteinheiten verwendet werden, wie in den Prozessmodellen.

Die Arbeit liefert keine strukturierten Ansätze zur Kopplung der einzelnen Teilmodelle untereinander, sondern konzentriert sich auf die Kopplung der Teilmodelle mit Prozessmodellen. Zur Kopplung der Teilmodelle wurde lediglich eine Schnittstelle bereitgestellt, um die Verknüpfung von Simulationsmodellen grundsätzlich zu ermöglichen. So wurde in dem entstandenen Prototyp zum Fallbeispiel durchaus die Windparkkomponente mit der Wetterkomponente verknüpft, um die exakt produzierte Strommenge errechnen zu können. Diese kann aber nur genutzt werden, wenn von den entsprechenden Teilmodellen Schnittstellen bereitgestellt sind und der Entwickler einer Komponente die interne Funktionsweise der entsprechenden Teilmodelle kennt. Eine Schnittstelle, mit der gezielt Entitäten oder Ereignisse aus einem Modell erfragt werden können, wäre ein möglicher Verbesserungsansatz.

Dies würde auch die Möglichkeiten der indirekten Kopplung von Modellen verbessern. Eine sehr wichtige Rolle in der Geschäftsprozesssimulation spielen die Ressourcen, denn nur ihre Modellierung ermöglicht eine Engpassanalyse und erzeugt realistische Verzögerungen bei der Abarbeitung von Prozessen. Die Spezifikation von zur Verfügung stehenden Ressourcen ist nicht Bestandteil der BPMN. Im Rahmen dieser Arbeit werden Ressourcen

selbst als Teilmodell modelliert und über die in dieser Arbeit vorgestellten Mechanismen mit den Aktivitäten der Prozesse verknüpft. Während bei der programmatischen Implementation von Simulationsmodellen Warteschlangen für Ressourcen explizit modelliert werden müssen, entstehen in der BPMN-Simulation implizit Warteschlangen für Prozesse an denjenigen Aktivitäten, an denen Ressourcen benötigt werden. Eine mögliche Konkurrenz um Ressourcen zwischen verschiedenen Teilmodellen stellt eine mögliche indirekte Interaktion dar. Die im vorherigen Absatz beschriebene strukturierte Schnittstelle würde es vereinfachen, eine Belegung von Ressourcen durch Teilmodelle zu realisieren, die für die entsprechende Zeit nicht zur Abarbeitung von Prozessen zur Verfügung stehen.

In der Arbeit wurde davon ausgegangen, dass alle Teilmodelle auf dem gleichen Simulationskern beruhen. In der prototypischen Entwicklung wurde als Simulationskern hierfür die weit verbreitete und flexible Bibliothek DESMO-J verwendet. Bei der Neuentwicklung von Komponenten hat der Entwickler nicht die Möglichkeit, die Simulationsbibliothek seiner Wahl zu verwenden. Eine besonders starke Einschränkung ist, dass keine bereits bestehenden Simulationskomponenten an die Prozesse gekoppelt werden können, welche auf anderen Simulationsbibliotheken beruhen. Daher können auch keine bereits bestehenden Simulationsmodelle eingebunden werden, wenn sie nicht auf dem gleichen Simulationskern wie die Prozesssimulationsbibliothek beruhen.

Auch wurde davon ausgegangen, dass alle Teilmodelle zeitdiskret sind. Die Verwendung kontinuierlicher Modelle wurde explizit ausgeschlossen. Daher wurden mit erweiterbaren Ereignissen und Aktivitäten auch nur Anknüpfungspunkte geschaffen, welche die Anbindung von diskreten Teilmodellen ermöglichen. Wenn kontinuierliche Modelle verwendet werden, so müssen diese auf der Zeitachse diskretisiert werden. Die Struktur der BPMN, welche auf einer Repräsentation durch Ereignisse beruht, legt nahe, dass eine andere Lösung nicht möglich ist, was im Rahmen einer künftigen Arbeit jedoch bestätigt oder widerlegt werden kann.

8.3. Anwendungsszenarien

Die möglichen Einsatzszenarien der Geschäftsprozesssimulation wurden bereits in den Neunziger Jahren spezifiziert (Gladwin und Tumay (1994)). Im Regelfall wird die Simulation im Rahmen von BPI-Projekten verwendet, um mögliche Prozessalternativen vor ihrer Implementation mit der bestehenden Vorgehensweise zu vergleichen. Grundlage für die Bewertung kann eine Vielzahl von Kriterien sein. Durch diese Arbeit wird auch die Integration von Kennzahlen möglich, welche abhängig von der Prozessumgebung sind.

Der mögliche Einsatzbereich der Simulation wurde durch diese Arbeit vergrößert, denn durch die bereitgestellten Konzepte werden die möglichen Einsatzszenarien der Geschäftsprozesssimulation um diejenigen Branchen erweitert, welche sich durch eine starke

Abhängigkeit zu ihrer Systemumgebung auszeichnen. Sofern eine kybernetische Kopplung zwischen Prozessen und Prozessumgebung besteht, war die Aussagekraft von Simulationsexperimenten bisher sehr begrenzt. Die Wirkung eines Prozesses auf die interagierenden Systeme und die entsprechende Rückkopplung dieser Systeme konnten nicht abgebildet werden. Durch diese Arbeit sind auch in diesen Domänen Leistungsmessungen mit Hilfe der Geschäftsprozesssimulation durch branchenspezifische Lösungen möglich.

Ein Beispiel für solch eine Branche ist die Offshore Windpark Industrie. Die Simulation kann nun auch hier im Rahmen der Leistungsmessung von Prozessen zur Errechnung von Kennzahlen verwendet werden, wie im Rahmen der Fallstudie gezeigt wurde. Der entstandene Prototyp wird in dem vom BMU geförderten Projekt „Systop Offshore Wind“ erfolgreich eingesetzt. Hier werden Kennzahlen zu den Prozessen des Leistungssystems bereitgestellt; dies wäre mangels ausreichend existierender Windparkanlagen und somit in der Summe nur weniger bereits getätigter Betriebsjahre am Originalsystem nicht möglich.

Im Rahmen solcher branchenspezifischer Lösungen können nun auch die durch das Teilsystem generierten Kennzahlen zur Bewertung der Prozesse herangezogen werden. In einigen Fällen übersteigt die Aussagekraft der domänenspezifischen Performanzkriterien die Aussagekraft der in der herkömmlichen Geschäftsprozesssimulation generierten Kennzahlen. Im Falle der Windenergiebranche sind Kennzahlen zur Stromerzeugung im Vergleich zu den Instandhaltungskosten beispielsweise als wichtiger zu bewerten, als die bloße Laufzeit von Prozessen oder die Ressourcenbelegung.

Die im Fallbeispiel entstandenen Komponenten könnten ebenfalls für den Onshore Bereich eingesetzt werden, jedoch sind die Instandhaltungsarbeiten an Onshore-Anlagen nicht im gleichen Maße den Wetterbedingungen ausgesetzt und auch nicht so komplex wie im Offshore Bereich. Mit den vorliegenden Erweiterungen wurde grundsätzlich die Anwendbarkeit der BPMN 2.0 im Rahmen der Simulation von Instandhaltungsprozessen technischer Anlagen gezeigt; dies lässt sich auf beliebige Anlagentypen übertragen.

Es sind jedoch Erweiterungen für viele weitere Domänen denkbar. Mit jeder geschaffenen Erweiterung werden die Einsatzmöglichkeiten der BPMN entsprechend vergrößert. In Haan (2014) entsteht derzeit eine Erweiterung für logistische Simulationen, die noch auf Einsatztauglichkeit geprüft wird. Hier wurden Logistiknetze als Teilmodelle abgebildet, welche sich auf die Laufzeit von Transportaktivitäten auswirken. Multimodale Transportketten können somit komfortabel mit Hilfe der BPMN 2.0 modelliert und mit geeigneten Geoinformationssystemen verknüpft werden.

Die Bereitstellung von Erweitungskomponenten für andere Industriebereiche liegt nahe. Die Konzepte für den stochastischen Wettergenerator sind beispielsweise Anwendungen aus der Agrarindustrie entnommen. Durch zusätzliche Komponenten zur Anlage von Saatgutsorten, welche die jeweils benötigten Arbeiten zu gegebenen Zeitpunkten

definieren und eine Komponente zur Definition von Anbauflächen, welche sich auf die Laufzeit von Aktivitäten auswirken, wären bereits wesentliche Teilmodelle definiert. Wie bei Offshore-Anlagen können auch hier die Arbeiten durch eine entsprechende Wetterlage verhindert werden. Unter Umständen werden auch zusätzliche Arbeiten notwendig (zum Beispiel durch Sturmschäden oder Schäden durch Schädlinge). Ein möglicher Untersuchungsgegenstand ist die Ressourcenallokation, wie die Zuteilung von Maschinen oder Hilfsarbeitern.

Die Betrachtung von Teilmodellen kann auch innerbetriebliche Systeme der BPMN betreffen. Zwei Erweiterungen zur Abbildung innerbetrieblicher Systemelemente wurden im Rahmen der Arbeit entwickelt, um die Flexibilität der Einsatzmöglichkeiten der Erweiterungskonzepte zusätzlich zu untermauern: Ressourcenmodelle und Kostenstellen. Bei beiden wurden lediglich BPMN-Aktivitäten zur Verfügung gestellt, Ereignisse mussten nicht implementiert werden. Obwohl beide Aspekte nicht Bestandteil der BPMN Spezifikation sind, werden sie in anderen Lösungen teilweise fest in die Simulationsbibliothek integriert. Hier wurde gezeigt, dass die bereitgestellten Kopplungsmechanismen flexibel genug sind, um auch derartige Erweiterungen ohne eine Änderung an der bestehenden Code-Basis zu ermöglichen.

Weitere innerbetriebliche Teilmodelle, wie Erweiterungen für Lagerhaltungssysteme oder Produktionsanlagen, sind ebenso denkbar. So könnte bei der Definition von Teilmodellen für solche technischen Systeme die Fehlerwahrscheinlichkeit oder das Auftreten sonstiger Risiken beim Betrieb der Anlagen gemäß der Fehlermöglichkeits- und Einflussanalyse (FMEA) beschrieben werden, wie in Greiner (2013) vorgeschlagen wurde.

8.4. Ausblick

Für die in Abschnitt 8.2 angesprochenen Einschränkungen können in künftigen Arbeiten Lösungen bereitgestellt werden. Die noch offenen Punkte sind:

- Eine Schnittstelle zur Abfrage von Entitäten aus Teilmodellen
- Die Verwendbarkeit von verschiedenen Simulationskernen zur Simulation der Teilmodelle
- Die Verwendung von kontinuierlichen Simulationsmodellen

Eine Lösung für diese Probleme würde die Flexibilität des vorgestellten Konzeptes noch weiter erhöhen. Es gibt jedoch noch zahlreiche weitere Aspekte, die die Anwendbarkeit und die Akzeptanz der vorgestellten Konzepte erhöhen könnte. Drei hiervon werden im folgenden aufgelistet.

Um die zu verknüpfende Prozessumgebung auch ohne die Entwicklung einer speziellen Erweiterungskomponente beschreiben zu können, wäre es denkbar, generische Modelleditoren bereit zu stellen, mit denen Modelle für verschiedene Domänen entwickelt werden können. Dies könnten zum Beispiel graphische Editoren sein, welche Notationen wie ausgewählte Diagrammformen der UML unterstützen. Die Aktivitätsdiagramme der UML mit ihrer Erweiterung für Simulationsstudien (vgl. Knaak und Page (2006)) wären ein gangbarer Weg. Auch die Nutzung der DEVS wäre eine erfolgversprechende Möglichkeit. Beide Notationen haben bereits ihre Einsatzfähigkeit für Simulationszwecke unter Beweis gestellt. Zu untersuchen wäre hierbei, welche Erweiterungen für diese Notationen notwendig wären, um die Kopplung an BPMN-Modelle zu ermöglichen. Dies wurde im Rahmen der Softwareentwicklung in Bode et al. (2011) bereits untersucht, eine Fokussierung des Themas auf die Zwecke der Simulation wäre notwendig. Es wäre ebenfalls zu untersuchen, ob der Entwicklungsaufwand für Teilmodelle in einem Maße sinkt, der den Verlust der domänenspezifischen Symbole und Modelleditoren rechtfertigt. Eventuell könnten solcherart Komponenten genutzt werden, um eine erste prototypische Evaluation der Konzepte zu ermöglichen, bevor eine auf den Einsatzzweck zugeschnittene Domänenkomponente geschaffen wird. Durch Im- und Export der DEVS bzw. UML-Modelle könnte der Aufwand der Implementation der Simulationsteilmodelle verringert werden, so dass nur noch ein Editor zur Parametrierung der Modelle bereitgestellt werden muss.

Das Empinia-Rahmenwerk ist grundsätzlich zur Entwicklung einer Software wie in dieser Arbeit vorgestellt geeignet. Die Konzepte sind jedoch übertragbar auf andere Plug-in-Plattformen und nicht auf die in dieser Arbeit genutzte .NET-basierte BPMN-Bibliothek beschränkt. So könnten diese Konzepte zum Beispiel mit der Eclipse Plattform in Java reimplementiert werden. Mit dem Werkzeug „BPMN2Modeler“ steht hier ein Plug-in zur Manipulation von BPMN-Modellen zur Verfügung. Das Graphical Editing Framework (GEF) könnte zur Implementation von Editoren für die domänenspezifischen Teilmodelle genutzt werden. Das Plug-in SimTools bietet eine Experimentplanungsumgebung zur Parametrierung von Experimentläufen. Die Simulationsbibliothek DESMO-J kann in ihrer Java-Version genutzt werden. Der Vorteil des Eclipse-Rahmenwerks gegenüber dem Empinia-Rahmenwerk ist die Vielzahl zur Verfügung stehender Komponenten, welche zur Entwicklung von branchenspezifischen Lösungen wiederverwendet werden können. Unter Umständen können der Entwicklungsaufwand und damit die Kosten für neue Erweiterungen verringert werden. Jedoch fehlt eine BPMN-Simulationsbibliothek, denn derzeit zur Verfügung stehende Simulationskomponenten für Eclipse setzen nicht auf der BPMN, sondern auf Sprachen wie jBPM auf.

In dieser Arbeit wurde eine Erweiterung der BPMN zur Simulation genutzt. Leider ist die Simulation explizit nicht im Standard der BPMN 2.0 enthalten. Da die Ausführungssemantik bereits exakt beschrieben ist, wäre es für eine künftige Version der BPMN wünschenswert, diejenigen Erweiterungsattribute, die für die Simulation benötigt wer-

den, in den Standard aufzunehmen. Besonders wichtig hierbei sind die Laufzeiten von Aktivitäten und die Zwischenankunftszeiten von Ereignissen mit Hilfe von stochastischen Verteilungen. Hierfür müsste eine einheitliche Sprachregelung zur Benennung von stochastischen Verteilungen und deren Parametern festgelegt werden. Zudem müsste festgelegt werden, welche stochastischen Verteilungen in einer standardkonformen Simulationsanwendung zwingend enthalten sein müssen, um die Austauschbarkeit der genutzten Werkzeuge zu ermöglichen. Wie schon bei der Erzeugung von Workflows aus BPMN-Modellen, bleibt auch bei der Simulation zudem das Problem, dass mit der Abfragesprache XPath zwar ein Vorschlag zur Definition von Bedingungen an Gateways gemacht wurde, dieser jedoch von eher wenigen Werkzeugen unterstützt wird. Eine Festlegung auf eine solche Notation wäre ebenfalls notwendig, um einen Standard zur Simulation von BPMN-Prozessen zu schaffen. Dieses Vorhaben soll mit entsprechenden Publikation vorangetrieben werden.

Die Findung weiterer möglicher Anwendungsszenarien kann im Rahmen dieser Arbeit nicht vollumfänglich geleistet werden. Die Konzepte wurden geliefert; ob diese in weitere innovative Anwendungen überführt werden können, muss in künftigen Arbeiten geklärt werden. Obwohl ökonomische Kennzahlen im Rahmen der Geschäftsprozessmodellierung und -simulation häufig die entscheidende Rolle spielen, ist auch die Etablierung von ökologischen oder soziologischen Kennzahlen im Rahmen der Nachhaltigkeitsanalyse von Geschäftsprozessen ein denkbares Anwendungsfeld. Die Ökobilanzierung von Geschäftsprozessen kann zur detaillierten Berechnung von Carbon Footprints bei der Entwicklung und Fertigung von Produkten eine Rolle spielen (vgl. Wohlgemuth (2005)). Die Betrachtung soziologischer Faktoren wie der Gesundheit oder Zufriedenheit der Arbeitnehmer kann ein langfristiger Faktor für die Akzeptanz und den Erfolg einer Unternehmung sein (vgl. Widok (2009)). Beide Ansätze bedürfen aber dringend der Integration entsprechend wissenschaftlich fundierter Teilmodelle. Hierfür wurde mit dieser Arbeit ein Grundstein gelegt.

Anhang

A. Beispielmodell zur BPMN-Bibliothek

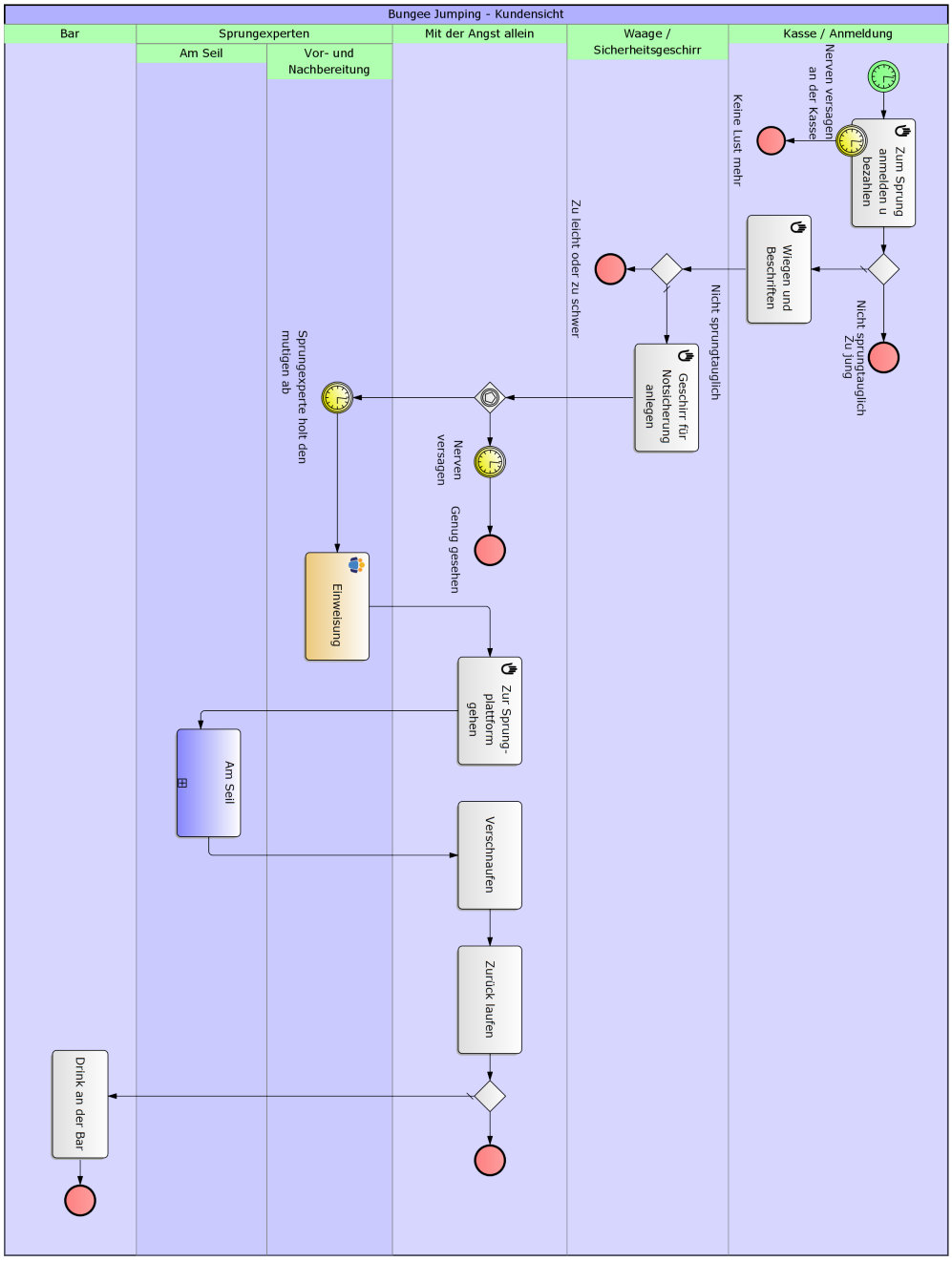


Abbildung A.1.: Kundenprozess beim Durchlaufen einer Bungee Jumping Anlage

Histogram for Processing time of AmSeil#1 (started at event: #1)

[0,2 - 0,3)	[0,3 - 0,4)	[0,4 - 0,5]
3	267	46

top

Process cycle times

Title	(Re)set	Obs	Mean	Std.Dv	Min	Max	Lower Quartile	Median	Upper Quartile
Processing time of AmSeil#1 (started at event: #1)	Donnerstag, 01.01.2015, 00:00:00 Uhr	316	0d, 0h, 22m and 12s	0d, 0h, 1m and 40s	0d, 0h, 17m and 28s	0d, 0h, 26m and 29s	0d, 0h, 21m and 2s	0d, 0h, 22m and 12s	0d, 0h, 23m and 25s

top

Concurrently existing processes

Title	(Re)set	Obs	Mean	Std.Dv	Min	Max	Period
Coexistent instances of processes in AmSeil#1	Donnerstag, 01.01.2015, 00:00:00 Uhr	633	0,9827	0,17173	0	2	4d, 23h, 2m and 5s

top

Abbildung A.2.: Simulationsergebnisse zu Prozesslaufzeiten

Resource Pools

Title	Idle	Waiting	In use	Other role	Blocked	Setup	Post-processing	Basic Load
Resource Pool for Sicherheitsexperte	167,377618439496	0	72,622381560504	0	0	0	0	0
Resource Pool for Sprungexperte	27,7437984347343	151,200180366635	181,056021198 63	0	0	0	0	0
Resource Pool for Kassenleitung	6,49543854594231	0	113,504561454058	0	0	0	0	0
Resource Pool for Bungeeseil	3,76756736636162	0	116,232432633638	0	0	0	0	0

top

Queues

Title	Qorder	(Re)set	Obs	QLimit	Qmax	Qnow	Qavg.	Zeros	max.Wait	avg.Wait	Refused
Waiting Queue at Resource Pool for Sicherheitsexperte	FIFO	Donnerstag, 01.01.2015, 00:00:00 Uhr	9	unlimit.	1	0	0,00373	0	0d, 0h, 5m and 55s	0d, 0h, 2m and 59s	0
Waiting Queue at Resource Pool for Sprungexperte	FIFO	Donnerstag, 01.01.2015, 00:00:00 Uhr	592	unlimit.	46	41	17,79206	0	0d, 9h, 5m and 10s	0d, 3h, 19m and 16s	0
Waiting Queue at Resource Pool for Kassenleitung	FIFO	Donnerstag, 01.01.2015, 00:00:00 Uhr	882	unlimit.	8	5	1,99577	0	0d, 0h, 41m and 55s	0d, 0h, 16m and 11s	0
Waiting Queue at Resource Pool for Bungeeseil	FIFO	Donnerstag, 01.01.2015, 00:00:00 Uhr	307	unlimit.	16	4	5,0092	0	0d, 5h, 46m and 57s	0d, 1h, 56m and 4s	0

top

Abbildung A.3.: Simulationsergebnisse zu Ressourcen

Event counter

Title	Type	(Re)set	Obs	Current Value	Min	Max
Triggered event "#9"	Count	Donnerstag, 01.01.2015, 00:00:21 Uhr	586	586	0	586
Triggered event "Nerven versagen an der Kasse#1"	Count	Donnerstag, 01.01.2015, 00:07:45 Uhr	78	78	0	78
Triggered event "VolVolle Stunde#1"	Count	Donnerstag, 01.01.2015, 00:30:00 Uhr	352	352	0	352
Triggered event "Nerven versagen beim warten mit Notsicherung#1"	Count	Donnerstag, 01.01.2015, 00:30:00 Uhr	2	2	0	2
Triggered event "Zu jung#1"	Count	Donnerstag, 01.01.2015, 00:38:48 Uhr	67	67	0	67
Triggered event "Zu leicht oder zu schwer#1"	Count	Donnerstag, 01.01.2015, 00:50:18 Uhr	80	80	0	80
Triggered event "Keine Lust mehr#1"	Count	Donnerstag, 01.01.2015, 00:50:48 Uhr	78	78	0	78
Triggered event "Ende mit Drink an der Bar#1"	Count	Donnerstag, 01.01.2015, 02:32:33 Uhr	261	261	0	261
Triggered event "Ende ohne Drink an der Bar#1"	Count	Donnerstag, 01.01.2015, 03:51:55 Uhr	52	52	0	52
Triggered event "Genug gesehen#1"	Count	Freitag, 02.01.2015, 16:22:12 Uhr	2	2	0	2

top

Abbildung A.4.: Simulationsergebnisse zu Ereignissen

B. Auszüge aus dem Programmcode des Prototpyen

Quellcode B.1: Schnittstelle des SolutionSercvie

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using Empinia.UI.Workbench;
6  using System.ComponentModel;
7  using Empinia.UI;
8
9  namespace Simulation.DESMOWindparkSolutions.Api
10 {
11  public interface ISolutionService
12  {
13   ISelectionService SelectionService { get; }
14   Solution CurrentSolution { get; set; }
15   List<ModelTypeConfiguration> KnownModelTypeConfigurations { get; }
16   IDWSModel SelectedModel { get; set; }
17   void OnPerspectiveChanged(object sender, WorkbenchPartEventArgs e);
18   void TryLoadLastSolution();
19   void OnSolutionChanged(object sender, PropertyChangedEventArgs e);
20   event PropertyChangedEventHandler SolutionChanged;
21   ModelTypeConfiguration GetConfigurationFor(IDWSModel iDWSModel);
22   ModelTypeConfiguration GetConfigurationFor(Type type);
23  }
24 }
```

Quellcode B.2: Klasse ModelTypeConfiguration

```
1 namespace Simulation.DESMOWindparkSolutions.Api
2 {
3  public class ModelTypeConfiguration
4  {
5   public ModelTypeConfiguration()
6   {
7    DomainSpecificElements = new List<DomainSpecificElement>();
8   }
9   public List<DomainSpecificElement> DomainSpecificElements{get; set
        ;}
10  public string ImplementatinClass { get; set; }  public string
        LoadCommmandID { get; set; }
11  public string ModelConfigurerID { get; set; }
12  public string ModelConnectorID { get; set; }
13  public string ModelConverterID { get; set; }
14  public string ModelDeletedCallbackCommand { get; set; }
15  public string ModelDescription {get; set; }
16  public string ModelType { get; set; }
17  public string PostFix { get; set; }
18  public string ShowModelCommandID { get; set; }
19 }
20 }
```

Quellcode B.3: Anmeldung des Erweiterungspunktes für ModelTypeConfigurations

```
1 <extensionPoint id="simulation.desmoWindparkSolutions.api.
      ModelTypeConfiguration">
2  <name translationId="{translate::simulation.
       desmoWindparkSolutions.api.modelTypeConfiguration.name}">
       Model-type Configuration</name>
3  <description translationId="{simulation.desmoWindparkSolutions
       .api.modelTypeConfiguration.description}">provides meta
       information on a model-type like the ids for proper load
       and save commands</description>
4 </extensionPoint>
```

Quellcode B.4: Auszug der Klasse BPMNActivity

```
1 namespace Desmoj.Extensions.Bpmn.Activities
2 {
3  public class BPMNActivity :
       Desmoj.Extensions.Bpmn.Internal.BPMNActiveNode
4  {
5    [...]
6
7    public delegate void ActivityStartEventHandler
       (BPMNActivity sender, BPMNProcess process);
8    public delegate void ActivityFinishEventHandler
       (BPMNActivity sender, double duration, BPMNProcess process);
9    public delegate void MultipleActivityStartEventHandler
       (BPMNActivity sender, BPMNProcess process);
10   public delegate void MultipleActivityFinishEventHandler
       (BPMNActivity sender, double duration, BPMNProcess process);
11
12   public event ActivityStartEventHandler StartActivity;
13   public event ActivityFinishEventHandler FinishedActivity;
14   public event MultipleActivityStartEventHandler
       StartMultipleActivity;
15   public event MultipleActivityFinishEventHandler
       FinishedMultipleActivity;
16
17   [...]
18
19 }
```

Quellcode B.5: Klasse DomainSpecificActivity

```
1 [...]
2 using Desmoj.Extensions.Bpmn.Activities;
3 using Desmoj.Extensions.Bpmn.Processes;
4
5 namespace Desmoj.Extensions.Bpmn.DomainSpecificExtensions
6 {
7  public class DomainSpecificActivity : IDomainSpecificActivity
8  {
9   protected BPMNActivity _activity;
10  protected string _domainId;
11
12  public DomainSpecificActivity(BPMNActivity activity, string id)
13  {
14    _domainId = id;
15    _activity=activity;
16    _activity.AddDomainSpecificActivity(_domainId, this);
```

```
17      _activity.StartActivity += new BPMNActivity.
            ActivityStartEventHandler(this.StartActivity);
18      _activity.FinishedActivity += new BPMNActivity.
            ActivityFinishEventHandler(this.FinishActivity);
19      _activity.StartMultipleActivity += new BPMNActivity.
            MultipleActivityStartEventHandler(this.StartMultipleActivity);
20      _activity.FinishedMultipleActivity += new BPMNActivity.
            MultipleActivityFinishEventHandler(this.FinishMultipleActivity
            );
21    }
22
23    public virtual void StartActivity(BPMNActivity activity,
          BPMNProcess process)
24    {
25    }
26
27    public virtual void FinishActivity(BPMNActivity activity,
          BPMNProcess process)
28    {
29    }
30
31    public virtual void StartMultipleActivity(BPMNActivity activity,
          BPMNProcess process)
32    {
33    }
34
35    public virtual void FinishMultipleActivity(BPMNActivity activity,
          double duration, BPMNProcess process)
36    {
37    }
38  }
39 }
```

Quellcode B.6: Klasse DomainSpecificArtifact

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using Desmoj.Extensions.Bpmn.Internal;
7  using Desmoj.Core.Simulator;
8
9  namespace Simulation.DesmoJ.Extensions.Bpmn.DomainSpecificExtensions
10 {
11   public abstract class DomainSpecificArtifact: BPMNElement {
12
13     public DomainSpecificArtifact(DomainArtifact artifact, Desmoj.Core.
         Simulator.Model model, string name, bool showInTrace) : base(
         model, name, showInTrace)
14     {
15       DomainArtifact = artifact;
16       DomainArtifact.DomainSpecificArtifact = this;
17     }
18
19     public Queue EntityQueue { get; set; }
20
21     public DomainArtifact DomainArtifact { get; set; }
22
23   }
24 }
```

Quellcode B.7: Schnittstelle des Parametrierers

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using Simulation.Experimentation.DomainModel;
6 using Simulation.DESMOWindparkSolutions.Api;
7
8 namespace Simulation.Experimentation.Api
9 {
10  public interface IModelConfigurer
11  {
12   List<ParameterValueDescriptor> GetParameterRange(IDWSModel model);
13
14   IDWSModel GetConfiguredModel(IDWSModel model, List<
       ParameterValueDescriptor> parameters);
15  }
16 }
```

Quellcode B.8: Schnittstelle der Simulationsfabrik

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using Simulation.BPMN.Scripting;
6 using Desmoj.Core.Dist;
7 using Desmoj.Core.Simulator;
8 using Simulation.DESMOWindparkSolutions.Api;
9
10 namespace Simulation.Experimentation.Api
11 {
12  public interface IModelConverter
13  {
14    Desmoj.Core.Simulator.Model CreateSimulationModel(IDWSModel
        document);
15    void Initialize(DistributionManager distributionManager, Model
        supermodel);
16    Desmoj.Core.Simulator.Model GetConvertedModel();
17
18    Dictionary<object, object> GetStaticSimMapping();
19  }
20 }
```

Quellcode B.9: Schnittstelle des Modellkopplers

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using Simulation.DESMOWindparkSolutions.Api;
6
7 namespace Simulation.Experimentation.Api
8 {
9  public interface IModelConnector
10  {
11   IEnumerable<Type> RequiredTypes{get;}
12
13   Type ProvidedType { get; }
14
15   void ConnectModels(IEnumerable<Desmoj.Core.Simulator.Model>
        ownDomainSimulationModels, IEnumerable<KeyValuePair<Type,
        Dictionary<object, object>>> foreignModelsVisualToSimMappings);
16  }
17 }
```

Quellcode B.10: Abstrakte Klasse für den Modellkoppler

```
1 [...]
2 using Desmoj.Extensions.Bpmn.Containers;
3 using Simulation.DesmoJ.Extensions.Bpmn.DomainSpecificExtensions;
4 using Desmoj.Core.Simulator;
5 using BPMN.Model;
6 using Desmoj.Extensions.Bpmn.DomainSpecificExtensions;
7 using Desmoj.Extensions.Bpmn.Activities;
8 using Desmoj.Extensions.Bpmn.Events;
9
10 namespace Simulation.Experimentation.Api
11 {
12  public abstract class AbstractModelConnector : IModelConnector
13  {
14   protected SimulationOutputService _errorService;
15
16   public AbstractModelConnector()
17   {
18    _errorService = SimulationOutputService.GetInstance();
19   }
20
21   public IEnumerable<Type> RequiredTypes
22   {
23    get { return RequiredModelTypes; }
```

```
24  }
25
26  public virtual IEnumerable<Type> RequiredModelTypes
27  {
28   get { return new List<Type>() { typeof(BPMNSimulationModel) }; }
29  }
30
31  public Type ProvidedType
32  {
33   get { return ProvidedModelType; }
34  }
35
36  public abstract Type ProvidedModelType { get; }
37
38  public void ConnectModels(IEnumerable<Model>
        ownDomainSimulationModels, IEnumerable<KeyValuePair<Type,
        Dictionary<object, object>>> foreignModelsVisualToSimMappings)
39  {
40   foreach (KeyValuePair<Type, Dictionary<object, object>> modelType
         in foreignModelsVisualToSimMappings)
41   {
42   if (modelType.Key == typeof(BPMNSimulationModel))
43   {
44    foreach (object graphicalBPMN in modelType.Value.Keys)
45    {
46     object desmoBPMN = modelType.Value[graphicalBPMN];
47     if (desmoBPMN.GetType() == typeof(DomainSpecificStartEvent))
48     {
49      ConnectDomainSpecificStartEvent((DomainSpecificStartEvent)
            desmoBPMN, (tStartEvent)graphicalBPMN,
            ownDomainSimulationModels);
50     }
51     else if (desmoBPMN.GetType() == typeof(DomainArtifact))
52     {
53      ConnectDomainSpecificArtifact((DomainArtifact)desmoBPMN, (
            tDataObject)graphicalBPMN, ownDomainSimulationModels);
54     }
55     else if (desmoBPMN.GetType() == typeof(BPMNActivity) && ((
           tActivity)graphicalBPMN).IsDomainSpecificElement)
56     {
57      ConnectDomainSpecificActivity((BPMNActivity)desmoBPMN, (
            tActivity)graphicalBPMN, ownDomainSimulationModels);
58     }
59     else if (desmoBPMN.GetType() == typeof(BPMNIntermediateEvent) &&
            ((tEvent)graphicalBPMN).IsDomainSpecificElement)
60     {
```

```
61          ConnectDomainSpecificIntermediateEvent((BPMNIntermediateEvent)
                desmoBPMN, (tEvent)graphicalBPMN, ownDomainSimulationModels)
                ;
62        }
63        else if (desmoBPMN.GetType() == typeof(BPMNEndEvent) && ((tEvent
              )graphicalBPMN).IsDomainSpecificElement)
64        {
65          ConnectDomainSpecificEndEvent((BPMNIntermediateEvent)desmoBPMN,
                (tEvent)graphicalBPMN, ownDomainSimulationModels);
66        }
67      }
68    }
69    else
70    {
71      ConnectNonBPMNModel(ownDomainSimulationModels, modelType);
72    }
73  }
74  return;
75 }
76
77 public abstract void ConnectDomainSpecificStartEvent(
      DomainSpecificStartEvent startEvent, tStartEvent graphical,
      IEnumerable<Model> ownDomainSimulationModels);
78
79 public abstract void ConnectDomainSpecificIntermediateEvent(
      BPMNIntermediateEvent intEvent, tEvent graphical, IEnumerable<
      Model> ownDomainSimulationModels);
80
81 public abstract void ConnectDomainSpecificEndEvent(
      BPMNIntermediateEvent intEvent, tEvent graphical, IEnumerable<
      Model> ownDomainSimulationModels);
82
83 public abstract void ConnectDomainSpecificArtifact(DomainArtifact
      artifact, tDataObject graphical, IEnumerable<Model>
      ownDomainSimulationModels);
84
85 public abstract void ConnectDomainSpecificActivity(BPMNActivity
      activity, tActivity graphical, IEnumerable<Model>
      ownDomainSimulationModels);
86
87 public abstract void ConnectNonBPMNModel(IEnumerable<Model>
      ownDomainSimulationModels, KeyValuePair<Type, Dictionary<object,
      object>> foreignModel);
88 }
89 }
```

C. Benutzeroberfäche des Prototypen

Wesentliche Beiträge zur Implementation der Benutzeroberfläche des Prototypen wurden in den studentischen Arbeiten Oeser (2013) und Stehle (2014) geleistet. Die folgenden Abbildungen sind daher der Arbeit Stehle (2014) entnommen.

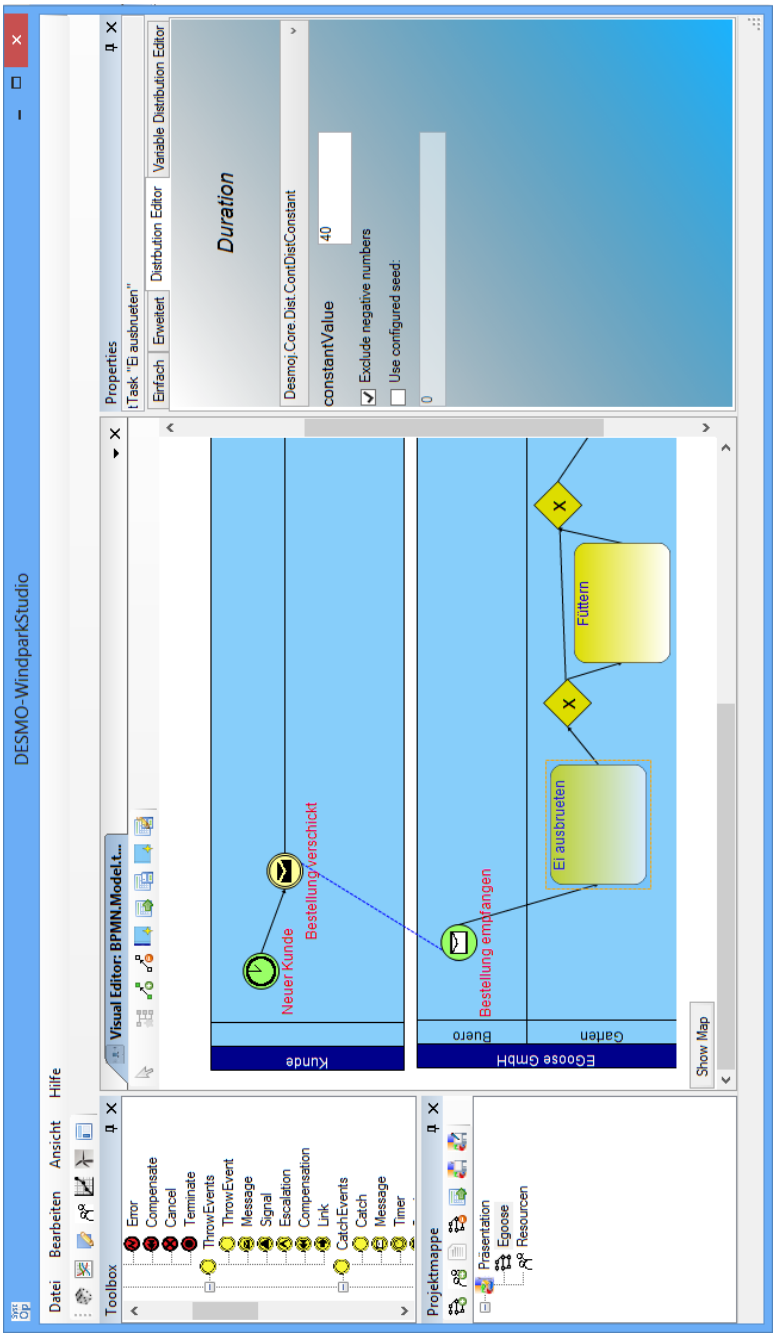


Abbildung C.1.: Benutzeroberfläche zur Modellierung von BPMN-Prozessen (Abbildung aus Stehle (2014))

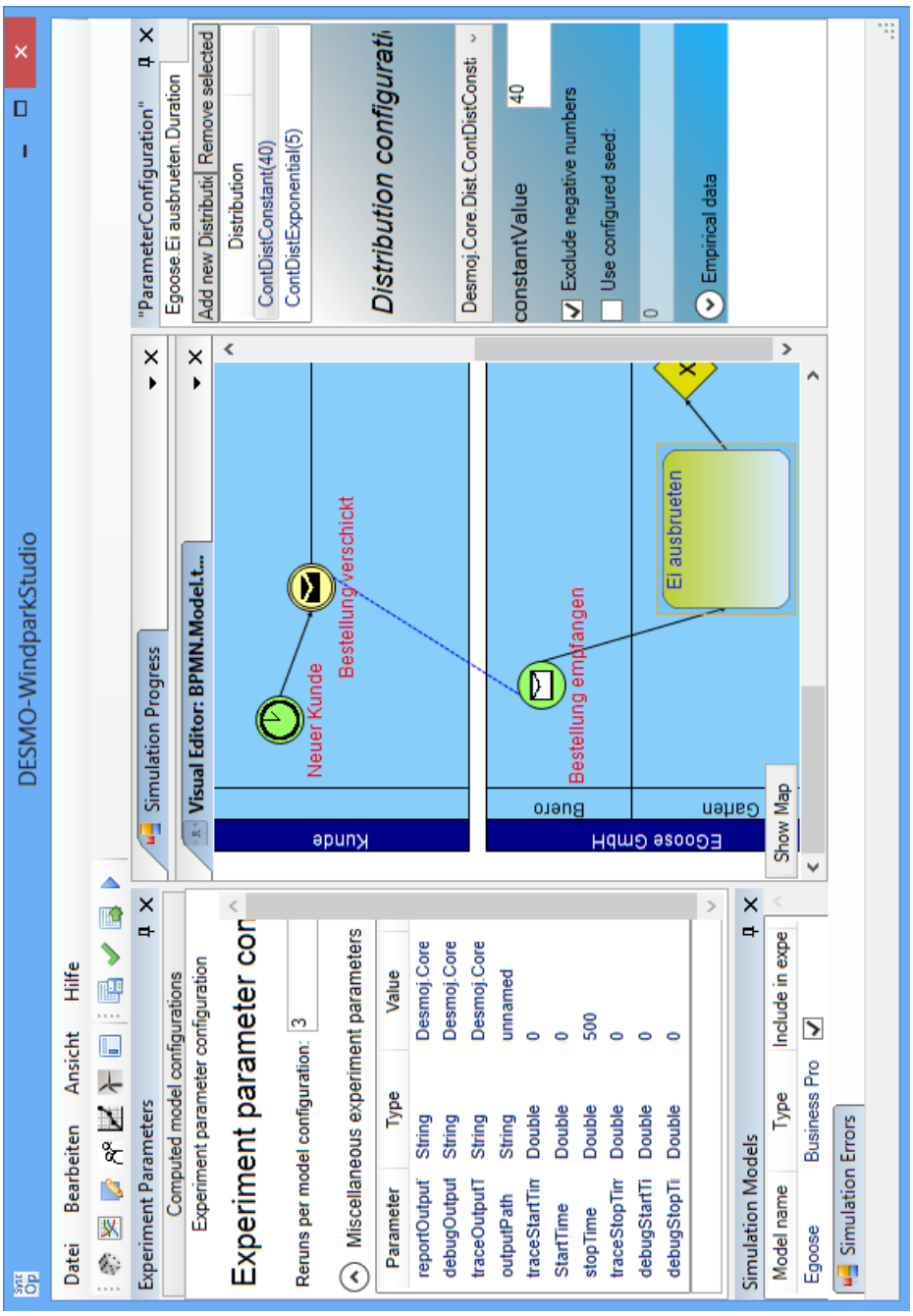


Abbildung C.2.: Benutzeroberfläche zur Experimentplanung (Abbildung aus Stehle (2014))

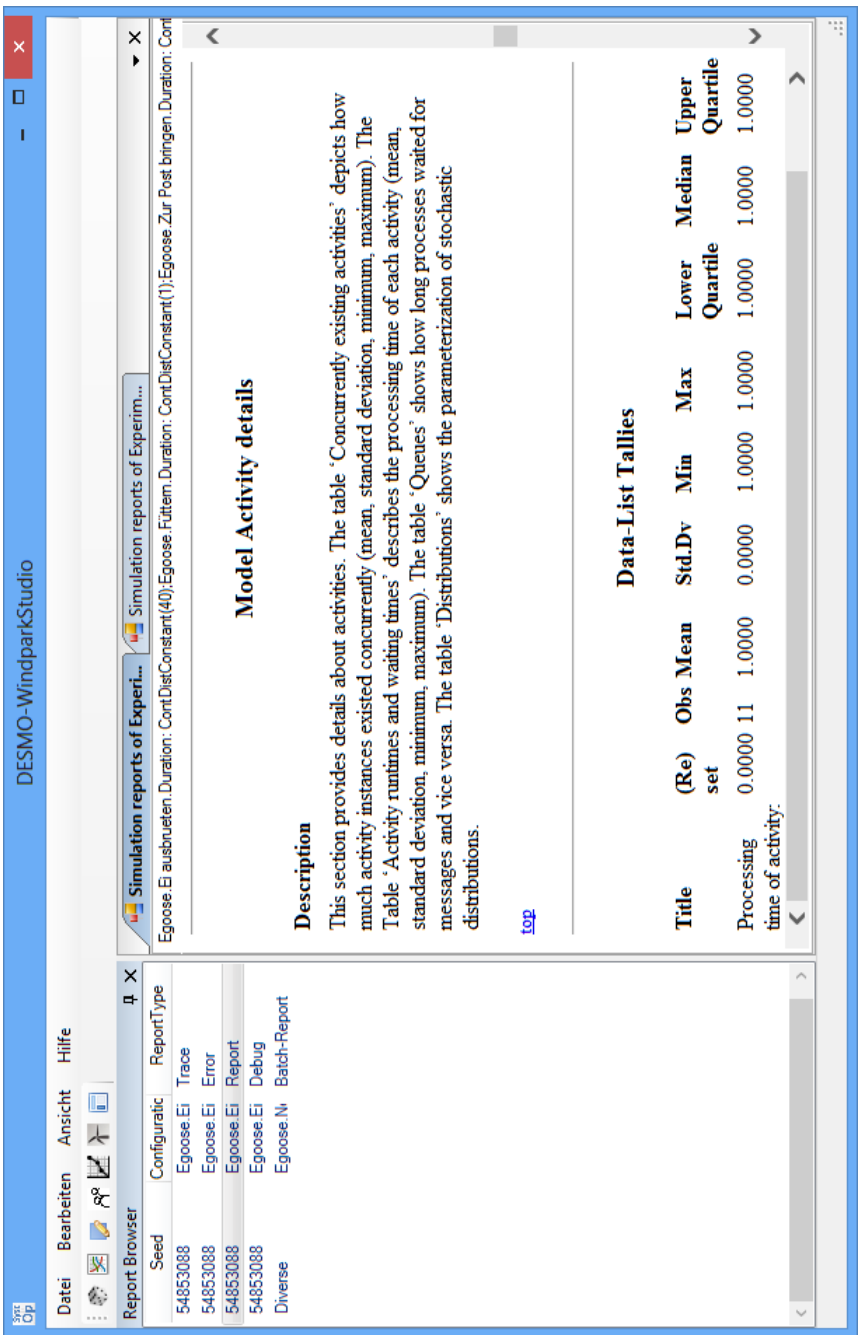


Abbildung C.3.: Benutzeroberfläche zur Darstellung von Simulationsergebnissen (Abbildung aus Stehle (2014))

D. Benutzeroberfläche zur Wettererweiterung

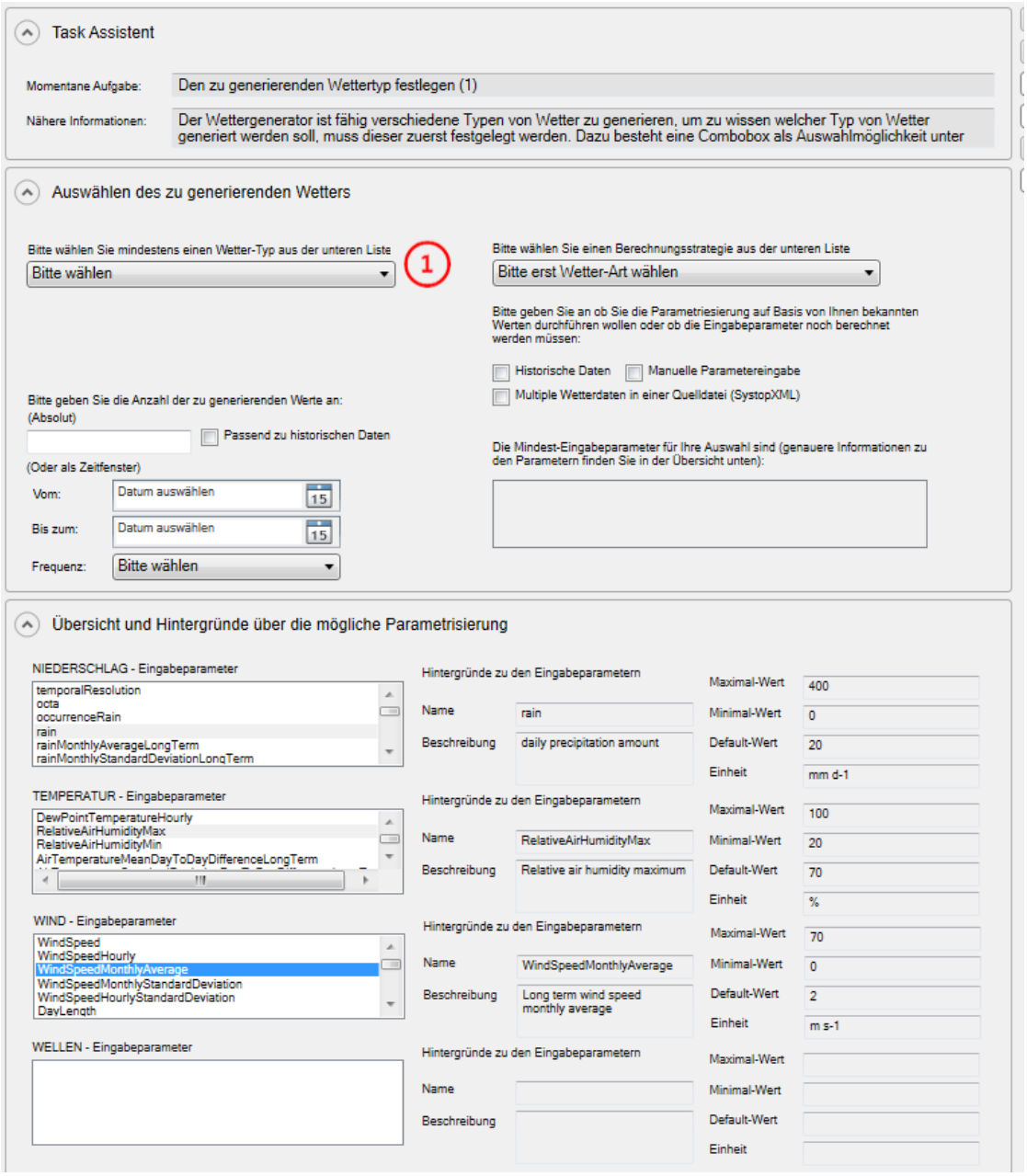


Abbildung D.1.: Benutzeroberfläche Wettergenerator - Schritt 1 - Auswahl der zu erzeugenden Daten

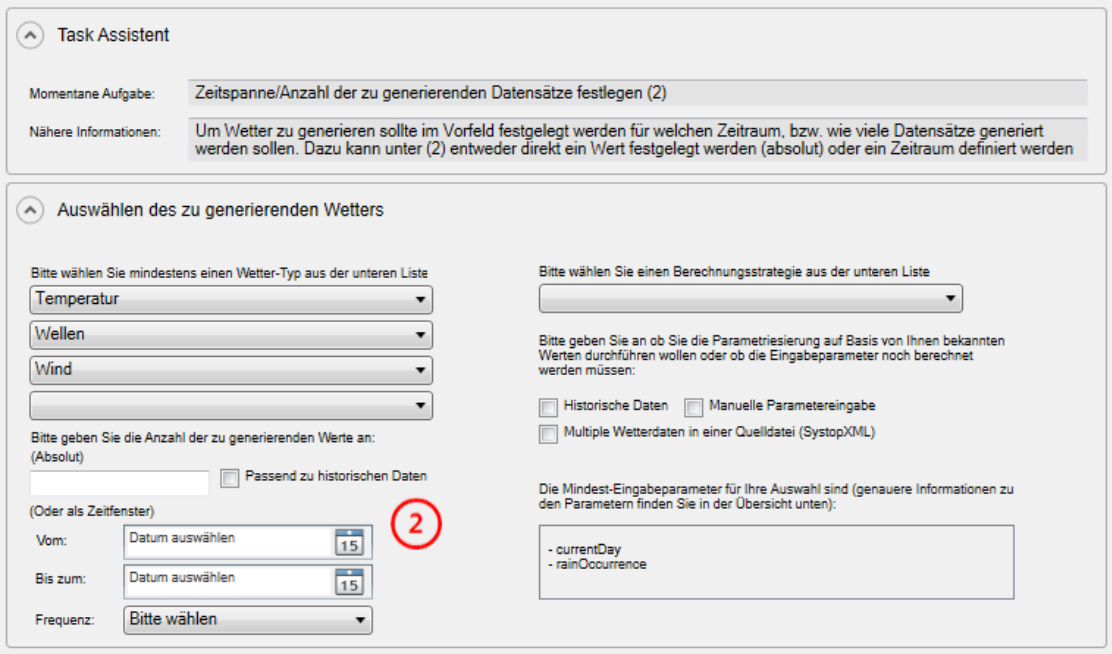


Abbildung D.2.: Benutzeroberfläche Wettergenerator - Schritt 2 - Auswahl des Zeitraums

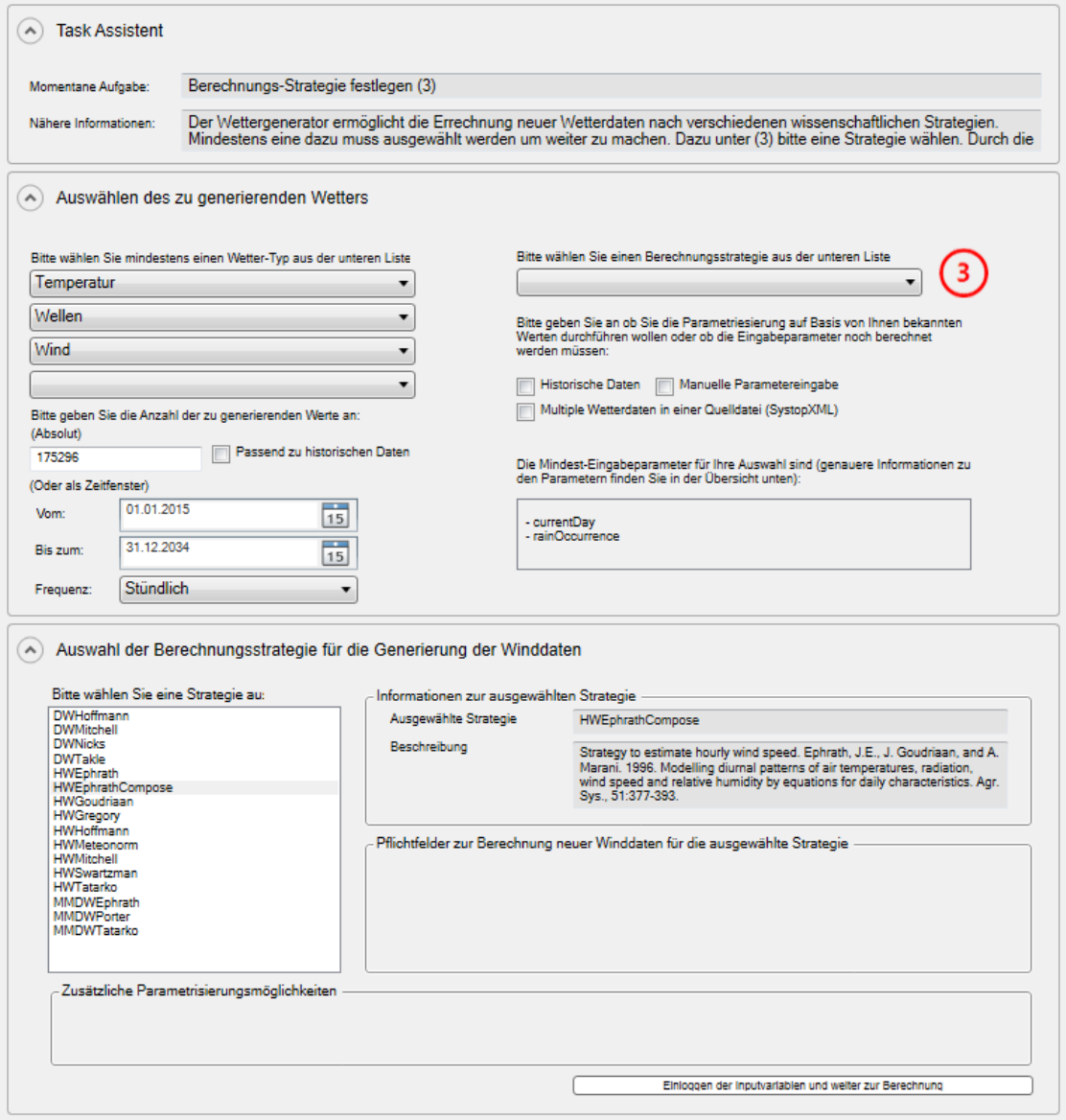


Abbildung D.3.: Benutzeroberfläche Wettergenerator - Schritt 3 - Auswahl der Berechnungsstrategie

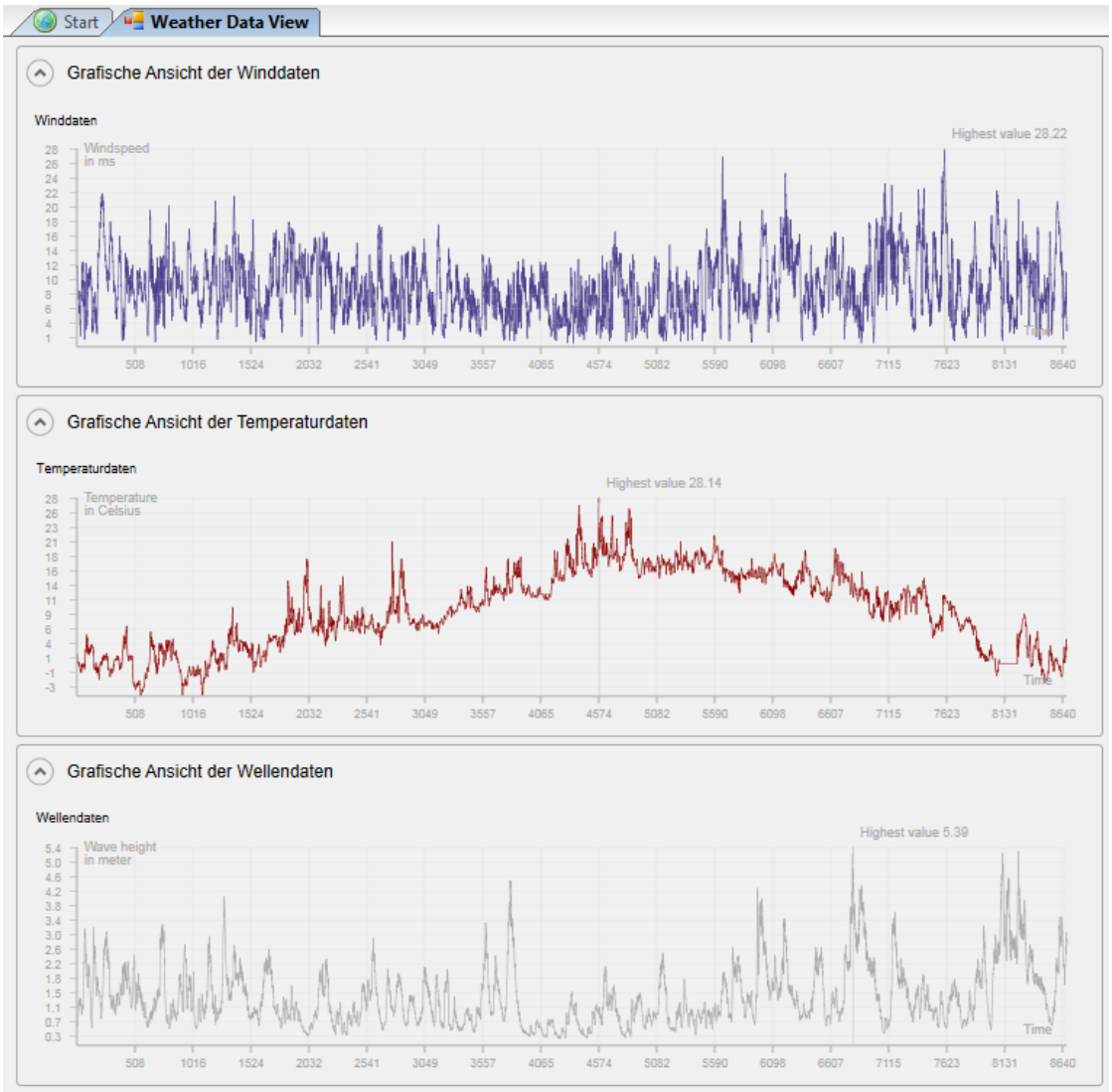


Abbildung D.4.: Benutzeroberfläche Wettergenerator - Ansicht der generierten Daten

E. Auszüge aus dem Programmcode zur Wettererweiterung

Quellcode E.1: Klasse BPMNWeatherConditionEvent

```
1 namespace Simulation.WeatherGeneration
2 {
3  public class BPMNWeatherConditionEvent :
       DomainSpecificIntermediateCatchEvent
4  {
5   public void TriggerEvent(String weatherExpression)
6   {
7    List<BPMNProcess> _passivatedProcessesTemp = new List<BPMNProcess
       >();
8    _passivatedProcessesTemp.AddRange(_passivatedProcesses);
9    foreach (SimProcess process in _passivatedProcessesTemp)
10   {
11    if (process is BPMNProcess)
12    {
13     ((BPMNModel)this.GetEvent().GetModel()).GetRuleEvaluator().
         Evaluate(((BPMNProcess)process).GetMainProcess(),
         weatherExpression, this.PresentTime());
14     bool value = ((BPMNModel)this.GetEvent().GetModel()).
         GetRuleEvaluator().EvaluateToBool(((BPMNProcess)process).
         GetMainProcess(), Condition, this.PresentTime());
15     if (value)
16     {
17      Release((BPMNProcess)process);
18     }
19    }
20   }
21  }
22 }
23 }
```

Quellcode E.2: Klasse WeatherConditionChangedEvent

```
1  namespace Simulation.WeatherGeneration
2  {
3   public class WeatherConditionChangedEvent : Desmoj.Core.Simulator.
       EventOf3Entities<Entity, Entity, Entity>
4   {
5    public override void EventRoutine(Entity wind, Entity wave, Entity
       temp)
6    {
7     string expression = "";
8     expression += CurrentValueToExpression((
         DesmoWeatherDataSeriesEntity)wind);
9     expression += CurrentValueToExpression((
         DesmoWeatherDataSeriesEntity)wave);
10    expression += CurrentValueToExpression((
         DesmoWeatherDataSeriesEntity)temp);
11    foreach (BPMNWeatherConditionEvent specEvent in ((
         DesmoWeatherModel)this.GetModel()).BPMNWeatherConditionEvents)
12    {
13     specEvent.TriggerEvent(expression);
14    }
15    ScheduleNextEvent(entity1, entity2, entity3);
16   }
17
18   private string CurrentValueToExpression(
       DesmoWeatherDataSeriesEntity entity)
19   {
20    double value = entity.WeatherDataSingleEntry[entity.nextIndex];
21    string expression = entity.ObjectOfInvestigation + "=" + value + "
         ;";
22    entity.nextIndex++;
23    return expression;
24   }
25
26   private void ScheduleNextEvent(Entity entity1, Entity entity2,
       Entity entity3)
27   {
28    var e = new WeatherConditionChangedEvent(this.GetModel(), "Weather
         Condition Event", false);
29    e.Schedule(entity1, entity2, entity3, new Desmoj.Core.Simulator.
         TimeSpan(intervallInHours));
30   }
31   }
32 }
```

Quellcode E.3: Visuelles Element zur Anzeige eines Wettereignisses im BPMN-Editor

```
1  namespace WeatherGeneration.DomainModel
2  {
3   public class WeatherEventVisualizer : BPMN.UI.Visualisations.Event
4   {
5    private string textfield = null;
6    private Empinia.UI.GraphicalModeling.Figures.Label m_Label;
7
8    public WeatherEventVisualizer():base()
9    {
10    this.Anchor = new AbsoluteAnchor(0, 0);
11    this.LocationChanged += onElementMoved;
12   }
13
14   public override bool CanVisualize(object logicalElement)
15   {
16    if ((LogicalElement is tEvent) && ((tEvent)logicalElement).
        IsDomainSpecificElement && "WeatherEvent".Equals(((tEvent)
        logicalElement).DomainSpecificType) && !
        EqualVisualizationOfThisTypeExistsFor(logicalElement))
17     return true;
18    else
19     return false;
20   }
21
22   public override bool CanInitialize()
23   {
24    CanVisualize(LogicalElement);
25   }
26
27   public override object Clone()
28   {
29    var clone = new EventVisualizer();
30    clone.Style = this.Style;
31    clone.Configure(Extension);
32    return clone;
33   }
34
35   public override void SetSymbolforEvent(List<tEventDefinition>
       Eventtypes)
36   {
37    drawSymbolForEvent(Assembly.GetExecutingAssembly(), "
        WeatherGeneration.Resources.IntermediateEvent.png");
38   }
39  }
40 }
```

Quellcode E.4: Anmeldung des Wettereignisses im grafischen BPMN-Editor über eine Bundle.XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <bundle
3  id="WeatherGeneration.DomainModel"
4  provider=""
5  xml:lang="en-US"
6  xmlns="http://www.empinia.org/release/0.3/schemas/Bundle.xsd"
7  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8  <name translationId="{translate::WeatherGeneration.DomainModel
       }">WeatherGeneration.DomainModel</name>
9  <description translationId="{translate::WeatherGeneration␣
       DomainModel␣Bundle}">WeatherGeneration.DomainModel Bundle</
       description>
10
11  <!-- VISUALIZATIONS -->
12  <extension
13   id="empinia.Weather.visualizations.WeatherVisuals.
        WeatherEvent"
14   point="empinia.ui.graphicalModeling.visualizations"
15   implementation="WeatherGeneration.DomainModel.EventVisualizer
        ,␣WeatherGeneration.DomainModel">
16    <name>Weather Event</name>
17    <description/>
18    <visualization
19     id="empinia.Weather.visualizations.WeatherEvent"
20     visualizedType="BPMN.Model.tStartEvent,␣␣BPMN.DomainModel">
21   </visualization>
22  </extension>
23
24  <!-- EDITOR CONFIGURATION -->
25  <extension
26   id="empinia.Weather.editorConfigExtensions"
27   point="empinia.bpmn.graphicalModeling.
        visualEditorConfigurationExtensions">
28   <name>Additional BPMN Visualizations</name>
29   <description>Additional Visualizations for extended BPMN-
        elements like windparkevents are defined here</description
       >
30
```

```
31   <visualization
32     id="empinia.Weather.visualizations.WeatherVisuals.
         WeatherEvent"
33     visualizedType="BPMN.Model.tEvent, ␣␣BPMN.DomainModel">
34     <style
35       borderColor="FF000000"
36       backColor="FFDDDDDD"
37       lineStrength="1"
38       lineDashStyle="Solid"
39       foreColor="FF888888" />
40   </visualization>
41 </extension>
42 </bundle>
```

Quellcode E.5: Anmeldung der Wettererweiterung über eine Bundle.XML

```
1 <bundle
2  id="WeatherGeneration.Editor"
3  provider=""
4  xml:lang="en-US"
5  xmlns="http://www.empinia.org/release/0.3/schemas/Bundle.xsd"
6  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
7  <name translationId="{translate::WeatherGeneration.Editor}">
       WeatherGeneration.Editor</name>
8  <description translationId="{translate::WeatherGeneration␣
       Editor␣Bundle}">WeatherGeneration.Editor Bundle</
       description>
9
10 <!-- MODELTYPECONFIGURATION -->
11 <extension
12   id="WeatherGeneration.windparkModelTypeConfiguration"
13   point="simulation.desmoWindparkSolutions.api.
         ModelTypeConfiguration"
14   name="WeatherGeneration.modelConfiguration"
15   description="Meta␣information␣for␣dealing␣with␣Weather␣models
         ">
16   <ModelTypeConfiguration
17     Postfix="wtr"
18     LoadCommandID ="WeatherGeneration.Editor.Commands.
         LoadWeatherModelCommand"
19     ModelType ="Weather␣Model"
```

```
20      ModelConnectorID="simulation.weathergeneration.
            modelConnector"
21      ModelConverterID="simulation.weathergeneration.
            modelConverter"
22      ModelConfigurerID="simulation.weathergeneration.
            modelConfigurer"
23      ShowModelCommandID="WeatherGeneration.Editor.Commands.
            ShowWeatherModelCommand"
24      ImplementingClass="WeatherGeneration.DomainModel.
            WeatherModel"
25    </ModelTypeConfiguration>
26  </extension>
27
28  <!-- COMMANDS -->
29  <extension
30    id="WeatherGeneration.Editor.commands"
31    point="empinia.core.runtime.commands">
32    <name>WeatherGeneration Editor Commands</name>
33    <description>Commands contributed by the WeatherGeneration.
          Editor bundle</description>
34
35    <command
36      id="WeatherGeneration.Editor.Commands.
            LoadWeatherModelCommand"
37      implementation="WeatherGeneration.Editor.Commands.
            LoadWeatherModelCommand, WeatherGeneration.Editor">
38    </command>
39
40    <command
41      id="WeatherGeneration.Editor.Commands.
            ShowWeatherModelCommand"
42      implementation="WeatherGeneration.Editor.Commands.
            ShowWeatherModelCommand, WeatherGeneration.Editor">
43    </command>
44
45  </extension>
46 </bundle>
```

Quellcode E.6: Simulationsfabrik der Wettererweiterung

```
1 namespace Simulation.WeatherGeneration
2 {
3  class ModelConverter : IModelConverter
4  {
5   readonly Dictionary<object, object> _staticSimulationMapping = new
        Dictionary<object, object>();
6
7   private SimulationOutputService _errorService;
8   private DesmoWeatherModel _desmoModell;
9   private IDWSModel _document;
10  private IReportService _reportService;
11
12  public ModelConverter(IReportService reportServ)
13  {
14   _errorService = SimulationOutputService.GetInstance();
15   _reportService = reportServ;
16  }
17
18  public Model CreateSimulationModel(IDWSModel document)
19  {
20   var statischesModell = (WeatherModel)document;
21   var windDataModel = new DesmoWindDataModel(_desmoModell,
        statischesModell.WindDataModel, true);
22   var tempDataModel = new DesmoTempDataModel(_desmoModell,
        statischesModell.TempDataModel, true);
23   var rainDataModel = new DesmoRainDataModel(_desmoModell,
        statischesModell.RainDataModel.Name, true);
24   var waveDataModel = new DesmoWaveDataModel(_desmoModell,
        statischesModell.WaveDataModel.Name, true);
25
26   _desmoModell.WindDataModel = windDataModel;
27   _desmoModell.TempDataModel = tempDataModel;
28   _desmoModell.WaveDataModel = waveDataModel;
29   _desmoModell.RainDataModel = rainDataModel;
30
31   _staticSimulationMapping.Add(document, _desmoModell);
32          _staticSimulationMapping.Add(statischesModell.WindDataModel,
                windDataModel);
33          _staticSimulationMapping.Add(statischesModell.TempDataModel,
                tempDataModel);
34          _staticSimulationMapping.Add(statischesModell.RainDataModel,
                rainDataModel);
35          _staticSimulationMapping.Add(statischesModell.WaveDataModel,
                waveDataModel);
36
```

```
37      return _desmoModell;
38    }
39
40    public void initialize(DistributionManager distributionManager,
          Model supermodel)
41    {
42      _desmoModell = new DesmoWeatherModel(supermodel, "
            DesmoWeatherDataModell", true, true);
43      _desmoModell.Reset();
44    }
45
46    public Model GetConvertedModel()
47    {
48      return _desmoModell;
49    }
50
51    public Dictionary<object, object> GetStaticSimMapping()
52    {
53      return _staticSimulationMapping;
54    }
55
56    public IDWSModel GetOriginalModel()
57    {
58      return _document;
59    }
60
61    public Type GetSimulationModelType()
62    {
63      return typeof(DesmoWeatherModel);
64    }
65  }
66 }
```

Quellcode E.7: Modellkoppler der Wettererweiterung

```
1  namespace Simulation.WeatherGeneration
2  {
3   class ModelConnector : AbstractModelConnector
4   {
5    public ModelConnector() : base()
6    {}
7
8    public override Type ProvidedModelType
9    {
10    get { return typeof(DesmoWeatherModel); }
11   }
12
13   public override void ConnectDomainSpecificStartEvent(
         DomainSpecificStartEvent startEvent, tStartEvent graphical,
         IEnumerable<Model> ownDomainSimulationModels)
14   {}
15
16   public override void ConnectDomainSpecificArtifact(DomainArtifact
         artifact, tDataObject graphical, IEnumerable<Model>
         ownDomainSimulationModels)
17   {}
18
19   public override void ConnectDomainSpecificActivity(BPMNActivity
         activity, tActivity graphical, IEnumerable<Model>
         ownDomainSimulationModels)
20   {}
21
22   public override void ConnectDomainSpecificIntermediateEvent(
         BPMNIntermediateEvent intEvent, tEvent graphical, IEnumerable<
         Model> ownDomainSimulationModels)
23   {
24    if (graphical.DomainSpecificType.Equals("WeatherEvent"))
25    {
26     var domainSpecificEvent = new BPMNWeatherConditionEvent(intEvent,
           "Weather Condition Changed Event", true);
27     domainSpecificEvent.Condition = graphical.DomainSpecificCondition
           ;
28     intEvent.ConnectSpecifiedEvent(domainSpecificEvent);
29     foreach (Model ownModel in ownDomainSimulationModels)
30     {
31      ((DesmoWeatherModel)ownModel).BPMNWeatherConditionEvents.Add(
           domainSpecificEvent);
32     }
33    }
34   }
```

```
35
36    public override void ConnectNonBPMNModel(IEnumerable<Model>
          ownDomainSimulationModels, KeyValuePair<Type, Dictionary<object,
           object>> foreignModel)
37    {}
38  }
39 }
```

F. Benutzeroberfläche zur Windparkerweiterung

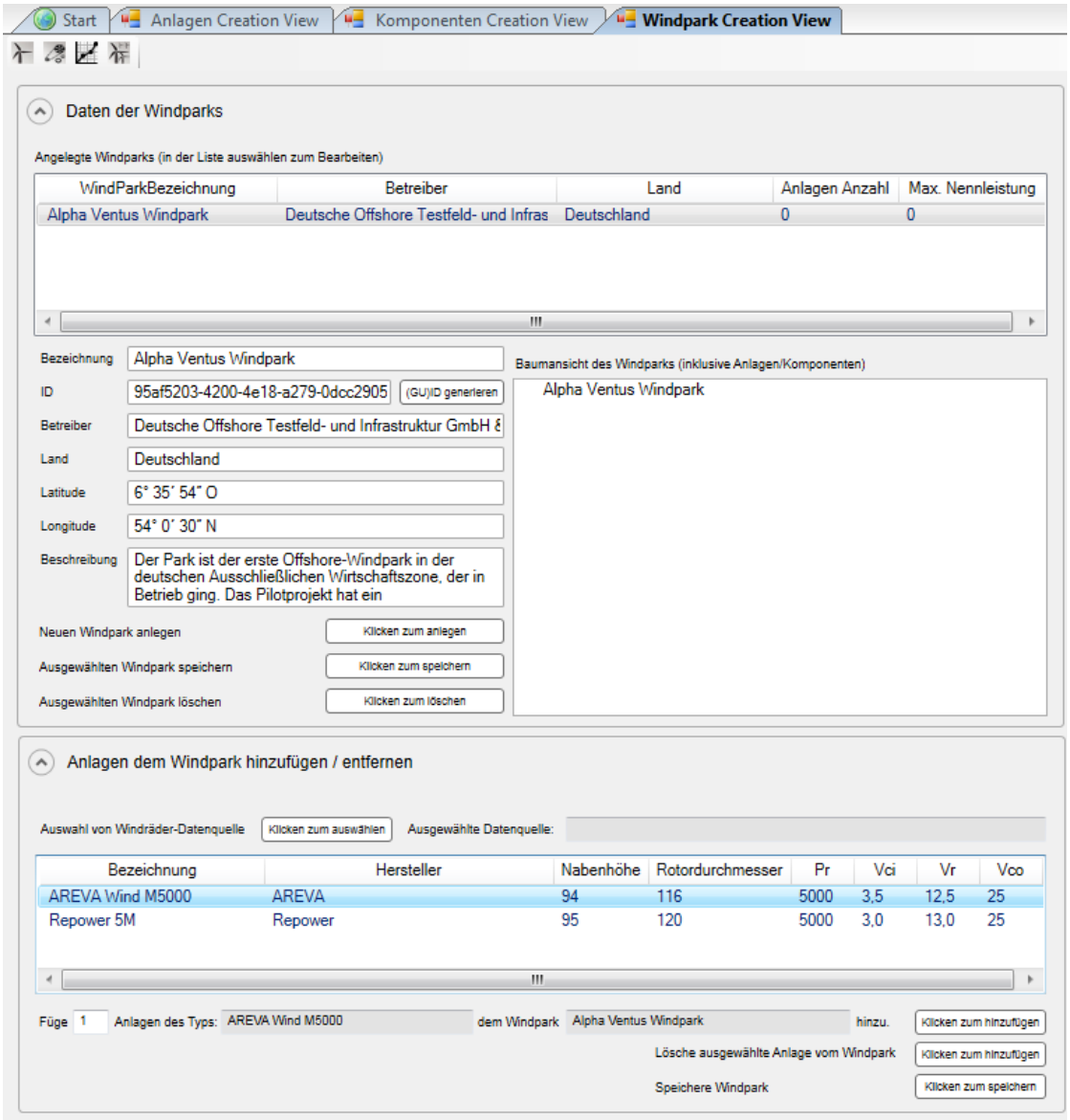


Abbildung F.1.: Benutzeroberfläche zur Definition von Offshore Windparks

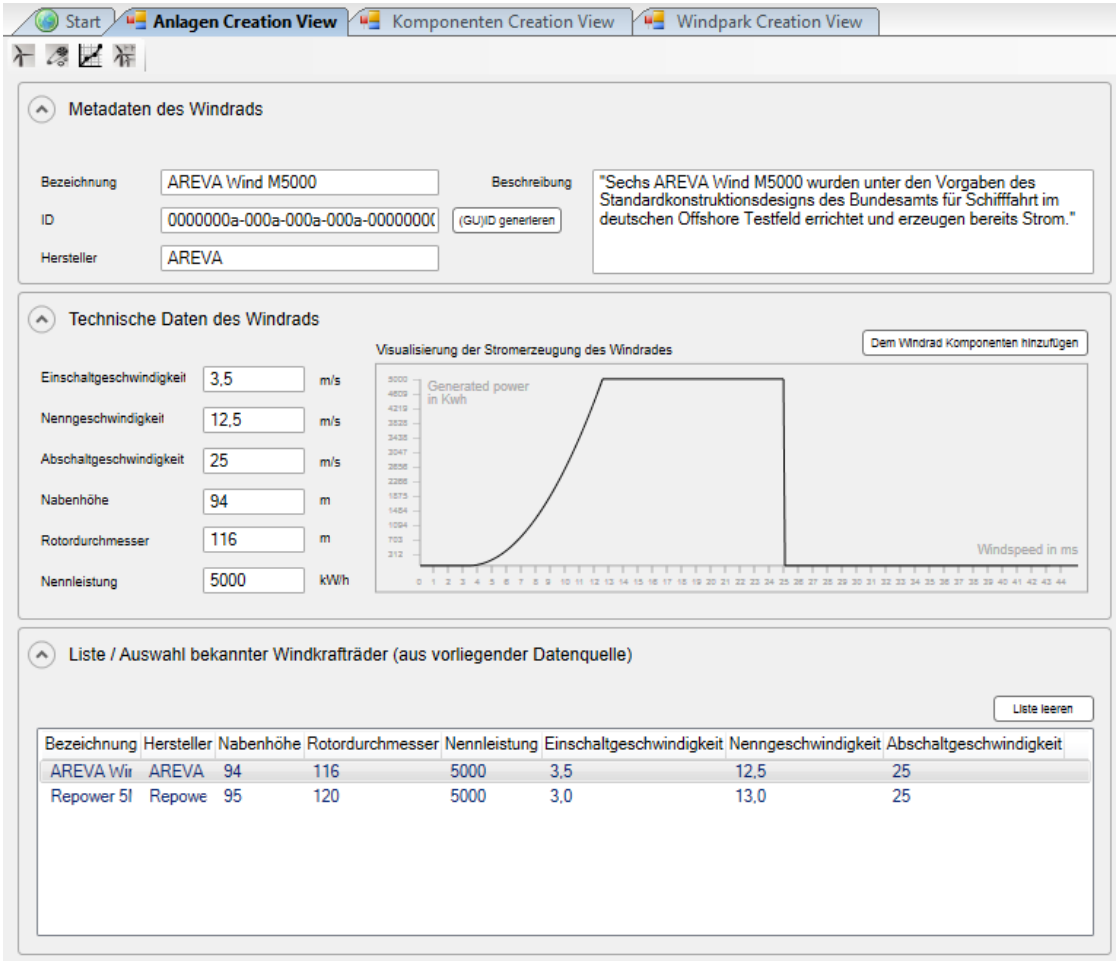


Abbildung F.2.: Benutzeroberläche zur Definition von Windenergieanlagen

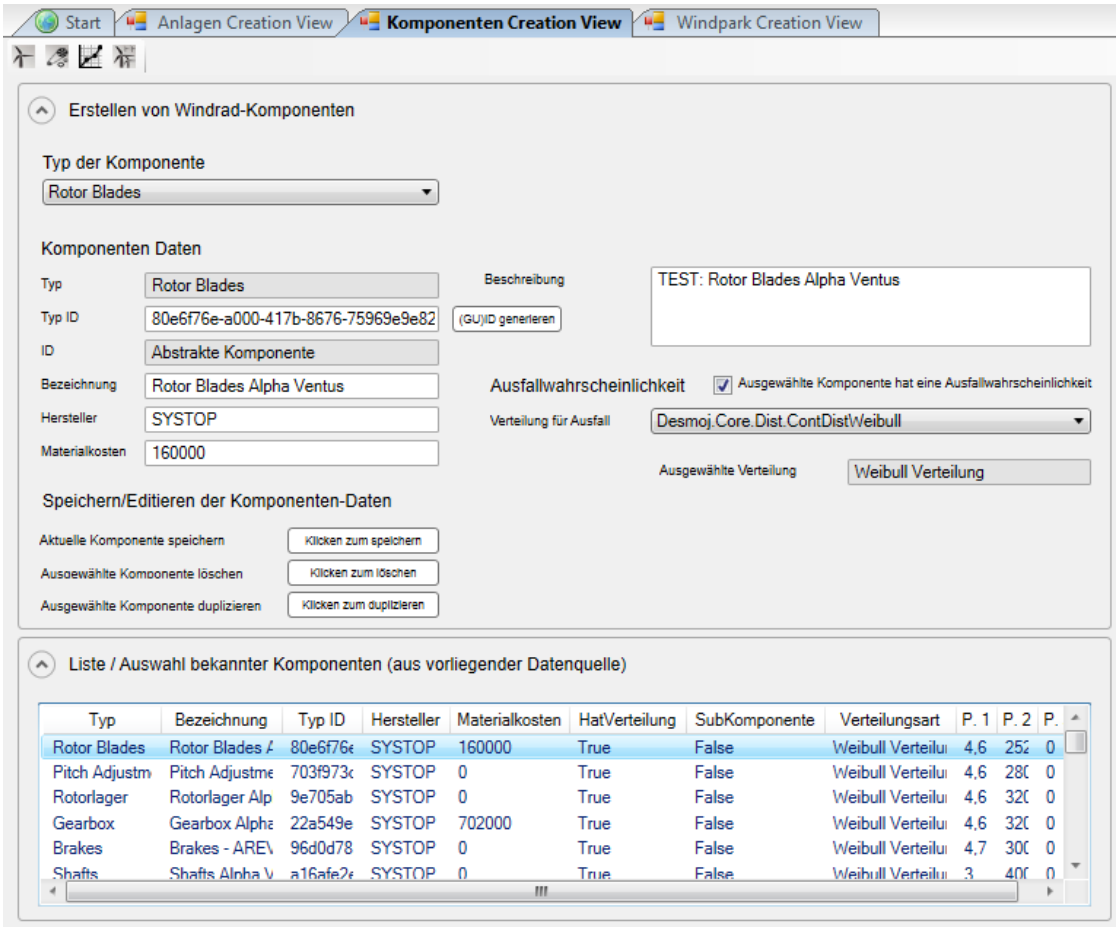


Abbildung F.3.: Benutzeroberfläche zur Definition von Anlagenkomponenten

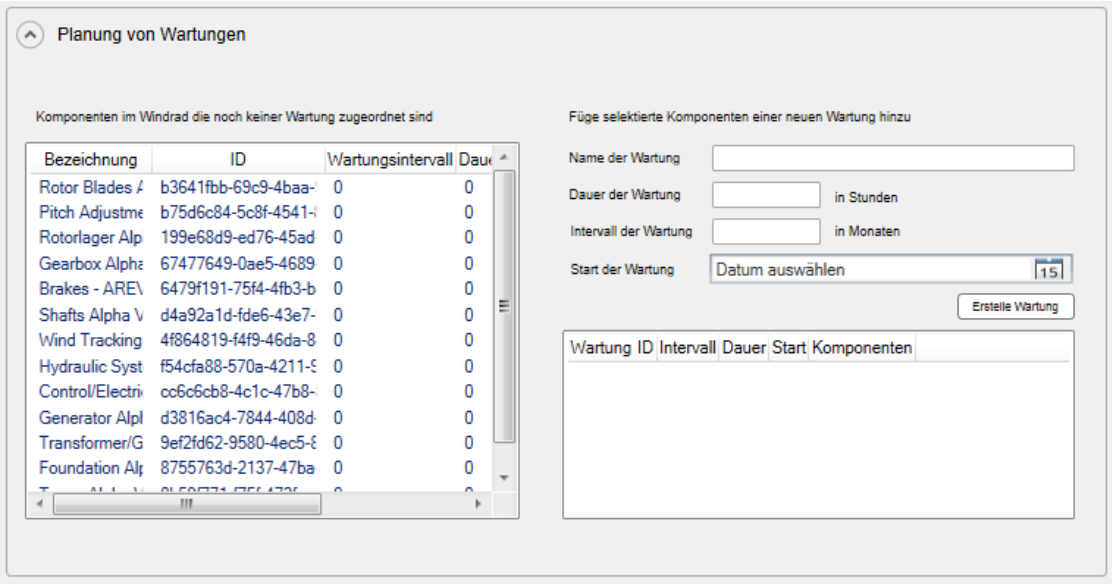


Abbildung F.4.: Benutzeroberfläche zum Anlegen eines Wartungsplans

G. Auszüge aus dem Programmcode zur Windparkerweiterung

Quellcode G.1: Klasse BPMNWindparkArtifact

```
1
2 namespace Simulation.Windpark
3 {
4  class BPMNWindparkArtifact: DomainSpecificArtifact
5  {
6   public BPMNWindparkArtifact(DomainArtifact artifact, string name,
        bool showInTrace): base(artifact, artifact.GetModel(), name,
        showInTrace)
7   {
8    EntityQueue = new Desmoj.Core.Simulator.Queue(this.GetModel(), "
        Windpark Entity Queue", true, true);
9   }
10  }
11 }
```

Quellcode G.2: Klasse BPMNWindparkFailureReceiveEvent

```
1 namespace Simulation.Windpark
2 {
3  class BPMNWindparkFailureReceiveEvent : DomainSpecificStartEvent
4  {
5   public void TriggerEvent(DesmoWindparkComponent who)
6   {
7    var associations = this.GetEvent().GetDataAssociations();
8    foreach (BPMNDataAssociation association in associations)
9    {
10    if (association.GetSuccessor() is DomainArtifact)
11    {
12     ((DomainArtifact)association.GetSuccessor()).EntityQueue.Insert(
          who);
13    }
14   }
15   foreach (SimProcess process in _passivatedProcesses)
16   {
```

```
17      Release(process);
18      }
19      _passivatedProcesses = new List<SimProcess>();
20    }
21  }
22 }
```

Quellcode G.3: Klasse BPMNWindparkStateManipulationActivity

```
1  namespace Simulation.Windpark
2  {
3   public class BPMNWindparkStateManipulationActivity :
        DomainSpecificActivity
4   {
5    public override void StartActivity(BPMNActivity activity,
        BPMNProcess process)
6    {
7    }
8
9    public override void FinishActivity(BPMNActivity activity,
        BPMNProcess process)
10   {
11    if (!process.IsInterrupted())
12    {
13     var associations = this._activity.GetDataAssociations();
14     foreach (BPMNDataAssociation association in associations)
15     {
16      if (association.GetPredecessor() is DomainArtifact)
17      {
18       Entity entity = ((DomainArtifact)association.GetPredecessor()).
            EntityQueue.RemoveFirst();
19       if (entity is DesmoWindparkComponent)
20       {
21        ((DesmoWindparkComponent)entity).ComponentRepaired();
22        break;
23       }
24       if (entity is DesmoWindparkTurbine)
25       {
26        ((DesmoWindparkTurbine) entity).ScheduleNextMaintenance();
27        break;
28       }
29      }
30     }
31    }
32   }
33  }
34 }
```

Quellcode G.4: Klasse WindparkFailureEvent

```
1 namespace Simulation.Windpark
2 {
3  class WindparkFailureEvent : Event<Entity>
4  {
5   public override void EventRoutine(Entity who)
6   {
7    var entity = (DesmoWindparkComponent)who;
8    entity.ComponentBreakDown();
9    foreach (var specEvent in ((DesmoWindparkModel)GetModel()).
         BPMNWindparkFailureReceiveEvents)
10   {
11    specEvent.TriggerEvent((DesmoWindparkComponent)who);
12   }
13  }
14 }
15 }
```

Quellcode G.5: Anmeldung von Visualisierungsklassen für BPMN-Elemente der Windparkerweiterung

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <bundle
3  id="Windpark.UI"
4  provider=""
5  xml:lang="en-US"
6  xmlns="http://www.empinia.org/release/0.3/schemas/Bundle.xsd"
7  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8  <name>Windpark UI</name>
9  <description>Windpark UI Bundle</description>
10
11  <!-- VISUALIZATIONS -->
12  <extension
13   id="empinia.Windpark.visualizations.WindparkVisuals.
        WindparkEvent"
14   point="empinia.ui.graphicalModeling.visualizations"
15   implementation="Windpark.UI.EventVisualizer, Windpark.UI">
16   <name>Windpark Event</name>
17   <description/>
18   <visualization
19    id="empinia.Windpark.visualizations.WindparkEvent"
20    visualizedType="BPMN.Model.tStartEvent, BPMN.DomainModel">
21   </visualization>
22  </extension>
23
24   <extension
25   id="empinia.Windpark.visualizations.WindparkVisuals.
        WindparkActivity"
26   point="empinia.ui.graphicalModeling.visualizations"
27   implementation="Windpark.UI.ActivityVisualizer, Windpark.UI">
28    <name>Windpark Activity</name>
29    <description/>
30   <visualization
31    id="empinia.Windpark.visualizations.WindparkActivity"
32    visualizedType="BPMN.Model.tTask, BPMN.DomainModel">
33   </visualization>
34  </extension>
35
36  <extension
```

```
37    id="empinia.Windpark.visualizations.WindparkVisuals.
          WindparkEntity"
38    point="empinia.ui.graphicalModeling.visualizations"
39    implementation="Windpark.UI.ArtifactVisualizer, Windpark.UI">
40    <name>Windpark Entity</name>
41    <description/>
42    <visualization
43      id="empinia.Windpark.visualizations.WindparkEntity"
44      visualizedType="BPMN.Model.tDataObject, BPMN.DomainModel">
45    </visualization>
46  </extension>
47
48
49  <!-- EDITOR CONFIGURATION -->
50  <extension
51    id="empinia.Windpark.editorConfigExtensions"
52    point="empinia.bpmn.graphicalModeling.
          visualEditorConfigurationExtensions">
53    <name>Additional BPMN Visualizations</name>
54    <description>Additional Visualizations for extended BPMN-
          elements like windparkevents are defined here</description
          >
55
56    <visualization
57      id="empinia.Windpark.visualizations.WindparkVisuals.
            WindparkEvent"
58      visualizedType="BPMN.Model.tEvent, BPMN.DomainModel">
59      <style
60      borderColor="FF000000"
61      backColor="FFDDDDDD"
62      lineStrength="1"
63      lineDashStyle="Solid"
64      foreColor="FF888888"/>
65    </visualization>
66
67  <visualization
68      id="empinia.Windpark.visualizations.WindparkVisuals.
            WindparkActivity"
69      visualizedType="BPMN.Model.tActivity, BPMN.DomainModel">
70      <style
71        borderColor="FF000000"
```

```
72          backColor="FFDDDD00"
73          lineStrength="1"
74          lineDashStyle="Solid"
75          foreColor="FF0101DF" />
76      </visualization>
77
78      <visualization
79      id="empinia.Windpark.visualizations.WindparkVisuals.
            WindparkEntity"
80      visualizedType="BPMN.Model.tDataObject, BPMN.DomainModel">
81       <style
82          borderColor="FF000000"
83          backColor="FFDDDD00"
84          lineStrength="1"
85          lineDashStyle="Solid"
86          foreColor="FF0101DF" />
87      </visualization>
88   </extension>
89 </bundle>
```

Quellcode G.6: Anmeldung der ModelTypeConfiguration zur Windparkerweiterung

```
1
2 <bundle
3   id="Windpark.AnlagenModel"
4   provider=""
5   xml:lang="en-US"
6   xmlns="http://www.empinia.org/release/0.3/schemas/Bundle.xsd"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8   <name translationId="{translate::Windpark AnlagenModel}">
        Windpark AnlagenModel</name>
9   <description translationId="{translate::Windpark AnlagenModel
        Bundle}">Windpark AnlagenModel Bundle</description>
10
11  <extension
12      id="Windpark.AnlagenModel.windparkModelTypeConfiguration"
13      point="simulation.desmoWindparkSolutions.api.
            ModelTypeConfiguration"
14      name="Windpark.AnlagenModel.modelConfiguration"
15      description="Meta information for dealing with windpark
            models">
16    <ModelTypeConfiguration
```

```
17      Postfix="park"
18      LoadCommandID ="windpark.anlagenmodel.commands.
            LoadWindparkModelCommand"
19      ModelType ="Windpark Model"
20      ModelConnectorID="simulation.Windpark.modelConnector"
21      ModelConverterID="simulation.windpark.modelConverter"
22      ModelConfigurerID="simulation.windpark.modelConfigurer"
23      ShowModelCommandID="windpark.anlagenmodel.commands.
            ShowWindparkModelCommand"
24      ImplementingClass="Windpark.DomainModel.WindparkModel"
25    </ModelTypeConfiguration>
26  </extension>
27
28  <!--___ Commands ___-->
29  <extension
30      id="Windpark.AnlagenModel.commands"
31      point="empinia.core.runtime.commands">
32    <name>Windpark.EnergyOutput Commands</name>
33    <description>Commands contributed by the Windpark.
          AnlagenModel bundle</description>
34
35    <command
36      id="windpark.anlagenmodel.commands.LoadWindparkModelCommand"
37      implementation="Windpark.AnlagenModel.Commands.
            LoadWindparkModelCommand, Windpark.AnlagenModel">
38    </command>
39
40    <command
41      id="windpark.anlagenmodel.commands.ShowWindparkModelCommand"
42      implementation="Windpark.AnlagenModel.Commands.
            ShowWindparkModelCommand, Windpark.AnlagenModel">
43    </command>
44  </extension>
45
46 </bundle>
```

Quellcode G.7: Simulationsfabrik zum Windparkmodell

```
1  namespace Simulation.Windpark
2  {
3   class ModelConverter : IModelConverter
4   {
5    readonly Dictionary<object, object> _staticSimulationMapping = new
         Dictionary<object, object>();
6    private static Dictionary<string, DesmoWindparkModel>
         _windparksInModel = new Dictionary<string, DesmoWindparkModel>()
         ;
7    private SimulationOutputService _errorService;
8    private DesmoWindparkModel _desmoModell;
9    private IDWSModel _document;
10   private IReportService _reportService;
11   private Model _superModel;
12
13   public ModelConverter(IReportService reportServ)
14   {
15    _errorService = SimulationOutputService.GetInstance();
16    _reportService = reportServ;
17   }
18
19   public Model CreateSimulationModel(IDWSModel document)
20   {
21    var statischesModell = (WindparkModel)document;
22    _desmoModell = new DesmoWindparkModel(_superModel,
          statischesModell.Name, true, true);
23    _desmoModell.Reset();
24    foreach (var turbinenImStatischenModell in statischesModell.
          WindturbineModel)
25    {
26     for (int i = 0; i < Double.Parse(turbinenImStatischenModell.
           AnzahlAnlagenImPark); i++)
27     {
28      var newTurbine = new DesmoWindparkTurbine(_desmoModell,
            turbinenImStatischenModell, true);
29
30      foreach (var komponentInDerStatischenTurbine in
            turbinenImStatischenModell.WindturbineComponentModel)
31      {
32       var newComponent = new DesmoWindparkComponent(_desmoModell,
             komponentInDerStatischenTurbine, true);
33       var verteilungsName = komponentInDerStatischenTurbine.
             DesmoVerteilungAusfall;
34
35       switch (verteilungsName)
```

```
36          {
37           case "Desmoj.Core.Dist.ContDistNormal":
38            newComponent.FailureProbability = (Distribution)(new
                  ContDistNormal(_desmoModell, newComponent.Name,
                  komponentInDerStatischenTurbine.Parameter1 * 24,
                  komponentInDerStatischenTurbine.Parameter2 * 24, true,
                  true));
39            break;
40
41           case "Desmoj.Core.Dist.ContDistExponential":
42            newComponent.FailureProbability = (Distribution)(new
                  ContDistExponential(_desmoModell, newComponent.Name,
                  komponentInDerStatischenTurbine.Parameter1 * 24, true,
                  true));
43            break;
44
45            [...]
46
47           case "Desmoj.Core.Dist.ContDistWeibull":
48            newComponent.FailureProbability = (Distribution)(new
                  ContDistWeibull(_desmoModell, newComponent.Name,
                  komponentInDerStatischenTurbine.Parameter1 * 24,
                  komponentInDerStatischenTurbine.Parameter2 * 24, true,
                  true));
49            break;
50
51           }
52           newComponent.TurbineParent = newTurbine;
53           newTurbine.WindturbineComponentModel.Add(newComponent);
54          }
55          _desmoModell.Windparkturbinen.Add(newTurbine);
56         }
57        }
58       return _desmoModell;
59      }
60
61      public void initialize(DistributionManager distributionManager,
            Model supermodel)
62      {
63       _superModel = supermodel;
64      }
65
66      public Model GetConvertedModel()
67      {
68       return _desmoModell;
69      }
```

```
70
71    public Dictionary<object, object> GetStaticSimMapping()
72    {
73      return _staticSimulationMapping;
74    }
75
76    public IDWSModel GetOriginalModel()
77    {
78      return _document;
79    }
80
81    public Type GetSimulationModelType()
82    {
83      return typeof(BPMNWindparkModel);
84    }
85  }
86 }
```

Quellcode G.8: Modellkoppler zum Windparkmodell

```
1  namespace Simulation.Windpark
2  {
3   class ModelConnector : AbstractModelConnector
4   {
5    public override Type ProvidedModelType
6    {
7     get { return typeof(DesmoWindparkModel); }
8    }
9
10   public override IEnumerable<Type> RequiredModelTypes
11   {
12    get { return new List<Type>() { typeof(BPMNSimulationModel),
          typeof(DesmoWeatherModel) }; }
13   }
14
15   public override void ConnectDomainSpecificStartEvent(
        DomainSpecificStartEvent startEvent, BPMN.Model.tStartEvent
        graphical, IEnumerable<Model> ownDomainSimulationModels)
16   {
17    if (graphical.DomainSpecificType.Equals("WindparkFailureEvent"))
18    {
19     var domainSpecificEvent = new BPMNWindparkFailureReceiveEvent(
          startEvent, "Windpark Failure Receive Event", true);
20     startEvent.ConnectSpecifiedEvent(domainSpecificEvent);
21     foreach (Model ownModel in ownDomainSimulationModels)
22     {
23      ((DesmoWindparkModel)ownModel).BPMNWindparkFailureReceiveEvents.
          Add(domainSpecificEvent);
24     }
25    }
26
27    if (graphical.DomainSpecificType.Equals("WindparkMaintenanceEvent"
        ))
28    {
29     var domainSpecificEvent = new BPMNWindparkMaintenanceReceiveEvent
          (startEvent, "Windpark Maintenance Receive Event", true);
30     startEvent.ConnectSpecifiedEvent(domainSpecificEvent);
31     foreach (Model ownModel in ownDomainSimulationModels)
32     {
33      ((DesmoWindparkModel) ownModel).
          BPMNWindparkMaintenanceReceiveEvents.Add(domainSpecificEvent)
          ;
34     }
35    }
36   }
```

```
37
38    public override void ConnectDomainSpecificArtifact(DomainArtifact
          artifact, BPMN.Model.tDataObject graphical, IEnumerable<Model>
          ownDomainSimulationModels)
39    {
40      if (graphical.DomainSpecificType.Equals("WindparkEntity"))
41      {
42        var domainSpecificArtifact = new BPMNWindparkArtifact(artifact, "
              Windpark Entity Artifact", true);
43      }
44    }
45
46    public override void ConnectDomainSpecificActivity(BPMNActivity
          activity, BPMN.Model.tActivity graphical, IEnumerable<Model>
          ownDomainSimulationModels)
47    {
48      if (graphical.DomainSpecificType.Equals("WindparkActivity"))
49      {
50        var domainSpecificActivity = new
              BPMNWindparkStateManipulationActivity(activity);
51      }
52    }
53
54    public override void ConnectDomainSpecificIntermediateEvent(Desmoj.
          Extensions.Bpmn.Events.BPMNIntermediateEvent intEvent, BPMN.
          Model.tEvent graphical, IEnumerable<Model>
          ownDomainSimulationModels)
55    {
56    }
57
58    public override void ConnectNonBPMNModel(IEnumerable<Model>
          ownDomainSimulationModels, KeyValuePair<Type, Dictionary<object,
           object>> foreignModel)
59    {
60      if(foreignModel.Key == typeof(DesmoWeatherModel)) {
61        foreach(Model model in ownDomainSimulationModels) {
62          foreach (object key in foreignModel.Value.Keys)
63          {
64            ((DesmoWeatherModel)foreignModel.Value[key]).WindPerformer.Add
                  ((DesmoWindparkModel)model);
65          }
66        }
67      }
68    }
69  }
70 }
```

Literaturverzeichnis

Allweyer, T. (2009). *BPMN 2.0 - Business Process Modeling Notation: Einführung in den Standard für die Geschäftsprozessmodellierung.* Books on Demand, Norderstedt, 2nd edition.

Ardhaldjian, R. und Fahner, M. (1994). Using Simulation in the Business Process Reengineering Effort. *Industrial Engineering-Norcross*, 26(7):60–61.

Bearing Point (2012). Business Process Management-Studie 2012. Online: http://toolbox.bearingpoint.de/images/pdf/NN_12021_0731_WP_DE_MS_BPO_final_web.pdf, letzter Zugriff am 22. März 2016.

Becker, J., Kugeler, M., und Rosemann, M. (2011). *Process management: A guide for the design of business processes.* Springer, Berlin, 2nd edition.

Blokdijk, G. und Menken, I. (2008). *The service level agreement SLA guide: SLA book, templates for service level management and service level agreement forms : fast and easy way to write your SLA.* Lulu Press Inc.

Blum, K. (2010). *Open and Extensible Business Process Simulator.* Master thesis, University of Tartu.

Bode, S., Lehnert, S., und Riebisch, M. (2011). Comprehensive model integration for dependency identification with EMFTrace. In *Joint Proc. of the First Int. Workshop on Model-Driven Software Migration and the Fifth Int. Workshop on Software Quality and Maintainability*, pages 17–20.

Booch, G., Rumbaugh, R., und Jacobsen, I. (1998). *The Unified Modelling Language Users Guide.* Addison-Wesley, New York.

Bornhöft, N. A. (2009). *Analyse eines neuartigen Horizontalkrankonzepts für Containerterminals mithilfe der Simulationstechnik.* Diplomarbeit, Universität Hamburg.

Bossel, H. (1992). *Modellbildung und Simulation.* Vieweg Verlag, Braunschweig.

Bradley, P., Browne, J., Jackson, S., und Jagdev, H. (1995). Business process reengineering (BPR) - A stuy of the software tools currently available. *Computers in Industry*, 25(3):309–330.

BSH (2014). FINO Wetterdatenbank. Online: http://www.bsh.de/de/Meeresdaten/Projekte/FINO/index.jsp, letzter Zugriff am 16. März 2016.

Bundesregierung (2002). Strategie der Bundesregierung zur Windenergienutzung auf See. Online: http://www.erneuerbare-energien.de/fileadmin/ee-import/files/pdfs/allgemein/application/pdf/windenergie_strategie_br_020100.pdf, zuletzt abgerufen am 22. März 2014.

Byon, E., Perez, E., Ding, Y., und Ntaimo, L. (2011). Simulation of wind farm operations and maintenance using discrete event system specification. *Simulation*, 87(12):1093–1117.

CRA-CIN (2014). CLIMA Tools for Agrometeorology and Agricultural Modelling. Online: http://www.sipeaa.it/ASP/ASP2/Clima.asp, letzter Zugriff am 16. März 2016.

Cuntz, N. (2004). *Über die effiziente Simulation von Ereignisgesteuerten Prozessketten.* Master thesis, Univsersität Paderborn, Münser.

Czogalla, R., Knaak, N., und Page, B. (2006). Simulating the Eclipse way-A generic experimentation environment based on the eclipse platform. *Proceedings of the 20th European Conference on Modelling and Simulation.*

Decker, G., Tscheschner, W., und Puchan, J. (2009). Migration von EPK zu BPMN. In *GI-Workshop Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK).*

DESMO-J (2014). DESMO-J Download-Site. Online: http://www.desmo-j.de, letzter Zugriff am 22. März 2016.

Deutscher Bundestag (2013). Antwort der Bundesregierung auf die Kleine Anfrage der Abeordneten Uwe Beckmeyer, Sören Bartol, Martin Burkert, weiterer Abgeordneter und der Fratkion der SPD - Drucksache 17/13289.

Dijkman, R. und Gorp, P. V. (2011). BPMN 2.0 execution semantics formalized as graph rewrite rules. *Business Process Modeling Notation*, pages 16–30.

EABPM (2009). *Business Process Management Common Body of Knowledge - BPM CBOK.* Verlag Dr. Götz Schmidt, Gießen.

Empinia (2014). Empinia Homepage. Online: http://www.empinia.org, letzter Zugriff am 22. März 2016.

Enstone, L. und Clark, M. (2006). BPMN and Simulation. Online: http://cfile7.ut.tistory.com, zuletzt abgerufen am 22. März 2014.

Falk, T. (2007). Perspektiven für die Windenergie 2020. Online: http://www.economics.phil.uni-erlangen.de/forschung/energie/ko_e_strw07/Falk.pdf, zuletzt zugegriffen am 22. März 2014.

Feijóo, A., Villanueva, D., Pazos, J. L., und Sobolewski, R. (2011). Simulation of correlated wind speeds: A review. *Renewable and Sustainable Energy Reviews*, 15(6):2826–2832.

Floyd, C. und Züllighoven, H. (2002). *Softwaretechnik*. Hanser Verlag, München, Wien.

Frank, U. und Laak, B. V. (2003). Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen. *Arbeitsberichte des Instituts für Wirtschaftsinformatik Universität Koblenz Landau*.

Freund, J. und Rücker, B. (2012). *Praxishandbuch BPMN 2.0*. Hanser Verlag, München.

Friedenstab, J.-P., Janiesch, C., Matzner, M., und Muller, O. (2012). Extending BPMN for Business Activity Monitoring. *2012 45th Hawaii International Conference on System Sciences*, pages 4158–4167.

Gamma, E. (2013). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, Boston (USA), 41 edition.

Gladwin, B. und Tumay, K. (1994). Modeling business processes with simulation tools. *Proceedings of the 26th conference on Winter simulation*, pages 114–121.

Göbel, J., Joschko, P., Koors, A., und Page, B. (2012). The Discrete Event Simulation Framework DESMO-J: Review, Comparison to other Frameworks and Latest Development. *Proceedings 27 European Conference in Modelling and Simulaition*.

Goebel, J. (2013). *Self-Organizing Transport Networks – Decentralized Optimization based on Genetic Programming*. PhD thesis, Universität Hamburg.

Greiner, S. (2013). Risikoorientierte Prozessmodelle in BPMN – Stand des Wissens und Potenziale. In Marx Gómez, J., Lang, C., und Wohlgemuth, V., editors, *IT-gestütztes Ressourcen- und Energiemanagement*, pages 209–218. Springer Berlin Heidelberg, Berlin, Heidelberg.

Greiner, S., Albers, H., Albers, H., und Renz, T. (2014). Low-risk Processes, Customized for Operation of Offshore Wind Farms. *DEWI-Magazin*, 44:60–64.

Griffel, F. (1998). *Componentware - Konzepte und Techniken eines Softwareparadigmas*. dpunkt.verlag, Heidelberg.

Grimm, C. und Schroll, R. (2007). Mixed-level-simulation heterogener systeme. *Proceedins of the ITG/GI/GMM-Workshop: Multi-Nature Systems.*

Guizzardi, G. und Wagner, G. (2011). Can BPMN Be Used for Making Simulation Models? *Enterprise and Organizational Modeling and Simulation*, pages 100–115.

Haan, J. (2012). *Konzeption und Implementation einer XML-Schnittstelle zum Importieren und Exportieren von BPMN 2.0-Modellen für .NET.* Projektbericht, Universität Hamburg.

Haan, J. (2014). *Untersuchung der Eignung der BPMN 2.0 als Modellierungssprache zur Logistiksimulation.* Master thesis in progress, Universität Hamburg.

Häuslein, A. (1993). *Wissensbasierte Unterstützung der Modellbildung und Simulation im Umweltbereich.* Verlag Peter Lang, Frankfurt am Main.

Hedstück, U. (2013). *Simulation diskreter Prozesse.* Springer Verlag, Berlin Heidelberg.

Hiatt, J. (1999). *Benchmarking Study - Best Practices in Business Process Reengineering and Process Design.* ProSci.

Hightower, R. (2008). *Accounting and finance policies and procedures.* J. Wiley and Sons, Hoboken (USA).

Hlupic, V. und Robinson, S. (1998). Business process modelling and analysis using discrete-event simulation. In *Proceedings of the 30th conference on Winter simulation*, pages 1363–1370. IEEE Computer Society Press.

Hobohm, J., Krampe, L., Peter, F., Gerken, A., Heinrich, P., und Richter, M. (2013). Kostensenkungspotentiale der Offshore-Windenergie in Deutschland. Online: http://www.offshore-stiftung.com/60005/Uploaded/SOW_Download|Langfassung_Studie_Kostensenkungspotenziale_Offshore-Windenergie.pdf, zuletzt zugegriffen am 22. März 2014.

Horton, G. (2003). Simulation: Das virtuelle Labor. *Magdeburger Wissenschaftsjournal*, 8:45–52.

IYOPRO (2014). IYOPRO Webanwendung. Online: http://www.iyopro.de, letzter Zugriff am 22. März 2016.

Jahr, P., Schiemann, L., und Wohlgemuth, V. (2009). Development of simulation components for material flow simulation of production systems based on the plugin architecture framework EMPINIA. In *Proceedings of the EnviroInfo 2009 (Berlin) - Environmental Informatics and Industrial Environmental Protection: Concepts.*

Jander, K., Braubach, L., Pokahr, A., und Lamersdorf, W. (2011). Validation of Agile Workflows using Simulation. *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, pages 39–55.

Jansen-Vullers, M. und Netjes, M. (2006). Business process simulation–a tool survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark*.

Joschko, P. (2008). *Entwurf einer Editierungskomponente zur Modellierung spezialisierter Graphen*. Diplomarbeit, Universität Hamburg.

Joschko, P., Brandt, C., und Page, B. (2009). Combining Logistic Container Terminal Simulation and Device Emulation using an Open-Source Java Framework. In Bruzzone, Cunha, Martínez, und Merkuryev, editors, *Proceedings of the International Conference on Harbor, Maritime & Multimodal Logistic Modelling and Simulation*, La Laguna.

Joschko, P., Haan, J., Janz, T., und Page, B. (2012). Business Process Simulation with DESMO-J and IYOPRO. In *Proceedings of the International Workshop on Applied Modeling and Simulation*, Rome.

Joschko, P., Widok, A. H., und Page, B. (2013a). A Simulation Tool for Maintenance Processes of Offshore Wind Farms. In *Proceedings of the 15th International Conference on Harbor, Maritime & Multimodal Logistic Modelling and Simulation*, volume 2009.

Joschko, P., Widok, A. H., Page, B., Appel, S., Greiner, S., und Albers, H. (2013b). Modeling and Simulation of Offshore Wind Farms including the Mapping and Analysis of relevant O & M Processes. *Proceedings of 27th Conference Environmental Informatics*, pages 1–12.

Kalnins, A., Kalnina, D., und Kalis, A. (1998). Comparison of tools and languages for business process reengineering. *Proceedings of the Third International Baltic Workshop on Databases and Information Systems*, pages 24–38.

Keller, G., Nüttgens, M., und Scheer, A.-W. (1992). Semantische Prozeßmodellierung auf der Basis Ereignisgesteuerter Prozeßketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik, Universität Saarland*.

Klinke, J. und Klarmann, M. (2012). SystOp Offshore Wind : German Wind Power Plant Model. Online: https://www.btc-ag.com/de/documents/SystOp_BTC_GermanWindPowerPlantModel.pdf, zuletzt abgerufen am 22. März 2014.

Klückmann, F. (2009). *Realzeitsynchrone Simulation – Begriffe , Anwendungen und exemplarische Umsetzung anhand des Simulationsframework DESMO-J.* Diplomarbeit, Universität Hamburg.

Knaak, N., Kruse, S., und Page, B. (2006). An agent-based simulation tool for modelling sustainable logistics systems. In *Proceedings of the iEMSs Third Biennial Meeting: Summit on Environmental Modelling and Software. International Environmental Modelling and Software Society*, Burlington (USA).

Knaak, N. und Page, B. (2006). Applications and extensions of the unified modelling language UML 2 for discrete event simulation. *International Journal of Simulation*, 7(6):33–43.

Komus, A., editor (2011). *BPM best practice: Wie führende Unternehmen ihre Geschäftsprozesse managen.* Springer, Berlin, Heidelberg.

Kunert, D.-i. A. (2010). *Prozessorientierte optimistisch-parallele Simulation.* PhD thesis, Humboldt-Universität Berlin.

Langner, P., Schneider, C., und Wehler, J. (1998). Petri net based certification of event-driven process chains. *Application and Theory of Petri Nets*, pages 286–305.

Law, A. M., Kelton, W. D., und Kelton, W. D. (2000). *Simulation modeling and analysis.* McGraw-Hill series in industrial engineering and management science. McGraw-Hill Higher Education, Boston Mass. u.a, 3rd edition.

Licher, V., Strackenbrock, B., Weiß, J., Claus, V., und Schwill, A. (2001). *Informatik Duden.* Duden Verlag, Mannheim, Leipzig, Wien, Zürich.

Liebig, S., Hedulser, S., Stüwing, M., und Dronka, S. (2001). Die Modellierung und Simulation gekoppelter mechanischer und hydraulischer Systeme. In *ASIM Workshop.*

List, B. und Korherr, B., editors (2006). *An evaluation of conceptual business process modelling languages*, SAC '06, New York (USA). ACM.

Loos, P. und Allweyer, T. (1998). Process orientation and object-orientation-an Approach for Integrating UML and Event-Driven Process Chains. *Publications of the Institute für Wirtschaftsinformatik, Universität Saarland*, 144(March):1–17.

McIlroy, M. (1968). Mass produced software components. *Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*, pages 138–156.

Mengel, C. E. (2013). *Simulation von Betriebs- und Instandhaltungsprozessen von Offshore-Windkraftanlagen mit der Modellierungssprache BPMN 2.0.* Bachelorarbeit, Universität Hamburg.

Mitchel, G., Ray, H., Griggs, V. B., und Williams, J. (2000). *EPIC Documentation*. Texas AatM Blackland Research and Extension Center.

Müller, C. (2012). Generierung von Simulationsmodellen aus ereignisgesteuerten Prozessketten. In Barton, T., Herrmann, F., Alm, W., Arnold, C., Augustin, Y., und Bentz, Y., editors, *Management und IT*, page 55ff.

Nickel, M. (2013). Entwicklungen in der deutschen Stromwirtschaft 2013 Entwicklung des Gesamt-Stromverbrauchs. Technical report, BDEW Bundesverband der Energie- und Wasserwirtschaft e.V.

Nicolae, O., Diaconescu, M., und Wagner, G. (2009). Simulation Modelling for Businesses using BPMN and AORS. *Annals of the University of Craiova-Mathematics and Computer Science Series*, 36(2):69–78.

Oeser, A. (2013). *Konzeption und Implementation eines grafischen Editors zur Manipulation von BPMN-Modellen im Projekt SystOp Offshore Wind*. Bachelor thesis, Universität Hamburg.

OMG (2011). Business Process Model and Notation (BPMN) Version 2.0. Online: http://www.omg.org/spec/BPMN/, zuletzt abgerufen am 22. März 2014.

Onggo, B. und Karpat, O. (2011). Agent-based conceptual model representation using BPMN. In *Proceedings of the 2011 Winter Simulation Conference*, number 2001, pages 671–682.

Page, B. (1991). *Diskrete Simulation*. Springer Verlag, Berlin, Heidelberg, New York.

Page, B. und Kreutzer, W. (2005). *The Java Simulation Handbook - Simulating Discrete Event Systems with UML and Java*. Shaker Verlag, Aachen.

Page, B., Lechler, T., und Claassen, S. (2000). *Objektorientierte Simulation in Java*. Libri Books on Demand, Norderstedt.

Panic, D., Wohlgemuth, V., und Schnackenbeck, T. (2008). *Erweiterung eines Open-Source-Rahmenwerkes um Simulationsfunktionalität für betriebliche Umweltinformationssysteme (BUIS)*. Shaker Verlag, Aachen.

Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg.

Pingel, U. und Schmidt, S. (2007). Geschäftspozessmanagement Umfrage 2007. Technical report, Competence Center Virtual Organisation and Engineering.

Racsko, P.; Semenow, M. (1989). Analysis of Mathematical Principles in Crop Growth Simulation Models. *Simulation*, 47:291–302.

Res, C., Semenov, M. A., Brooks, R. J., Barrow, E. M., und Richardson, C. W. (1998). Comparison of the WGEN and LARS-WG stochastic weather generators for diverse climates. *Climate Research*, 10:95–107.

Riebisch, M., Bode, S., Farooq, Q.-U.-A., und Lehnert, S. (2011). Towards Comprehensive Modelling by Inter-model Links Using an Integrating Repository. *2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, (2):284–291.

Robinson, S. (1994). *Successful simulation : a practical approach to simulation projects*. McGraw-Hill Higher Education, London.

Rohleder, T. R. und Silver, E. A. (1997). A tutorial on business process improvement. *Journal of Operations Management*, 15:139–154.

Rücker, B. (2008a). Business Process Simulation in Action. *Java-Magazin*, 6:94–99.

Rücker, B. (2008b). Business Process Simulation-selbst gemacht. *Java-Magazin*, 6:20–25.

Rücker, B. (2008c). *Business Process Simulation tool with JBoss jBPM*. Master thesis, Stuttgart University of Applied Science.

Sandu, T. (2007). *Modellgetriebene Entwicklung von Simulationsprogrammen am Beispiel des DESMO-J-Frameworks*. Diplomarbeit, Universität Hamburg.

Schäfer, H. (2012). *Kooperation in Geschäftsprozessmodellierungsansätzen*. Diplomarbeit, Universität Hamburg.

Schmauder, R. und Schmidt, P.-M. (1998). Regelsteuerungen in der ARIS-Simulation. In *Simulationstechnik: 12. Symposim, in Zürich*. vdf Hochschulverlag AG.

Schnackenbeck, T., Panic, D., und Wohlgemuth, V. (2007). Eine offene Anwendungsarchitektur als Fundament eines Methodenbaukastens für betriebliche Umweltinformationssysteme. In *Smulation in den Umwelt- und Geowissenschaften*, pages 49–59. Shaker, Aachen.

Schwarz, S., Abecker, A., Muas, H., und Sintek, M. (2001). Anforderungen an die Workflow-Unterstützung für wissensintensive Geschäftsprozesse. In *Deutsches Forschungsinstitut für künstliche Intelligenz*, volume 01-02. Kaiserslautern.

Seidlmeier, H. (2010). *Prozessmodellierung mit ARIS: Eine beispielorientierte Einführung für Studium und Praxis*. Studium IT-Management und Anwendung. Vieweg + Teubner, Wiesbaden, 3rd edition.

Smith, M., Erwin, J., und Diaferio, S. (2007). Role & responsibility charting (RACI). Online: http://www.pmforum.org/library/tips/pdf_files/RACI_R_Web3_1.pdf, zuletzt abgerufen am 22. März 2014.

Stehle, T. (2013). Experimentplanung und -durchführung mit BPMN-Modellen im Windpark-Simulationswerkzeug DESMO-Windpark-Studio. In Page, B., editor, *Proceedings of the 27th Conference on Environmental Informatics*, pages 902–909. Shaker, Aachen.

Stehle, T. (2014). *Unterstützung von Business Process Improvement-Vorhaben als Erweiterung einer BPMN-Simulatonssoftware.* Master thesis, Universität Hamburg.

Sun, F. (2010). *Raumkonzept und 3D-Visualisierung für die ereignis-diskrete Simulationsengine.* Diplomarbeit, University of Hamburg.

SystOp (2014). SystOp Projektziele. Online: http://www.systop-wind.de/index.php?id=54, letzter Zugriff am 22. März 2016.

Szyperski, C. (1997). *Component Software, Beyound Object Oriented Programming.* Addison-Wesley, New York.

Tumay, K. (1995). Business Process Simulation. In *Proceedings of the 1995 Winter Simulation Conference ed. C. Alexopoulos,*, pages 55–60.

Tumay, K. (1996). Business Process Simulation. In Charnes, J. M., Morrice, D. J., Brunner, D., und Swain, J., editors, *Proceedings of the 1996 Winter Simulation Conference*, pages 93–98.

Vacek, J., Wohlegmuth, V., Joschko, P., Mäusbacer, M., und Page, B. (2007). Einsatz eines Werkzeuges zur Stoffstromsimulation bei einem Halbleiterhersteller. In *Simulation in Umwelt- und Geowissenschaften*, pages 23–36.

van der Aalst, W. M. P. (2000). *Workflow patterns*, volume 47 of *BETA publicaties reprints.* BETA Research Institute, Eindhoven.

Weber, T. (2013). Gezeitenwende. *Erneuerbare Energien*, 06:38–45.

Widok, A. (2009). Social Sustainability: Theories, Concepts, Practicability. In *Proceedings of the EnviroInfo 2009 (Berlin) - Environmental Informatics and Industrial Environmental Protection: Concepts*, pages 43–51.

Wohlgemuth, V. (2005). *Komponentenbasierte Unterstützung von Methoden der Modellbildung und Simulation im Einsatzkontext des betrieblichen Umweltschutzes.* PhD thesis, Universität Hamburg.

Wohlgemuth, V., Page, B., und Kreutzer, W. (2006). Combining Discrete Event Simulation and Material Flow Analysis based on a Component-Oriented Approach to Industrial Environmental Protection.

Youngblood, M. D. (1994). *Eating the chocolate elephant: Take charge of change through total process management*. Micrografax, Richardson and Tex.

Ziesing, H.-J. (2013). Energieverbrauch in Deutschland im Jahr 2012. Technical report, Arbeitsgemeinschaft Energiebilanzen e.V.

Zwintzscher, O. (2005). *Software-Komponenten im Überblick*. W3L-Verlag, Bochum.

Publikationsverzeichnis

Liste der wissenschaftlichen Veröffentlichungen des Autors gemäß §7, Absatz 5 der Promotionsordnung

Publikationen zum Thema

2013

Joschko, P., Widok, A., Appel, S., Greiner, S., Albers, H., Page, B. (2013). *„Analysis of O&M processes of Offshore Wind Farms“* (Poster). EWEA Offshore 2013, 19th-21st November, Frankfurt, Germany.

Joschko, P., Widok, A., Page, B. (2013). *„A Simulation Tool for Maintenance Processes of Offshore Wind Farms“*, in: Proceedings of the 15th International Conference on Harbor, Maritime & Multimodal Logistic Modelling and Simulation, organized within the 10th International Mediterranean and Latin American Modeling Multiconference, 25th-27th Septembre, Athens, Greece.

Joschko, P., Widok, A., Page, B., Appel, S., Greiner, S., Albers, H. (2013). *„Modelling and Simulation of Offshore Windfarms including the Analysis of relevant O&M Processes“.* Proceedings of 27th International Conference Environmental Informatics, 2nd-4th Septembre, Hamburg. Shaker Verlag, Aachen.

2012

Joschko, P., Haan, J., Janz, T., Page, B. (2012). *"Business Process Simulation with IYOPRO und DESMO-J"*. In Bruzzone, Buck, Cayirci, Longo (Eds.) "Proceedings of the International Workshop on Applied Modeling & Simulation", 24th-27th September, Rome (Italy).

Vorträge zum Thema

2. BPM Symposium für den Mittelstand. 28.11.2012, Iserlohn: *„Prozesskostenrechnung mit BPMN 2.0 und IYOPRO."*

EnviroInfo 2013 Conference. 02.09.2013, Hamburg: *„Modelling and Simulation of Offshore Windfarms"* (Keynote)

BPM für den Mittelstand - Geschäftsprozessmanagement in der Cloud. 22.11.2012, Iserlohn: *„Modellbildung und Simulation mit BPMN 2.0 und IYOPRO."*

Process Solution Days. 16.5.2012, Frankfurt: *„Modellbildung und Simulation mit BPMN 2.0 und IYOPRO"*

Berliner BUIS Tage. 26.-27.4.2012, Berlin: *„Simulation der Verfügbarkeit von Offshore-Windparks als Planungsgrundlage für Instandhaltungsstrategien"*

ASIM Workshop Simulation in Umwelt- und Geowissenschaften. 29.-30.3.2012, Hamburg: *„Simulation der betrieblichen Prozesse im Leistungssystem Offshore Windpark"*

Sonstige Publikationen

2013

Göbel, J., Joschko, P., Koors, A., Page, B. (2013). *„The Discrete Event Simulaiton Framework DESMO-J: Review, Comparison to other Frameworks and Latest Development"*. Proceedings of the 27th European Conference on Modelling and Simulation, 27-30 May, Ålesund, Norway.

2012

Joschko, P.; Page, B.; Brandt, C. (2012). *„Reusing model components in logistics simulation and device emulation in container terminal operation with open-source framework DESMO-J"*. In Bruzzone, Rodriguez (Eds.): International Journal of Simulation and Process Modelling, 2012 Vol. 7 No.3 (Special Issue on Modelling and Applied Simulation: Advanced Methodologies and Techniques), Inderscience Publishers.

2011

Joschko, P., Schäfer, H., Nicolas Denz, Schmitz, C., Page, B. (2011). *„Emission Trade Assistant: Prototype Implemention and Lessons learned“*. In Pillmann, Schade, Smits (Eds.): Proceedings of 25th International Conference Environmental Informatics, 5th-8th October, Ispra (Italy). Shaker Verlag, Aachen.

2010

Joschko, P., Schmitz, C., Page, B., Nicolas Denz (2010). *„A Workflow-based Compliance Assistant for Facilites in EU Emission Trading System“*. International Congress on Environmental Modelling and Software, July 5-8, Ottawa (Canada), 2010. Überarbeitete Version in deutscher Sprache: *„Implementation eines Workflow-basierten IT-Assistenten für Anlagenbetreiber im Europäischen Emissionshandel“*. 3. Berliner BUIS-Tage, 6-8 October, Köln/Bonn.

Page, B., Joschko, P., Schmitz, C. (2010). *„Design concepts for an IT-Assistant System for CO2 Emission Trading“*, 5th Annual International Symposium in Environment, 20-23 May, Athens (Greece).

Joschko, P., Kruse, S., Page, B. (2010). *„Modellierung und Simulation des EU-Emissionshandelsmarktes mit einer agentenbasierten Erweiterung zu DESMO-J“*, in Wittmann (Eds): Proceedings des Workshops Simulation in Umwelt- und Geowissenschaften, March 18-19, Shaker Verlag, Aachen.

2009

Joschko, P., Page, B., Wohlgemuth, V. (2009). *„Combination of Job Oriented Simulation with Ecological Material Flow Analysis as Integrated Analysis Tool for Business Production Processes“*, in Rossetti, Hill, Johansson, Dunkin, Ingals (Eds.): Proceedings of the 2009 Winter Simulation Conference, Dec 13-16, Austin, Texas.

Joschko, P., Brandt, C., Page, B. (2009). *„Combining Logistic Container Terminal Simulation and Device Emulation using an Open-Source Java Framework“*, in Bruzzone, Cunha, Martínez, Merkuryev (Eds.): Proceedings of the International Conference on Harbor, Maritime & Multimodal Logistic Modelling and Simulation, organized within the 6th International Mediterranean and Latin American Modeling Multiconference, Sep 23-25, Universidad de La Laguna, La Laguna 2009, pp. 106-115.

Page, B., Schmitz, C., Joschko, P., Nicolas Denz (2009). *„Design of an IT-Assistant System for CO2 Emission Trading“*, in Wohlgemuth, Page, Voigt (Eds.): Proceedings

of the 23rd International Conference Environmental Informatics - Informatics for environmental protection, sustainable development and risk management Volume 2, Sep 10-12, Shaker Verlag, Aachen, pp. 25-32.

2008

Joschko, P. (2008). *„Entwurf einer Editierungskomponente zur Modellierung spezialisierter Graphen“*, Diplomarbeit an der Universität Hamburg.

2007

Vacek, J., Wohlgemuth, V., Joschko, P., Mäusbacher, M., Page, B. (2007). *„Einsatz eines Werkzeuges zur Stoffstromsimulation bei einem Halbleiterhersteller“*, in Wittmann, Wohlgemuth (Eds.): Proceedings des Workshops Simulation in Umwelt- und Geowissenschaften, March 22-23, Shaker Verlag, Aachen, pp. 23-36.

Eidesstattliche Erklärung gemäß §7, Absatz 4 der Promotionsordnung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 24. März 2014

Philip Joschko