UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik
Betriebliche Informationssysteme

Berechnung und Anwendung von Modelldifferenzen im Geschäftsprozessmanagement

Bachelorarbeit

Leipzig, 9. September 2008

vorgelegt von
Stanley Hillner
geb. am 05. März 1984
Studiengang Informatik

Betreuender Hochschullehrer: Univ.-Prof. Klaus-Peter Fähnrich
Betreuender Assistent: Dipl.-Inf. Heiko Kern

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

ARIS	Architektur integrierter Informationssysteme
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPMN	Business Process Modeling Notation
CASE	Computer Aided Software Engineering
DSML	Domain Specific Modeling Language
eEPK	erweiterte Ereignisgesteuerte Prozesskette
EMF	Eclipse Modeling Framework
EMFT	Eclipse Modeling Framework Technology
EPK	Ereignisgesteuerte Prozesskette
GUI	Graphical User Interface
IDE	Integrated Development Environment
M2C	Modell-zu-Code (Transformation)
M2M	Modell-zu-Modell (Transformation)
M2P	Modell-zu-Plattform (Transformation)
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
OASIS	Organization for the Advancement of Structured Information Standards
ODF	Open Document Format
OMG	Object Management Group
OWL	Web Ontology Language
QVT	Query View Transformations
SOA	Service-oriented Architecture
UML	Unified Modeling Language
UUID	Universally Unique Identifier
VWL	Volkswirtschaftslehre
W3C	Worl Wide Web Consortium
WSBPEL	Web Service Business Process Execution Language
WSDL	Web Service Definition Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformation

1 Einleitung

1.1 Anwendungsfall: EPK-zu-BPEL-Transformation

Die Softwareentwicklung ist seit den Anfängen der Informatik stetig effizienter geworden. Beispielsweise wird heute neue Software fast schon vollautomatisch entwickelt. Neben der stark geförderten Wiederverwendung von Systemkomponenten und anderen mehr oder weniger verbreiteten Methoden, welche die Produktivität oder auch Softwarequalität verbessern sollen[1], ist die modellgetriebene Softwareentwicklung ein sehr effizientes und weit verbreitetes Konzept, qualitativ hochwertige Softwaresysteme zu entwickeln. Bei der modellgetriebenen Softwareentwicklung spielen die Modelle der abzubildenden Realitätsausschnitte eine wichtigere Rolle als in der klassischen Softwareentwicklung. Hier werden die erstellten Modelle dazu verwendet Code, Dokumentationen oder andere Artefakte mittels Transformationen zu erzeugen. Beispielsweise können so auch Modelle anderer Modellierungssprachen aus den bestehenden Modellen erzeugt werden[2].

Da sich das derzeitige Software-Engineering stark am Wiederverwendungskonzept orientiert und die Einbindung unternehmesexterner Dienstleistungen ebenfalls eine hohe Wirtschaftlichkeit verspricht, werden Softwaresysteme zunehmend unter Verwendung von Geschäftsprozessen entwickelt, welche oft durch die Inanspruchnahme verschiedener Web Services umgesetzt werden. Zur Modellierung solcher Softwaresysteme stehen in der Geschäftsprozessmodellierung verschiedene Modellierungssprachen zur Verfügung, wie z. B. ereignisgesteuerten Prozessketten (EPK) oder auch die Business Process Execution Language (BPEL), welche heute von vielen Entwicklern standardmäßig bei der Modellierung von Geschäftsprozessen eingesetzt werden[3]. Dabei werden fachliche Aspekte der Software unter Zuhilfenahme ereignisgesteuerter Prozessketten modelliert, wohingegen BPEL die technischen Eigenschaften des Systems berücksichtigt. Der hier beschriebene Anwendungsfall verwendet genau dieses Vorgehen bei der Erstellung eines Auskunftssystems. Wie in Abbildung 1 dargestellt, wurden die fachlichen EPK-Modelle des Systems durch Modell-zu-Modell-Transformationen zuerst in BPEL-Modelle überführt. Dies ist durch die Transformation von *EPK* nach *BPEL* dargestellt. *BPEL* enthält dabei alle fachlichen Informationen aus *EPK* und meist noch Standardwerte für technische Details, wie z. B. die Webadressen, unter welchen die Web Services erreichbar sind. Im zweiten Schritt des Entwicklungsprozesses werden die technischen Informationen im entstandenen BPEL-Modell angepasst, wodurch ein verfeinertes BPEL-Modell entsteht, hier durch *BPEL"* dargestellt. Da nun aber die Softwareentwicklung nach der Erstellung des ersten lauffähigen Systems oft nicht abgeschlossen ist, sondern Software meist in einem

[1] Hierfür kann man z. B. das Computer Aided Software Engineering (CASE) nennen, welches dem Entwickler durch Computergestützte Werkzeuge viele Aufgaben bei der Erstellung neuer Software abnimmt bzw. erleichtert[Balz93].

[2] Vgl. [StVö06]

[3] Der Begriff der Geschäftsprozessmodellierung, sowie die Modellierungssprachen EPK und BPEL werden in Abschnitt 2.2 detailliert behandelt.

iterativen Prozess entwickelt wird, können sich z. B. die EPK-Modelle im Zuge der Entwicklung durch das Hinzufügen oder Löschen von Services oder ähnlichem ändern. Ein so geändertes Modell wird hier durch *EPK'* dargestellt. Dieses Modell wird nun erneut in ein BPEL-Modell transformiert (*BPEL'*), welches jedoch wieder nur die Standardwerte für Adressen, Ports usw. beinhaltet. Nun müsste die Definition der technischen Eigenschaften des Systems erneut für das gesamte Modell durchgeführt werden. Dies entspricht aber überhaupt nicht dem Gedanken der Wiederverwendung. Aus diesem Grund ist es von Vorteil, lediglich die Änderungen, welche *BPEL'* gegenüber *BPEL* beinhaltet, zu *BPEL"* hinzuzufügen. Dadurch müsssen lediglich die hinzugekommenen Modellelemente betrachtet werden, was vor allem bei großen Projekten mit sehr umfangreichen Modellen eine enorme Ersparnis an Entwicklerressourcen bedeutet. In Abbildung 1 soll dies ebenfalls verdeutlicht werden. Abschließend resultiert *BPEL"'* aus BPEL" und den hinzugefügten fachlichen Änderungen. *BPEL"'* enthält dann auch die technischen Erweiterungen, welche sich auf die geänderten Modellelemente beziehen.

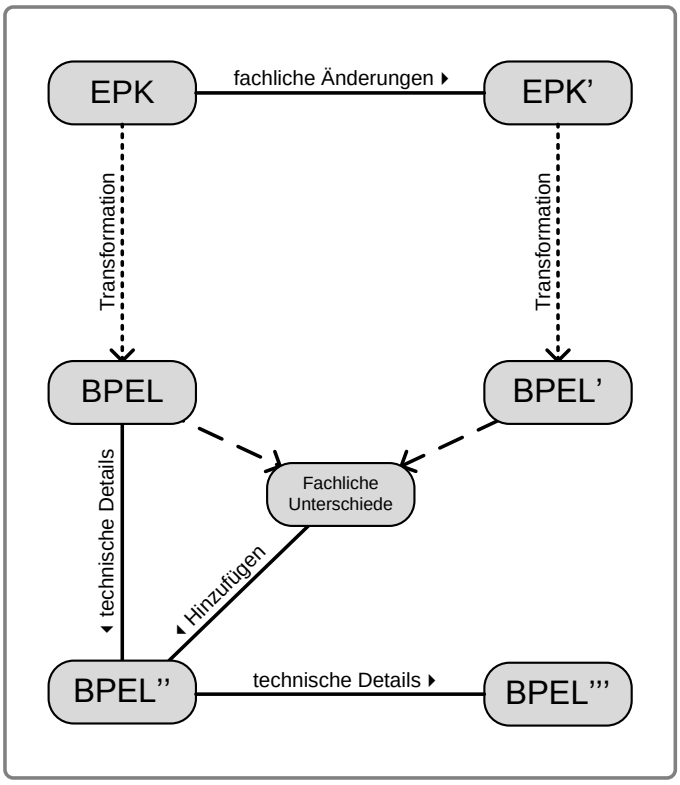

Abb. 1: EPK-zu-BPEL-Transformation

Wie zuvor bereits kurz angesprochen, wurde in dem für dieser Arbeit relevanten Anwendungsfall ein Auskunftssystem entwickelt. Dieses System wird durch die Modelle 2 und 3 abgebildet und stellt im praktischen Teil dieser Arbeit den Ausgangspunkt der Testfälle für die Evaluation der Differenzberechnung unter Nutzung zweier Werkzeuge dar.

Ziel der Software, welche durch das erste EPK-Modell dargestellt wird ist es, Auskunftsanfragen von Nutzern des Systems zu beantworten. Dabei wird für jede Anfrage eine Prüfung der Auskunftsberechtigung für den anfragenden Nutzer veranlasst, welche durch

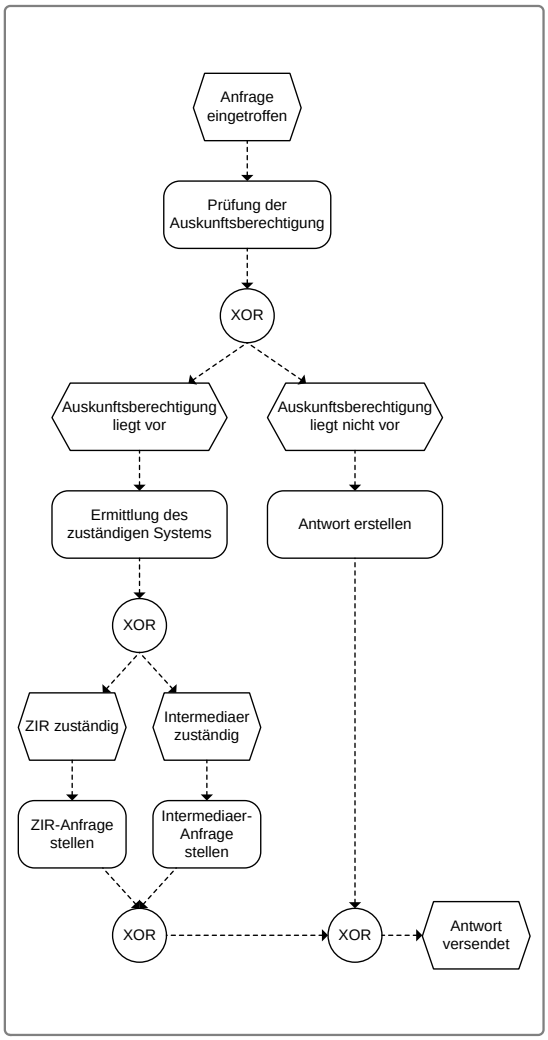

Abb. 2: Beispielsystem ohne Protokollierung

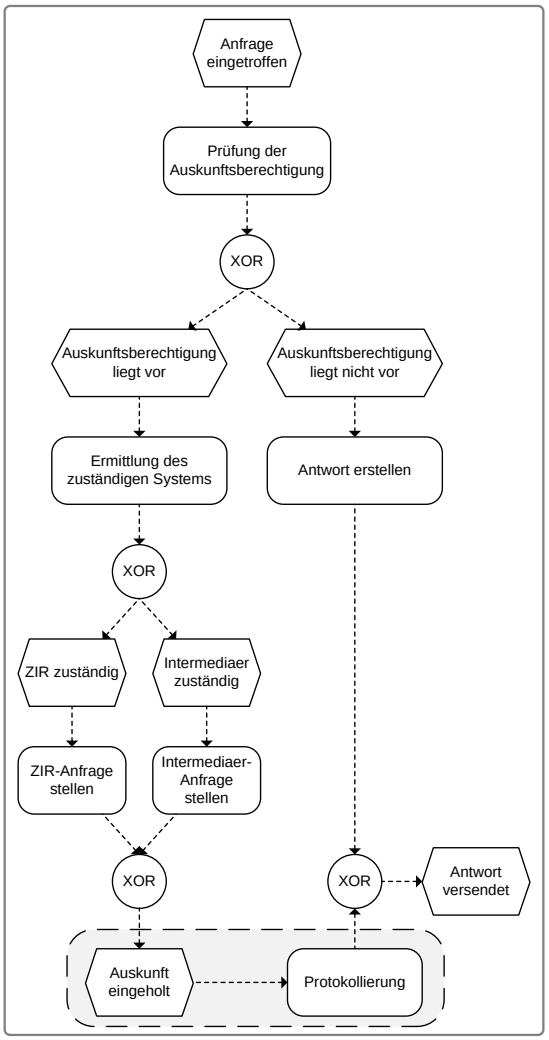

Abb. 3: Beispielsystem mit Protokollierung

einen Berechtigungsdienst ausgeführt wird. Falls der Nutzer über eine Auskunftsberechtigung verfügt, wird das zuständige Auskunftssystem ermittelt. In diesem Beispiel gibt es zwei mögliche Auskunftsdienste, die zentrale Datenbank und den Auskunftsdienst Intermediaer. Die Ermittlung des zuständigen Auskunftsdienstes wird durch den Verzeichnisdienst vorgenommen. Sobald der zuständige Auskunftsdienst bekannt ist, wird an diesen eine Anfrage nach der vom Nutzer gewünschten Information gesendet. Die vom zuständigen Dienst erzeugte Antwort wird dann an den Nutzer weitergeleitet. Falls der Nutzer jedoch keine Auskunftsberechtigung für den Nutzer vorliegt, wird diesem die Verweigerung der Auskunft über eine Nachricht mitgeteilt.

Nachdem das erste Modell in ein BPEL-Modell transformiert und daraus das System erzeugt wurde, soll nun noch eine Protokollierung für die erstellten Auskünfte hinzugefügt werden. Das Modell aus Abbildung 2 wurde um diese Protokollierung erweitert, welche nun ebenfalls dem BPEL-Modell hinzugefügt werden muss. Aus diesem Grund wurde die zweite EPK in ein weiteres BPEL-Modell transformiert. Hier müssen nun die Änderungen gegenüber dem ersten BPEL-Modell ermittelt und dem verfeinerte BPEL-Modell

des ersten Systems hinzugefügt werden, um eine erneute Anpassung des gesamten Systems zu verhindern. Diese Berechnung wird in Kapitel 5 zu Testzwecken durchgeführt und erläutert.

Anmerkend sei gesagt, dass die in Abbildung 2 und 3 dargestellten EPK-Modelle der Übersichtlickeit halber nicht vollständig sind. In diesen Abbildungen wurden die erforderlichen Nachrichten und Dienstanbieter nicht dargestellt. Im Anhang finden sich in den Abbildungen 38 und 39 die vollständigen Modelle des Systems.

1.2 Problemstellung

Softwaresysteme werden heute vermehrt modellgetrieben entwickelt. Dies bedeutet, dass die zu erstellenden Systeme durch Modelle beschrieben werden. Danach werden aus diesen Modellen der eigentliche Quellcode der Software, Dokumentationen oder andere Artefakte generiert. Dabei kommen bei der Entwicklung von Softwaresystemen in der Regel mehrere Probleme auf, welche durch den Einsatz von Modell-Differenzen lösbar sind. So entstehen beispielsweise während des Entwicklungszyklus der Software verschiedene Modellversionen, oder die Modelle werden verteilt von mehreren Modellierern entwickelt.

Der erste der beiden Fälle, welcher auch schon im Anwendungsfall eine Rolle spielte, resultiert beispielsweise daraus, dass große Systeme im allgemeinen durch mehrere Modelle beschrieben werden und der Entwicklungsprozess iterativ umgesetzt wird. Die im Zuge der Systementwicklung entstandenen Modelle müssen nun geeignet verwaltet werden. Da an den im Verlauf solch einer iterativen Systementwicklung erstellten Modellen, welche oft auch aus einer Modell-zu-Modell-Transformation hervorgehen[4], und dem aus den Modellen generierten Code oft noch manuelle Änderungen vorzunehmen sind, ist es erforderlich bei Änderungen der Modelle nicht die gesamte Software, sondern nur die Teile des Systems neu zu generieren, welche von den Änderungen betroffen sind. Dies soll die bereits manuell hinzugefügten Änderungen schützen und so eine Einsparung von Mitarbeiterressourcen ermöglichen. Um solche Anforderungen umzusetzen bietet sich die Nutzung von Modell-Differenzen an. Diese stellen die Änderungen unterschiedlicher Modellversionen dar, welche dann für weitere Generierungen verwendet werden können.

Der zweite Fall entsteht beispielsweise, wenn ein Modell von zwei Entwicklern simultan geändert wird. Deren Änderungen müssen dann so genau wie möglich in einem neuen Modell zusammengeführt werden, wofür eine auf Differenzen basierende Modellvereinigung benötigt wird. Dieser Anwendungsfall von Modell-Differenzen wurde von Alanen und Porres in [AlPo03] aufgeführt und soll später in Kapitel 3 weiter verdeutlicht werden.

[4] siehe Anwendungsfall

1.3 Zielstellung der Arbeit

Ziel dieser Arbeit ist es, verschiedene Ansätze für die Berechnung von Modell-Differenzen zu untersuchen, zu erläutern und auf ausgewählte Testfälle anzuwenden. Dazu werden verschiedene Werkzeuge angesprochen und deren Umsetzungen der Differenzberechnung näher betrachtet. Weiter sollen die ermittelten Ansätze zweier Werkzeuge durch Tests, welche auf der EPK-zu-BPEL-Transformation des Anwendungsfalls aufbauen, evaluiert werden. Es soll ebenfalls eine beispielhafter Ablauf der Differenzberechnung aufgezeigt werden. Des Weiteren werden verschiedene Anwendungsbereiche von Modell-Differenzen betrachtet.

Dieser Arbeit liegen unter anderem folgende Fragestellungen zugrunde, welche im Verlauf der Ausarbeitungen beantwortet werden sollen:

- Was ist eine Differenz von Modellen?
- Wie kann eine Modell-Differenz berechnet werden?
- Wie können Differenzen dargestellt werden?
- Wo können Modell-Differenzen eingesetzt werden?
- Welche Werkzeuge zur Differenzberechnung existieren?
- Wie unterscheiden sich die Ergebnisse der einzelnen Werkzeuge und wie lassen sich diese Unterschiede begründen?

1.4 Aufbau der Arbeit

Die Arbeit gliedert sich im Wesentlichen in drei Abschnitte. Der erste Teil soll zunächst Grundlagen für den weiteren Verlauf der Arbeit legen. Dabei wird zuerst auf das Paradigma der modellgetriebenen Softwareentwicklung eingegangen. Hier werden unter anderem Begriffe wie Modell, Metamodell, Metamodellierung oder auch Modelloperatoren erläutert, damit diese später Verwendung finden können. Danach Folgt ein kurzer Exkurs in die Geschäftsprozessmodellierung. Während dieses Abschnitts werden Modellierungsmöglichkeiten, wie ereignisgesteuerte Prozessketten und die Business Process Execution Language, sowie allgemeine Abläufe bei der Erstellung von Systemen auf Basis von Geschäftsprozessen betrachtet.

Im zweiten Teil der Arbeit wird konkret auf die Berechnung von Modell-Differenzen eingegangen. Es werden sowohl theoretische als auch praxisrelevante Ansätze bei der Differenzberechnung untersucht. Im darauf folgenden Kapitel werden einige Werkzeuge, welche sich sowohl im praktischen Einsatz als auch in der Entwicklung befinden, genannt und näher betrachtet.

Im letzten Kapitel dieser Arbeit wird anhand des Anwendungsbeispiels (EPK-zu-BPEL-Transformation) aus 1.1 und der darauf aufbauenden Testfälle die Berechnung von Modell-Differenzen unter Verwendung zweier Werkzeuge evaluiert. Es werden ausgehend von den Modellen der EPK-zu-BPEL-Transformation verschiedene Änderungen der Modelle vorgenommen, mit deren Hilfe die Berechnung von Differenzen durch die Werkzeuge überprüft werden soll.

2 Grundlagen

2.1 Model-Driven Engineering

In der Softwareentwicklung haben sich in den letzten 50 Jahren viele Konzepte und Methoden herausgebildet, welche es ermöglichen, Software immer effizienter und qualitativ hochwertiger zu entwickeln[5]. Dabei wurde der Grad der Abstraktion bei der Entwicklung von Softwaresystemen stetig erhöht. Es wurden zahlreiche Methoden und Verfahrensweisen entwickelt, welche die Produktqualität und die Produktivität steigern sollen. So kamen z. B. sogenannte CASE-Tools auf, welche den Nutzer aktiv bei der Softwareentwicklung unterstützen oder die Unified Modeling Language (UML), die heute vor allem in der objektorientierten Softwareentwicklung quasi als Standard für die modellbasierte Beschreibung der Systeme verwendet wird. Ein weiteres Beispiel für einen Ansatz zur Produktivitäts- und Qualitätssteigerung, welcher vor allem im letzten Jahrzehnt stark in den Fokus der Softwareentwicklung gerückt ist, ist die Wiederverwendung bestehender Systemkomponenten oder ähnlichem. Diese Vorgehensweise bei der Softwareentwicklung ermöglicht eine hohe Steigerung der Produktivität und der Softwarequalität.

Die modellgetriebene Softwareentwicklung[6] ist ebenfalls in der Lage, eine derartige Verringerung der Entwicklungszeit bei gleichzeitig hoher Softwarequalität zu erreichen. MDE stellt eine Methode der Softwareentwicklung dar, bei welcher der Fokus während der Entwicklung auf der Erstellung von Modellen eines eher unübersichtlichen Originals liegt. Bei der modellgetriebenen Softwareentwicklung entsteht dabei gegenüber der klassischen Softwareentwicklung ein großer Vorteil aus der konsequenten Nutzung der Modelle des zu entwickelnden Systems, welche durch das Model-Driven Engineering erstellt wurden. Hier werden Modelle nicht nur zur Beschreibung der Software verwendet. Die entwickelten Modelle werden vielmehr dazu genutzt, das System fast schon vollautomatisch zu erstellen. Dabei wird die Software mittels geeigneter Generatoren aus den Modellen erzeugt, wodurch (fast) keine manuellen Implementierungen mehr vorgenommen werden müssen. Mittels dieser Modelle ist es ebenfalls möglich das System domänenspezifisch, also auf den jeweiligen Anwendungsbereich zugeschnitten zu beschreiben und durch Validierung der Modelle Fehler der Software bereits frühzeitig im Entwicklungsprozess zu vermeiden. Dies wird beispielsweise durch Douglas C. Schmidt postuliert. Nachfolgend ein Zitat aus [Schm06]:

> *„MDE tools impose domain-specific constraints and perform model checking that can detect and prevent many errors early in the life cycle.“*

[5] Vgl. [StVö06]

[6] in der Literatur meistens als Model-Driven Engineering (MDE) oder Model-Driven Software Development (MDSD) bezeichnet. Dabei stellt MDSD einen Oberbegriff für alle Techniken dar, deren Ziel es ist, aus formalen Modellen lauffähige Systeme zu generieren [StVö06].

In den nächsten Abschnitten wird der Modellbegriff, sowie der Begriff des Metamodells zur weiteren Verwendung in dieser Arbeit definiert. Weiter wird der Begriff des Modelloperators erläutert sowie verschiedene Operatoren genannt.

2.1.1 Modelle

Modelle können allgemein als ein Muster, Entwurf oder Vorbild für etwas bezeichnet werden[7]. Ein Ziel von Modellen ist die Vereinfachung eines komplexen Sachverhaltes, um diesen für den Menschen anschaulich darzustellen. So hat z. B. der Philosoph Klaus Dieter Wüsteneck[8] 1963 den Modellbegriff wie folgt allgemein definiert:

> *„Ein Modell ist ein System, das als Repräsentant eines komplizierten Originals auf Grund mit diesem gemeinsamer, für eine bestimmte Aufgabe wesentlicher Eigenschaften von einem dritten System benutzt, ausgewählt oder geschaffen wird, um letzterem die Erfassung oder Beherrschung des Originals zu ermöglichen oder zu erleichtern, beziehungsweise um es zu ersetzen.“*

In vielen Wissenschaftsbereichen ist es durchaus üblich, Modelle zur Beschreibung bestimmter Sachverhalte zu verwenden. Dies soll die nachfolgende beispielhafte Auflistung einiger Wissenschaften, in welchen Modelle verwendet werden, verdeutlichen.

- **Wissenschaftstheorie:** In der Wissenschaftstheorie, werden Modelle beispielsweise zur Darstellung einer Annahme verwendet, welche bereits relativ sicher ist. Diese Annahmen werden als Theorien bezeichnet. Im Gegensatz dazu werden hypothetische Annahmen als Hypothesen bezeichnet. Modelle dienen dabei dazu, komplexe Theorien semantisch über ihren Anwendungsbereich zu beschreiben. Es wird hier lediglich eine strukturelle Ähnlichkeit der Modelle mit der Wirklichkeit verlangt, welche unter Anderem durch Analogien und Idealisierungen[9] beschrieben wird.[10]
- **Logik:** „Eine Belegung der atomaren Formeln mit Wahrheitswerten, für die eine Gesamtformel F den Wert „wahr“ ergibt, nennt man ein Modell für F.“ [KrKü06]
- **Wirtschaftswissenschaften:** Modelle werden ebenfalls in den Wirtschaftswissenschaften verwendet. Durch Modelle werden z. B. ökonomische Strukturen und Prozesse beschrieben und analysiert. Ein Modellbeispiel in der Volkswirtschaftslehre (VWL) ist der *Homo oeconomicus*, welches einen fiktiven Akteur der Volkswirtschaft durch seine Eigenschaften beschreibt.
- **Informatik:** Die Informatik nutzt Modelle zur Abbildung eines Ausschnitts der Realität, um in diesem Aufgaben durch Nutzung der Informationstechnologie zu lösen. Modelle in der Informatik sind beispielsweise Sequenzdiagramme, Klassendiagramme, Aktivitätsdiagramme, uvm., welche Bestandteil der UML sind. Auch EPKs können hier angeführt werden, die auch im weiteren Verlauf dieser Arbeit eine wichtige Rolle spielen werden. Des Weiteren wurde bereits in Kapitel 2.1 ver-

[7] Vgl. [Bro91]
[8] Vgl. [Wüst63]
[9] Idealisierungen können z. B. die Annahme eines Idealen Gases in der Physik oder der vollkommene Markt in der Volkswirtschaftslehre sein.
[10] Vgl. [Kühne99]

deutlicht, dass Modelle die primären Artefakte der Modellgetriebenen Softwareentwicklung darstellen.

1973 hat Herbert Stachoviak in *Allgemeine Modelltheorie* [Sta73] den Begriff des Modells etwas genauer betrachtet. Er unterscheidet dabei drei primäre Merkmale eines Modells, welche nachfolgend kurz erläutert werden.

Abbildungsmerkmal: Modelle sind stets Abbildungen von natürlichen oder künstlichen Objekten. Diese Objekte können ebenfalls Modelle sein. Des Weiteren muss ein Objekt, welches durch ein Modell beschrieben wird, nicht zwangsläufig bereits existieren. In diesem Fall ist das Modell das Vorbild eines zu schaffenden Objektes. (Dieser Ansatz wird im MDSD verwendet um Systeme zu generieren, wohingegen in der klassischen Softwareentwicklung Modelle oft dafür verwendet werden, existierende Systeme anschaulich darzustellen.)

Verkürzungsmerkmal: Im Allgemeinen erfassen Modelle nur die Objekte und Attribute des durch sie repräsentierten Originals, die vom jeweiligen Modellerschaffer und/oder Modellbenutzer für relevant erachtet werden. Diese Technik der Modellierung wird allgemeinhin als Abstraktion bezeichnet und trennt, wie zuvor bereits beschrieben, die wichtigen von den unwichtigen Merkmalen.

Pragmatisches Merkmal: Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion in dreierlei Hinsicht:

(a) für bestimmte – erkennende und/oder handelnde, modellbenutzende – Subjekte
(Dies bedeutet, dass Modelle nicht nur Modelle von etwas sind, sondern auch Modelle für jemanden, z. B. einen Menschen oder einen künstlichen Modellbenutzer, dem dadurch die Erfassung des Sachverhaltes oder dessen Verarbeitung ermöglicht wird.)

(b) innerhalb bestimmter Zeitintervalle
(Modelle erfüllen ihren Zweck nur zu einem bestimmten Zeitpunkt oder in einem Zeitintervall, nämlich solange, wie die Beschreibung des Sachverhaltes benötigt wird und gültig ist. Es kann durchaus vorkommen, dass sich der abgebildete Ausschnitt der Realität verändert, woraufhin die Modelle verworfen oder angepasst werden müssen.)

(c) unter Einschränktung auf bestimmte gedankliche oder tatsächliche Operationen
(Modelle haben immer einen bestimmten Zweck. Ausschließlich diesem Zweck sind sie zugeordnet.)

Da nun der Modellbegriff für diese Arbeit ausreichend definiert und mit einigen Beispielen hinterlegt wurde, werden abschließend, bevor im nächsten Kapitel die Definition des Begriffs Metamodell vorgenommen wird, zwei weitere Zitate angebracht. Diese beiden Aussagen bekräftigen die zuvor genannten Eigenschaften von Modellen und betrachten Modelle vor allem aus Sicht der Softwareentwicklung.

> *„Models consist of sets of elements that describe some physical, abstract, or hypothetical rality. Good models serve as means of communication; they´re cheaper to build than the real thing; and they suit the plan of attack that the team takes toward solving the problem at hand.“* [MSUW04]

„In the software development domain, models of software systems are typically used to understand and predict properties of the system or to produce implementations.“ [FrRu06]

2.1.2 Metamodelle

Modelle werden, wie bereits bekannt, im MDSD dazu verwendet, das zu erstellende System darzustellen und weitestgehend automatisiert zu erstellen. Grundlage für die Erstellung und die Verwendung von Modellen sind Modellierungssprachen, welche die Syntax und die Semantik der verwendbaren Modellelemente definieren[11]. Somit stellt die Entwicklung von Modellierungssprachen im Model Driven Engineering einen wichtigen Arbeitsschritt dar, da ohne diese meist auf bestimmte Anwendungsdomänen zugeschnittenen Modellierungssprachen, die domänenspezifische Modellierung von Softwaresystemen nicht möglich ist[12]. Die Entwicklung von Modellierungssprachen wird im Gegensatz zur Entwicklung von Modellen, welche konkrete Anwendungssysteme beschreiben, nicht als Modellierung, sondern als Metamodellierung bezeichnet. Metamodelle sind selbst wieder Modelle, die nun jedoch die Sprachkonzepte definieren, welche von den eigentlich benötigten Modellen verwendet werden können. Stephen J. Mellor hat Metamodelle wie folgt beschrieben:

„A metamodel is simply a model of a modelling language. It defines the structure, semantics, and constraints for a family of models.“ [MSUW04]

(Der Begriff *family* wurde in diesem Zusammenhang genutzt, um Modelle zu gruppieren, welche eine gemeinsame Syntax und Semantik besitzen.)

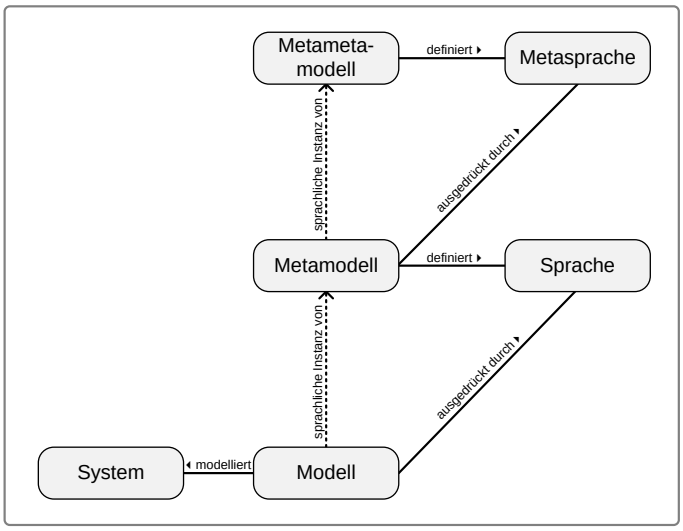

Abb. 4: Metamodellhierarchie [Kühne06]

Metamodelle selbst sind also Modelle von Modellen und müssen ihrerseits wieder durch eine Modellierungssprache beschrieben werden. Diese Sprache wird allgemein als Metasprache bezeichnet. Diese Metasprache setzt wieder die Existenz eines Modells voraus,

[11] Vgl. [Kühne06]
[12] Solche Modellierungssprachen werden als Domain-Specific Modeling Languages (DSML) bezeichnet

welches deren zugrundeliegende Sprachkonzepte definiert. Dieses Modell heißt Metametamodell. Durch die soeben identifizierten (Meta-)Modelle und die zugehörigen Modellierungssprachen lässt sich eine Modellhierarchie aufstellen, welche oben genannte Abhängigkeiten verdeutlicht. Diese Hierarchie ist in Abbildung 4 dargestellt.

Beispiele für die Umsetzung dieses Konzeptes sind unter Anderem die sehr weit verbreitete UML der Object Managemenet Group (OMG)[13] oder auch das recht populäre Eclipse Modeling Framework (EMF)[14], welches später noch etwas näher betrachtet wird.

2.1.3 Modelloperatoren

In der modellgetriebenen Softwareentwicklung existieren verschiedene Arten von Modelloperatoren, die es überhaupt erst ermöglichen, die modellgetriebene Entwicklung von Software zu realisieren. Beispiele solcher Operatoren sind Modelltransformationen, die Vereinigung von Modellen oder auch Differenzen[15] von Modellen. Diese Operatoren spielen alle eine wichtige Rolle während des Softwareentwicklungsprozesses, da beispielsweise durch verschiedene Transformationen weitere Modelle mit einem anderen Betrachtungsfokus erzeugt werden können oder auch Code generiert wird. Differenzen und Vereinigungen sind unter Anderem bei der iterativen bzw. verteilten Entwicklung von Software wichtige Hilfsmittel zur Verwaltung der Modellversionen und deren Verarbeitung im weiteren Verlauf der Entwicklung. Weiter gibt es noch Operatoren, wie das Merging[16] von Modellen, die Validierung[17] der erstellten Modelle oder auch den Split[18] von Modellen. Im Folgenden wird jedoch lediglich die zuerst genannte Gruppe von Operatoren näher btrachtet. Dabei wird zunächst auf verschiedene Modelltransformationen eingegangen, bevor daran anschließend ein kurzer Ausblick auf die Modell-Differenz gegeben wird. Diese wird in Kapitel 3 noch einmal gesondert betrachtet.

Bei Modelltransformationen handelt es sich um Operatoren, die bestehende Modelle in andere Modelle oder auch in Text überführen. Transformationen, welche Modelle in andere Modelle überführen, heißen Modell-zu-Modell-Transformationen (M2M)[19]. Bei dieser Art der Transformation gibt es immer ein Quellmodell mit dem dazu gehörenden Quell-Metamodell und ein Zielmodell, welches oft ein anderes Metamodell besitzt als das Quellmodell. Ziel der M2M-Transformation ist es, eine Abbildung der Elemente des Quell-Metamodells auf die des Ziel-Metamodells zu finden. Ist einmal eine solche Abbildung beschrieben worden, lassen sich sämtliche Instanzen des Quell-Metamodells in In-

[13] Die Object Management Group ist ein Konsortium, welches sich mit der Entwicklung von Standards für die systemübergreifende und herstellerunabhängige objektorientierte Programmierung befasst. [OMG]

[14] EMF ist ein Open-Source Framework, basieren auf Java, mit dessen Hilfe strukturierte Modelle für die modellgetriebene Softwareentwicklung erstellt werden können. Ebenfalls ist es möglich, aus diesen Modellen Quelltext zu generieren. [EMF]

[15] Die Differenz von Modellen wird in der Literatur häufig nur mit einem Δ bezeichnet.

[16] Model-Merging mischt die Inhalte eines Modells oder die Inhalte einer berechneten Modell-Differenz in das Originalmodell [BCE+06]. In dem dieser Arbeit zugrundeliegenden Anwendungsfall können auf diese Weise die ermittelten Differenzen dem verfeinerten Originalmodell hinzugefügt werden.

[17] Die Validierung der Modelle hat das Ziel sicherzustellen, dass die modellierte Software den Anforderungen entspricht. Durch die Validierung der Modelle können so bereits zu einem frühen Zeitpunkt der Entwicklung Fehler vermieden und Kosten gespart werden. [StVö06]

[18] Wird ein Split auf ein Modell angewendet, wird das Modell in mehrere Partitionen zwelegt, deren Abhängigkeit über eine Relation beschrieben ist. Ein Split kann als Inverse Operation eins Merge aufgefass werden. [BCE+06]

[19] Vgl. [StVö06]

stanzen des Ziel-Metamodells transformieren. M2M-Transformationen werden meistens vor der Codegenerierung genutzt, um die Modelle für den Generator aufzubereiten oder mit dem Ziel, Modelle von einer Modellierungssprache in eine andere zu transformieren. Dies ist beispielsweise dann sinnvoll, wenn zu Beginn der Arbeit ein fachliches Modell des Systems erstellt wurde, welches dann in eine technische Modellierungssprache transformiert wird, um dem Modell technische Eigenschaften für die Codegenerierung hinzuzufügen. Als Beispiel für M2M-Transformationen kann man die Query View Transformations (QVT) nennen. QVT stellt eine Spezifikation für eine Metamodell-basierte Sprache zur Modell-zu-Modell-Transformation dar und wurde von der Object Management Group (OMG) spezifiziert.

Eine weitere Modelltransformation ist das Generieren von Code, Dokumentationen oder ähnlichem aus den Modellen. Das Generieren von Softwareartefakten gehört zu den Modell-zu-Plattform-Transformationen (M2P) oder auch Modell-zu-Code (M2C). M2P-Transformationen besitzen eine Art „Hintergrundwissen“ über die, dem System zugrundeliegende Plattform und sind in der Lage, textuelle Artefakte basierend auf der Plattform zu generieren. Eine Art dieser Artefakte ist z. B. der ausführbare Quellcode. (Anmerkend soll hier noch gesagt werden, dass nicht zwangsläufig Quellcode zur Ausführung des Systems generiert werden muss. Es ist ebenfalls möglich, lauffähige Modelle zu erstellen, welche dann mittels eines Interpreters ausgeführt werden. Eine Beispielsprache für ausführbare Modelle ist BPEL4WS, welche in Kapitel 2.2.2 noch näher betrachtet wird.) In dem dieser Arbeit zugrundeliegenden Anwendungsfall wurde ebenfalls eine M2M-Transformation angewendet. Es wurden Modelle von einer fachlichen Modellierungsebene auf eine technische Modellierungsebene transformiert. Die dort verwendeten Modellierungssprachen werden in Kapitel 2.2 detaillierter betrachtet, um die weitere Verwendung in dieser Arbeit vorzubereiten.

Wichtiger als Modelltransformationen ist für diese Arbeit der Differenzoperator. Dieser kann z. B. bei der Versionskontrolle von Modellen eingesetzt werden. Als kurzer Ausblick auf Kapitel 3 sei gesagt, dass der Differenzoperator dazu dient, Unterschiede zwischen zwei Modellen zu berechnen. Diese Differenz kann dann Beispielsweise dazu verwendet werden, verschiedene Versionen eines Modells zu identifizieren und zu verwalten oder Änderungen eines Modells durch verschiedene Modellierer zusammenzuführen.

2.2 Geschäftsprozessmodellierung

Geschäftsprozessmodellierung ist ein wichtiges Mittel, um unternehmensinterne Abläufe zu betrachten und darauf aufbauend Verbesserungen in der Ablauforganisation zu erreichen. Dies kann die Effizienz, den Service oder auch die Qualität der Produkte steigern. Aus diesem Grund ist es auch nicht überraschend, dass viele Unternehmen trotz starkem Kostendrucks zunehmend in die Geschäftsprozessoptimierung investieren, für welche die Modellierung von Geschäftsprozessen die Grundlage bildet.[20]

Zur ersten Bestimmung des Geschäftsprozessbegriffs wird hier die Definition nach Frank J. Rump [Rump99] verwendet:

[20] Vgl. [Rump99, MaSV96, Gada08]

> *„Ein Geschäftsprozess ist eine zeitlich und sachlogisch abhängige Menge von Unternehmensaktivitäten, die ein bestimmtes, unternehmensrelevantes Ziel verfolgen und zur Bearbeitung auf Unternehmensressourcen zurückgreifen.“*

Diese Definition liegt jedoch bereits einige Jahre zurück und sollte laut Andreas Gadatsch [Gada08] verfeinert werden, um sie der heutigen Auffassung von Geschäftsprozessen anzupassen. Gadatsch begründet dies durch aktuelle Umfragen in der Wirtschaft, durch welche gezeigt wurde, dass sich die Schwerpunkte bei der Betrachtung von Geschäftsprozessen verlagert haben. Sein Verständnis von einem Geschäftsprozess ist in nachfolgender Definition zusammengefasst:

> *„Ein Geschäftsprozess ist eine zielgerichtete, zeitlich-logische Abfolge von Aufgaben, die arbeitsteilig von mehreren Organisationen oder Organisationseinheiten unter Nutzung von Informations- und Kommunikationstechnologien ausgeführt werden können. Er dient der Erstellung von Leistungen entsprechend den vorgegebenen, aus der Unternehmensstrategie abgeleiteten Prozesszielen. Ein Geschäftsprozess kann formal auf unterschiedlichen Detailierungsebenen und aus mehreren Sichten beschrieben werden. Ein maximaler Detailierungsgrad der Beschreibung ist dann erreicht, wenn die ausgewiesenen Aufgaben je in einem Zug von einem Mitarbeiter ohne Wechsel des Arbeitsplatzes ausgeführt werden können.“* [Gada08]

Gadatsch betont in seiner Definition z. B. die Nutzung von Informations- und Kommunikationstechnologien zur Abarbeitung der Aufgaben eines Prozesses. Des Weiteren präzisiert er das unternehmensrelevante Ziel dahingehend, dass er sagt, Geschäftsprozesse dienen der betriebswirtschaftlichen Leistungserstellung, wodurch der betriebswirtschaftlichen Charakter von Geschäftsprozessen noch einmal verstärkt wird. Als wichtigste Erweiterung der voranstehenden Definition von Frank J. Rump sind aber die unterschiedlichen Detailierungsebenen hervorzuheben, welche Gadatsch in seiner Definition eingebracht hat. Diese Ebenen könnten insofern wichtig sein, da es mit solch einer Unterscheidung der Detailliertheit der Aufgaben möglich wäre, die Planung und Bearbeitung der Abläufe stark zu vereinfachen und somit rationaler zu gestalten.

Der Begriff des Geschäftsprozesses wurde durch die oben stehenden Definitionen für diese Arbeit ausreichend definiert. Des Weiteren wurde die Wichtigkeit der Geschäftsprozessmodellierung bzw. der darauf aufbauenden Optimierung von Geschäftsprozessen kurz angesprochen. Im Folgenden wird nun eine weit verbreitete Sprache der Geschäftsprozessmodellierung etwas näher betrachtet.

2.2.1 Ereignisgesteuerte Prozessketten

Ereignisgesteuerte Prozessketten wurden von einer Arbeitsgruppe am Institut für Wirtschaftsinformatik der Universität des Saarlandes in Saarbrücken unter Leitung von Prof. Dr. A.-W. Scheer entwickelt und zum ersten mal 1992 in einem technischen Bericht mit dem Titel „Semantische Prozessmodellierung auf der Grundlage ’Ereignisgesteuerter Prozeßketten (EPK)’“ [ScKN92] veröffentlicht. EPKs spielen heute eine wichtige Rolle bei der Modellierung von Geschäftsprozessen auf rein fachlicher Ebene und sind

unter anderem ein Standard der ARIS[21] Platform. Des Weiteren werden EPKs im R/3-Referenzmodell[22] von SAP[23] verwendet. Ereignisgesteuerte Prozessketten werden dabei zur Darstellung der Ablauforganisation von Unternehmen genutzt.

Die Geschäftsprozesse eines Unternehmens werden bei EPKs hauptsächlich unter Verwendung von Ereignissen, Funktionen und Konnektoren (oder auch Regeln) beschrieben (Abbildung 5), welche durch einen Kontrollfluss verbunden sind. Der Kontrollfluss wird als gestrichelte, gerichtete Kante dargestellt. Konnektoren sind in EPKs nicht nur als grafische Mittel zur Verbindung von Modellelementen vorgesehen, sondern dienen auch als logische Verknüpfungen, wie z. B. „und“ „oder“ oder „exklusiv-oder“. Weiter kann eine EPK z. B. auch Informationsobjekte oder Organisationseinheiten (Abbildung 6) enthalten. Falls solche Elemente verwendet werden, nennt man das Modell eine erweiterte ereignisgesteuerte Prozesskette (eEPK).

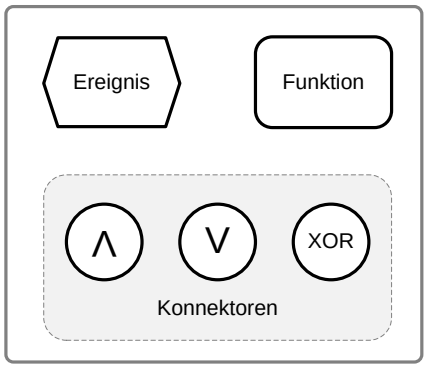

Abb. 5: EPK-Elemente

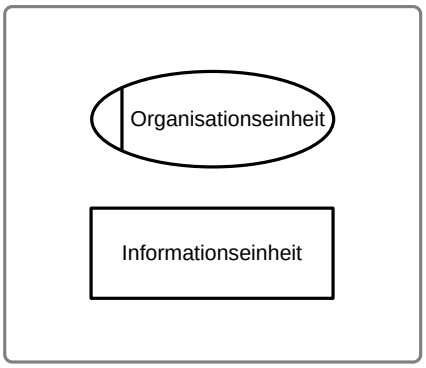

Abb. 6: weitere EPK-Elemente

EPKs bzw. eEPKs können noch weitere Elemente, wie beispielsweise Prozesspfade oder Komponenten enthalten. Diese Elemente werden jedoch in den Testfällen dieser Arbeit nicht verwendet, weshalb im Folgenden auch nicht weiter auf diese Bestandteile einer eEPK eingegangen wird. In der Fachliteratur gibt es viele ausführliche Definitionen von ereignisgesteuerten Prozessketten wie z. B. [ScKN92, NüRu02, Rump99].

Im Folgenden werden die hier relevanten Elemente noch einmal aufgelistet und erläutert[24]:

- ***Ereignisse*** stellen eingetretene betriebswirtschaftliche Zustände dar. Ein Ereignis, also ein bestimmter Zustand, kann eine Folge von Aktivitäten auslösen und ist immer eine zeitpunktbezogene, passive Komponente des Systems.
  Ein Beispiel für ein Ereignis ist „Anfrage ist eingetroffen“.

[21] Die von der IDS-Scheer AG entwickelte ARIS-Platform stellt integrierte Softwareprodukte zur Verfügung, welche Unternehmen bei der Optimierung ihrer Geschäftsprozesse unterstützen sollen. Dabei verwendet ARIS viele Standardsprachen zu Geschäftsprozessmodellierung, wie z. B. EPK, BPEL, WSDL und UML.[ARI08]

[22] Vgl. [ARI08]

[23] SAP ist mit derzeit weltweit über 75.000 Kunden eines der größten unabhängigen Softwarehäuser für Kollaborationssoftware im Geschäftsbereich. [SAP]

[24] Vgl. [Rump99]

- ***Funktionen*** beschreiben in einem EPK-Modell eine Aktivität des Geschäftsprozesses oder einen komplexeren betriebswirtschaftlichen Vorgang. Funktionen sind also aktive Komponenten des Systems.
  Ein Beispiel für eine Funktion ist „Antwort erstellen“.
- ***Konnektoren*** oder auch Verknüpfungsoperatoren sind für die Ablauflogik eines Geschäftsprozesses zuständig. Dabei kann man zwischen den aus der Logik bekannten konjunktiven („AND“ $\wedge$), disjunktiven („OR“ $\vee$) und kontravalenten („XOR“) Operatoren unterscheiden.
- ***Informationsobjekte*** gehören zu Funktionen und werden mittels Pfeilen mit diesen verbunden. Je nach Pfeilrichtung zeigt das Informationsobjekt die für die Funktion benötigten Daten oder die von der Funktion produzierten Daten auf.
- ***Organisationseinheiten*** werden unter Verwendung ungerichteter Kanten mit Funktionen verbunden und zeigen dadurch an, wer für die Abarbeitung dieser Funktion zuständig ist.

Weiter sind bei der Erstellung von ereignisgesteuerten Prozessketten einige Regeln zu beachten, von welchen die wichtigsten nachfolgend aufgeführt sind[25]:

- Eine EPK beginnt und endet immer mit einem Ereignis.
- Funktionen besitzen immer auslösende Ereignisse und erzeugen immer neue Ereignisse.
- Es ergibt sich eine alternierende Folge von Ereignissen und Funktionen, die nur durch Verknüpfungsoperatoren unterbrochen werden kann.
- Ein Verknüpfungsoperator hat entweder eine Eingangs- und mehrere Ausgangskanten (diese Art von Konnektoren werden als Verteiler bezeichnet) oder mehrere Eingangs- und eine Ausgangskante (diese werden Verknüpfer genannt).

Zwei Beispielhafte EPKs sind in Abbildung 2 und 3 im Kapitel 1.1 dargestellt. Diese EPKs sind im Anhang in den Abbildungen 38 und 39 noch einmal vollständig unter Verwendung von Informationsobjekten und Organisationseinheiten als eEPKs dargestellt.

2.2.2 Business Process Execution Language

Die Business Process Execution Language[26] ist ein auf der Extensible Markup Language (XML) basierender Industriestandard zur Beschreibung von Geschäftsprozessen. Ziel ist es, verschiedene Anwendungssysteme unter Zuhilfenahme von Web Service-Technologien zu integrieren und somit Plattformunabhängigkeit und die Wiederverwendung bestehender Systeme zu ermöglichen[27]. Es soll ermöglicht werden, Workflows für Komponenten zu erstellen, wobei die einzelnen Geschäftsprozesse die Komponenten darstellen. Dabei die Web Services die Implementierung der Funktionalität der Prozesse bereit. Mittels BPEL wird lediglich eine Beschreibung eines Web Service angegeben, der dafür verantwortlich ist, Web Services in ihrem zeitlichen Ablauf zu koordinieren, um einen

[25] Vgl. [Rump99]

[26] Die ursprüngliche Bezeichnung von BPEL war BPEL4WS (BPEL for Web Services).
Des Weiteren ist BPEL in der Literatur häufig auch unter dem Namen WSBPEL zu finden.

[27] Diese Technik wird als Orchestrierung von Web Services bezeichnet.

bestimmten Geschäftsprozess zu realisieren. Dabei ist zu beachten, dass BPEL-Prozesse lediglich eine automatisierte Kommunikation mit den verwendeten Web Services bereitstellen. Eine Mensch-zu-Maschine-Kommunikation kann erst von den verwendeten Web Services realisiert werden.

Die Business Process Execution Language stellt zwei unterschiedliche Arten von Prozessen zur Verfügung, ausführbare und abstrakte Prozesse. Ausführbare Prozesse können, wie der Name schon sagt, auf Rechnern (sogenannten Workflowmaschinen) ausgeführt werden, während abstrakte Prozesse dazu dienen, das nach außen sichtbare Verhalten des Prozesses zu beschreiben. Mit diesen beiden Arten von Prozessen wird eine Trennung der internen Umsetzung eines Prozesses und des nach außen sichtbaren Verhaltens erreicht. Abstrakte Prozesse könnten also als eine Art Interface aufgefasst werden, mit deren Hilfe es beispielsweise möglich ist, die konkrete Realisierung eines Prozesses vor einem Geschäftspartner zu verbergen, welcher später eventuell zu einem Konkurrenten werden könnte.

Da BPEL stark IT-fokussiert ist (dies wird z. B. durch die Behandlung von Exceptions deutlich) und über keine standardisierte graphische Notation verfügt, wird eine einfache, fachliche Modellierung von Geschäftsprozessen unter Nutzung von BPEL für den Fachanwender fast schon unmöglich. Wie bereits im vorherigen Kapitel verdeutlicht wurde, bieten sich für die fachliche Ebene der Geschäftsprozessmodellierung andere, einfachere Modellierungsstandards wie ereignisgesteuerte Prozessketten an. Diese Modelle können dann durch geeignete M2M-Transformationen, wie z. B. im ARIS SOA Architekt[28] enthalten, in andere Modellierungssprachen wie BPEL übertragen und mit weiteren technischen Details versehen werden. In dem Anwendungsbeispiel, welches den Tests in dieser Arbeit zugrundeliegt, wurde ebenfalls eine solche Transformation angewendet.

Auch wenn BPEL über keine standardisierte graphische Notation verfügt, gibt es doch Möglichkeiten, BPEL graphisch darzustellen. Eine dieser Möglichkeiten ist die Business Process Modeling Notation (BPMN). Der BPMN-Standard definiert eine Abbildung von BPMN nach BPEL, durch die in BPMN modellierte Prozesse in BPEL-Prozesse transformiert werden können. BPMN wird hier nicht weiter verwendet und wurde nur ergänzend erwähnt. Ein Beispielhaftes Modell unter Verwendung des BPMN-Standards befindet sich im Anhang (Abbildung 40). In der Literatur finden sich viele Dokumentationen der Business Process Modeling Notation, so z. B. in [White04].

Die eigentliche BPEL-Notation erfolgt im XML-Format und ist selbst für einen kleinen Geschäftsprozess relativ unübersichtlich. Im Anhang (Listing 10 befindet sich ein passendes BPEL-Listing zu dem in Abbildung 2 dargestellten Anwendungsbeispiel. Aufbauend auf diesem Listing werden nun folgend die wichtigsten Elemente eines BPEL-Prozesses erklärt und mit einem kurzen Beispiellisting verdeutlicht. Die Elemente eines BPEL-Prozesses werden dafür in zwei Hauptkategorien unterteil.

1. **Basisaktivitäten des Prozesses:**
    - **`<receive>`:** Diese Aktivität erlaubt es dem Prozess, auf eine passende eingehende Nachricht zu warten. Erst wenn diese Nachricht angekommen ist, wird die `<receive>` Aktivität beendet. Eine Nachricht ist passend, wenn sie die vom `<receive>`-Konstrukt definierten Attribute erfüllt (siehe z. B. unten stehendes

[28] Vgl. [ARI08]

Listing 1). Hier wird auf eine konkrete Anfrage gewartet. Es sind verschiedene Attribute wie z. B. der `partnerLink`, der `portType` oder auch die `variable` definiert. Erst wenn die Anfrage diese Anforderungen erfüllt, wird sie angenommen.

```
1 <receive name="AnfrageEingetroffen" createInstance="yes"
    partnerLink="sampleProcessPL" portType="
    tns:anfrageEingetroffenPT" operation="
    AnfrageEingetroffenOperation" variable="Anforderung-0600">
2   <documentation>AnfrageEingetroffen</documentation>
3 </receive>
```

Listing 1: BPEL: Receive

- **`<reply>`:** Diese Aktivität erlaubt es einem Prozess, auf eine eingegangene Nachricht zu antworten. Die Nachricht muss von einer eingebundenen Nachrichtenaktivität, wie z. B. `<receive>` empfangen worden sein. Weitere Nachrichtenaktivitäten sind `<onMessage>` und `<onEvent>`. Ein `<reply>` enthält ebenfalls (teilweise optionale) Attribute, welche wieder gewisse technische Eigenschaften oder auch den Bezug auf einen bestimmten Nachrichteneingang angeben. Im Beispiellisting ist ein `reply` mit Namen Antwort versendet angegeben. Dieses definiert verschiedene Attribute wie z. B. den `partnerLink` mit zugehörigem `portType`, an den die Antwort versendet wird.

```
1 <reply name="Antwort␣versendet" partnerLink="sampleProcessPL"
    portType="tns:anfrageEingetroffenPT" operation="
    AnfrageEingetroffenOperation" variable="Antwort-0601">
2   <documentation>Antwort versendet</documentation>
3 </reply>
```

Listing 2: BPEL: Reply

- **`<invoke>`:** Ein `<invoke>` ist dafür zuständig, eine Ein- oder Zweiwegeoperation auf einem bestimmten `portType` durchzuführen, welcher von einem Partner-Service angeboten wird. Falls es sich um eine Zweiwege-Kommunikation handelt (`request-response`), wird die Aktivität damit abgeschlossen, dass eine Antwort des Partners (`response`) eingeht. In unten stehendem Listing wird die Operation zur Prüfung der Auskunftsberechtigung für den anfragenden Nutzer eingeleitet. Dazu werden verschiedene Attribute, wie der zuständige `partnerLink` mit `portType`, sowie eine `inputVariable` und eine `outputVariable` definiert. Diese beiden Variablen Definieren beispielsweise Voraussetzung und das Ergebnis der Operation. Weiter hat das `<invoke>` ein Kindelement, welches die Dokumentation dieser Aktivität angibt.

```
1 <invoke name="PruefungAuskunftsberechtigung" partnerLink="
    berechtigungsdienstPL" portType="
    tns:pruefungAuskunftsberechtigungPT" operation="
    PruefungAuskunftsberechtigungOperation" inputVariable="
    Anforderung-0600" outputVariable="InfoAuskunftsberechtigung
    ">
2   <documentation>PruefungAuskunftsberechtigung</documentation>
3 </invoke>
```

Listing 3: BPEL: Invoke

- **<assign>:** Eine <assign>-Aktivität wird dafür genutzt, einer Variablen einen neuen Wert zuzuweisen. Ein <assign> kann beliebig viele elementare Wertzuweisungen enthalten. Eine dieser Zuweisungen ist beispielsweise eine <copy>-Aktivität. Das hier gelistete <assign> kopiert z. B. den Inhalt der Variablen Anforderung-0600 in die Variable Antwort-0601.

```
1 <assign name="AntwortErstellen">
2   <copy>
3     <from variable="Anforderung-0600" part="Anforderung-0600
        Part"></from>
4     <to variable="Antwort-0601" part="Antwort-0601Part"/>
5   </copy>
6 </assign>
```

Listing 4: BPEL: Assign

2. **Kontrollflussaktivitäten des Prozesses:**
   - **<partnerLinks>:** Dieser Abschnitt des BPEL-Prozesses ist eine Sammlung der verschiedenen Partner, die mit diesem Prozess in Interaktion treten. Jeder <partnerLink> beschreibt einen Web Service und kapselt verschiedene Informationen, wie z. B. den partnerLinkType. Nachdem die einzelnen partnerLinks definiert wurden können sie später im Prozess weiter verwendet werden, um die Dienste eines Service abzurufen. Dieses Konstrukt ist eigentlich kein Kontrollflusselement. Das Listing zeigt einen Ausschnitt der Sammlung von partnerLinks des Anwendungsfalls. Der dargestellte partnerLink besitzt einen Namen, einen Typ sowie eine Rolle im System.

```
1 <partnerLinks>
2   <partnerLink name="berechtigungsdienstPL" partnerLinkType="
      tns:berechtigungsdienstPLT" partnerRole="partnerRole"/>
3   ...
4 </partnerLinks>
```

Listing 5: BPEL: PartnerLinks

- **<variables>:** Diese Sektion beschreibt die Datenvariablen, welche der Prozess verwendet. Die Datenvariablen dienen der Aufrechterhaltung des Status

während des Austauschens, sowie zwischen dem Austausch von Nachrichten. Eine <variable> kann wie in unserem Beispiel eine bestimmte Nachricht, oder auch eine Systemanforderung sein. Das Listing zeigt eine von mehreren Variablen des Beispielsystems, welche einen Namen und einen Nachrichtentyp besitzen.

```
1 <variables>
2   <variable name="Anforderung-0600" messageType="
        tns:anforderung-0600MT"/>
3   ...
4 </variables>
```

Listing 6: BPEL: Variables

- **<sequence>:** Eine <sequence> ist eine Aktivität, welche eine Menge von Aktivitäten definiert, die sequentiell in der angegebenen Reihenfolge abgearbeitet werden. Die hier dargestellte <sequence> enthählt zwei Aktivitäten, welche in der gegebenen Reihenfolge abgearbeitet werden. Zuerst wird auf den Eingang einer Nachricht gewartet (<receive>). Erst wenn diese Anfrage eingetroffen ist, wird die Aktivität zur Prüfung der Auskunftsberechtigung vom System gestartet (<invoke>).

```
1 <sequence>
2   <receive name="AnfrageEingetroffen" createInstance="yes"
        partnerLink="sampleProcessPL" portType="
        tns:anfrageEingetroffenPT" operation="
        AnfrageEingetroffenOperation" variable="Anforderung-0600"
        >
3   </receive>
4   <invoke name="PruefungAuskunftsberechtigung" partnerLink="
        berechtigungsdienstPL" portType="
        tns:pruefungAuskunftsberechtigungPT" operation="
        PruefungAuskunftsberechtigungOperation" inputVariable="
        Anforderung-0600" outputVariable="
        InfoAuskunftsberechtigung">
5   </invoke>
6 </sequence>
```

Listing 7: BPEL: Sequence

- **<if>:** Die <if>-Aktivität dient dazu, genau eine von mehreren alternativen Aktivitäten auszuwählen. Hier wird das aus Programmiersprachen allgemein bekannte if-else-Format verwendet. In unten stehendem Listing wird auf jeden Fall eine der beiden alternativen Aktiviäten ausgeführt. Falls die in <condition> geforderte Bedingung erfüllt ist, wird die Aktivität ZirAnfrageStellen gestartet. Falls die Bedingung nicht erfüllt ist, wird der <else>-Zweig ausgelöst und die Aktivität IntermediaerAnfrageStellen gestartet.

```
1 <if>
2   <condition>...</condition>
3   <invoke name="ZirAnfrageStellen" partnerLink="
        zentraleDatenbankPL" portType="tns:zirAnfrageStellenPT"
        operation="ZirAnfrageStellenOperation" inputVariable="
        Anforderung-0600" outputVariable="Antwort-0601">
4   </invoke>
5   <else>
6     <invoke name="IntermediearAnfrageStellen" partnerLink="
          intermediaerPL" portType="
          tns:intermediearAnfrageStellenPT" operation="
          IntermediearAnfrageStellenOperation" inputVariable="
          Anforderung-0600" outputVariable="Antwort-0601">
7     </invoke>
8   </else>
9 </if>
```

Listing 8: BPEL: If

Diese Auflistung zeigt einige der im Anwendungsfall dieser Arbeit verwendeten BPEL-Elemente auf. Eine umfangreiche Dokumentation der Bestandteile eines BPEL-Modells findet sich in „Web Services Business Process Execution Language Version 2.0“[JoEv07] von OASIS[29].

[29] OASIS (Organization for the Advancement of Structured Information Standards) ist eine internationale Organisation, welche sich mit der Weiterentwicklung von Web Service- und EBusiness-Standards beschäftigt. Bekannte Standards sind beispielsweise BPEL und das Open Document Format (ODF) [OASIS]

3 Berechnung von Modell-Differenzen

3.1 Einsatzgebiete von Modell-Differenzen

Im Folgenden soll die Berechnung von Modell-Differenzen näher betrachtet werden. Zuvor wird jedoch kurz auf mögliche Einsatzgebiete der Differenzen von Modellen eingegangen, wie diese bereits in der Problemstellung dieser Arbeit in Kapitel 1.2 angesprochen wurden.

Wie bereits erwähnt werden Modell-Differenzen häufig bei der Versionskontrolle eingesetzt. Dabei ist es notwendig, Änderungen von Modellen ausfindig zu machen, um gegebenenfalls Modelle mit neuen Versionsnummern zu versehen. Weiter werden Differenzen dazu verwendet werden, bestehende Systeme mittels geänderter Modelle anzupassen, ohne die Software vollständig neu generieren zu müssen. Dies geht bereits aus dem Anwendungsfall aus Kapitel 1.1 hervor, in welchem das bereits lauffähige Auskunftssystem um die Fuktion der Protokollierung erweitert wird. Bei diesem Vorgehen ist es erforderlich, die Differenz der Modelle zu entwickeln, um die Änderungen des neuen Modells dem alten Modell, welches bereits weiter verfeinert wurde, hinzuzufügen. Auf diese Weise wird eine vollständige Neugenerierung des Systems und die daraus resultierende Notwendigkeit, manuelle Anpassungen für das gesamte System vorzunehmen vermieden. Eine weitere Anwendung von Modell-Differenzen zeigen Marcus Alanen und Ivan Porres in [AlPo03] auf. Alanen und Porres gehen davon aus, dass ein Quellmodell (im Folgenden als $M_{original}$ bezeichnet) von zwei verschiedenen Modellierern simultan geändert wird. Dabei entstehen die beiden geänderten Modelle, welche nachfolgend als M_1 und M_2 bezeichnet werden. Nun ist es erforderlich, die Änderungen beider Modellierer in dem Zielmodell (Nachfolgend M_{final} genannt) zusammenzufassen[30]. Nutzt man für diese Aufgabe die mengentheoretische Vereinigung, so ist es möglich, dass einige Änderungen in M_{final} nicht berücksichtigt werden. Deutlich wird dies in [AlPo03] dadurch, dass der erste Designer Modellelement B löscht, wohingegen der zweite Designer dieses Element im Modell belässt und ein weiteres Element D hinzufügt. Wendet man nun die mengentheoretische Vereinigung auf beide Modelle an, enthält M_{final} die Elemente A, B, C und D. Hier werden die Änderungen des ersten Modellierers nicht berücksichtigt. Dies ist der Fall, da die Vereinigung aus mengentheoretischer Sicht wie folgt definiert ist:

$$M_{final} = M_1 \cup M_2 := \{x | (x \in M_1) \vee (x \in M_2)\}$$

Es werden also in $M_1 \cup M_2$ sowohl alle Elemente der Menge M_1, als auch alle Elemente der Menge M_2 zusammengefasst. Da nun die Änderung in M_1 ein gelöschtes Element ist, welches in M_2 jedoch nicht gelöscht wurde, haben Alanen und Porres die hier benötigte Vereinigung beider Modelle wie folgt definiert:

[30] Abbildung 7 zeigt die simultane Änderung des Quellmodells.

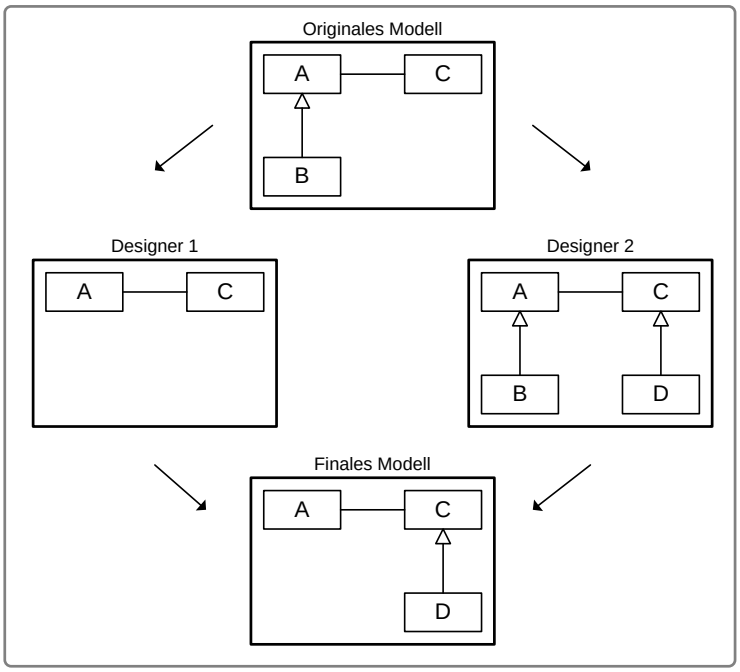

Abb. 7: Vereinigung zweier Versionen eines Modells (aus [AlPo03])

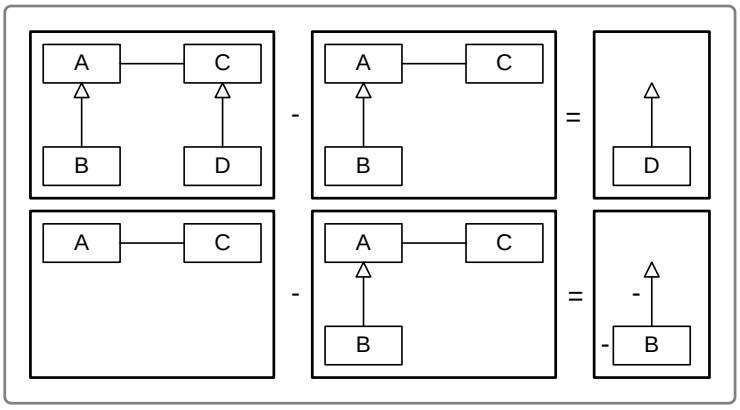

Abb. 8: Differenz zweier Modelle (aus [AlPo03])

$$M_{final} := M_{original} + (M_1 - M_{original}) + (M_2 - M_{original})$$

Die Berechnung von M_{final} erfolgt nun durch das in Abschnitt 2.1.3 bereits angesprochene Merging von $M_{original}$ mit den beiden berechneten Modell-Differenzen. Durch diese Vereinigung auf Basis der Differenzen werden die Änderungen beider Modellierer in M_{final} berücksichtigt und es gehen keine Informationen verloren. Die beiden Differenzen $(M_1 - M_{original})$ und $(M_2 - M_{original})$ sind in Abbildung 8 zu finden. Des Weiteren ist obige Berechnung von M_{final} in Abbildung 9 graphisch verdeutlicht.

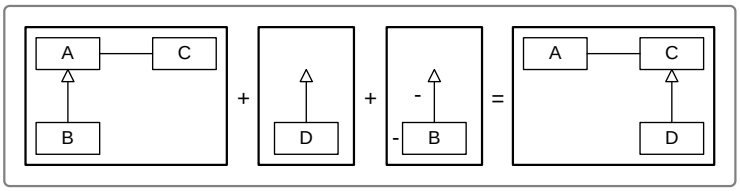

Abb. 9: Beispiel einer Diff-basierten Vereinigung (aus [AlPo03])

Nachdem zuvor einige Einsatzmöglichkeiten von Modell-Differenzen aufgezeigt wurden, soll nun die Berechnung von Modell-Differenzen näher betrachtet werden. Diese wird nachfolgend in drei möglichen Schritten erläutert, welche sich durch Analyse einiger theoretischer, sowie praktischer Ansätze ergeben haben. Als erstes ist die Berechnung der Ähnlichkeiten von Elementen des ersten und des zweiten Modells notwendig[31]. Danach wird eine Abbildung der Modellelemente aufeinander erläutert, bevor abschließend in Abschnitt 3.4 die eigentliche Berechnung und Darstellung der Differenz zweier Modelle näher erläutert wird.

3.2 Modell-Matching

Einleitend wurde bereits erwähnt, dass zur Berechnung der Modell-Differenz mehrere Schritte nötig sind. Der erste dieser Arbeitsschritte ist die Ermittlung von Ähnlichkeiten der Modellelemente. Oft werden Differenzen in der Literatur auf UML-Modellen berechnet. Dabei wird manchmal, wie beispielsweise in [AlPo03], die Existenz eines Universally Unique Identifiers (UUID) vorausgesetzt, durch welchen die Modellelemente eindeutig identifiziert werden können. Durch die Tatsache, dass UUIDs vom System generiert und jede ID genau einem Element zugeordnet ist, kann jedes Element eindeutig und ohne Missverständnisse über diese UUID identifiziert werden. Aus diesem Grund kann bei Verwendung solcher IDs der Vergleich der einzelnen Modellelemente entfallen. Es ist lediglich notwendig einen Test der IDs durchzuführen, um die Gleichheit der Elemente zu prüfen. Dies ist in der Praxis jedoch in der Regel nicht möglich, da Modelle meist von vielen Modellierern auf verschiedenen Systemen erstellt oder geändert werden[32]. In diesem Fall kann es vorkommen, dass eine UUID mehrfach und für unterschiedliche Elemente vergeben wird oder dass semantisch gleiche Elemente verschiedene IDs zugewiesen bekommen, wodurch die Abbildung der Elemente aufeinander fehlerhaft erfolgt. Aus diesem Grund ist für eine Differenzermittlung in der Praxis in der Regel der Vergleich der einzelnen Modellelemente notwendig. Bevor jedoch eine beispielhafte Berechnung des Vergleichs erfolgen kann, ist es erforderlich, mögliche Elemente und deren Eigenschaften zu identifizieren. In der Regel wird die Berechnung der Differenz unter Zuhilfenahme eines konkreten Metamodells erläutert. Meist wird dafür die Unified Modeling Language herangezogen[33]. Die nachfolgende Differenzierung der hier verwendeten Modellelemente ergab sich sowohl durch Recherche als auch durch eigene Tests:

[31] In der Literatur entfällt dieser Schritt häufig unter der Annahme, dass es möglich ist, jedes Element eindeutig zu identifizieren. Mehr dazu in Kapitel 3.2.
[32] Vgl. [Toul06]
[33] UML wird beispielsweise in [AlPo03] und [KoPP06] für die Berechnung der Differenz genutzt.

- **Container:** Container beinhalten in der Regel andere Modellelemente. In EMF sind Container typischerweise EPackages und EClasses.
- **Objekte:** Diese werden ab sofort als Klassen bezeichnet. Klassen können Attribute, Referenzen oder Operationen kapseln. Ein Beispiel eines Objektes in EMF ist eine EClass.
- **Attribute:** Attribute sind analog zu Attributen in Klassendiagrammen zu sehen und verweisen auf einfache Datentypen. Sie beschreiben den Zustand einer Klasse. Das EMF Metamodell Ecore bietet hierfür EAttribues an.
- **Referenzen:** Referenzen sind Verweise auf andere Klassen im Modell. Hierfür gibt es EReferences.
- **Operationen:** Operationen sind analog zu Methoden in Klassendiagrammen zu betrachten und kapseln die Funtionalität einer Klasse. In EMF sind dies EOperations.

EMF wurde als „Metamodell“[34] gewählt, da die später bei den Tests verwendeten Werkzeuge auf diesem Framework aufsetzen.

Diese fünf Typen von Elementen sind im Allgemeinen ausreichend für die Berechnung von Ähnlichkeiten zwischen den Modellelementen. Um den Vergleich der Modellelemente nicht nur für eine bestimmte Klasse von Modellen zu gewährleisten, darf die Differenzierung der Modellelemente nicht zu stark zu verfeinert werden. Beispielsweise ist es in EMF möglich, Ecore-spezifisch die Namen der Modellelemente zu vergleichen, da laut Ecore jedes Element einen Namen besitzt. Falls der Vergleich jedoch nicht auf Ebene der Metamodelle, welche ihre Eigenschaften direkt von Ecore erben, sondern auf den Modellen angewendet wird, ist die Existenz eines Namens für jedes Modellelement nicht zwangsläufig gewährleistet. Hier kommt es bei Berücksichtigung der Namen unter Umständen zu Fehlern in der Berechnung.

Im Folgenden soll ausgehend von der obigen Differenzierung der Modellelemente ein beispielhafter Vergleich der Elemente aufgezeigt werden. Dieser Vergleich ergab sich aus Recherchen über verschiedene Ansätze, sowie eigener Arbeit am Werkzeug ModelMatcher.

Bei einem Vergleich zweier Modelle müssen prinzipiell alle Modellelemente des ersten Modells mit allen Modellelementen des zweiten Modells verglichen werden. Für diese Vergleiche bietet es sich an, jeweils ein Tupel aus beiden Modellelementen zu bilden, zu welchem ein Vergleichswert gespeichert wird. Bei einem Vergleich sämtlicher Modellelemente des ersten Modells mit sämtlichen Modellelementen des zweiten Modells entsteht schnell ein sehr umfangreiches Kreuzprodukt beider Modelle, was einen großen Rechenaufwand bedeutet. Wenn man jedoch das entstandene Kreuzprodukt durch das Eliminieren nicht relevanter Kombinationen von Elementen verkleinert, ist eine große Steigerung der Effizienz dieses Vergleichs möglich. Laut Alanen und Porres ist es möglich, Tupel aus dem Kreuzprodukt zu entfernen, welche nicht den gleichen Metatyp besitzen, welche also Instanzen unterschiedlicher Elemente des Metamodells sind. Diese Vorgehensweise wird auch in der Praxis angewendet[35] und soll nachfolgend am Beispiel des Systems aus dem Anwendungsfall verdeutlicht werden.

[34] EMF selbst ist kein Metamodell. Das Metametamodell des Eclipse Modeling Framework ist Ecore und in Abbildung 41 dargestellt. Eine detaillierte Dokumentation findet sich in [Ecore, EMF, BSM+03].

[35] z. B. bei EMF Compare

Bei einem Vergleich der beiden EPKs aus Kapitel 1.1 (Abbildungen 2 und 3) sind bei der Bildung des Kreuzproduktes eine Gesamtzahl von 255 Vergleichen notwendig, da das erste Modell 15 Elemente und das zweite Modell 17 Elemente enthält[36]. Geht man davon aus, dass aus einem Ereignis einer erignisgesteuerten Prozesskette nach Änderung des Modells nie eine Funktion oder ein Konnektor entstehen kann, so müssen lediglich die Ereignisse des ersten Modells mit den Ereignissen des zweiten Modells verglichen werden. Selbiges gilt für alle anderen Modellelemente. Unter dieser Voraussetzung ergeben sich 42 Vergleiche von Ereignissen, 30 Vergleiche von Funktionen und 16 Vergleiche von XOR-Konnektoren. Die daraus resultierende Gesamtzahl von nur noch 88 Vergleichen bedeutet eine verkleinerung der Kreuzproduktes um 167 Elemente und somit eine Verringerung des Berechnungsaufwandes um ca. 66%[37]. Im Folgenden wird davon ausgegangen, dass das gebildete Kreuzprodukt beider Modelle bereits bereinigt wurde.

Da die Tupel nun bekannt sind, deren Elemente verglichen werden sollen, kann jetzt der eigentliche Vergleich beginnen. Der Vergleich wird hier beispielhaft erläutert, wie er sich aus Recherche über verschiedene Werkzeuge und durch eigene Arbeit ergeben hat. Die bei der Berechnung der Vergleichswerte herangezogenen Eigenschaften der Elemente werden in Kapitel 4 für einige Werkzeuge noch einmal detailliert betrachtet. Für einen Vergleich der Elemente eines Tupels haben sich im Wesentlichen die Folgenden Schritte ergeben:

1. Vergleich aller Attribute des ersten Elements mit allen Attributen des zweiten Elements. Dabei ist darauf zu achten, dass nur korrespondierende Attribute miteinander verglichen werden. Beispielsweise darf niemals ein Name-Attribut mit einem Attribut verglichen werden, welches ein Element als abstrakt deklariert.
2. Als nächstes bietet sich der Vergleich des Containers an, welcher das aktuelle Element beinhaltet. Hier kann man, um nicht den Vergleich für den Container selbst wieder durchführen zu müssen, den vorherigen Vergleichswert beider Container-Elemente aus der Vergleichstabelle nutzen.
3. Im nächsten Schritt kann ein Vergleich der Referenzen der beiden Elemente folgen.

   In unserem Anwendungsfall muss davon ausgegangen werden, dass es erforderlich ist, die Modelle für einen automatisierten Vergleich in eine geeignetere Darstellung zu transformieren. Hier werden oft Baumdarstellungen verwendet, wie z. B. in EMF. Dadurch müssen natürlich Modellelemente wie die Kanten, die den Kontrollfluss darstellen, aufgelöst und anderweitig dargestellt werden. Dieses Problem wird im Allgemeienen über Referenzen gelöst. Es ist beispielsweise möglich für eine Kontrollflusskante ein Modellelement vom Typ Kontrollfluss zu erstellen, welches jeweils eine Referenz auf das Startelement und das Endelement enthält.

   Beim Vergleichen der Referenzen kann der Einfachheit halber wieder auf die Vergleichswerte der referenzierten Elemente aus den vorhergehenden Schritten zurückgegriffen werden. Die Referenzen eines Modellelements können zwar ebenfalls als eine Art Attribut gesehen werden, jedoch müssen Referenzen und Attribute beim Vergleich der Werte grundsätzlich verschieden behandelt werden. Attribute referen-

[36] Die Kontrollflusskanten wurden in dieser beispielhaften Berechnung nicht als zu vergleichende Modellelemente berücksichtigt.

[37] Bei der Verringerung des Rechenaufwandes sind die Vergleiche auf Metatypen nicht berücksichtigt worden, da diese im Gegensatz zu einem vollständigen Elementvergleich einen sehr geringen Rechenaufwand ausmachen.

zieren im Gegensatz zu Referenzen einfache Datentypen wie z. B. Integer, Float oder Boolean. Diese lassen sich durch die vom Datentyp bereitgestellten Operatoren sehr einfach auf Gleichheit testen. Bei dem Gleichheitstest der Referenzen ist es jedoch erforderlich zu prüfen, ob die beiden referenzierten Elemente gleich sind, da der Gleichheitstest der Objekte, als die die Elemente in den Speicher geladen wurden, über den „=“-Operator fehlschlagen würde.

4. Als letzter Schritt soll der Vergleich des Inhalts der Beiden Elemente genannt werden. Unter Inhalt sind die Kindelemente eines Containers zu verstehen, die sich in einer Baumnotation ergeben. Es ist auch hier möglich, für den Vergleich dieser Kindelemente auf die Vergleichswerte der vorhergehenden Schritte zurückzugreifen.

Nachdem diese Schritte durchlaufen wurden, liegt für jedes Tupel ein Vergleichswert vor. Danach ist es möglich, diese Werte auf einen bestimmten Referenzwert zu normieren, damit eine Vergleichbarkeit der Werte erreicht wird. In der Praxis wird oft eine Normierung auf den Wert 1.0 vorgenommen[38], was einer 100% igen Übereinstimmung der Elemente entspricht. Der Datentyp, welcher die Tupel und die zugehörigen Vergleichswerte verwaltet wird im Folgenden als Matching bezeichnet. Das Matching bildet die Grundlage für den nächsten Schritt der Differenzermittlung, das Mapping der Modellelemente.

3.3 Modell-Mapping

Ausgehend vom zuvor ermittelten Matching der Modelle soll hier ein Mapping, also eine Abbildung zwischen den Modellen erarbeitet werden. Beim Mapping der Modelle wird versucht, jedes Element des ersten Modells auf Elemente des zweiten Modells abzubilden, also korrespondierende Elemente der beiden Modelle zu finden. Elemente aus dem ersten und dem zweiten Modell können aufeinander abgebildet werden, falls sie dasselbe konzeptionelle Artefakt des abgebildeten Realitätsausschnitts beschreiben. Elemente, welche keine korrespondierenden Elemente im gegenüberliegenden Modell besitzen, werden während dieser Phase der Differenzermittlung ebenfalls verzeichnet und später bei der Berechnung der Differenz entweder als neue oder gelöschte Modellelemente berücksichtigt. Bei der Ermittlung der Abbildung der Modellelemente werden die Vergleichswerte, welche im Matching zu jedem Tupel von Modellelementen gespeichert sind, als eine Wahrscheinlichkeit angesehen, mit der die Elemente dasselbe Artefakt beschreiben[39]. Zur Ermittlung des Mappings ist noch eine Klassifikation der Modellelemente notwendig. Eine solche Klassifikation wird beispielsweise von Kolovos, Paige und Polack in [KoPP06] vorgenommen. Diese Klassifikation ist nachfolgend aufgelistet und erläutert.

Als initiale Bedingung an einen Vergleichsalgorithmus führen Kolovos, Paige und Polack die Verarbeitung zweier Modelle (linkes und rechtes Modell) und die folgende Partitionierung der Modellelemente auf:

[38] Eine Normierung der Vergleichswerte findet sich z. B. in COMA++, EMF Compare oder auch in UMLDiff

[39] Wahrscheinlich werden auch aus diesem Grund die Vergleichswerte auf einen Wert von 1.0 normiert. Ein Vergleichswert von 1.0 würde in dem Fall eine 100% ige Wahrscheinlichkeit anzeigen, mit der die Elemente übereinstimmen.

1. Elemente, die auf andere Elemente im gegenüberliegenden Modell abgebildet werden können, die also dasselbe konzeptuelle Artefakt beschreiben.
2. Elemente, die nicht auf Elemente des gegenüberliegenden Modells abgebildet werden können.

Die Elemente der ersten Kategorie werden in Abbildung 10 durch weiße Knoten dargestellt, Elemente der zweiten Kategorie durch schwarze Knoten.

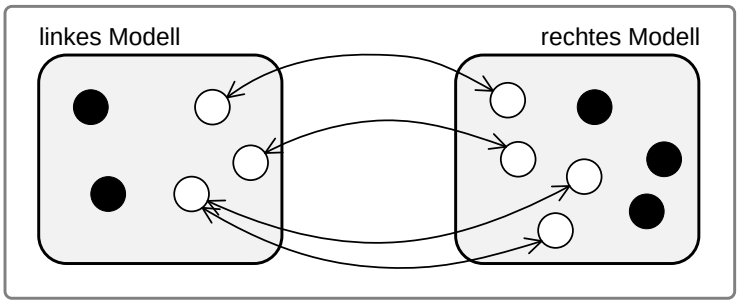

Abb. 10: Klassifikation der Elemente zweier Modelle (nach [KoPP06])

Im nächsten Schritt wird die Klassifikation weiter verfeinert. Dies ergibt sich ganz einfach durch den Fakt, dass bei Änderungen an Modellen nicht nur Elemente gelöscht oder hinzugefügt werden. Es werden vielmehr einzelne Eigenschaften von Elementen geändert, wie beispielsweise die Deklaration einer Klasse als abstrakt im neuen Modell. Da beide Elemente prinzipiell dasselbe Artefakt beschreiben, jedoch gewisse Unterschiede zwischen ihnen bestehen, wird in [KoPP06] die erste Elementkategorie noch einmal in zwei Teilkategorien unterteilt.

1. Elemente, die auf andere Elemente im gegenüberliegenden Modell abgebildet werden können.
    a) Elemente, die zu ihren Mapping-Partnern im gegenüberliegenden Modell konform sind.
    b) Elemente, die zu ihren Mapping-Partnern im gegenüberliegenden Modell nicht konform sind.

Elemente werden in diesem Zusammenhang als konform bezeichnet, falls sie eine 100% ige Übereinstimmung aufweisen. Es dürfen also keine Änderungen der Modellelemente festgestellt werden. Zwei Modellelemente sind beispielsweise nicht konform, falls eines der beiden Elemente als abstrakt deklariert ist. In Abbildung 11 sind die Elemente der Kategorie 1a weiterhin durch weiße Knoten dargestellt. Die Elemente der Kategorie 1b werden nun jedoch durch graue Knoten dargestellt und mittels eines gepunkteten Pfeils verbunden.

Als letzte Verfeinerung dieser Kategorisierung werden die Elemente der zweiten Kategorie weiter unterschieden. Die zweite Kategorie wird in zwei weitere Gruppen von Elementen unterteilt. Hier werden Elemente unterschieden, welche der Domain der Vergleichsoperation angehören und Elemente, die der Domain der Vergleichsoperation nicht angehören. Dies lässt sich dadurch erklären, dass in [KoPP06] nicht die Existenz eines gemeinsamen Metamodells vorausgesetzt wird, wodurch es Elemente geben kann, die von

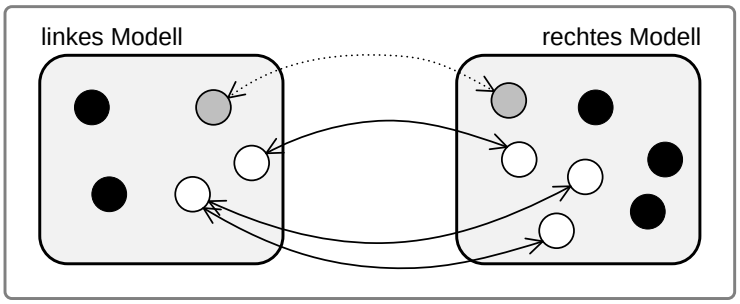

Abb. 11: Verfeinerte Elementklassifikation (nach [KoPP06])

der Vergleichsoperation nicht verarbeitet werden können. Diese Elemente werden dann einfach ignoriert, wohingegen für die Elemente, welche in Kategorie 2 eingeordnet sind und der Domain der Vergleichsoperation angehören, kein passender Mapping-Partner gefunden wurde. Abbildung 12 verdeutlicht diese Unterscheidung graphisch.

2. Elemente, die nicht auf Elemente des gegenüberliegenden Modells abgebildet werden können.
   a) Elemente, die der Domain der Vergleichsoperation angehören.
   b) Elemente, die nicht der Domain der Vergleichsoperation angehören.

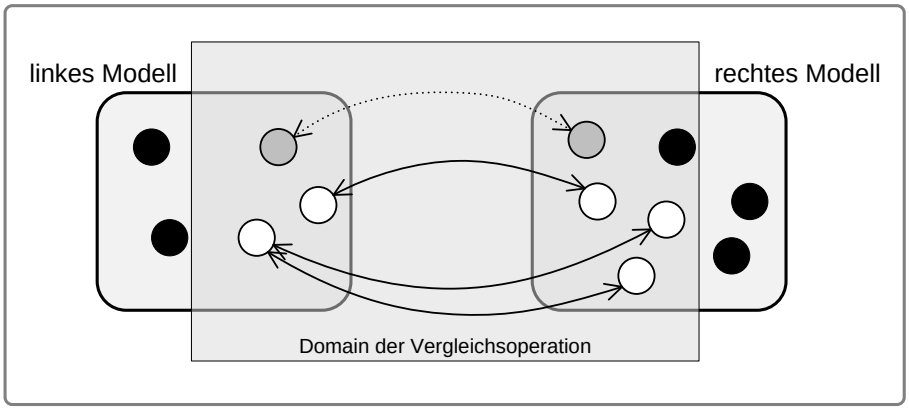

Abb. 12: Weiter verfeinerte Elementklassifikation (nach [KoPP06])

Die Verfeinerte Unterscheidung der Kategorie 2 spielt bei der Betrachtung des Mappings in dieser Arbeit jedoch keine Rolle, da hier immer ein gemeinsames Metamodell beider Modelle vorausgesetzt wird. Letztere Kategorisierung sollte nur der Vollständigkeit halber aufgeführt werden. Für das Mapping, wie es hier durchgeführt wird, ergibt sich, bezogen auf [KoPP06], folgende Kategorisierung der Elemente:

1. a) Elemente aus Tupeln, deren Vergleichswert 1.0 ist, da hier keine Unterschiede gefunden wurden.
   b) Elemente aus Tupeln, die prinzipiell dasselbe Artefakt beschreiben, deren Vergleichswert jedoch kleiner als 1.0 ist. Diese Elemente können aufeinander abgebildet werden.

2. Elemente, deren Vergleichswert kleiner als 1.0 ist und die nicht aufeinander abgebildet werden dürfen, da sie andere Artefakte des abgebildeten Realitätsausschnitts beschreiben.

Die Elemente der Kategorie 1a zu finden ist sehr einfach, da hier lediglich das Matching nach Tupeln durchsucht werden muss, zu denen ein Vergleichswert von 1.0 gespeichert ist. Die Elemente des Tupels werden dann im Mapping aufeinander abgebildet. Das Problem besteht darin, die restlichen Elemente auf die Kategorien 1b und 2 zu verteilen. Da bei beiden Kategorien die Vergleichswerte kleiner als 1.0 sind, ist vorerst kein sichtbares Erkennungsmerkmal vorhanden. Eine mögliche Lösung dieses Problem, wie sie bei der Entwicklung des ModelMatcher erarbeitet wurde, berücksichtigt beim Mapping der Elemente die jeweils höchsten Vergleichswerte für ein Element eines Modells. Nachdem der höchste Vergleichswert für ein Element gefunden wurde, wird aus dem entsprechenden Tupel, zu welchem dieser Vergleichswert gespeichert ist, das zweite Element extrahiert. Nun wird zu diesem Element ebenfalls der höchste Vergleichswert ermittelt. Falls die beiden Tupel, zu denen die jeweils höchsten Vergleichswerte gespeichert sind, jeweils dieselben Elemente enthalten, also identisch sind, können die beiden Elemente des Tupels in die Kategorie 1b eingeordnet und somit aufeinander abgebildet werden. Falls die beiden Tupel jedoch nicht identisch sind, existiert zum ersten Element des Tupels kein korrespondierendes Element des gegenüberliegenden Modells und das Element wird in Kategorie 2 eingeordnet. Um das Mapping weiter zu verdeutlichen, ist ein kommentierter Ausschnitt aus dem zuvor beschriebenen Algorithmus nachfolgend aufgelistet.

```
1  /*Das Matching befindet sich in der Variable matching*/
2
3  //Erzeuge eine Kopie des Matching, auf welcher weiter gearbeitet wird.
4  Matching matchCopy = (Matching) matching.clone();
5
6  while(matchCopy.size() != 0) {
7     /* Nimm erstes Element des ersten Tupels aus dem kopierten Matching.
8      * Erzeuge temporäres Matching (leftMatching), welches alle Tupel
           aus der Kopie des Matchings enthält,
9      * deren erstes Element mit dem ersten Element des Tupels identisch
           ist.
10     * Speichere anschließend in t1 das Tupel, welches den höchsten
           Vergleichswert aus leftMatching besitzt.
11     */
12    Matching leftMatching = matchCopy.getMatchingByFirstElement(((Tuple)
          matchCopy.keySet().toArray()[0]).getFirstElement());
13    Tuple t1 = (Tuple) leftMatching.maximumMatchingEntry().getKey();
14
15    /* Nimm das zweite Element des Tupels t1 und suche alle Tupel, deren
           zweite Elemente mit diesem übereinstimmen.
16     * Speichere diese Menge in rightMatching.
17     * Speichere das Tupel mit dem größten Vergleichswert aus
           rightMatching in t2.
18     */
19    Matching rightMatching = matching.getMatchingBySecondElement(t1.
          getSecondElement());
```

```
20      Tuple t2 = (Tuple) rightMatching.maximumMatchingEntry().getKey();
21
22
23      /* Falls nun t1 und t2 identisch sind, kann man die Elemente
            aufeinander mappen.
24       * Falls die beiden Tupel nicht identisch sind, bedeutet dies, dass
            eines der Elemente
25       * einen größeren Vergleichswert mit einem anderen Element besitzt,
            wodurch das Mapping
26       * der beiden Elemente nicht möglich ist.
27       */
28      if(t2.equals(t1)) {
29                      ...
30           Hier wird das Mapping zwischen den beiden Elementen verarbeitet
                .
31                      ...
32      }
33
34
35      /* Anschließend müssen alle Einträge aus leftMatching aus matchCopy
            gelöscht werden.
36       * Dadurch wird die Schleife beendet, wenn alle Elemente gemapped
            sind, zu welchen
37       * ein korrespondierendes Element existiert.
38       */
39      for(Iterator j=tmp.keySet().iterator(); j.hasNext(); ) {
40           matchCopy.remove(j.next());
41      }
42  }
```

Listing 9: Mapping-Algorithmus des ModelMatcher

In der Testphase des ModelMatcher hat der zuvor gelistete Algorithmus die Mappings vieler Modelle zuverlässig und genau berechnet. Anmerkend sei jedoch gesagt, dass während des Mappings Probleme auftreten, falls mehrere Elemente desselben Metatyps vorhanden sind, welche keine Attribute oder sonstige Merkmale besitzen, durch die sie eindeutig identifiziert werden könnten. In diesen Fällen sind die Vergleichswerte immer identisch, wodurch beim Mapping durch diesen Algorithmus keine eindeutige Entscheidung getroffen werden kann.

Egal wie das Mapping umgesetzt wird, steht nach dem Mapping der Elemente fest, welche Modellelemente hinzugefügt, gelöscht oder geändert wurden. Abschließend muss ausgehend von diesem Mapping der Modelle die angestrebte Modell-Differenz berechnet werden. Diese Berechnung wird im nächsten Abschnitt genauer Betrachtet.

3.4 Modell-Differenz

In den beiden vorhergehenden Schritten wurde die Grundlage zur Ermittlung der Differenz zwischen zwei Modellen gelegt. Mit dem Mapping der Modelle sind korrespondierende Elemente der beiden Modelle aufeinander abgebildet worden. Ebenso wurden Elemente identifiziert, zu denen keine korrespondierenden Elemente existieren. Ausge-

hend von diesem Mapping kann die Differenz beider Modelle erarbeitet werden. Bevor jedoch die Differenz der Modelle berechnet werden kann, muss die Frage geklärt werden, was eine Modell-Differenz ist und wie sie dargestellt werden kann. Die Antwort auf die erste der beiden Fragen lautet wie folgt: Die Differenz zwischen zwei Modellen M_1 und M_2 zeigt die Änderungen von M_2 gegenüber M_1 auf[40]. Änderungen des Modells können sowohl das Hinzufügen oder Löschen von Modellelementen, als auch die Änderung von Eigenschaften von Modellelementen umfassen. Beispielsweise können Attributwerte oder Referenzziele geändert werden. Eine Antwort auf die Frage nach der Darstellung von Modell-Differenzen soll nun gefunden werden. Es gibt die Möglichkeit, die Differenz als eigenständiges Modell darzustellen. Dabei ist es jedoch notwendig, die gelöschten Modellelemente als negative Modellelemente auszuzeichnen, wie dies z. B. in [AlPo03] gezeigt wurde und in Abbildung 8 dargestellt ist. Hier stellt sich die Frage ob es sinnvoll ist, ein Modell mit negativen Elementen zu erstellen und wie in einem solchen Modell beispielsweise Änderungen von Attributwerten dargestellt werden. Alanen und Porres sind aus diesem Grund zu dem Schluss gekommen, dass es intuitiver und für eine automatisierte Verarbeitung effektiver ist, Differenzen als Sequenzen von Änderungsoperationen darzustellen. Eine initiale Bedingung an Modell-Differenzen ist, dass aus einer Anwendung dieser Operationen auf das Modell M_1 das Modell M_2 resultiert. Alanen und Porres haben für Ihre Darstellung von Modell-Differenzen sieben elementare Transformationen identifiziert, welche nachfolgend aufgelistet sind. Dabei sind sie von der Prämisse ausgegangen, dass der Metatyp eines Elements nicht änderbar ist. Diese Einschränkung wurde in dieser Arbeit bereits in Kapitel 3.2 bei der Berechnung des Matching von Modellen angenommen, um den Rechenaufwand bei dem Vergleich der Modelle zu minimieren. Weiter wird in [AlPo03] die Existenz einer UUID für jedes Element vorausgesetzt, welche bei den nachfolgenden Operationen berücksichtigt werden. Unter der Annahme, dass diese UUIDs nicht existieren, was in der Praxis häufiger der Fall ist, können die Operationen auch ohne die angegebenen IDs verwendet werden.

1. Erstellen und Löschen von Elementen
   - $new(e,t)$: Erzeuge Element vom Typ t mit UUID e.
   - $del(e,t)$: Lösche Element vom Typ t mit UUID e.
2. Änderung einer Eigenschaft vom Typ f, eines Elements e mit UUID u.
   - $set(e,f,v_o,v_n)$: Ändere den Wert des Features $e.f$ von v_o zu v_n.
     Dies gilt für Attribute primitiven Typs.
   - $insert(e,f,e_t)$: Füge eine Verbindung von $e.f$ nach e_t für ein ungeordnetes Feature hinzu.
   - $remove(e,f,e_t)$: Lösche eine Verbindung von $e.f$ nach e_t, für ein ungeordnetes Feature.
   - $insertAt(e,f,e_t,i)$: Füge eine Verbindung von $e.f$ nach e_t an Position i für ein geordnetes Feature hinzu.
   - $removeAt(e,f,e_t,i)$: Lösche eine Verbindung von $e.f$ nach e_t an Position i, für ein geordnetes Feature.

[40] Vorausgesetzt, M_1 ist das Original und M_2 die neue Version. Die Differenz wird dann wie folgt berechnet: $\Delta = M_2 - M_1$.

Bei der Änderung von Elementeigenschaften ist zu beachten, dass Porres und Alanen keine Bidirektionalität der Operationen anstreben, da vorausgesetzt wird, dass die jeweilige Gegenrichtung während der Berechnung ebenfalls durch eine Operation geändert wird.

Die hier aufgeführte Unterscheidung der Änderungsoperationen nach [AlPo03] ist sehr fein granuliert. Dabei wird beispielsweise von der Existenz geordneter Listen für Attribute oder Referenzen ausgegangen. Eine solche Ordnung ist jedoch in der Praxis eher selten vorhanden[41] und wird aus diesem Grund hier nicht weiter berücksichtigt. Aus weiterer Recherche der praxisrelevanten Werkzeuge und Arbeiten und eigener Entwicklung ergibt sich folgende Unterscheidung der Änderungsoperationen, aus welchen sich eine Differenz im Wesentlichen zusammensetzt:

1. **Erstellen von Elementen:** Dieser Operator hat das Ziel, neue Elemente im ursprünglichen Modell zu erzeugen. Dabei werden alle Elemente, welche im neuen Modell M_2, jedoch nicht im ursprünglichen Modell M_1 enthalten sind, durch eine Änderungsoperation dieser Art erfasst. Diese Elemente sind entweder gar nicht, oder nur als einzelne Elemente ohne Mapping-Partner im Mapping verzeichnet.
2. **Löschen von Elementen:** Bei diesem Operator ist das Ziel analog zum Erzeugen von Modellelementen zu sehen. Jedoch werden hier Elemente entfernt. Es werden alle Elemente des Originalmodells M_1 verarbeitet, welche nicht im geänderten Modell M_2 enthalten sind. Diese Elemente sind im Mapping der Modelle gar nicht oder ebenfalls als einzelne Elemente ohne Mapping-Partner verzeichnet.
3. **Änderung von Elementeigenschaften:** Ziel dieses Operators ist es, alle Änderungen von Modellelementen beider Modelle als Änderungsoperationen zu erfassen. Dazu werden alle Elemente verarbeitet, welche im Mapping auf andere Elemente abgebildet sind. Falls der Vergleichswert der aufeinander abgebildeten Elemente 1.0 beträgt, sind die Elemente vollkommen identisch und es muss keine Änderungsoperation erstellt werden. Falls dieser Wert jedoch kleiner als 1.0 ist, müssen die Änderungen beider Elemente zusätzlich kalkuliert werden. Zu diesem Zweck müssen alle Attribute und Referenzen der Elemente betrachtet werden. Bei Änderungen von Attributen oder Referenzen wird lediglich eine Änderungsoperation erzeugt, welche die betroffene Eigenschaft benennt und deren neuen Wert angibt. Falls jedoch ein Attribut oder eine Referenz hinzukommt oder gelöscht wird, muss eine Einfüge- oder Löschoperation für das jeweilige Feature erzeugt werden, welches angibt wo es eingefügt oder gelöscht werden soll und welchen Wert das Feature erhalten soll. Diese Betrachtung spielt jedoch nachfolgend keine Rolle. Die in den Tests verwendeten Werkzeuge verwenden das Eclipse Modeling Framework, welches für nicht verwendete Eigenschaften der Modellelemente Null- bzw. Standardwerte einsetzt.

Nach diesem Schritt liegt eine vollständige Modell-Differenz vor, welche beispielsweise in einer Versionskontrolle verwendet werden kann. Wie diese Differenzen in der Praxis berechnet und dargestellt werden, soll im nächsten Kapitel durch die Betrachtung einiger praxisrelevanter Werkzeuge geklärt werden.

[41] Vgl. [Toul06]

4 Werkzeuge

4.1 Einleitung

In diesem Abschnitt werden vier Werkzeuge vorgestellt, mit deren Hilfe der Vergleich von Modellen und die Berechnung von Modell-Differenzen möglich sind. Drei der vier genannten Werkzeuge spielen bereits im Praxiseinsatz eine wichtige Rolle, lediglich der ModelMatcher befindet sich noch in der Entwicklung. Dabei sind die beiden Werkzeuge COMA++ und SiDiff bereits in zahlreichen Veröffentlichungen behandelt worden und genießen ein durchaus hohes Ansehen in Fachkreisen, wohingegen sich EMF Compare durch eine hohe Praxisrelevanz auszeichnet. Die Werkzeuge EMF Compare und ModelMatcher wurden gewählt, da sie einerseits einen ähnlichen Vergleichsansatz nutzen und Modelle des Eclipse Modeling Framework verarbeiten. Andererseits wird das Werkzeug EMF Compare, wie bereits erwähnt, in der Praxis bereits häufig verwendet. Der ModelMatcher ist hingegen ein eigenes Projekt welches im Rahmen dieser Arbeit getestet und validiert werden soll. Diese beiden Werkzeuge werden ebenfalls in den Tests dieser Arbeit verwendet. Die beiden Werkzeuge SiDiff und COMA++ sollen hier alternative Ansätze des Modellvergleichs aufzeigen. Beide arbeiten auf anderen Modellarten als die ersten beiden Werkzeuge.

In Kapitel 5 werden Differenzen verschiedener Modelle, welche alle an der EPK-zu-BPEL-Transformation des Anwendungsfalls ausgerichtet sind, berechnet. Dort werden die Ergebnisse der beiden Werkzeuge EMF Compare und ModelMatcher miteinander verglichen. Die Verwendung der Werkzeuge COMA++ und SiDiff in den Tests ist aus verschiedenen Gründen nicht möglich. Ein Grund, weshalb diese Werkzeuge nicht verwendet werden können ist dass beide nicht auf dem Eclipse Modeling Framework aufbauen und keine EMF-Modelle verarbeiten können. Da aber die Modelle, zwischen welchen hier die Differenzen berechnet werden sollen, nach einer Transformation im EMF-Format vorliegen, fiel unter anderem aus diesem Grund die Entscheidung auf die Verwendung der ersten beiden Werkzeuge. In den nachfolgenden Abschnitten werden die vier zuvor genannten Werkzeuge näher betrachtet. Dabei liegen die Schwerpunkte der Betrachtung auf den beiden Tools EMF Compare und ModelMatcher. Bei der Analyse dieser beiden Werkzeuge wird das jeweilige Werkzeug kurz vorgestellt und eine Betrachtung der Benutzungsoberfläche vorgenommen. Danach werden jeweils die Architektur und der Vergleichsansatz, sowie die Differenzberechnung und deren Darstellung betrachtet. Die Betrachtung der Werkzeuge SiDiff und COMA++ wird nicht in diesem Umfang stattfinden. Hier wird jeweils zu Beginn eine kurze Einführung in die Werkzeuge vorgenommen. Danach folgt zu beiden Werkzeugen eine kurze Betrachtung der Architektur und des Vergleichs.

4.2 EMF Compare

4.2.1 Einführung und Benutzungsoberfläche

Das erste Werkzeug, welches hier vorgestellt werden soll, nennt sich EMF Compare. EMF Compare ist ein Plugin für die Eclipse IDE[42] und wurde in Zusammenarbeit von Obeo[43] und Intalio[44] entwickelt. Dabei ist das Projekt EMF Compare ein Teil des Eclipse Modeling Framework Technology (EMFT)[45] Projektes, dessen Zielstellung es ist, neue Technologien aufbauend auf EMF einzuführen. Beispiele für neue Technologien, welche durch EMFT eingeführt wurden, sind das verteilte Arbeiten an Modellen auf Basis eines Modell-Repository oder das Ausführen von Suchanfragen auf EMF-Modellen, sowie eine Möglichkeit, Modelle in relationalen Datenbanken persistent zu speichern. Sowohl die Obeo Company als auch Intalio sind stark in der modellgetriebenen Softwareentwicklung aktiv, weshalb beide Unternehmen die Erforderlichkeit einer geeigneten Umsetzung von Modelloperatoren für die modellgetriebene Softwareentwicklung erkannt und verwirklicht haben.

EMF Compare nutzt einen sehr effizienten und generischen Vergleichsalgorithmus, mit welchem es möglich ist, Modelle zu vergleichen, welche auf beliebigen Metamodellen basieren. Des Weiteren ist es möglich, das Plugin für den eigenen Gebrauch über sogenannte Extension Points[46] zu erweitern. Eine Nutzung des Tools ist sowohl unter Verwendung einer vorgefertigten Benutzungsoberfläche, als auch durch Einbindung der Funktionalität über Quellcode möglich. Um einen ersten Einblick in das Werkzeug zu ermöglichen, ist in Abbildung 13 die bereitgestellte Benutzungsoberfläche dargestellt und wird anschließend kurz erläutert.

Die Benutzungsoberfläche des Eclipse-Plugins untergliedert sich im Wesentlichen in zwei Teile. Der erste der beiden Teile ist die eigentliche Auflistung der berechneten Modell-Differenz. Diese, in Abbildung 13 mit der Nummer 1 gekennzeichnete Fläche der Benutzungsoberfläche, stellt die Unterschiede der Modelle in Baumform textuell dar. Hier werden Attributänderungen mit dem jeweils alten und neuen Wert, Referenzänderungen mit dem alten und neuen Ziel der Referenz und das Löschen und Hinzufügen von Elementen aufgelistet. der Baum verdeutlicht dabei die Position des Elements, welches geändert wurde. Wenn man in dieser Darstellung der Differenz eine Änderung per Mausklick auswählt, wird diese in der unteren Arbeitsfläche (2) durch verschiedenfarbige Linien visualisiert. Die Linienfarben richten sich nach der Art der Änderung und ermöglichen somit einen einfacheren Überblick über die Differenz der Modelle. Beispielsweise werden gelöschte Elemente rot, hinzugefügte Modellelemente grün und geänderte Modellelemente blau dargestellt. Weiter kann sich der Nutzer die Eigenschaften zu jedem der Modellelemente anzeigen lassen, um sich einen genauen Überblick über die Elemente zu verschaffen. Dies ist durch einen Klick auf den Tab mit dem Namen Properties am unteren Bildrand mög-

[42] Die Eclipse IDE ist eine erweiterbare Open-Source Entwicklungs-Plattform, welche von der Eclipse-Open-Source Gemeinschaft entwickelt wurde und stetig weiterentwickelt wird.[Eclipse]

[43] Obeo [Obeo]

[44] Intalio [Intalio]

[45] Vgl. [EMFT]

[46] Extension Points stellen eine Möglichkeit dar, Eclipse oder Eclipse-Plugins zu erweitern. Eine genauere Beschreibung und Dokumentation der Eclipse Architektur findet sich in [DFK+04] oder direkt auf der Eclipse-Webseite [Eclipse]

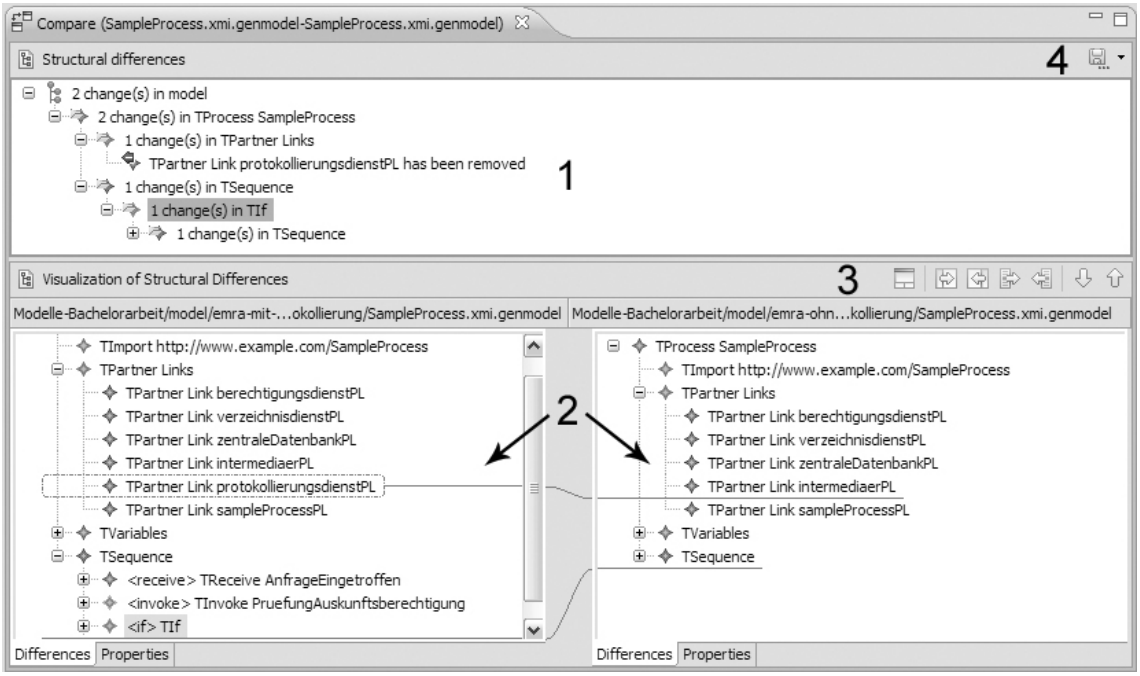

Abb. 13: Benutzungsoberfläche – EMF Compare

lich. Die graphische Darstellung der Modell-Differenzen unterteilt sich in die Darstellung beider Modelle in Baumnotation und eine Zeichenfläche, auf welcher die Differenzen angezeigt werden. Einige weiterführende Funktionen werden durch die rechts von Nummer 3 befindlichen Buttons bereitgestellt. Hier ist es beispielsweise möglich, die ermittelten Differenzen entweder komplett oder einzeln in das jeweils gegenüberliegende Modell zu übertragen[47]. Abschließend soll noch die Möglichkeit erwähnt werden, die ermittelte Differenz als eigenständiges EMF-Modell zu serialisieren. Dies ist mit dem Button rechts von Nummer 4 möglich und erzeugt ein Modell mit der Dateinamenserweiterung *.emfdiff*. Dieses Modell enthält sowohl die Differenz der Modelle, als auch das zuvor berechnete Mapping mit den Vergleichswerten der Modellelemente.

4.2.2 Architektur und Vergleichsansatz

Nachdem zuvor eine allgemeine Einführung in dieses Werkzeug erfolgt ist und die Benutzungsoberfläche des Werkzeugs erläutert wurde, wird im Folgenden die Architektur und der von diesem Werkzeug verwendete Vergleichsansatz näher betrachtet.

Die Berechnung von Modell-Differenzen erfolgt in EMF Compare in zwei Phasen. Die erste dieser Phasen ist das Matching der Modelle, bei welchem die Vergleiche der Modellelemente durchgeführt werden. In dieser Phase wird ebenfalls das Mapping der Modellemente umgesetzt, in welchem die Modellelemente der beiden Modelle aufeinander abgebildet werden. Am Ende der ersten Phase steht das hier als Match-Modell bezeichnete Modell. Dieses wird später bei der zuvor bereits angesprochenen Serialisierung der Modell-Differenz zusätzlich gespeichert. In der zweiten und letzten Phase wird die Diffe-

[47] Diese Funktion implementiert ein Merging der Differenzen mit den Modellen.

renz beider Modelle berechnet. Während dieser Phase werden die gelöschten oder hinzugefügten Elemente, sowie die Änderungen der aufeinander abgebildeten Modellelemente identifiziert. Am Ende dieser Phase steht die hier als Diff-Modell bezeichnete Modell-Differenz. Der zuvor beschriebene Ablauf der Differenzberechnung soll durch Abbildung 14 noch einmal graphisch verdeutlicht werden. Nachfolgend wird die Matching- und Mapping-Phase genauer betrachtet.

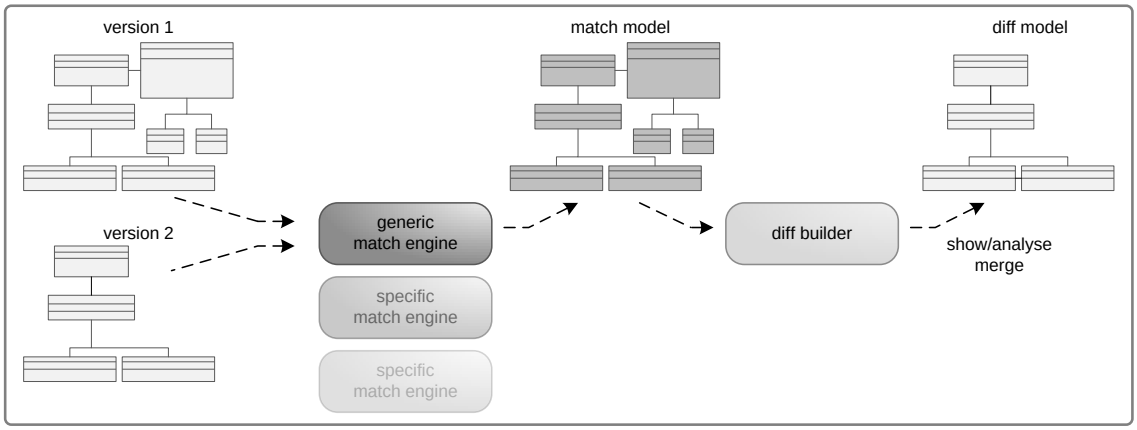

Abb. 14: Architektur von EMF Compare (nach [EMFC])

In [Toul06] hat Antoine Toulmé bereits betont, dass die Verwendung von UUIDs zur Identifikation von Modellelementen, wie dies z. B. in [AlPo03] von Alanen und Porres vorausgesetzt wurde, häufig nicht der Praxis entspricht, ja sogar oft nicht umsetzbar ist. Aus diesem Grund wurde bei der Entwicklung von EMF Compare ein eigener Vergleichsalgorithmus implementiert, welcher sämtliche Modellelemente miteinander vergleicht, um eine Abbildung der Elemente aufeinander zu finden. Toulmé erläutert diesen Vergleich der Modelle und geht dabei beispielsweise auch auf die verschiedenen Anforderungen an den Vergleich von Attributen und Referenzen ein. In [Toul06] werden z. B. die Unterschiede zwischen Attributen als Werte zwischen 0 und 1 berechnet. Dazu werden sämtliche Attributwerte wie String-Werte behandelt und unter Zuhilfenahme von Levenshteins Editierdistanz[48] verglichen. Das Ergebnis des Vergleichs beider Strings wird anschließend auf 1 normiert. Laut Toulmé sollten Zahlenwerte ebenfalls als String-Werte behandelt werden, da beispielsweise ein Vergleich von Postleitzahlen und Raumnummern als Integer-Werte keine Aussagekraft besitzt. Ein weiterer Punkt, welcher den Algorithmus auszeichnet, ist die Modellabhängige Gewichtung von Attributen. Die Gewichtung der Attribute hängt hier von der Anzahl des Vorkommens und deren Informationsgehalt ab. Der Vergleichs- und Matching-Algorithmus ist in [Toul06] zur Verdeutlichung als Quellcode-Ausschnitt gelistet. Die Umsetzung des Vergleichs in EMF Compare verläuft im Detail wie folgt[49]:

Die Entwickler von EMF Compare verwenden standardmäßig vier Vergleichsmetriken, um die Elemente beider Modelle miteinander zu vergleichen und anschließend abzubilden. Die erste Metrik ist der Test der Elemente auf Typgleichheit. Hier wird geprüft, ob die Modellelemente Instanzen desselben Metamodellelements sind. Ist dies der Fall, wird eine Übereinstimmung zurückgegeben. Danach wird der Namenstest der Elemente

[48] Vgl. [Leve65]
[49] Vgl.: [EMFC]

durchgeführt. Dieser Test ist besonders kritisch, da hier nicht vorausgesetzt werden kann, dass jedes Modellelement ein Name-Attribut besitzt. In diesem Werkzeug wird bei dem Namenstest nicht nur nach einem Name-Attribut gesucht, sondern nach einem Attribut, welches eine gute Chance hat, der Name des Modellelements zu sein. Dieses Attribut kann durchaus auch eine andere Bezeichnung besitzen. Wenn ein solches Attribut gefunden wurde, werden beide Elemente anhand dieses Attributes verglichen. Wie bereits aus [Toul06] hervorgeht, wird dabei Levenshteins Editierdistanz verwendet und der Ergebniswert auf 1 normiert, sodass aus diesem Vergleich ein Wert zwischen 0 und 1 resultiert. Als nächstes folgt ein Vergleich sämtlicher Attribute beider zu vergleichender Elemente. Wie bereits zuvor angesprochen erhalten die Attribute verschiedene Gewichtungen, abhängig davon, wie oft sie auftreten oder welche Informationen sie tragen können. Diese Informationen werden aus einer Analyse der Modelle abgeleitet. Abschließend folgt der Vergleich der Relationen. Hier werden Abhängigkeiten gesucht und verglichen, welche ein Element mit anderen besitzt. Diese Abhängigkeiten werden in EMF typischerweise durch EReferences, also Referenzen, oder durch Containment-Beziehungen, also durch Kindelemente umgesetzt. Die durch diese Vergleiche ermittelten Vergleichswerte werden anschließend zusammengefasst und auf einen Wert von 1.0 normiert.

Die hier verwendeten Vergleichskriterien sind bei Verwendung der Funktionalität des Werkzeugs über die angebotenen Programmierschnittstellen durch den Nutzer erweiterbar und können somit auf die spezifischen Bedürfnisse der Nutzer zugeschnitten werden. Der Nutzer hat beispielsweise die Möglichkeit, bestimmte Attribute beim Vergleich stärker oder schwächer zu gewichten, oder sogar ganz und gar vom Vergleich auszuschließen. Weiter ist es möglich, einen selbst definierten Vergleich einzubinden.

4.2.3 Differenzberechnung

Die hier erläuterte zweite Phase erzeugt das Diff-Modell. Wie bereits angesprochen werden hier die Elemente als gelöscht bzw. hinzugefügt gekennzeichnet, welche im Match-Modell nicht auf andere Elemente abgebildet werden konnten. Ob ein Element als gelöscht oder hinzugefügt in die Differenz übernommen wird, hängt davon ab, in welchem der beiden Modelle sich das Element befindet. Falls das Element im linken Modell vorhanden ist, im rechten jedoch nicht, wird es als gelöscht gekennzeichnet. Analog werden Elemente aus dem rechten Modell, welche jedoch nicht im linken Modell vorhanden sind, als hinzugefügt angesehen. Weiter werden alle im Match-Modell aufeinander abgebildeten Elemente durchlaufen und die Änderungen der Modellelemente gegenüber ihrem Partnerelement ermittelt. Dabei wird das rechte Modell als neue Version und das linke Modell als ursprüngliche Version betrachtet. Dies wurde ebenfalls zuvor bei der Ermittlung der gelöschten und hinzugefügten Elemente vorausgesetzt. Die Änderungen der aufeinander abgebildeten Elemente werden als Instanzen der jeweiligen Modellelemente des Diff-Metamodells erzeugt und zeigen u.a. geänderte Attributwerte oder geänderte Referenzziele an. In Abbildung 15 ist die Modell-Differenz nach der Serialisierung, zusammengesetzt aus dem Diff-Modell und dem Match-Modell dargestellt.

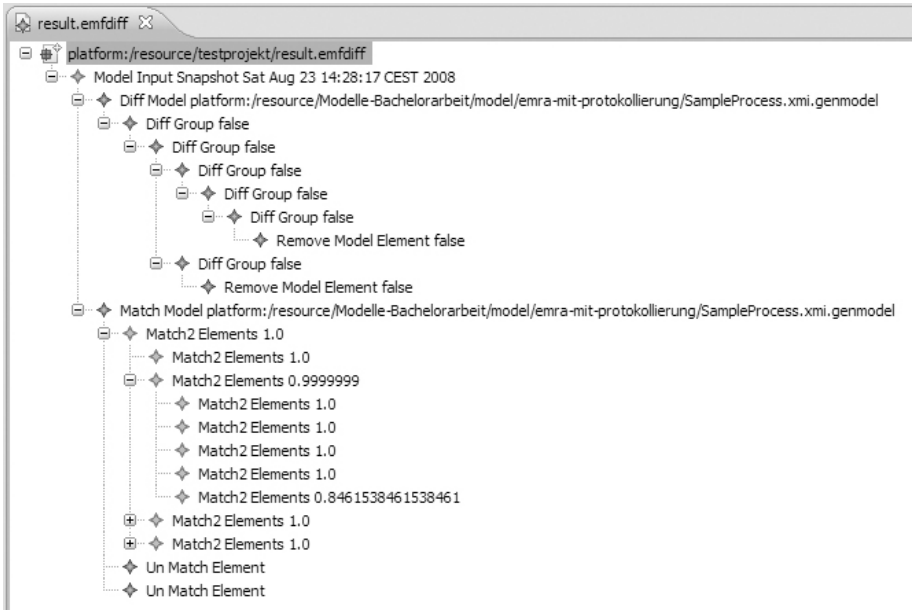

Abb. 15: Serialisierte Modell-Differenz – EMF Compare

4.3 ModelMatcher

4.3.1 Einführung und Benutzungsoberfläche

Der ModelMatcher wird am Lehrstuhl für Betriebliche Informationssysteme der Universität Leipzig entwickelt und befindet sich aktuell noch in der Entwicklungs- bzw. Testphase. Bei der Entwicklung dieses Werkzeugs wurden Konzepte des zuvor genannten Tools EMF Compare, die Ergebnisse umfangreicher Recherchen sowie eigene Ansätze verwendet. Mit der Entwicklung dieses Werkzeugs wird das Ziel verfolgt, einen eigenen, performanten und konfigurierbaren Vergleich zwischen Modellen des Eclipse Modeling Framework zu implementieren. Die Ergebnisse dieses Vergleichs stehen dann zur Umsetzung von Modelloperatoren wie der Modell-Differenz, der Vereinigung von Modellen, oder auch dem Merge von Modellen zur Verfügung. Um einen ersten Einblick in den ModelMatcher zu erlangen ist in den Abbildungen 16 und 17 die Benutzungsoberfläche dargestellt, welche im Folgenden kurz erläutert wird.

Der ModelMatcher wird als Eclipse-Plugin entwickelt und arbeitet auf dem Eclipse Modeling Framework. Dabei ist das Tool sowohl über die graphische Benutzerschnittstelle[50] als auch durch Zugriff auf Quellcode-Ebene verwendbar. Die eingebaute Benutzungsoberfläche ist als Multipage-Editor umgesetzt. Diese Art von Editoren in der Eclipse IDE bieten eine Unterstüzung für mehrere Seiten, auf welche die GUI verteilt werden kann[51]. Die hier wichtigen Funktionen des ModelMatcher sind auf zwei Seiten des Editors untergebracht, welche in den obigen Abbildungen 16 und 17 dargestellt sind. Die erste Abbildung zeigt die visuelle Darstellung der Vergleichsergebnisse beider Modelle. Hier sind im Wesentlichen die vier markierten Bestandteile (1-4) von Interesse:

[50] Graphical User Interface (GUI)

[51] Eine detailliertere Beschreibung des Aufbaus der Eclipse IDE gibt beispielsweise [DFK+04].

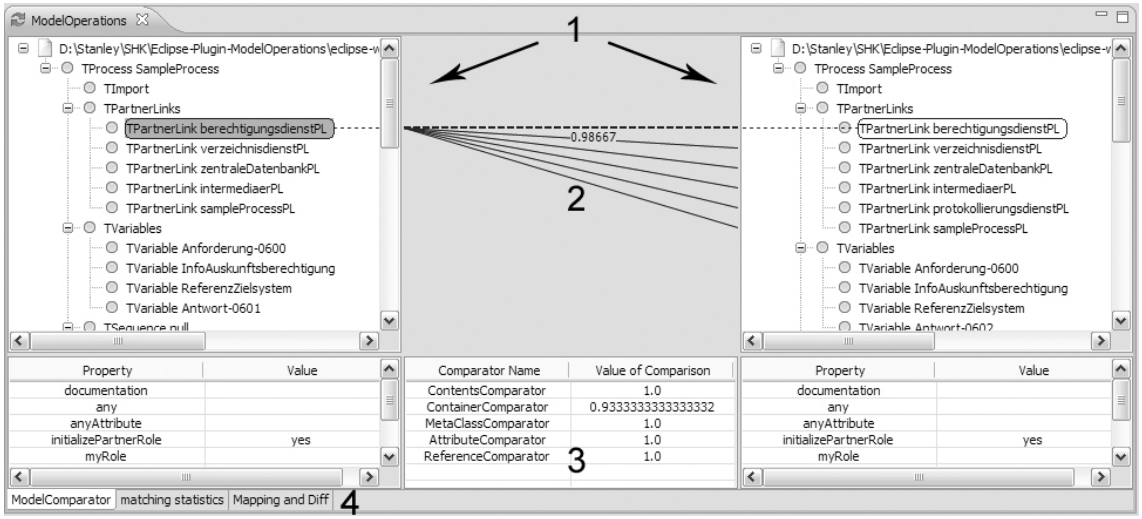

Abb. 16: Benutzungsoberfläche des ModelMatcher (Vergleichspanel)

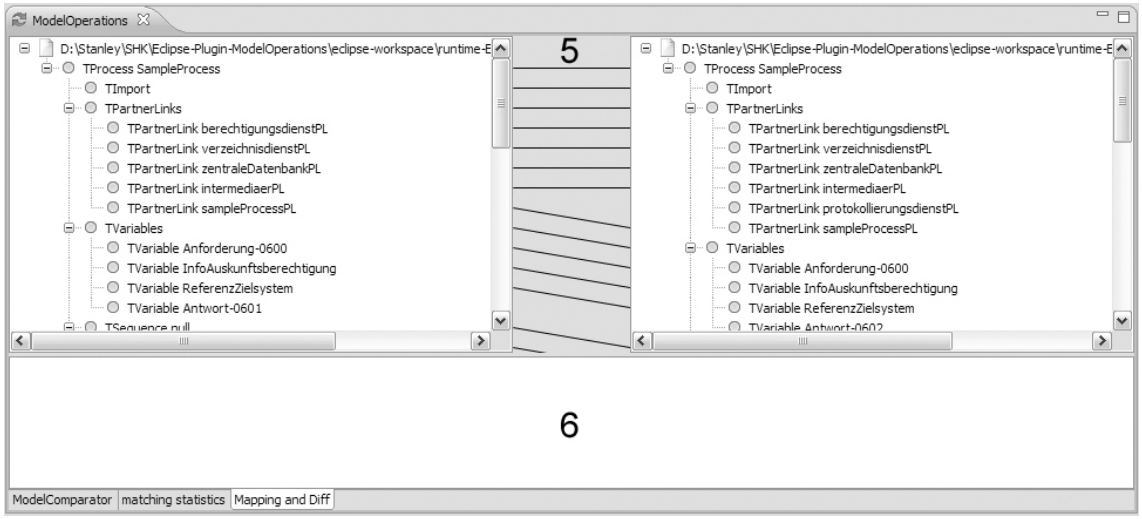

Abb. 17: Benutzungsoberfläche des ModelMatcher (Mapping und Diff)

1 Diese Nummer markiert die beiden Modellbäume. Sie dienen der Darstellung der verglichenen Modelle in Baumform. Unter jedem dieser Bäume befindet sich eine Tabelle, welche die Attribute und Referenzen zu dem aktuell gewählten Modellelement auflistet.

2 Zwischen beiden Modellbäumen befindet sich eine Zeichenfläche, welche die Vergleichswerte der Elemente durch Linien visuell darstellt. Die Vergleichslinien zwischen den Elementen werden gezeichnet, sobald in einem der beiden Modellbäume ein Element ausgewählt wird. Hier ist es möglich, eine der gezeichneten Linien zu selektieren. Daraufhin wird der Vergleichswert beider Elemente auf der Linie angezeigt.

3 Falls eine der oben beschriebenen Vergleichslinien gewählt wird, werden in der mit Nummer 3 gekennzeichneten Tabelle die einzelnen Vergleichswerte, aus welchen

sich der finale Vergleichswert zusammensetzt, für das gewählte Elementpaar dargestellt. Die in Abbildung 16 dargestellten Einträge repräsentieren die Standard-Vergleichskriterien, welche jedoch durch den Nutzer erweiterbar sind.

4 Diese Nummer markiert die Tabs, welche dazu dienen, zwischen den einzelnen Seiten des Editors zu wechseln.

Die nächste Seite, welche hier von Interesse ist, wird in Abbildung 17 dargestellt. Diese teilt sich im Wesentlichen in zwei Abschnitte auf:

5 Diese Zeichenfläche, welche sich zwischen den beiden Modellbäumen befindet, dient der Darstellung des Mappings zwischen beiden Modellen. Korrespondierende Elemente werden auf der Zeichenfläche durch Linien verbunden. Hier soll zukünftig noch die Möglichkeit bestehen, das Mapping der Modelle per drag and drop manuell zu beeinflussen.

6 Die Funktionalität der unteren Fläche ist bisher nicht implementiert. Hier wird künftig die Differenz der Modelle in Baumform dargestellt. Des Weiteren werden in diesem Bereich weitere Funktionen, wie das Serialisieren der Differenz oder ein Merging der Modelle angeboten.

Der Nutzer hat die Möglichkeit, die graphische Darstellung der Vergleichswerte, wie z. B. den Schwellenwert, ab welchem die Vergleichslinien dargestellt werden sollen, zu beeinflussen. Weiterhin ist es möglich, den Vergleich der Modelle an die Bedürfnisse des Nutzers anzupassen. Bei Nutzung der graphischen Oberfläche ist dies während des Ladens der Modelle für den Vergleich möglich. In diesem Dialog kann der Nutzer spezifische Attribute angeben, für welche er einen separaten Vergleich wünscht oder auch die Gewichtung der einzelnen Vergleichsoperationen regulieren. Im Detail wird dies im nachfolgenden Abschnitt während der Erläuterung der Architektur des ModelMatcher beschrieben.

4.3.2 Architektur und Vergleichsansatz

Wie bereits erwähnt, wird in diesem Abschnitt der Arbeit die Architektur des ModelMatcher näher betrachtet. Des Weiteren werden hier die Vergleichskriterien erläutert, welche in diesem Werkzeug verwendet werden. Abbildung 18 stellt die Architektur des Model Matcher dar. Diese wird im Folgenden erläutert.

Der ModelMatcher vergleicht zwei Modelle des Eclipse Modeling Framework. Diese werden serialisiert, beispielsweise in Form von *.ecore* oder *.xmi* Dateien eingelesen. Nachdem das Einlesen der Elemente in die intern verwendete Datenstruktur abgeschlossen ist, wird ein Kreuzprodukt beider Modelle in Form einer Matrix erstellt. Diese Matrix wird gebildet, um für jedes Element des linken Modells jeweils einen Vergleichswert für jedes Element des rechten Modells zu verwalten[52]. Nachdem diese Matrix erstellt wurde, wird der erste Comparator[53] mit dem Namen *MetaClass-Comparator* auf die Modelle, bzw. die einzelnen Paare von Modellelementen angewendet. Der *MetaClass-Comparator*

[52] Die hier angesprochene Matrix dient zur Speicherung des endgültigen Vergleichswertes der Elementtupel. Im Verlauf des Vergleichs wird für jede Vergleichsoperation eine separate Vergleichsmatrix erstellt.

[53] Eine hier als Comparator bezeichnete Klasse kapselt jeweils ein Vergleichskriterium, welches auf zwei Elemente angewendet werden soll.

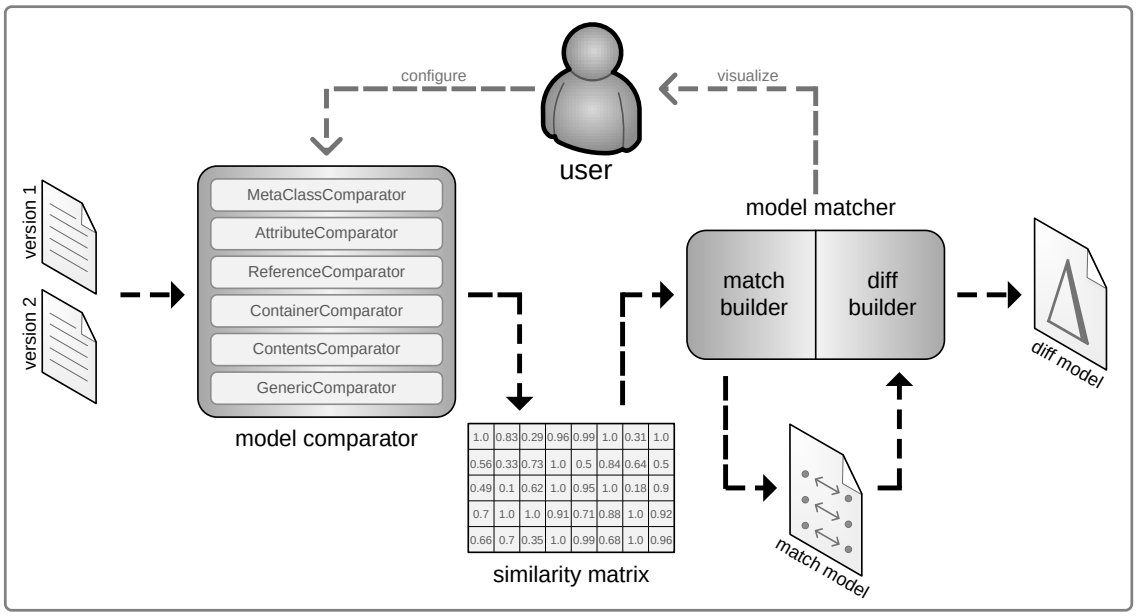

Abb. 18: Architektur des ModelMatcher

ermittelt den Metatyp der Modellelemente, also das Metamodellelement, welches die Modellelemente instanziieren. Sind beide Metatypen gleich, wird ein Wert von 1.0 zurückgegeben. Der Nutzer hat auch hier im Vorfeld die Wahl, ein komplettes Kreuzprodukt beider Modelle für den Vergleich zu verwenden oder die in Abschnitt 3.2 angesprochene Einschränkung der Gleichheit der Metatypen zu berücksichtigen. Falls letztere Option gewählt wurde, werden nach dem ersten Vergleich durch den *MetaClass-Comparator* diejenigen Elementpaare aus der Matrix entfernt, für welche 0.0 in der Matrix gespeichert ist. Nachdem der erste Vergleich beendet ist und die Matrix eventuell von störenden Elementtupeln bereinigt wurde, finden vorerst vier weitere Standardvergleiche statt. Diese sind nachfolgend in der durchlaufenen Reihenfolge aufgelistet und erklärt.

- ***Attribute-Comparator*** Hier werden sämtliche Attribute beider Elemente miteinander verglichen. Dabei wird zu jedem Attribut sowohl die Existenz dieses Attributs im gegenüberliegenden Element als auch deren Wertegleichheit berücksichtigt. Für den Vergleich von Attributwerten wurde im Gegensatz zu dem bei EMF Compare verwendeten Ansatz bewusst auf Levenshteins Editierdistanz verzichtet. Attributwerte werden mithilfe der Operatoren verglichen, welche von deren Datentyp bereitgestellt werden. Dies liegt darin begründet, dass es beispielsweise als fraglich angesehen wird, ob eine durch die Editierdistanz berechnete 66%ige Gleichheit der beiden Zahlen 312 und 112 gerechtfertigt ist. Falls diese beiden Zahlen beispielsweise Raumnummern von Hotelzimmern darstellen, stehen sie in keinerlei relevantem Bezug zueinander und der Vergleich beider Nummern muss 0.0 als Resultat liefern.
- ***Reference-Comparator*** Dieser Comparator vergleicht sämtliche Referenzen, welche die Modellelemente besitzen. Referenzen sind beispielsweise die Verweise auf mögliche Superklassen, von denen das aktuelle Element erbt. Referenzen werden im

Reference-Comparator verglichen, indem die vorherigen Vergleichswerte für das Tupel ausgewertet werden, welches aus den referenzierten Elementen besteht. Im Falle von Referenzlisten werden sämtliche Kombinationen von Elementtupeln ausgewertet und die Längen der Referenzlisten berücksichtigt.

- ***Container-Comparator*** Durch die Baumnotation, in welcher EMF-Modelle hier verarbeitet werden, besitzt jedes Element, welches nicht das Wurzelelement ist, ein Vaterelement. Dieses Vaterelement ist der Container des zu vergleichenden Modellelements. Während des aktuellen Vergleichs wird die Übereinstimmung der beiden Container unter Zuhilfenahme der Vergleichswerte aus den vorhergehenden Vergleichen bewertet. Zurückgegeben wird auch hier wieder ein auf 1.0 normierter Vergleichswert.
- ***Contents-Comparator*** Abschließend werden von jedem der Elemente des Tupels die Kindelemente miteinander verglichen. Hier wird sowohl die Anzahl der Kinder im Baum, als auch deren Gleichheit berücksichtigt. Die Gleichheit der Kinder wird durch Analyse der zuvor durchlaufenen Vergleiche geprüft. Zwei Elemente sind bezüglich ihrer Kinder gleich, falls der *Contents-Comparator* für sie einen Vergleichswert von 1.0 liefert.

Nachdem die Standardvergleiche abgeschlossen sind, ist es möglich einige weitere Vergleiche von Attributen oder Referenzen durchzuführen. Dazu gibt es die Möglichkeit, dass der Nutzer die Namen der zu vergleichenden Attribute oder Referenzen beim Laden der Modelle explizit angibt. Dies wurde bereits bei der Erläuterung der Benutzungsoberfläche angesprochen. Daraufhin wird für jedes angegebene Attribut, bzw. jede Referenz ein *Generic-Comparator* instanziiert. Die Modelle werden wie zuvor nun mit jedem dieser *Generic-Comparator* verglichen.

Sobald der Vergleich der Modelle vollständig abgeschlossen ist, wird die zu Beginn erstellte Matrix, welche in Abbildung 18 als Similarity Matrix bezeichnet wird, mit den Vergleichswerten vervollständigt. Dabei werden die Werte sämtlicher Comparator addiert und durch deren Anzahl[54] dividiert, um eine Normierung der Vergleichswerte auf 1.0 zu erreichen. Die nun vollständig gefüllte Matrix wird im weiteren Verlauf als Input für den in Abbildung 18 als Model Matcher bezeichneten Teil des Werkzeugs verwendet. Dieser Teil der Berechnung unterteilt sich in die Ermittlung des Mappings der Modellelemente und die Erstellung der Modell-Differenz.

4.3.3 Mapping und Differenzberechnung

In dem hier erläuterten Teil des Werkzeugs wird das Mapping der Modelle und darauf aufbauend die Differenz beider Modelle ermittelt. Beim Mapping der Modelle werden zunächst sämtliche Tupel der zuvor erstellten Ähnlichkeitsmatrix aufeinander abgebildet, zu denen der Wert 1.0 verzeichnet ist. Diese Elemente können ohne Probleme aufeinander abgebildet werden, da zwischen ihnen keinerlei Unterschiede festgestellt wurden. Danach werden die restlichen Elemente des Mappings verarbeitet. Dies geschieht durch den Algorithmus aus Listing 9. Wie bereits in Abschnitt 3.3 erläutert, werden hier die höchsten

[54] Die Anzahl ist abhängig von der Gewichtung der einzelnen Vergleichskriterien. Beispielsweise wird bei einer 50% igen Gewichtung des *Attribute-Comparator* für die Gesamtzahl der Vergleichskriterien um 0.5 reduziert.

Vergleichswerte der Modellelemente ermittelt. Daraufhin wird der höchste Vergleichswert des zweiten Elements des daraus resultierenden Elementtupels gesucht. Falls die beiden ermittelten Tupel identisch sind, wird eine Abbildung beider Elemente der Tupel aufeinander vorgenommen. Falls diese Tupel nicht identisch sind, wird das erste Element des Tupels als einzelnes, nicht abgebildetes Element in das Mapping aufgenommen.

Wenn das Mapping der Modelle abgeschlossen ist, wurde das Mapping-Modell im Speicher erstellt. Dieses wird für die Ermittlung der Modell-Differenz verwendet, bei welcher alle Elemente, welche im Mapping als nicht abgebildet verzeichnet sind, in der Differenz als neu bzw. gelöscht gekennzeichnet werden. Alle anderen Modellelemente, zu welchen ein Mapping-Partner existiert, werden einzeln auf Änderungen geprüft. Diese Änderungen werden in das Diff-Modell aufgenommen. Das so entstandene Diff-Modell kann zusammen mit dem Mapping-Modell serialisiert werden. Solch ein Modell ist in Abbildung 19 dargestellt.

Im Vergleich zu EMF Compare benötigt der ModelMatcher für die Verarbeitung großer Modelle etwas mehr Zeit. Dies ist durch die umfangreicheren Vergleiche und einen noch nicht ganz ausgereiften Mapping-Mechanismus begründet. Jedoch besteht hier die Möglichkeit für den Nutzer, die Vergleichskriterien mit unterschiedlichen Gewichtungen zu versehen. Somit ist der Vergleich der Modelle nicht auf die durch Modellanalyse errechneten Gewichtungen der Attribute angewiesen und es können je nach Konfiguration und Einsatzzweck präzisere Ergebnisse erzielt werden.

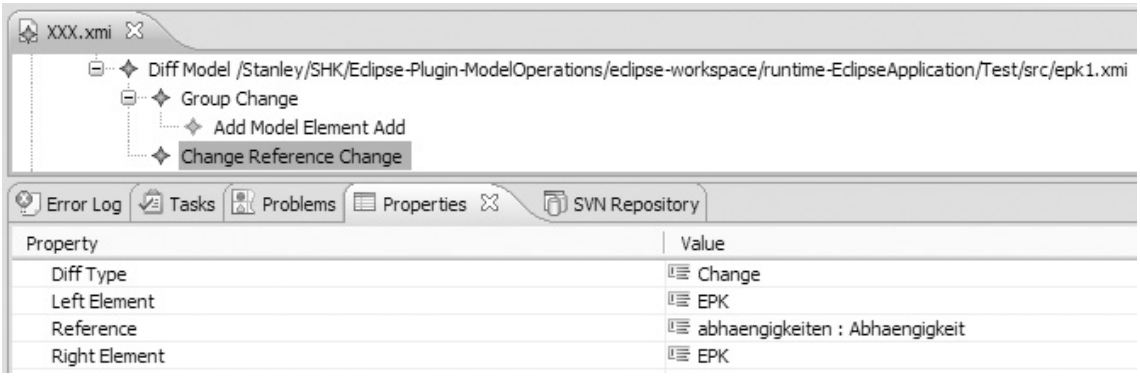

Abb. 19: Serialisierte Modell-Differenz – ModelMatcher

4.4 SiDiff

Das Werkzeug, welches hier vorgestellt wird, nennt sich SiDiff und ist ein Projekt der Universität Siegen. Im Gegensatz zu den anderen in dieser Arbeit betrachteten Werkzeugen ist SiDiff in der Lage, die Ermittlung von Modell-Differenzen auf graphischen Modellen durchzuführen. Dabei werden verschiedene UML-Diagramme[55], Matlab/Simulink-Diagramme[56] oder auch Stoffwechselreaktionsketten[57] unterstützt. Alle drei zuvor ge-

[55] wie z. B. Klassendiagramme oder auch Zustandsautomaten

[56] Simulink, vom Hersteller MathWorks ist eine Plattform für die Mehrdomänensimulation und das Model-Based Design dynamischer Systeme. Simulink wird für die Entwicklung eingebetteter Software verwendet. [Math]

[57] Stoffwechselreaktionsketten werden genutzt, um chemische Prozesse der Bioinformatik zu modellieren.

nannten Modellarten haben die Gemeinsamkeit, dass sie als Graphen dargestellt werden können, jedoch haben die Modelle auch viele mehr oder weniger offensichtliche Unterschiede, wie beispielsweise die verwendeten Elementarten (Klassen in Klassendiagrammen, Zustände in Zustandsdiagrammen usw.). Trotz dieser Unterschiede sind in den Modellen ausreichend Gemeinsamkeiten vorhanden, die es den Entwicklern von SiDiff ermöglichen einen generischen, hochgradig konfigurierbaren Vergleichsalgorithmus zu entwickeln, mit dessen Hilfe beispielsweise Differenzen solcher Modelle berechnet werden können. Der Vergleichsalgorithmus muss dabei für jeden Modelltyp speziell konfiguriert werden. Hier ist es beispielsweise erforderlich die Modellelemente dem Algorithmus „bekannt zu machen“ . Eine solche Konfiguration für SiDiff besteht unter anderem aus einer Transformation der Originalmodelle in eine intern verwendete Struktur, einer Definition der Ähnlichkeiten von Elementen und der Spezifikation des Ausgabeformats der Differenz. Die Konfigurierbarkeit und somit die Anwendung des Vergleichsalgorithmus auf die verschiedenen Modelltypen wird dadurch ermöglicht, dass diese Modelle im XML Metadata Interchange (XMI)-Format serialisiert sind. Dadurch lässt sich die Transformation der Modelle in die intern verwendete Struktur beispielsweise durch eine Extensible Stylesheet Language Transformation (XSLT) realisieren. Die durch die Transformation entstandenen Modelle im XML-Format können nun unter Nutzung des Algorithmus und der weiteren Konfiguration Verglichen werden. Abschließend wird die Differenz oder beispielsweise auch die Vereinigung der Modelle erstellt und kann ebenfalls als Modell serialisiert werden. Eine automatische Weiterverarbeitung, beispielsweise zum Merge der Modelle ist ebenfalls möglich. Die Architektur dieses Werkzeugs ist in Abbildung 20 dargestellt.

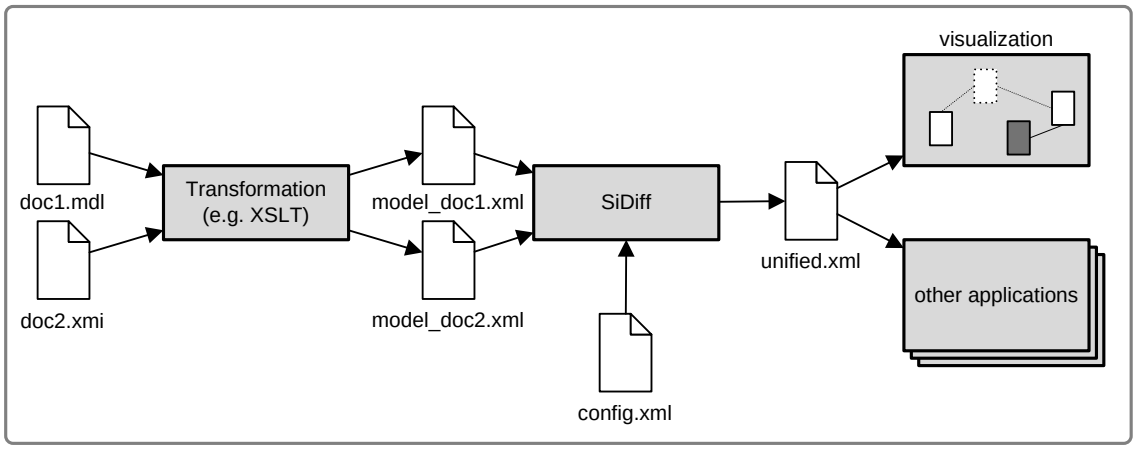

Abb. 20: Architektur – SiDiff (nach [SiDiff])

Der SiDiff-Vergleichsalgorithmus verlässt sich ebenso wie die beiden bisher genannten Werkzeuge nicht auf die Existenz von UUIDs, welche ein Element eindeutig identifizieren. In diesem Werkzeug werden ebenfalls die Unterschiede und somit auch das Mapping der Modellelemente berechnet. Dabei verwendet SiDiff die oben genannte flexible Ähnlichkeits definition aus der Konfiguration und einen Vergleichsalgorithmus, welcher die Änderungen von Knoten, Attributen, Referenzen und Bewegungen der Modellelemente ermittelt und dessen Schwellenwerte für die Ähnlichkeitsdefinitionen flexibel einstellbar sind. Eine weitere nützliche Eigenschaft ist das einfach zu ändernde Metamodell, welches

z. B. aus Knoten mit Attributen oder auch aus einem Baum mit Graphkanten bestehen kann, da hier verschiedene Modellarten verarbeitet werden können. Bildschirmausschnitte eines berechneten Vereinigungsmodells zweier Klassendiagramme bzw. der Differenz zweier Simulink-Diagramme sind in den Abbildungen 44 und 47 im Anhang dieser Arbeit dargestellt.

Abschließend soll zu diesem Abschnitt noch gesagt werden, dass SiDiff im eigentlichen Sinn kein eigenständiges Werkzeug ist. SiDiff ist vielmehr ein Algorithmus, welcher sich in verschiedene Plattformen integrieren oder auch als eigenständiges Werkzeug realisieren lässt. So ist SiDiff beispielsweise als Erweiterung für MATLAB[58] verfügbar oder auch als Erweiterung für FUJABA[59]. Des Weiteren ist auf der SiDiff-Webseite [SiDiff] ein WebService nutzbar, mit dem sich einige Modelltypen vergleichen lassen. Trotz des innovativen Ansatzes, graphische Modelle zu vergleichen und der Fähigkeit, verschiedenartige Modelltypen durch vorherige Konfiguration des Systems zu vergleichen, ist eine Verwendung von SiDiff in den Tests dieser Arbeit ist nicht möglich. Es ist beispielsweise nicht möglich, EPK- oder BPEL-Modelle zu vergleichen und die Formate des Eclipse Modeling Framework können nicht verarbeitet werden.

4.5 COMA++

Das Werkzeug COMA++ ist ein Projekt der Abteilung Datenbanken der Universität Leipzig unter Leitung von Prof. Dr. Erhard Rahm[60]. Ziel dieses Werkzeugs ist es, Schema- und Ontologie-Matchings zu berechnen. Da COMA++ auf den Metadaten arbeitet und nicht dazu bestimmt ist Modelle zu vergleichen, kann es in den nachfolgenden Tests ebenfalls nicht zur Berechnung der Modell-Differenzen verwendet werden. Der Grund warum dieses Werkzeug hier dennoch erwähnt wird liegt darin, dass es ein in Fachkreisen bekanntes und angesehenes Werkzeug ist und darüber hinaus einen etwas anderen Ansatz des Vergleichs verwendet als die anderen hier genannten Werkzeuge. COMA++ ist unter anderem in der Lage, semantische Übereinstimmungen zwischen Datenbankschemata, XML Nachrichtenformaten und Ontologien zu berechnen. Dabei bietet COMA++ neben anderen Formaten hauptsächlich eine Unterstützung der Formate XML Schema Definition (XSD[61]) und Web Ontology Language (OWL[62]) und ist in der Lage, selbst sehr große Matching-Probleme zu lösen. Um die Arbeitsweise von COMA++ zu verdeutlichen wird im Folgenden der Aufbau des Werkzeugs, sowie dessen Vergleichsansatz erläutert.

Die in Abbildung 21 dargestellte Architektur besteht aus fünf Hauptteilen. Diese fünf Teile sind nachfolgend aufgelistet und erklärt. Dabei ist es möglich, auf sämtliche Architekturbestandteile durch die graphische Nutzerschnittstelle zuzugreifen. Die COMA++-GUI ist in Abbildung 48 im Anhang dieser Arbeit dargestellt.

[58] MATLAB ist eine plattformunabhängige Software der Firma MathWorks [Math], welche zur Lösung mathematischer Probleme und der graphischen Darstellung der Lösungen genutzt wird.

[59] FUJABA ist eine von der Universität Paderborn entwickeltes CASE-Tool Suite. [FUJABA]

[60] Vgl. [COMA++]

[61] XSD ist ein Dateiformat zur Beschreibung von XML Schema [XSD]. XML Schema dient der Beschreibung der Struktur von XML Dokumenten und wurde von der W3C [W3C] eingeführt.

[62] OWL bezeichnet die Web Ontology Language, welche ebenfalls durch die W3C spezifiziert wurde [OWL]. OWL wurde entwickelt, um die Inhalte von Informationen automatisiert zu verarbeiten, statt die Informationen lediglich für den Nutzer darzustellen.

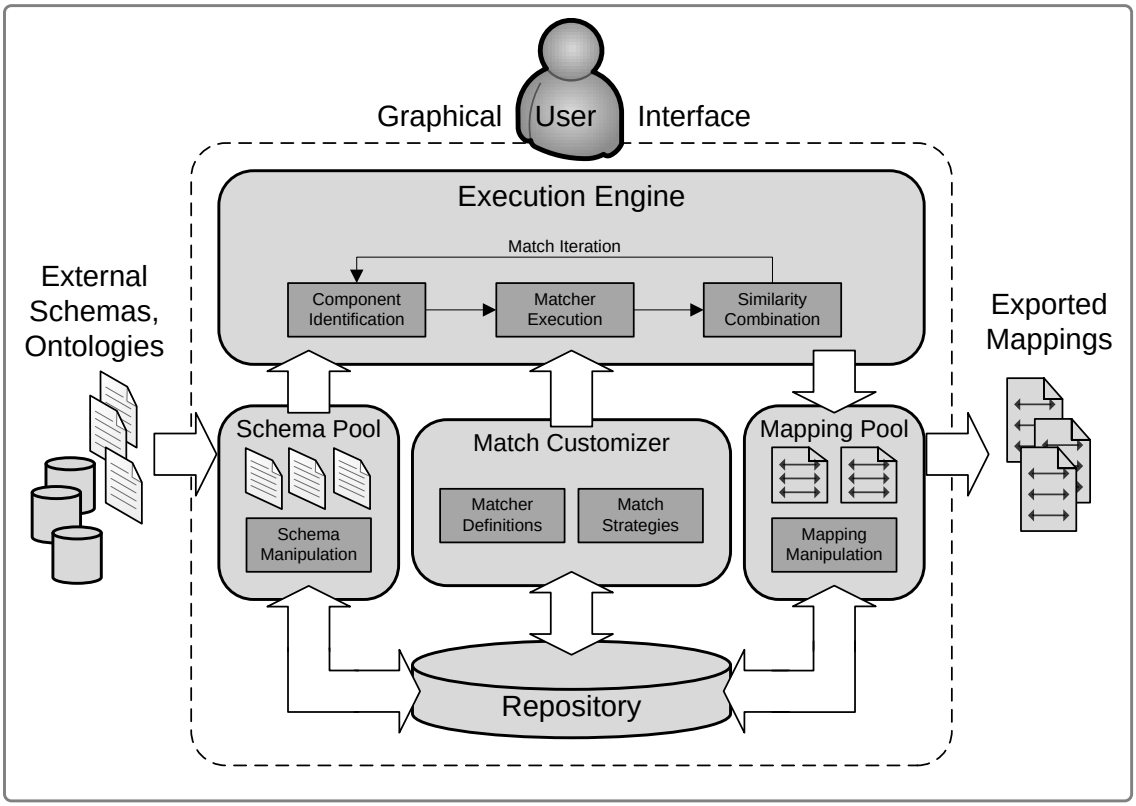

Abb. 21: Architektur – COMA++ (aus [COMA++])

1 **Repository:** In diesem Repository werden sämtliche für das Matching relevanten Daten verwaltet.

2 **Schema Pool:** Dieser dient dazu, alle verwendeten Schemata und Ontologien im Speicher zu verwalten.

3 **Mapping Pool:** Hier werden alle bereits berechneten Mappings von Schemata verwaltet.

4 **Match Customizer:** Dieser bietet dem Nutzer des Werkzeugs die Möglichkeit, Vergleichskriterien zu definieren oder auch eigene Vergleichsstrategien zu erstellen.

5 **Execution Engine:** Die Execution Engine führt das eigentliche Matching in einem iterativen Prozess aus. Dabei werden die Schemata und die im Match Customizer definierten Matcher als Input verwendet und ein Mapping der Schemata zurückgegeben. Jede Iteration der Execution Engine, also jedes Matching der Metamodelle besteht aus drei Teilschritten. Diese Phasen sind nachfolgend aufgelistet.

   1. **Component identification:** In dieser Phase werden die für das Matching und Mapping wichtigen Komponenten der Schemata identifiziert.
   2. **Matcher execution:** In diesem Abschnitt der Iteration werden die verschiedenen Vergleichskriterien (Matcher) auf die Komponenten angewendet, um die Ähnlichkeiten zwischen den Komponenten zu ermitteln.

3. **Similarity combination:** Die letzte Teilphase der Iteration dient dazu, die einzelnen, Matcher-spezifischen Ähnlichkeiten zu einem Wert zusammenzufassen, um mögliche korrespondierende Komponenten der Schemata zu finden.

Das in einer Iteration berechnete Mapping kann gegebenenfalls auch für die nächste Iteration als Eingabewert verwendet werden, um es weiter zu verfeinern. Des Weiteren ist es dem Nutzer möglich, jede Iteration spezifisch anzupassen, um bessere Vergleichsergebnisse zu erzielen. Hier kann mit den vom Match Customizer angebotenen Alternativen gearbeitet werden. So ist es beispielsweise möglich, die beim Vergleich zu berücksichtigenden Komponententypen anzugeben, die Matcher für jede Iteration spezifisch zu wählen oder auch die Strategien für die Similarity combination zu beeinflussen. Die drei zuvor angesprochenen Phasen jeder Iteration sind in Abbildung 22 noch einmal graphisch dargestellt und im Folgenden wird der Ablauf der Ermittlung des Mappings in COMA++ erläutert.

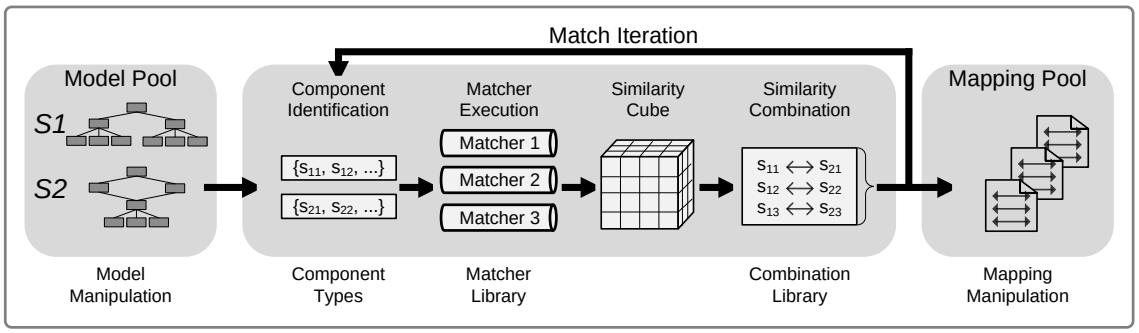

Abb. 22: COMA++ – Matching-Iteration (nach [COMA++])

Zu Beginn des Vergleichs ist es notwendig, die Modelle COMA++ als Eingabe zu übergeben, welche dann zur weiteren Verarbeitung einen Präprozessor durchlaufen, der die Modelle in ein intern verwendetes Datenformat überführt. Nachdem dies geschehen ist und der Nutzer die Vergleichskriterien usw. an seine Bedürfnisse angepasst hat, wird die erste Iteration des Matching gestartet. Dabei werden in der ersten Phase dieser Iteration die zu vergleichenden und aufeinander abzubildenden Komponenten der Modelle ermittelt. In Abbildung 22 sind die Komponenten als Listen für die beiden Eingabemodelle dargestellt. Komponenten können beispielsweise Knoten, Pfade, oder Fragmente der Modelle sein. Die nächste Phase der Iteration führt die zuvor gewählten Vergleiche auf den ermittelten Komponenten aus und überträgt die Vergleichsergebnisse in einen dafür geeigneten Datentyp, welcher hier als Similarity Cube bezeichnet wird. Als mögliche Vergleichskriterien sind hier beispielsweise die Namen oder auch die Namenspfade der Komponenten genannt. Als letzte Teilphase folgt die Similarity Combination, welche den Similarity Cube als Eingabe verwendet und anhand der dort verzeichneten Vergleichswerte unter Nutzung verschiedener Operationen, wie Aggregation oder auch Selektion die Ergebnisse zusammenfügt. Aus dieser Phase resultiert das Mapping der Modelle, welches letztendlich das Ergebnis einer Iteration darstellt. Dieses Mapping kann sowohl zu einer Differenz, als auch einem Durchschnitt oder anderen Artefakten verarbeitet und im Mapping Pool abgelegt werden. Des Weiteren dient das erstellte Mapping in der Regel als neuer Input für die nächste Matching-Iteration, bei welcher der Nutzer durch neue Konfiguration der Matcher Strategien für das Mapping eine weitere Verfeinerung erzielen kann. Die zuvor

genannten Matcher, sowie die Mapping-Strategien werden als Workflows beschrieben und in erweiterbaren Bibliotheken verwaltet. Auf diese Weise ist es möglich sehr umfangreiche Problemstellungen durch eine Zerlegung der Matching- und Mapping-Operationen in viele Teilschritte effektiv zu lösen.

Auch wenn sich das Werkzeug COMA++ dadurch von den zuvor angesprochenen Werkzeugen unterscheidet, dass die hier vorgenommenen Vergleiche und Abbildungen auf den Metamodellen und nicht den eigentlichen Modellen umgesetzt werden, wurde COMA++ dennoch aus zwei Gründen betrachtet. Einerseits sollte mit diesem Werkzeug eine etwas andere Anwendung von Modellvergleichen gezeigt werden und andererseits weichen sowohl die Architektur dieses Werkzeugs, als auch die Vergleichskriterien von den zuvor genannten Werkzeugen ab. Ein Einsatz dieser Vorgehensweise ist auch für den Vergleich auf Modellebene interessant, da beispielsweise aus der iterativen Ermittlung des finalen Mappings ein Performancevorteil resultieren kann.

5 Werkzeugtests

5.1 Testkriterien

In diesem letzten Kapitel der Arbeit wird die Arbeitsweise der beiden Werkzeuge EMF Compare und ModelMatcher an einigen Beispielmodellen getestet. Ausgangspunkt dieser Tests sind die aus der EPK-zu-BPEL-Transformation des Anwendungsfalls hervorgegangenen BPEL-Modelle. Wie in Abbildung 1 dargestellt ist, sind das die noch nicht manuell verfeinerten Modelle. Die Gründe für diese Vorgehensweise wurden bereits in den vorhergehenden Kapiteln dargelegt.

Um die Arbeitsweise der Werkzeuge ausreichend zu testen und gegebenenfalls einige Stärken bzw. Schwächen der Tools aufzudecken, sind neben der Differenzberechnung der in Abschnitt 1.1 vorgestellten Modelle einige weitere Testfälle nötig. Diese werden ausgehend von den im Anwendungsfall definierten Modellen erstellt. Dabei wird der Schwierigkeitsgrad der Änderungen in den Testfällen gesteigert. Anfangs werden lediglich Elemente hinzugefügt bzw. gelöscht, was eine sehr einfache Änderung der Modelle darstellt. Danach werden Elementeigenschaften geändert, bevor abschließend die Auswirkungen von Positionsänderungen von Modellelementen untersucht werden. Nachfolgend werden die Testfälle vorgestellt, bevor in Kapitel 5.2 die Durchführung der Tests sowie die Betrachtung der Ergebnisse beider Werkzeuge folgen.

5.1.1 Testfall 1

Durch diesen ersten Testfall soll geprüft werden, ob die Werkzeuge in der Lage sind, hinzugefügte bzw. gelöschte Modellelemente zu erkennen. Des Weiteren wird untersucht, wie diese Elemente in der Differenz dargestellt werden. Dazu bietet der in Abschnitt 1.1 genannte Anwendungsfall ausreichend Möglichkeiten. In diesen Modellen wurde ein System beschrieben, welches Anfragen von Nutzern verarbeitet und dem Anfragenden bei vorliegender Berechtigung eine Auskunft erteilt. Das System wurde aus dem ihm zugrundeliegenden Modell generiert und lauffähig gemacht. Danach sollte jedoch eine zusätzliche Protokollierung der erteilten Auskünfte in das System eingearbeitet werden. Dazu wurde das vorliegende EPK-Modell (Abbildung 2) um die benötigten Elemente erweitert, woraus das Modell aus Abbildung 3 resultierte. Dieses wurde ebenfalls in ein BPEL-Modell transformiert, um das System zu erweitern. Die erforderliche Modell-Differenz soll auf den transformierten BPEL-Modellen berechnet werden, um die Änderungen des zweiten Modells in das weiter verfeinerte BPEL-Modell einzubringen. Die hier verwendeten BPEL-Modelle liegen im EMF-Format vor und sind in den Abbildungen 23 und 24 dargestellt. Eine im eigentlichen BPEL-Format gespeicherte Version dieser Modelle ist im Anhang (Listing 10) abgebildet. Diese Darstellung der Modelle wird jedoch hier nicht weiter verwendet.

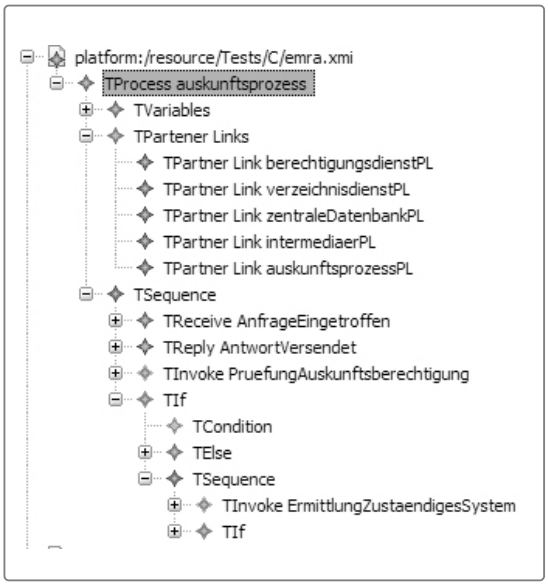

Abb. 23: Testfall 1 – Originalmodell

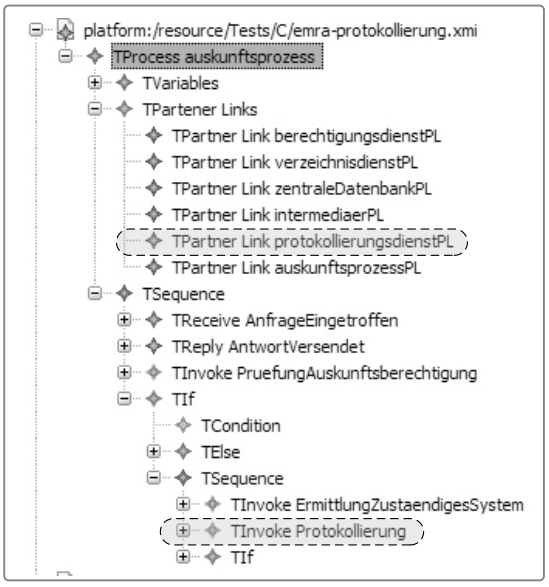

Abb. 24: Testfall 1 – geändertes Modell

In Abbildung 24 ist zu erkennen, dass zur Umsetzung der Protokollierung der Auskünfte ein PartnerLink namens *protokollierungsdienstPL* hinzukam. Des Weiteren ist ein Invoke, also die Einleitung einer Operation mit dem Namen *Protokollierung* eingefügt worden, welche ein Documentation-Element als Kind besitzt. Diese drei Elemente sollen durch die Werkzeuge als hinzugefügt erkannt und in der Modell-Differenz verzeichnet werden. Die Gesamtzahl der Änderungen beträgt 3.

Die folgenden Testfälle verwenden als Originalmodell jeweils das Modell aus Abbildung 24, bei welchem die Protokollierung der Auskünfte bereits hinzugefügt wurde. Aus diesem Grund werden in den nun folgenden drei Testfällen jeweils nur die Änderungen des neuen Modells aufgezeigt.

5.1.2 Testfall 2

Der zweite Testfall soll einfache Attributänderungen verschiedener Modellelemente untersuchen. Das Ändern von Attributen erfordert bei der Berechnung der Differenzen eine präzisere Abbildung der Modellelemente aufeinander, da sich die Vergleichswerte der einzelnen Elementpaare ändern. Unter Umständen kann es dazu kommen, dass zwei semantisch gleiche Modellelemente durch Änderungen ihrer Attribute nicht mehr aufeinander abgebildet werden können, oder dass sogar falsche Elementpaare gebildet werden.

In diesem Testfall wurden drei Attribute geändert. Die erste Änderung betrifft das Wurzelelement vom Typ TProcess. Hier wurde das Name-Attribut von *auskunftsprozess* auf den Wert *EMRA* geändert. Abbildung 25 stellt dieses Element mit dessen Änderung dar. Die weiteren Änderungen wurden im Element *auskunftsprozessPL* vom Typ TPartnerLink vorgenommen. In Abbildung 26 werden auch diese Änderungen verdeutlicht. Die Werkzeuge sollen drei Unterschiede zwischen den Modellen erkennen und in der Modell-Differenz ausweisen.

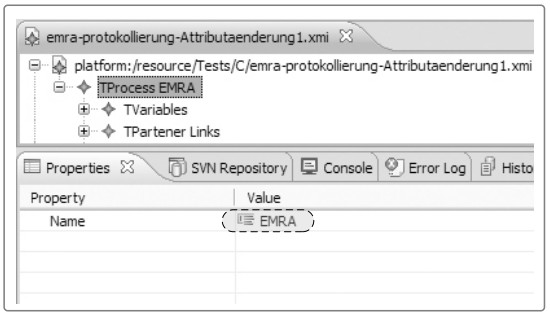

Abb. 25: Testfall 2 – erste Attributänderung

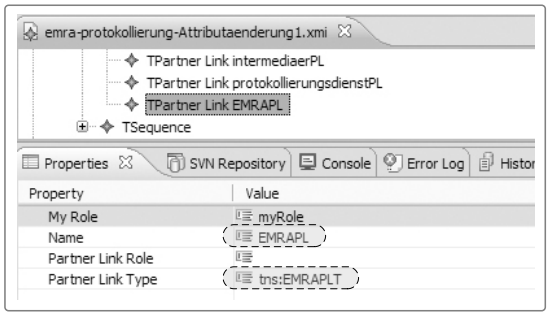

Abb. 26: Testfall 2 – weitere Attributänderungen

5.1.3 Testfall 3

In diesem Testfall wird erneut eine Attributänderungen untersucht. Diese Änderungen betreffen den PartnerLink *zentraleDatenbankPL* und sind in Abbildung 27 dargestellt. Hier wurden der Name und der Partner Link Type geändert. Die Differenz der Modelle müsste nun zwei Attributänderungen aufweisen. Des Weiteren kann man durch Betrachtung des vorhergehenden Testfalls erwarten, dass die von EMF Compare berechnete Modell-Differenz nicht die beiden Attribute als geädnert betrachtet, sondern wieder das Element *zentraleDatenbankPL* als gelöscht und das Element *DatenbankPL* als Hinzugefügt betrachtet. Im Folgenden soll untersucht werden, inwiefern sich die berechneten Differenzen voneinander und von den Differenzen aus Testfall 2 unterscheiden.

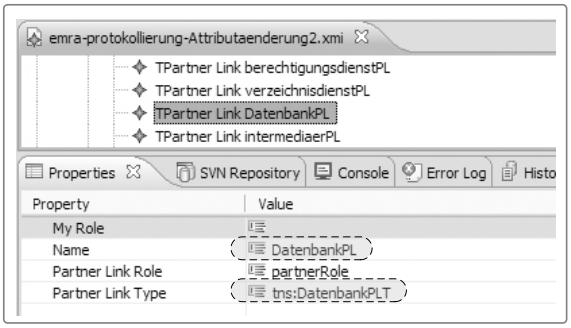

Abb. 27: Testfall 3 – Attributänderungen

5.1.4 Testfall 4

Durch diesen Testfall soll geprüft werden, inwiefern sich eine Positionsänderung der Elemente im Baum negativ auf die Ermittlung der Differenzen auswirken kann. Zu diesem Zweck wurde das Modell dahingehen verändert, dass lediglich der Partner Link *berechtigungsdienstPL* unter alle anderen Partner Links verschoben wurde. In Abbildung 28 ist diese Verschiebung dargestellt. Diese Verschiebung des PartnerLinks stellt keine semantische Änderung des Modells dar und sollte somit durch die Werkzeuge nicht als solche kenntlich gemacht werden. Die berechnete Differenz darf keine Unterschiede beider Modelle enthalten.

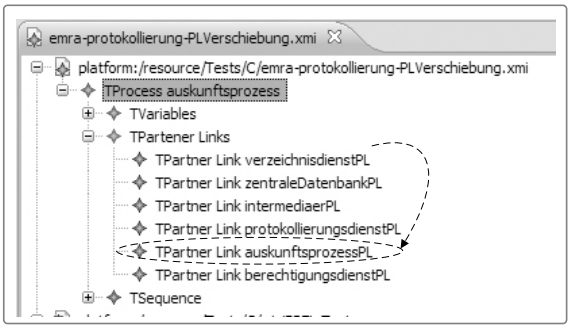

Abb. 28: Testfall 4 – Elementverschiebung

5.1.5 Testfall 5

In diesem Testfall wird die Verschiebung von Modellelementen in der Baumnotation noch einmal auf einer anderen Art von Modellen durchgeführt. Die Abbildungen 29 und 30 stellen die aus dem Anwendungsfall (EPK-zu-BPEL-Transformation) bekannte EPK (Abbildung 3 - vollständig in Abbildung 39 dargestellt) im EMF-Format dar. Bei der Serialisierung von graphischen Modellen in einer Baumnotation, wie sie hier verwendet wird, ergeben sich oft flache Baumstrukturen. Diese Strukturen entstehen vor allem dann, wenn die graphischen Modellelemente keine weiteren Elemente enthalten können, wie dies bei den hier verwendeten EPKs der Fall ist. Aus diesem Grund wirkt sich auch eine Verschiebung der Modellelmente im Baum nicht auf die graphische Representation des Modells aus. Die Abhängigkeiten der Modellelemente voneinander werden hier durch Referenzen gelöst, für die eine Änderung der Position von Elementen nicht relevant ist. Eine Solche Verschiebung eines Modellelements ist nachfolgend dargestellt. Hier wurde im zweiten Modell das Event *AnfrageEingetroffen* an das Ende der Event-Liste verschoben. Da bei dieser Verschiebung des Elements keinerlei semantische Änderungen am Modell vorgenommen wurden, darf von den Werkzeugen keine Änderung der Modelle erkannt werden.

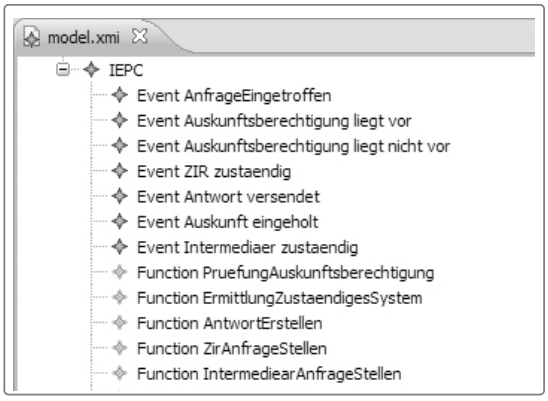

Abb. 29: Testfall 4 – Originalmodell

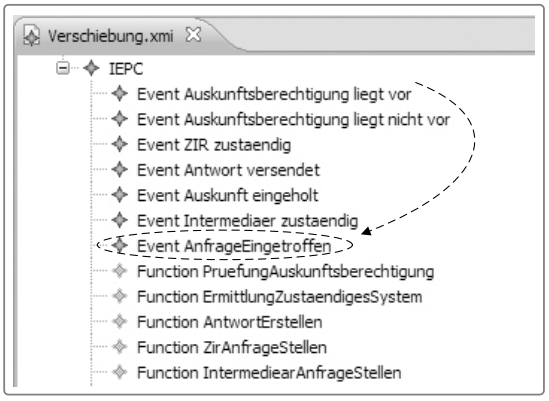

Abb. 30: Testfall 4 – geändertes Modell

5.2 Durchführung und Ergebnisse

In diesem Abschnitt werden die Differenzen der zuvor genannten Testfälle unter Nutzung der beiden Werkzeuge EMF Compare und ModelMatcher berechnet und analysiert. Die Ergebnisse der Berechnungen werden jeweils durch Abbildungen dargestellt. Während der Analyse der berechneten Differnzen sollen sowohl Unterschiede und Gemeinsamkeiten der erzielten Ergebnisse aufgezeigt als auch mögliche Ursachen für eventuelle Abweichungen von den geforderten Ergebnissen erläutert werden. Des Weiteren soll durch die erreichten Ergebnisse der Werkzeuge die Wahl der Testfälle weiter begründet werden.

5.2.1 Testfall 1

Zu Beginn der Auswertung dieses Tests sind die durch die Werkzeuge berechneten Differenzen in den Abbildungen 31 und 32 dargestellt. Im Folgenden werden diese Ergebnisse näher betrachtet.

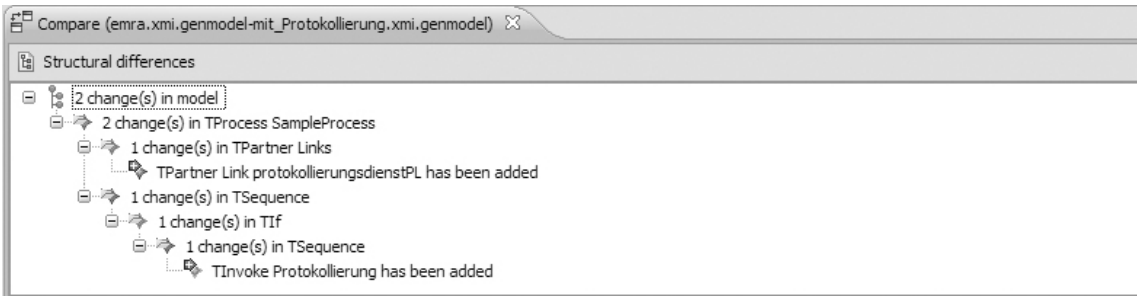

Abb. 31: Testfall 1 – durch EMF Compare berechnete Differenz

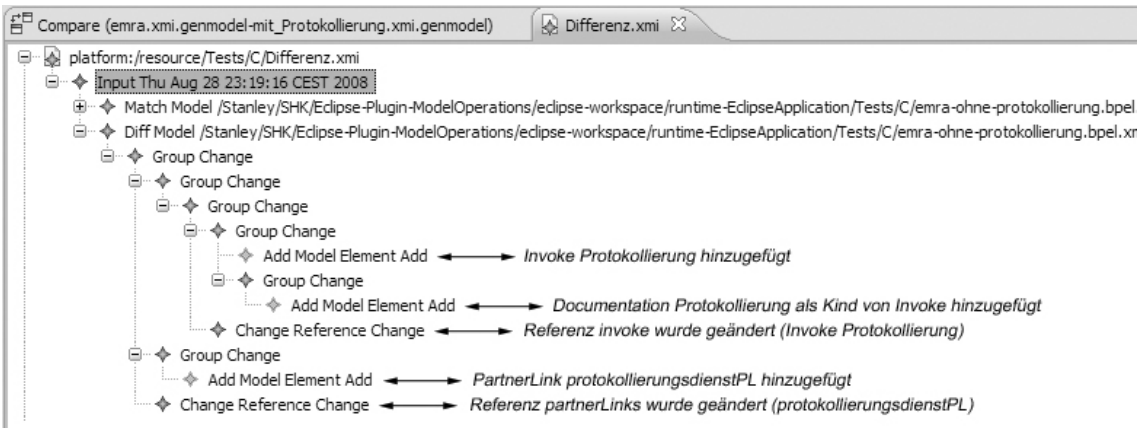

Abb. 32: Testfall 1 – vom ModelMatcher berechnete Differenz

Wie den beiden Abbildungen zu entnehmen ist, wurden von keinem der beiden Werkzeuge genau drei Änderungen des Originalmodells erfasst. Das Ergebnis des Tools EMF Compare weist lediglich zwei hinzugefügte Modellelemente aus, den PartnerLink und das Invoke. Dies ist jedoch dadurch begründet, dass ein Ziel der Entwickler von EMF Compare ist, die Modell-Differenzen so minimal wie möglich darzustellen. Das Kindelement

vom Typ Documentation wird dabei EMF-intern über das Element Invoke referenziert und kann bei der Übernahme des Invoke-Elements in das Originalmodell ermittelt und ebenfalls übertragen werden. Unter Beachtung dieser Funktion des Eclipse Modeling Framework kann davon ausgegangen werden, dass die Differenz durch EMF Compare korrekt ermittelt wurde. Der ModelMatcher weist hingegen in der berechneten Differenz fünf Unterschiede der Modelle aus. Hierunter befinden sich die drei hinzugefügten Elemente. Des Weiteren werden zwei Referenzänderungen angezeigt. Zum einen wurde die Referenz *partnerLinks* des Modellelements *PartnerLinks* geändert, da sich die Liste der als Kindelemente enthaltenen PartnerLinks geändert hat. Weiter wurde die Referenz *invoke* des Elements *ErmittlungZustaendigesSystem* vom Typ TSequenz aus selbigem Grund geändert. Diese beiden Änderungen wurden zwar korrekt erkannt und aufgelistet, jedoch ist hier in der Darstellung der Differenz beider Modelle eine Redundanz vorhanden, welche nach Möglichkeit vermieden werden soll[63]. Hier ist es lediglich erforderlich, die hinzugefügten Elemente anzuzeigen, da die Referenzen durch das Einfügen der Elemente durch EMF automatisch angepasst werden. Diese Redundanz in der Differenz entspricht zwar nicht einer minimalen Darstellung selbiger jedoch wird durch eine erneute Anpassung der Referenzlisten kein Schaden angerichtet. Somit kann die durch den ModelMatcher ermittelte Modell-Differenz prinzipiell als korrekt bezeichnet werden.

5.2.2 Testfall 2

Die Ergebnisse des zweiten Tests sind in den Abbildungen 33 und 34 dargestellt Nachfolgend werden auch diese erläutert.

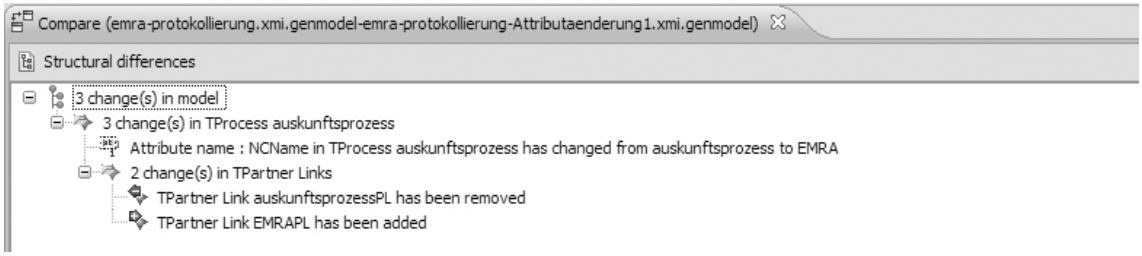

Abb. 33: Testfall 2 – durch EMF Compare berechnete Differenz

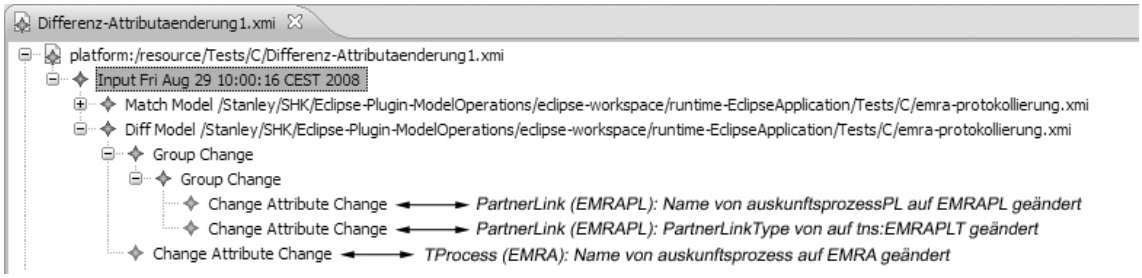

Abb. 34: Testfall 2 – vom ModelMatcher berechnete Differenz

[63] In [AlPo03] wird beispielsweise eine Minimierung der erstellten Differenzen gefordert.

Die bei diesem Test durch beide Werkzeuge ermittelten Differenzen weisen Unterschiede auf. Zwar enthalten beide Differenzen genau drei Änderungen der Modelle, jedoch weichen die erzielten Ergebnisse voneinander und von dem geforderten Ergebnis ab. Durch EMF Compare wurde zwar die erste Attributänderung (der Name des Prozesses) richtig erkannt, die letzten beiden Attributänderungen wurden jedoch nicht als solche angesehen. Hier kam kein Mapping der beiden Modellelemente *auskunftsprozessPL* und *EMRAPL* zustande, da der Vergleichswert beider Elemente scheinbar einen geforderten Schwellenwert nicht überschritten hat. Aus diesem Grund wurde das Element *auskunftsprozessPL* als gelöscht und anschließend das Element *EMRAPL* als hinzugefügt in die Differenz aufgenommen. Dies liegt unter anderem daran, dass EMF Compare bei dem Mapping der Elemente interne Schwellenwerte verwendet[64], welche einen Mindestvergleichswert der Elemente angeben, ab welchem diese aufeinander abgebildet werden können. Der ModelMatcher war hingegen in der Lage, ein korrektes Mapping der Elemente zu erstellen, sodass sämtliche Attributänderungen auch als solche angezeigt werden konnten. Hier war es durch den in Abschnitt 4.3 beschriebenen Mapping-Algorithmus möglich, die semantische Gleichheit der PartnerLinks zu erkennen. Dies liegt daran, dass im Gegensatz zu EMF Compare auf den Einsatz von Schwellenwerten für das Mapping der Elemente verzichtet wird.

5.2.3 Testfall 3

Die in den Abbildungen 35 und 36 dargestellten Differenzen weichen, wie bereits im Testfall zuvor, sowohl voneinander als auch vom geforderten Ergebnis ab. Bei der Ähnlichkeit dieses Tests zum vorhergehenden Test wurden ähnliche Differenzen wie im Testfall 2 erwartet. Hier wird jedoch deutlich, dass sich die erreichten Ergebnisse in die entgegengesetzte Richtung entwickelt haben. EMF Compare war bei diesem Test in der Lage, die beiden Attributänderungen korrekt auszuweisen, wohingegen der ModelMatcher das Element *zentraleDatenbankPL* als gelöscht und das Element *DatenbankPL* als hinzugefügt ausweist. Die beiden zusätzlichen Referenzänderungen sind bereits aus Testfall 1 bekannt und zeigen lediglich die geänderten Listen auf, welche die Kinder der beiden Modellelemente enthalten. Diese in der Differenz des ModelMatcher verzeichneten Änderungen werden hier nicht weiter berücksichtigt.

Das im Gegensatz zu Testfall 2 korrekt berechnete Ergebnis von EMF Compare lässt sich leicht erklären. Trotz der Ähnlichkeit dieses Tests zum vorhergehenden besteht ein unterschied, welchen sich EMF Compare zunutze macht. Der geänderte Name und der geänderte Partner Link Type weichen in diesem Beispiel relativ wenig von den ursprünglichen Werten ab. So wurde aus *zentraleDatenbankPL* der Wert *DatenbankPL* und aus *zentraleDatenbankPLT* der Wert *DatenbankPLT*. Wie bereits in Kapitel 4.2 erläutert, nutzt EMF Compare für den Vergleich von Attributwerten die Editierdistanz von Zeichenketten nach Levenshtein. Aus diesem Grund ist der Ähnlichkeitswert dieser Attribute, welcher nicht unwesentlich den Ähnlichkeitswert der Elemente beeinflusst, viel höher als dies im zweiten Test der Fall war. Des Weiteren kann die Berücksichtigung der Baumposition der Elemente hier ebenfalls ein entscheidendes Kriterium gewesen sein, weshalb ein korrektes Mapping gebildet wurde. Die durch den ModelMatcher berechnete Differenz ist hingegen

[64] Vgl. [Toul06]

inkorrekt. Die in Abbildung 36 dargestellte Differenz ist die Folge des noch nicht ausgereiften Mapping-Algorithmus des ModelMatcher. Beim Mapping des ModelMatcher wird immer der jeweils höchste Vergleichswert eines Elements betrachtet. Falls dieser nicht vergeben wurde, und mit der Gegenrichtung übereinstimmt, werden die Elemente aufeinander abgebildet. Hier ist jedoch der höchste Vergleichswert nicht diesen beiden Elementen zugeordnet, wodurch sie nicht aufeinander abgebildet werden konnten.

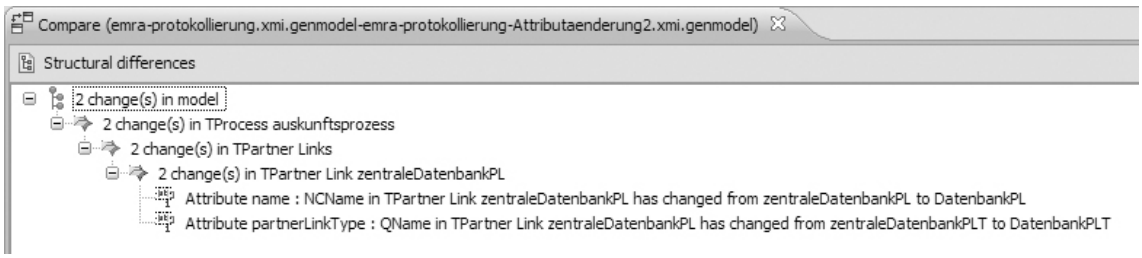

Abb. 35: Testfall 3 – durch EMF Compare berechnete Differenz

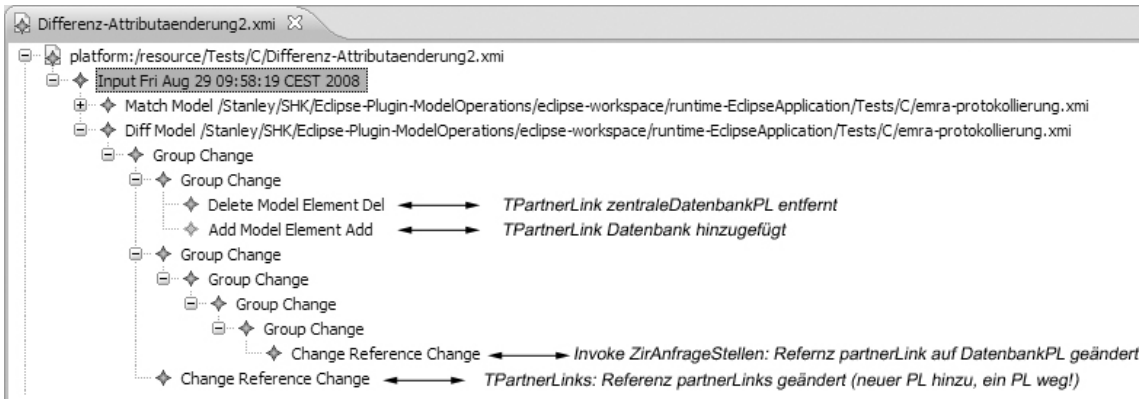

Abb. 36: Testfall 3 – vom ModelMatcher berechnete Differenz

5.2.4 Testfälle 4 und 5

Da die berechneten Differenzen der Werkzeuge im vierten Test der Vorgabe entsprechend leer sind und somit keine Analyse der Ergebnisse erforderlich sind, werden in diesem Abschnitt die Testfälle 4 und 5 aufgrund ihrer Ähnlichkeit zusammengefasst.

Wie bereits angesprochen sind die Ergebnisse des vierten Tests erwartungsgemäß korrekt. Da sich durch die Verschiebung des Modellelements innerhalb seines ursprünglichen Containers keinerlei semantische Änderungen ergeben, wurden von beiden Werkzeugen keine Unterschiede zum Originalmodell festgestellt. Die von EMF Compare genutzte Betrachtung der Positionen von Modellelementen im Baum spielt hier eine eher untergeordnete Rolle. Da die weiteren Vergleichskriterien (Attribute, Relationen und Namen der Elemente) keine Änderungen aufwiesen, konnte die Verschiebung des Elements den Vergleichswert nicht entscheidend beeinflussen. Dies liegt daran, dass die Baumposition der Elemente eher dazu genutzt wird, in zweifelhaften Fällen, bei denen mehrere Elementpaare denselben oder ähnliche Vergleichswerte besitzen, einen Hinweis auf ein mögliches

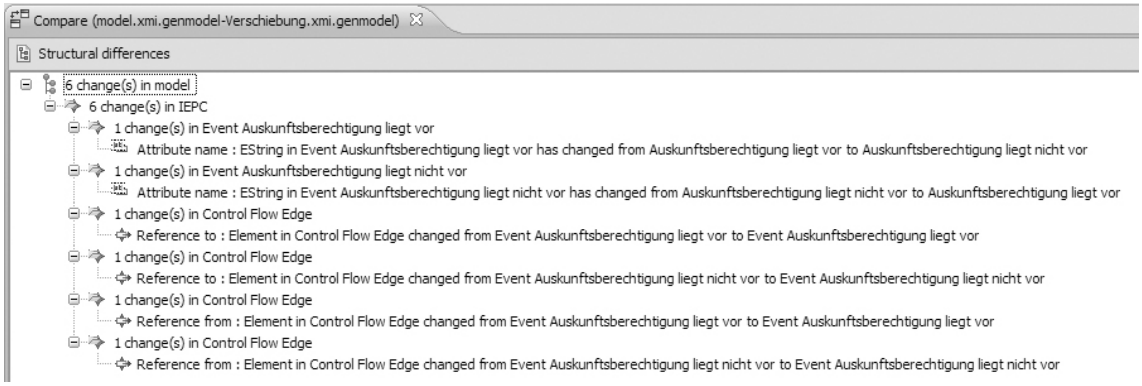

Abb. 37: Testfall 5 – durch EMF Compare berechnete Differenz

Mapping zu geben. Aus diesem Grund wird das Kriterium der Baumposition entweder gering gewichtet oder lediglich in den zuvor angesprochenen Fällen angewendet.

Um jedoch zu zeigen, dass durch die Berücksichtigung der Baumposition von Modellelementen durchaus negative Ergebnisse erzielt werden können, wurde in Testfall 5 eine andere Art von Modellen herangezogen. Die hier verwendeten graphischen EPK-Modelle ergeben im geforderten EMF-Format eine sehr flache Baumhierarchie. Wie bereits bei der Vorstellung des Testfalls angesprochen resultiert aus der Verschiebung des Elements keine semantische Änderung der EPK, wodurch bei diesem Test ebenfalls keinerlei Änderungen durch die Werkzeuge festgestellt werden dürfen. Dies ist bei der durch den ModelMatcher berechneten Differenz auch der Fall. Diese ist leer. Die durch EMF Compare berechnete Differenz (Abbildung 37) der beiden Modelle weist hingegen sechs Änderung auf. Eine Begründung hierfür liegt unter anderem darin, dass jedes Event einer EPK in dieser EMF-Notation lediglich ein Attribut (Name) enthält und sonst keine weiteren eindeutigen Merkmale besitzt. Dadurch können beispielsweise durch die Verwendung der Editierdistanz mit anschließender Betrachtung der Baumposition Elemente, wie z. B. die hier dargestellten Events *Auskunftsberechtigung liegt vor* und *Auskunftsberechtigung liegt nicht vor* falsch aufeinander abgebildet werden. Wie aus der Differenz ersichtlich ist, wurde die Abbildung dieser Elemente vertauscht, wodurch alle Referenzen, welche auf diese Elemente verweisen, als geändert angesehen werden.

5.3 Auswertung

In den zuvor durchlaufenen Tests sind die Unterschiedlichen Vergleichsansätze der beiden Werkzeuge EMF Compare und ModelMatcher verdeutlicht worden. Besonders wurden diese in den Testfällen 2, 3 und 5 deutlich. Hier hob sich beispielsweise die Nutzung der Editierdistanz beim Vergleich von Attributwerten, sowei die Betrachtung der Baumpositionen durch EMF Compare hervor. Dabei haben diese Kriterien sowohl positiven als auch negativen Einfluss auf das ermittelte Ergebnis ausgeübt. Dies hängt jedoch stark von der Art der Änderungen und deren Umfang sowie der verwendeten Modellart ab. Nachfolgende Tabelle soll einen Überblick über die erzielten Ergebnisse beider Werkzeuge ermöglichen. Dabei sind korrekt berechnete Modell-Differenzen durch ein Häkchen und

falsch berechnete Differenzen durch ein Kreuz dargestellt. Die Kreise zeigen Differenzen, welche durch die Werkzeuge prinzipiell richtig berechnet wurden, bei denen jedoch weitere Verbesserungen, wie beispielsweise eine Minimierung des Ergebnisses, möglich sind. Aufbauend auf dieser Übersicht wird abschließend ein Gesamtfazit beider Werkzeuge gezogen.

	EMF Compare	ModelMatcher
Testfall 1	❍	❍
Testfall 2	✗	✓
Testfall 3	✓	✗
Testfall 4	✓	✓
Testfall 5	✗	✓

Tabelle 1: Übersicht der erzielten Ergebnisse beider Werkzeuge

Zusammenfassend betrachtet ergeben keine deutlichen Vorteile für eines der beiden Werkzeuge. Zwar sind in obiger Tabelle geringfügig mehr korrekt berechnete Differenzen für den ModelMatcher verzeichnet, jedoch tendiert das Ergebnis von EMF Compare für den Testfall 1 eher zu einer korrekt berechneten Differenz, als das des ModelMatcher. Aus diesem Grund ist die Vergleichstabelle der berechneten Differenzen als nahezu ausgewogen anzusehen. Jedes der beiden Werkzeuge weist bei verschiedenen Anforderungen bestimmte Stärken, aber auch Schwächen auf. So wirkt sich beispielsweise die Betrachtung Baumpositionen in Kombination mit der Editierdistanz für Attribute bei EMF Compare für bestimmte Modellarten negativ auf das Mapping der Elemente aus. Beim ModelMatcher sind hingegen gelöschte oder hinzugefügte Elemente redundant in den berechneten Differenzen verzeichnet. Für beide Werkzeuge wurden jedoch die in Kapitel 4 erläuterten Vergleichskriterien bestätigt.

6 Zusammenfassung und Ausblick

Durch diese Arbeit wurde eine Einführung in die modellgetriebene Softwareentwicklung gegeben. Der Schwerpunkt lag jedoch auf der Berechnung und der Anwendung von Modell-Differenzen während der Modellierung von Geschäftsprozessen. Dabei wurde ausgehend von einer Einführung in die modellgetriebene Softwareentwicklung und die Konzepte der Geschäftsprozessmodellierung ein Ansatz der Berechnung von Modell-Differenzen erläutert. Hier wurde eine Vorgehensweise vorgeschlagen, bei der ausgehend von einem Vergleich der Modelle eine Abbildung selbiger erstellt wird. Aufbauend auf dieser Abbildung ist es möglich die Unterschiede der Modelle zu berechnen. Mögliche Anwendungen von Modell-Differenzen wurden beispielsweise durch Versionskontrollsysteme oder das Verteilte Entwickeln von Modellen genannt. Weiterhin wurden verschiedene Werkzeuge vorgestellt, mit welchen der Vergleich von Modellen sowie die Berechnung von Modell-Differenzen möglich ist. Diese Werkzeuge zeigten verschiedene Ansätze zur Berechnung von Modell-Differenzen in der Praxis. Dabei wurden zwei der genannten Werkzeuge detailliert beschrieben, um sie in den darauf folgenden Tests zu verwenden. Diese Testfälle wurden ausgehend von der EPK-zu-BPEL-Transformation des Anwendungsfalls erstellt, um die Arbeitsweise der beiden Werkzeuge EMF Compare und ModelMatcher zu evaluieren. Dabei wurden Änderungen der Modelle mit unterschiedlichen Schwierigkeitsgraden vorgenommen, welche eventuelle Vor- und Nachteile der einzelnen Vergleichskriterien der Werkzeuge beleuchteten. In diesem Kapitel wurden die Vergleichskriterien der Werkzeuge sowie der zuvor erläuterte Ansatz der Berechnung von Modell-Differenzen bekräftigt. Weiterhin wurden verschiedene Stärken und Schwächen der verwendeten Werkzeuge verdeutlicht, welche aber bezüglich der Ermittlung von Differenzen als größtenteils gleichwertige Konkurrenten anzusehen sind.

Die Ermittlung von Modell-Differenzen allein bringt aber in der modellgetriebenen Softwareentwicklung wenig Nutzen. Die bereits angesprochenen Anwendungen von Modell-Differenzen sind der Hauptgrund, warum diese überhaupt berechnet werden. Die Untersuchung bzw. Entwicklung eines Versionskontrollsystems, welches Modell-Differenzen zur Vergabe von Versionsnummern für Modelle nutzt oder die Betrachtung eines Systems, welches parallel geänderte Modelle unter Nutzung des auf Modell-Differenzen basierenden Vereinigungsoperators zusammenführt, sind mögliche nächste Schritte. Hier spielen Performanzbetrachtungen bei der Berechnung von Differenzen sowie bei deren Anwendung eine wichtige Rolle, durch welche sich die bestehenden Ansätze durchaus voneinander abgrenzen können.

Abschließend kann man jedoch sagen, dass der Differenzoperator ein unverzichtbares Hilfsmittel ist, welches die modellgetriebene Softwareentwicklung durch die Vermeidung von Redundanten Arbeitsschritten effektiver gestaltet.

Literaturverzeichnis

[AlPo03] Marcus Alanen and Ivan Porres. Difference and union of models. *UML 2003 – The Unified Modeling Language. Model Languages and Applications.*, pages 2–17, Springer Verlag, 2003.

[ARI08] The ARIS-platform. `http://www.ids-scheer.de/de/ARIS_Business_Performance_Edition_Software/89371.html`, 18.08.2008.

[Balz93] Helmut Balzert. CASE: Systeme und Werkzeuge BI-Wiss.-Verlag, 1993.

[BeMe07] Philip A. Bernstein and Sergey Melnik. Model management 2.0: Manipulating richer mappings. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data.*, pages 1–12, 2007.

[Bro91] *Brockhaus Enzyklopädie in 24 Bänden*. F. A. Brockhaus, 1991.

[BCE+06] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu and Mehrdad Sabetzadeh A Manifesto for Model Merging. *Proceedings of the 2006 international workshop on Global integrated model management*, pages 5–12, 2006.

[BSM+03] Frank Budinsky, Dave Steinberg, Ed Merks, Ray Ellersick and Timothy J. Grose. *Eclipse Modeling Framwork*. Addison-Wesley Professional, 2003.

[COMA++] COMA++. `http://dbs.uni-leipzig.de/research/coma.html`, 25.08.2008.

[DFK+04] Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman and Pat McCarthy. *The Java Developer's Guide to Eclipse (Second Edition)*. Addison-Wesley Professional, 2004.

[Eclipse] Eclipse – an open development platform. `http://www.eclipse.org`, 11.08.2008.

[Ecore] Ecore dokumentation. `http://download.eclipse.org/modeling/emf/emf/javadoc/2.5.0/org/eclipse/emf/ecore/package-summary.html`, 20.08.2008.

[EMF] Eclipse Modeling Framework (EMF). `http://www.eclipse.org/modeling/emf/?project=emf#emf`, 20.08.2008.

[EMFT] The Eclipse Modeling Framework Technology Project. `http://www.eclipse.org/modeling/emft`, 11.08.2008.

[EMFC] EMF Compare. `http://www.eclipse.org/modeling/emft/?project=compare#compare`, 23.08.2008.

[FrRu06] Robert France and Bernhard Rumpe. Modeling the complex living world. *Software & Systems Modeling*, volume 5, number 1, pages 1–2, 2006.

[FUJABA] FUJABA Tool Suite. http://www.fujaba.de/, 23.08.2008.

[Gada08] Andreas Gadatsch. *Grundkurs Geschäftsprozess-Management – Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. Vieweg Verlag, 5. Edition, 2008.

[JoEv07] Diane Jordan (IBM) and John Evdemon (Microsoft). *Web Services Business Process Execution Language Version 2.0*, volume 2.0. OASIS, 2007.

[Intalio] Intalio Company. http://www.intalio.com/, 21.08.2008.

[Kühne99] Ulrich Kühne. Wissenschaftstheorie. *Enzyklopädie Philosophie*, 1999.

[Kühne06] Thomas Kühne. Matters of (meta-) modeling. *Software & Systems Modeling*, volume 5, number 4, pages 395–401, 2006.

[KrKü06] Martin Kreuzer and Stefan Kühling. *Logik für Informatiker*. Pearson Studium, 2006.

[KoPP06] Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack. Model comparison: A foundation for model composition and model transformation testing. *Proceedings of the 2006 international workshop on Global integrated model management*, pages 13–20, 2006.

[Leve65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, pages 845–848, 1965.

[Math] Mathworks – matlab/simulink. http://www.mathworks.de/products/simulink/, 23.08.2008.

[MSUW04] Stephen J. Mellor, Kendall Scott, Axel Uhl and Dirk Weise. *MDA Distilled – Principles of Model-Driven Architecture*. Addison-Wesley Professional, 2004.

[MaSV96] Prof. Dr. Bernd Mahra, Prof. Dr. Alexander Schill and Prof. Dr. Gottfried Vossen. *Geschäftsprozessmodellierung und Workflow-Management*. Informatik Lehrbuch-Reihe. International Thomson Publishing, 1. Edition, 1996.

[NüRu02] Markus Nüttgens and Frank J. Rump. *Promise 2002 – Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, chapter Syntax und Semantik ereignisgesteuerter Prozessketten (EPK), pages 64–77. Gesellschaft für Informatik, 2002.

[OASIS] OASIS - Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/home/index.php, 18.08.2008.

[Obeo] Obeo Company. http://www.obeo.fr/?\&lang=en, 21.08.2008.

[OMG] The Object Management Group (OMG). http://www.omg.org, 28.08.2008.

[OWL] Web Ontology Language. http://www.w3.org/tr/owl-features/, 25.08.2008.

[Rump99] Frank J. Rump. *Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten*. Teubner-Reihe Wirtschaftsinformatik. Teubner, 1999.

[SAP] SAP AG. http://www.sap.com/germany/index.epx, 18.08.2008.

[Schm06] Douglas C. Schmidt. Model-Driven Engineering. *Computer*, volume 39, number 2, pages 25–31, 2006.

[SiDiff] The SiDiff-Project. `http://pi.informatik.uni-siegen.de/sidiff/`, 10.08.2008.

[ScKN92] A.-W. Scheer, G. Keller and M. Nüttgens. Semantische Prozessmodellierung auf Grundlage Ereignisgesteuerter Prozessketten (EPK). Technical Report, Institut für Wirtschaftsinformatik - Universität des Saarlandes, Saarbrücken, 1992.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer Verlag, 1973.

[StVö06] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase and Simon Helsen. *Model Driven Software Development – Technology, Engineering, Management*. John Wiley, 2006.

[Toul06] Antoine Toulmé. Presentation of EMF Compare utility. *position paper at Eclipse summit, Esslingen, Germany*, 2006.

[W3C] W3C – World Wide Web Consortium. `http://www.w3.org/`, 25.08.2008.

[White04] Stephen A. White. *Business Process Modeling Notation (BPMN)*. Business Process Management Initiative (BPMI), 2004.

[Wüst63] Klaus-Dieter Wüsteneck. Zur philosophischen Verallgemeinerung und Bestimmung des Modellbegriffs. *Deutsche Zeitschrift für Philosophie*, page 1504 ff., 1963.

[XSD] XML Schema Definition. `http://www.w3.org/xml/schema`, 25.08.2008.

Anhang

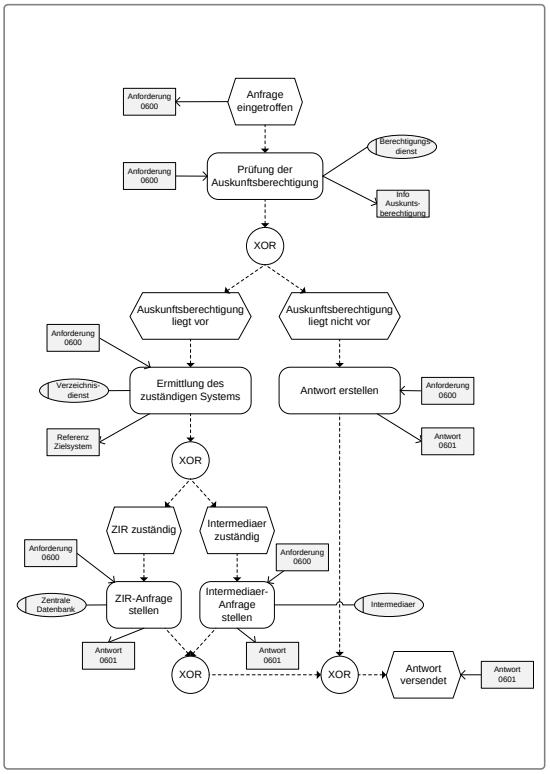

Abb. 38: Beispielsystem ohne Protokollierung (vollständige EPK)

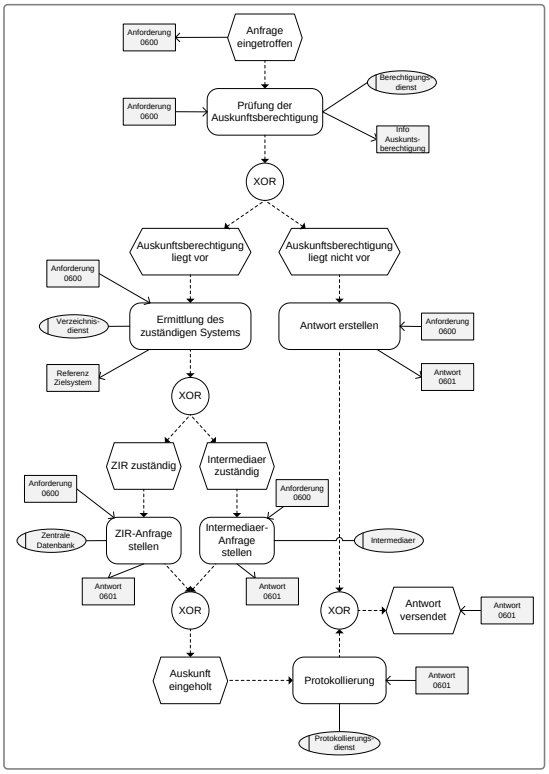

Abb. 39: Beispielsystem mit Protokollierung (vollständige EPK)

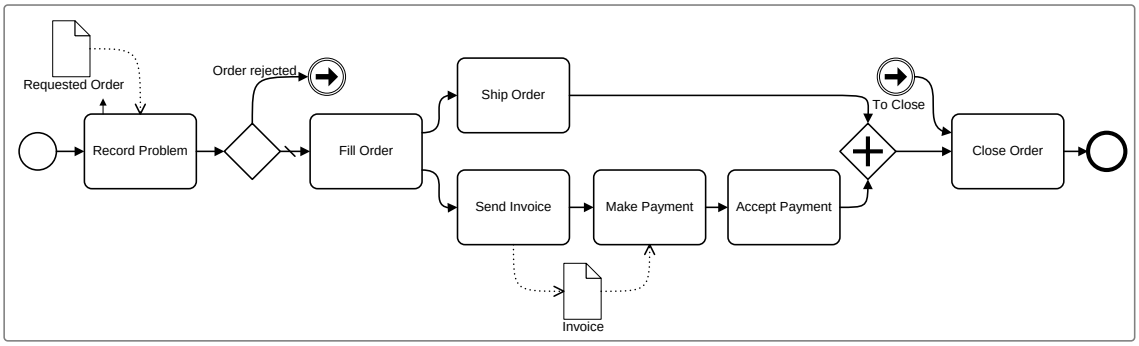

Abb. 40: BPMN-Beispielmodell aus [White04]

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!-- generated -->
4  <process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
       executable" xmlns:tns="http://www.example.com/SampleProcess"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SampleProcess"
       targetNamespace="http://www.example.com/SampleProcess">
5    <import importType="http://schemas.xmlsoap.org/wsdl/" location="
         SampleProcess.wsdl" namespace="http://www.example.com/
         SampleProcess"/>
6    <partnerLinks>
7      <partnerLink name="berechtigungsdienstPL" partnerLinkType="
           tns:berechtigungsdienstPLT" partnerRole="partnerRole"/>
8      <partnerLink name="verzeichnisdienstPL" partnerLinkType="
           tns:verzeichnisdienstPLT" partnerRole="partnerRole"/>
9      <partnerLink name="zentraleDatenbankPL" partnerLinkType="
           tns:zentraleDatenbankPLT" partnerRole="partnerRole"/>
10     <partnerLink name="intermediaerPL" partnerLinkType="
           tns:intermediaerPLT" partnerRole="partnerRole"/>
11     <partnerLink name="sampleProcessPL" partnerLinkType="
           tns:sampleProcessPLT" myRole="myRole"/>
12   </partnerLinks>
13   <variables>
14     <variable name="Anforderung-0600" messageType="tns:anforderung-0600
         MT"/>
15     <variable name="InfoAuskunftsberechtigung" messageType="
           tns:infoAuskunftsberechtigungMT"/>
16     <variable name="ReferenzZielsystem" messageType="
           tns:referenzZielsystemMT"/>
17     <variable name="Antwort-0601" messageType="tns:antwort-0601MT"/>
18   </variables>
19   <sequence>
20     <receive name="AnfrageEingetroffen" createInstance="yes"
           partnerLink="sampleProcessPL" portType="
           tns:anfrageEingetroffenPT" operation="
           AnfrageEingetroffenOperation" variable="Anforderung-0600">
21       <documentation>AnfrageEingetroffen</documentation>
22     </receive>
23     <invoke name="PruefungAuskunftsberechtigung" partnerLink="
           berechtigungsdienstPL" portType="
           tns:pruefungAuskunftsberechtigungPT" operation="
           PruefungAuskunftsberechtigungOperation" inputVariable="
           Anforderung-0600" outputVariable="InfoAuskunftsberechtigung">
24       <documentation>PruefungAuskunftsberechtigung</documentation>
25     </invoke>
26     <if>
27       <condition>
28         <!-- Auskunftsberechtigung liegt vor -->
29       </condition>
30       <sequence>
31         <invoke name="ErmittlungZustaendigesSystem" partnerLink="
               verzeichnisdienstPL" portType="
               tns:ermittlungZustaendigesSystemPT" operation="
```

```
            ErmittlungZustaendigesSystemOperation" inputVariable="
            Anforderung-0600" outputVariable="ReferenzZielsystem">
32          <documentation>ErmittlungZustaendigesSystem</documentation>
33        </invoke>
34        <if>
35          <condition>
36            <!-- ZIR zustaendig -->
37          </condition>
38          <invoke name="ZirAnfrageStellen" partnerLink="
              zentraleDatenbankPL" portType="tns:zirAnfrageStellenPT"
              operation="ZirAnfrageStellenOperation" inputVariable="
              Anforderung-0600" outputVariable="Antwort-0601">
39            <documentation>ZirAnfrageStellen</documentation>
40          </invoke>
41          <else>
42            <!-- Intermediaer zustaendig -->
43            <invoke name="IntermediearAnfrageStellen" partnerLink="
                intermediaerPL" portType="
                tns:intermediearAnfrageStellenPT" operation="
                IntermediearAnfrageStellenOperation" inputVariable="
                Anforderung-0600" outputVariable="Antwort-0601">
44              <documentation>IntermediearAnfrageStellen</documentation>
45            </invoke>
46          </else>
47        </if>
48      </sequence>
49      <else>
50        <!-- Auskunftsberechtigung liegt nicht vor -->
51        <assign name="AntwortErstellen">
52          <documentation>AntwortErstellen</documentation>
53          <copy>
54            <from variable="Anforderung-0600" part="Anforderung-0600
                Part"></from>
55            <to variable="Antwort-0601" part="Antwort-0601Part"/>
56          </copy>
57        </assign>
58      </else>
59    </if>
60    <reply name="Antwort versendet" partnerLink="sampleProcessPL"
        portType="tns:anfrageEingetroffenPT" operation="
        AnfrageEingetroffenOperation" variable="Antwort-0601">
61      <documentation>Antwort versendet</documentation>
62    </reply>
63  </sequence>
64</process>
```

Listing 10: BPEL-Beispielmodell (zu EPK aus Abbildung 2)

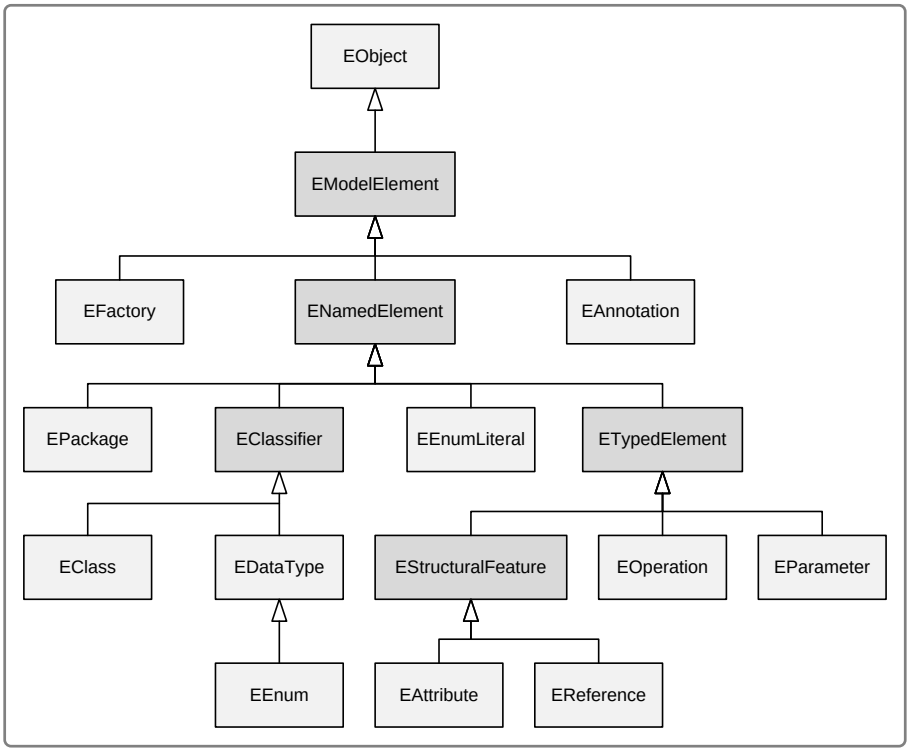

Abb. 41: Ecore-Klassenhierarchie [Ecore]

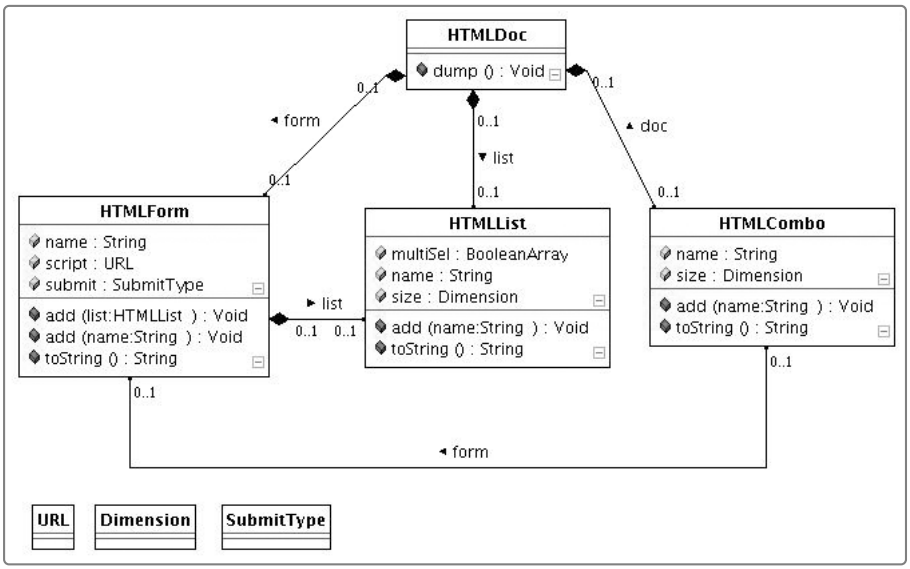

Abb. 42: SiDiff – Klassendiagramm 1 [SiDiff]

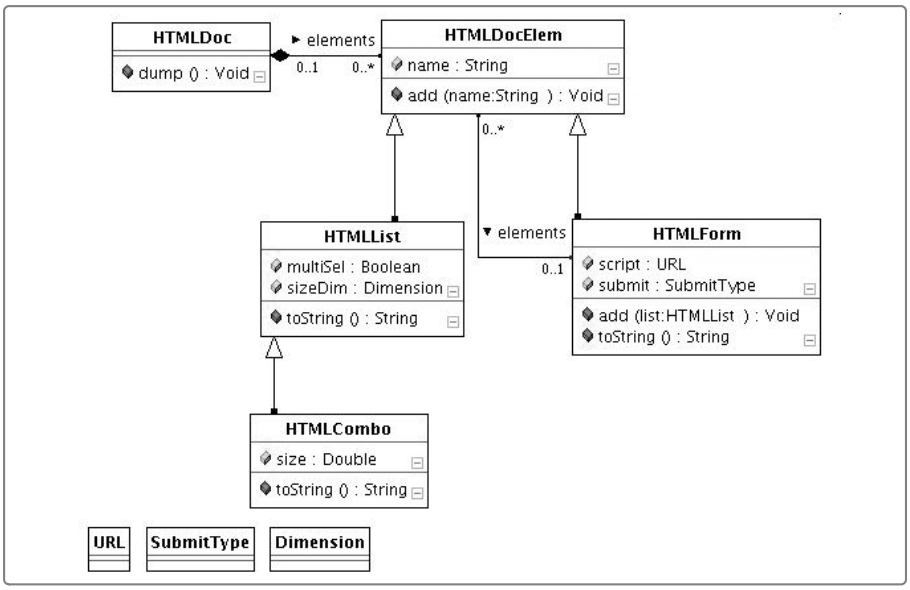

Abb. 43: SiDiff – Klassendiagramm 2 [SiDiff]

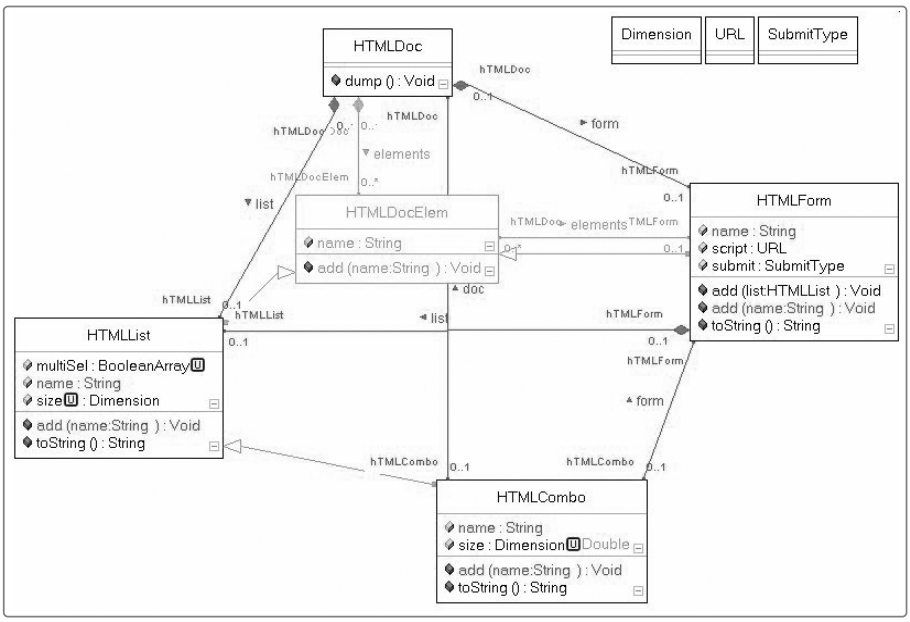

Abb. 44: SiDiff – Vereinigung-Klassendiagramme [SiDiff]

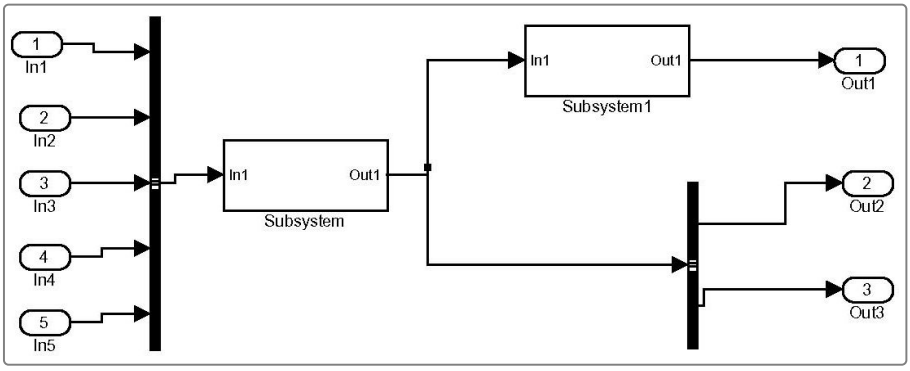

Abb. 45: SiDiff – Simulink-Diagramm 1 [SiDiff]

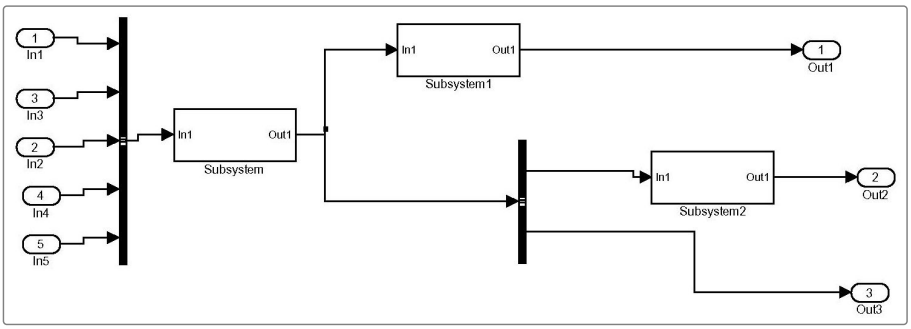

Abb. 46: SiDiff – Simulink-Diagramm 2 [SiDiff]

Update of attribute **LastModifiedDate** at Element **ModelRoot** (root) . Value changed form **"Wed Jun 01 17:07:51 2005"** to **"Fri Jun 17 16:13:58 2005"** .
Update of attribute **LastModifiedBy** at Element **ModelRoot** (root) . Value changed form **"mathlab"** to **"Allebrod"** .
Update of attribute **ModelVersionFormat** at Element **ModelRoot** (root) . Value changed form **"1.%<AutoIncrement:6>"** to **"1.%<AutoIncrement:9>"** .
Update of attribute **name** at Element **"simple2"** (S210) . Value changed form **"simple2"** to **"simple3"** .
Update of attribute **Location** at Element **"simple2"** (S210) . Value changed form **[419, 160, 999, 453]** to **[2, 82, 1014, 722]** .
Update of attribute **name** at Element (S346) . Value changed form **de.usi.simulink.Line3** to **de.usi.simulink.Line9** .
Update of attribute **Points** at Element (S363) . Value **[50, 0; 0, -25]** was removed.
Update of attribute **name** at Element (S363) . Value changed form **de.usi.simulink.Line5** to **de.usi.simulink.Line15** .
Reference **SrcBlock** of Object **de.usi.simulink.Line5** (S363) changed from **"Bus\nSelector"** (S255) to **"Subsystem2"** (S324)
Update of attribute **name** at Element (S391) . Value changed form **de.usi.simulink.Line10** to **de.usi.simulink.Line12** .
Update of attribute **DstPort** at Element (S386) . Value changed form **3** to **2** .
Update of attribute **name** at Element (S386) . Value changed form **de.usi.simulink.Line9** to **de.usi.simulink.Line17** .
Update of attribute **name** at Element (S340) . Value changed form **de.usi.simulink.Line2** to **de.usi.simulink.Line8** .
Update of attribute **name** at Element (S397) . Value changed form **de.usi.simulink.Line11** to **de.usi.simulink.Line13** .
Update of attribute **name** at Element (S351) . Value changed form **de.usi.simulink.Line4** to **de.usi.simulink.Line10** .
Update of attribute **DstPort** at Element (S381) . Value changed form **2** to **3** .
Update of attribute **name** at Element (S381) . Value changed form **de.usi.simulink.Line8** to **de.usi.simulink.Line18** .
Update of attribute **Points** at Element (S369) . Value changed form **[0, -10]** to **[135, 0; 0, 50]** .
Update of attribute **name** at Element (S369) . Value changed form **de.usi.simulink.Line6** to **de.usi.simulink.Line16** .
Update of attribute **name** at Element (S375) . Value changed form **de.usi.simulink.Line7** to **de.usi.simulink.Line11** .
Element **de.usi.simulink.Line14** (S480) was added.
Update of attribute **Position** at Element **"In3"** (S232) . Value changed form **[30, 108, 60, 122]** to **[25, 73, 55, 87]** .
Update of attribute **Position** at Element **"In2"** (S227) . Value changed form **[30, 68, 60, 82]** to **[25, 113, 55, 127]** .
Update of attribute **Position** at Element (S335) . Value changed form **[535, 173, 565, 187]** to **[715, 233, 745, 247]** .
Update of attribute **Position** at Element (S330) . Value changed form **[535, 108, 565, 122]** to **[720, 138, 750, 152]** .
Update of attribute **Position** at Element (S326) . Value changed form **[530, 33, 560, 47]** to **[675, 33, 705, 47]** .
Update of attribute **FontSize** at Element **"Subsystem"** (S262) . Value **10** was removed.
Update of attribute **FontSize** at Element (S294) . Value **10** was removed.
Element **"Subsystem2"** (S324) was added.
Element **de.usi.simulink.Line6** (S413) was added.
Element **de.usi.simulink.Line3** (S396) was added.

Abb. 47: SiDiff – Simulink-Modell-Differnz [SiDiff]

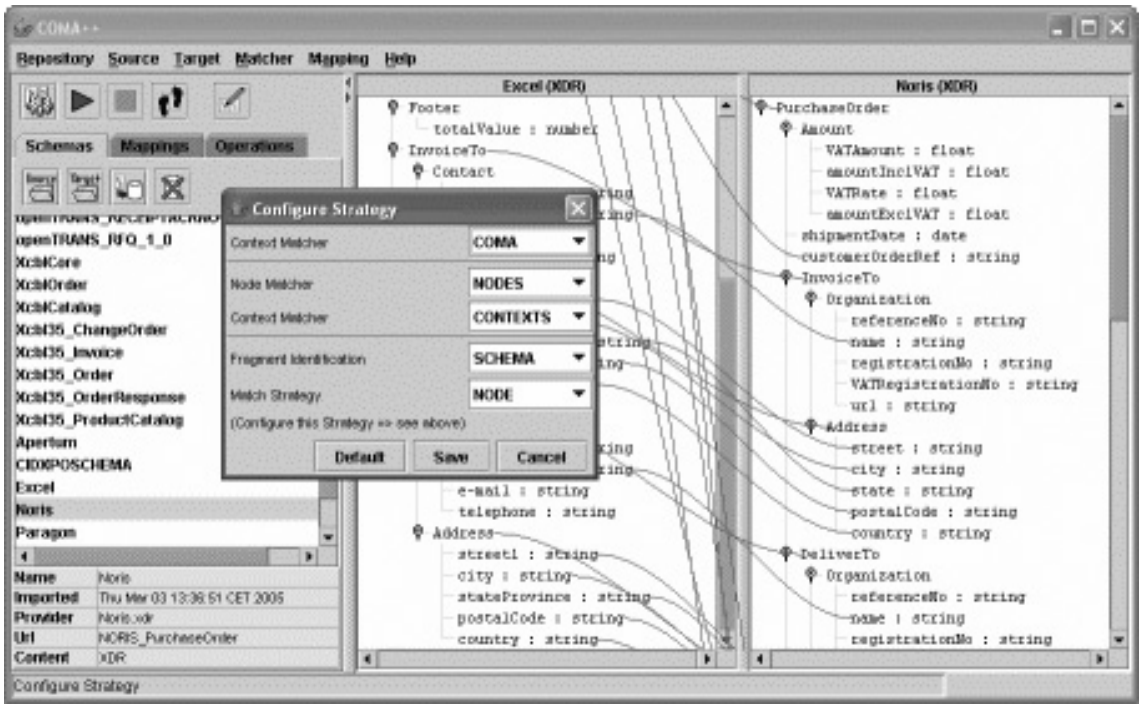

Abb. 48: COMA++ – Benutzungsoberfläche [COMA++]

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 9. September 2008 Stanley Hillner