Business Impact Analysis
Konzept und Realisierung einer ganzheitlichen Geschäftsanalyse

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Sylvia Natalie Radeschütz

aus Stuttgart

Hauptberichter:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Mitberichter:	Prof. Dr. rer. nat. Frank Leymann

Tag der mündlichen Prüfung: 12. Dezember 2011

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2011

Dankesworte

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftliche Mitarbeiterin in der Abteilung Anwendersoftware des Instituts für Parallele und Verteilte Systeme der Universität Stuttgart. Ich möchte den Menschen danken, die mich bei dem Entstehen dieser Arbeit begleitet haben.

Mein Dank gilt in erster Linie Herrn Professor Bernhard Mitschang für die Anregung zu dieser Arbeit, sein großes Interesse und die Betreuung meiner Arbeit. Danke für den interessanten Gedankenaustausch in zahlreichen Diskussionen und Ihre Unterstützung bei der Umsetzung dieser Arbeit!

Herrn Professor Frank Leymann möchte ich danken für das stets rege Interesse am Thema meiner Arbeit und für seine Bereitschaft zur Übernahme des Mitberichts. Danke für Ihren netten Zuspruch, wann immer wir uns an der Universität begegneten!

Allen Kollegen in der Abteilung Anwendersoftware danke ich für die inhaltliche Zusammenarbeit, aber vor allem für die lustigen Fahrradtouren und Kaffeepausen. Insbesondere gilt mein Dank Marko Vrhovnik, Holger Schwarz, Florian Niedermann, Matthias Wieland und Jing Lu für die kritischen Diskussionen und fachlichen Ratschläge im Verlauf der Arbeit sowie Florian Niedermann, Renate Widmaier und Steffen Rüger für die Anmerkungen beim Korrekturlesen. Danke für eure wertvollen Tipps!

Danken möchte ich auch meinen Studenten Florian Niedermann, Daojun Cui und Wolfgang Bischoff, die im Rahmen ihrer Diplomarbeit bzw. ihrer Hiwi-Tätigkeit wertvolle Hilfe bei der GUI des Prototyps leisteten. Danke für euer kreatives Engagement!

Besonders herzlich bedanken möchte ich mich bei meinem Ehemann, der mir stets geduldig zugehört und mich in meiner Arbeit bestärkt hat, sowie bei meinen Eltern und meinem Bruder, die mich immer bei der Verwirklichung meiner Ziele unterstützt haben. Danke für eure liebevolle Unterstützung und euer Verständnis!

Sindelfingen, im März 2011 Sylvia Radeschütz

Zusammenfassung

Immer mehr Unternehmen setzen auf die Etablierung von Geschäftsprozessen, die verschiedene Unternehmensanwendungen integrieren, anstatt die einzelnen Geschäftsfunktionen separat für sich auszuführen. Geschäftsprozesse werden hierzu als Workflows formalisiert und modelliert und auf einem Workflowmanagementsystem rechnergestützt zur Ausführung gebracht. Auf diese Weise können die Unternehmen schnell und möglichst noch vor der Konkurrenz auf neue Marktsituationen reagieren und ihre Produkte bzw. Geschäftsprozesse z.B. auf neue Kundenwünsche anpassen. Für eine bestmögliche Anpassung der Workflows an neue Anforderungen und eine wirkungsvolle Optimierung der Geschäftsprozesse ist eine genaue Analyse der Ablaufdaten erforderlich. Die Analyse umfasst neben den Zeitmessungen des Ablaufs u.a. auch die von den Geschäftsprozessen referenzierten Dateneingaben und Datenausgaben.

Um die Workflows jedoch wirklich wettbewerbsfähig zu halten, fehlt ein entscheidender Aspekt bei heutigen Analyseansätzen: die Berücksichtigung von Informationen aus anderen Unternehmensanwendungen, die zwar nicht im Workflow selbst integriert wurden, aber dennoch wertvolle Daten für wichtige Modellierungsentscheidungen enthalten. Solche ganzheitlichen Geschäftsprozessoptimierungen sind heutzutage nur mit Hilfe von großen manuellen Anstrengungen in der Integration und Analyse der Informationen möglich. Die Durchführung einer ganzheitlichen Optimierung birgt demnach zahlreiche neue Probleme in diesen Bereichen, die zunächst gelöst werden müssen: Das semiautomatische Matchen der Geschäftsprozessdaten mit den operativen Daten aus anderen Unternehmensanwendungen und die Umsetzung ihrer gemeinsamen Analyse mit Hilfe von neuen Verfahren.

In dieser Arbeit werden Konzepte zur Realisierung von Lösungen zu der Integration und der ganzheitlichen Analyse dieser Daten vorgestellt. Diese Art der umfassenden Unternehmensanalyse wird hier Business Impact Analysis (BIA) genannt.

Die Integration für BIA steht im Mittelpunkt dieser Arbeit. Das Integrationsverfahren wendet eine Matchregeln gemäß einer wohldefinierten Kontrollstrategie, die die optimale Reihenfolge der Regelanwendungen bestimmt, auf die Workflowdaten und operativen Daten an, um sie miteinander auf Schemaebene zu kombinieren. Bei jeder Kombination wird ein Ähnlichkeitswert zwischen den kombinierten Daten berechnet. Wenn der Ähnlichkeitswert über einem durch den Benutzer festgelegten Schwellwert liegt, wird die Kombination der Daten als neuer Match in die Ergebnismenge aufgenommen. Die Regelbasis enthält semi-automatische Regeln für das Kombinieren von Daten, die zuvor durch semantische Begriffe aus einer Ontologie annotiert wurden. Des Weiteren gehören automatische Regeln zur Regelbasis für das Matchen von nicht oder nur teilweise annotierten Daten. Außerdem sind Filterregeln in der Regelbasis dafür zuständig, die berechneten Ähnlichkeitswerte gemäß der Übereinstimmung in der Struktur der jeweiligen Kombinationselemente anzupassen. Die Kombinations- und Filterregeln beruhen auf existierenden und neuen Matchverfahren.

Die Analyse für BIA in dieser Arbeit setzt die Erstellung eines integrierten Data Warehouses voraus. Diese Arbeit stellt zwei Architekturmodelle vor, um solch ein Warehouse aufbauen zu können und gibt eine detaillierte Einschätzung ihrer Anwendbarkeit für BIA.

Die Analyseverfahren für BIA können auf beiden Architekturansätzen des Warehouses durchgeführt werden. Die Arbeit entwickelt SQL-Analyseoperatoren, um die integrierten Daten in OLAP-Anfragen so zu evaluieren, dass sich neue Erkenntnisse für die Geschäftsprozessoptimierung ergeben. Außerdem gibt die Arbeit Anhaltspunkte, wie die Operatoren im Mining-Verfahren für BIA verwendet werden können.

Die Integrations-, Warehousing- und Analyseansätze sind in einer prototypischen Implementierung umgesetzt. In Experimenten wird die Effektivität der Matchregeln untersucht und der Gewinn in der Menge und Qualität der Matchergebnisse gegenüber existierenden Verfahren erläutert. Weitere Messszenarien zeigen die Benutzbarkeit der Warehousing- und Analyseansätze für BIA. Insgesamt lässt sich zeigen, dass mit dieser Arbeit durch die Anwendung der Integrationsregeln und Analyseoperatoren auf dem integrierten Data Warehouse eine solide Grundlage geschaffen werden kann, auf der es möglich ist, eine gewinnbringende ganzheitliche Geschäftsprozessoptimierung durchführen zu können.

Abstract

More and more companies focus on the establishment of business processes, that integrate various enterprise applications, instead of executing all business management functions separately. Therefore, business processes are formalized and modeled as workflows and are executed on a computer-based workflow management system. In this way, companies can react very quickly (and hopefully before their business rivals) to new market situations e.g. to new customer requests. For the best possible adaptation of workflows to new requirements and an effective business processes optimization a detailed analysis of the executed data is required. The analysis includes the time measurements of the process execution, but also input and output data that is referenced by the business process.

In order to keep the workflows truly competitive, a crucial aspect is missing in today's analytical approaches: the consideration of information from other enterprise applications, which was not integrated in the workflow itself, but still contains valuable data for modeling important decisions during process design. Nowadays, such global business process improvements are only possible by means of large manual effort in the integration and analysis of this additional information. Enabling such a comprehensive optimization involves therefore many new problems in these areas that have to be solved first: The semi-automatic matching of the business process data with operational data from other enterprise applications and their joint analysis with the help of new methods.

In this thesis concepts are presented to realize solutions for the integration and global analysis of this data. This kind of comprehensive analysis in a company is called here Business Impact Analysis (BIA).

The focus of this work is the integration for BIA. The new integration methods apply a rule set to workflow data and operational data according to a well-defined control strategy that determines the optimal sequence of rule applications in order to combine the data at schema level. For each combination, a similarity value between the combined data is calculated. When the similarity value is above a threshold value that was given by a user,

the combination result of the data is included into the result set as a new match. The rule set contains semi-automatic rules for combining data, which was previously annotated by semantic concepts from an ontology. Furthermore, it includes automatic rules for the matching of non- or only partially-annotated data. In addition, filtering rules are in the rule set that are responsible for adjusting the calculated similarity values according to the correlations in the structure of the respective combination elements. The combination and filtering rules are based on existing and new matching approaches.

The analysis approach for BIA in this work requires the creation of an integrated warehouse. This work introduces two architectures in order to build such a warehouse and it provides a detailed estimation of their applicability to BIA.

The analysis methods for BIA can be performed on both architectural approaches of the warehouse. The work develops SQL analysis operators to evaluate the integrated data in OLAP queries so that new knowledge becomes evident for business process optimization. Moreover, the work shows how the operators can be used in the mining process for BIA.

The integration, warehousing and analytical approaches are implemented in a prototypical implementation. In experiments, the effectiveness of the matching rules is examined and the gain in the quantity and quality of the match results compared to existing methods is explained. Further measurement scenarios show the usability of the warehousing and analytical approaches to BIA. Overall, it should be emphasized that this work can be used to create a solid foundation by the application of integration rules and analysis operators on the integrated warehouse, on which it is possible to conduct a profitable integrated business process optimization.

Abkürzungsverzeichnis

API	Application Programming Interface
BAM	Business Activity Monitoring
BIA	Business Impact Analysis
BOID	Business Object Identifier
BOIDelem	Element des Business Objects
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
CFP	Elemente des Prozesskontrollflusses in BIA
COM	Combination-Result: Ausgabe der BIA-Matchphase
$\mathbf{COM_{filtered}}$	Combination-Result: Ausgabe nach Kombination der Ähnlichkeitswerte
$\mathbf{COM_P}$	Combination-Result: Ausgabe der BIA-Pairingphase
$\mathbf{COM_{Str}}$	Combination-Result: Ausgabe der strukturellen BIA-Filterphase
CWM	Common Warehouse Metamodel
dBDD	deep Business Data Description
DBMS	Datenbankmanagementsystem
DWH	Data Warehouse
ERP	Enterprise Resource Planning
ETL	Extraction - Transformation - Load
KPI	Key Performance Indicator
OLAP	Online Analytical Processing
Ont	Ontologie
OWL	Web Ontology Language
P	Geschäftsprozesse in BIA
$\mathbf{P_{Ont}}$	annotierte Geschäftsprozesse in BIA
PV	Prozessvariablen in BIA
$\mathbf{PV_{Ont}}$	annotierte Prozessvariablen in BIA
$\mathbf{PV_{reduced}}$	reduzierte Menge der Prozessvariablen in BIA
RDF	Resource Description Framework

S	operative Schemaelemente in BIA
$\mathbf{S_{Ont}}$	annotierte operative Schemaelemente in BIA
SAWSDL	Semantic Annotation for WSDL
sim	Ähnlichkeitswert in BIA
$\mathbf{sim_P}$	Ähnlichkeitswert nach der BIA-Pairingphase
$\mathbf{sim_{Str}}$	Ähnlichkeitswert nach der BIA-Filterphase
SQL	Structured Query Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WFMS	Workflowmanagementsystem
WSDL	Web Service Description Language
WSML	Web Service Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Inhaltsverzeichnis

KAPITEL 1

Einführung

Die Integration und gemeinsame Analyse von Daten in heterogenen IT-Systemen gewinnt ständig an Relevanz. Sie ist bis heute eines der anhaltenden Probleme der Datenbankforschung [Hal03, BH08]. Der Grund dafür ist, dass meist in jedem Bereich eines Unternehmens unterschiedliche IT-Systeme verwendet werden und daher nahezu in jedem Bereich wichtige heterogene Geschäftsdaten anfallen [LN07]. Die Menge der Anforderungen an die Integration wächst ständig, da die Entwicklung neuer Technologien die Entwicklung neuer Integrationssysteme notwendig macht.

Solch eine neue Technologie ist die Automatisierung von Geschäftsprozessen, die die Interaktion zwischen verschiedenen Anwendungen eines Unternehmens spezifizieren. Die Integration der Ausführungsdaten dieser Geschäftsprozesse mit Daten aus anderen IT-Systemen eines Unternehmens und ihre ganzheitliche Analyse ist ein notwendiger Schritt, um die Geschäftsprozesse an die sich schnell ändernden Marktanforderungen anpassen zu können. Diese Integration und die dadurch mögliche ganzheitliche Geschäftsanalyse stehen im Fokus dieser Arbeit. Dieses Kapitel zeigt auf, warum die ganzheitliche Analyse für ein Unternehmen so bedeutend ist und stellt die einzelnen Schritte vor, die zur Durchführung der Analyse notwendig sind. Es stellt außerdem das Szenario vor, das die Grundlage für die Beispiele in den folgenden Kapiteln bildet, und gibt einen Überblick über den Aufbau der Arbeit.

Wie wichtig die Integration von Daten immer noch ist, zeigen Schätzungen in [LN07], die besagen, dass heute mehr als die Hälfte aller IT-Kosten auf die Integration von existierenden Systemen aufgewendet werden. Dieser Sachverhalt unterstreicht die Wichtigkeit dieser Arbeit.

1.1 Motivation

Immer mehr Unternehmen setzen auf die Etablierung flexibler Geschäftsprozesse. Geschäftsprozesse können Einzelanwendungen orchestrieren, d.h. ein Geschäftsprozess beschreibt eine Folge von Einzeltätigkeiten, die schrittweise ausgeführt werden, um ein geschäftliches oder betriebliches Ziel zu erreichen. Ein Geschäftsprozess kann Teil eines anderen Geschäftsprozesses sein oder andere Geschäftsprozesse enthalten bzw. diese anstoßen. Die Geschäftsprozesse sind meist in einer Beschreibungssprache wie z.B. der Business Process Execution Language (BPEL) modelliert und werden auf einem Workflowsystem zur Ausführung gebracht. Eine schnelle Anpassung und Optimierung der Geschäftsprozesse an neue Marktsituationen entscheidet oftmals über den Erfolg eines Unternehmens [WCL+05]. Für diese schnelle Anpassung und Optimierung benötigen wir eine effiziente und effektive Analyse der Ablaufdaten, um den Optimierungsbedarf zu erfassen, d.h. um unnötige Aktivitäten im Prozess zu erkennen und sie durch effizientere zu ersetzen. Die Prozessanalyse kann mit Hilfe der Ausführungsdaten einige Probleme aufdecken, die für die Optimierung wertvoll sind.

Zur effektiven Optimierung der Geschäftsprozesse sollte die Analyse jedoch auf das gesamte Unternehmen ausgedehnt und für die Analyse nicht nur Prozessdaten, sondern auch andere operative Geschäftsdaten aus eigenständigen Anwendungen berücksichtigt werden. Eigenständige Anwendungen sind Unternehmensanwendungen, die nicht in den zu optimierenden Geschäftsprozess eingebettet sind, aber dennoch für die Optimierung nützliche Informationen bereithalten. Unter diese Anwendungen fallen unter anderem ERP-Systeme oder Tabellenkalkulationen. Sowohl die Anwendungen in Geschäftsprozessen als auch die eigenständigen Anwendungen arbeiten auf Unternehmensdaten (Geschäftsobjekte der Prozesse bzw. operative Daten der eigenständigen Anwendungen). Dabei können sich die Unternehmensdaten in beiden Anwendungen auf das gleiche reale Objekt beziehen, wie es Abb. 1.1 darstellt.

Abb. 1.1: Korrelation zwischen Objekten einer Firma

In einer ganzheitlichen Analyse sollen alle Daten aus allen Datenquellen des Unternehmens gemeinsam ausgewertet werden. Es ist möglich, die Einflussfaktoren für ineffiziente Aktivitäten aus Geschäftsprozessen mit Hilfe zusätzlicher Informationen aus operativen Daten zu finden und dadurch noch mehr Optimierungspotenzial für die Prozesse zu entdecken. Die ganzheitliche Analyse ist bisher nur mit Hilfe aufwändiger manueller Integrationen zur Analysezeit möglich, da sich heutige Analyseverfahren im Unternehmen entweder auf die Prozessanalyse (Monitoring und Workflow Mining) oder auf Analysen

über operative Anwendungsdaten (Business Intelligence mit OLAP-Analysen und Data Mining) fokussieren.

Somit verzichten viele Unternehmen bei der Steuerung ihrer Geschäftsprozesse auf das Potenzial solcher umfassenden Analysen und führen die Analyse der Prozesse und der operativen Daten in getrennten Systemumgebungen durch. Die Prozessanalyse allein zeigt zwar auf, wie das Ergebnis und die Ausführungszeit mancher Aktivitäten bzgl. verschiedener Werte der Geschäftsobjekte variiert, z.B. der Bearbeitungszeit der Aktivitäten durch bestimmte Angestellte, aber sie gibt keinen Aufschluss darüber, welche Faktoren (z.B. Arbeitserfahrung oder Fortbildung) dabei eine Rolle spielen. Die Faktoren können evtl. durch Informationen aus den operativen Geschäftsdaten, die bereits in einem operativem Data Warehouse (DWH) konsolidiert sein können, herausgefunden werden. Auch eine rein operative Datenanalyse führt allerdings nicht zu einer Prozessoptimierung, da im DWH Anhaltspunkte fehlen, auf welche Geschäftsobjekte im Prozess sich die operativen Informationen beziehen. Außerdem fehlen Ablaufdaten zum Prozess wie z.B. die Dauer der Aktivitäten oder die ausgeführten Pfade bei Verzweigungen im Prozess. Werden Prozessdaten und operative Daten jedoch während des ETL (Extraktion, Transformation, Laden) zusammengeführt und in einem DWH für BIA integriert, können ganzheitliche, unternehmensweite Analysen einen wertvollen Beitrag zu weiteren Prozessoptimierungen leisten, indem Faktoren aus beiden Datenquellen berücksichtigt werden. Prozesseigenschaften wie die Prozesslaufzeit können nun bezüglich ihrer Kausalitäten organisiert werden. So kann nun u.a. geprüft werden, welchen Einfluss Arbeitserfahrung und Fortbildungen auf die Bearbeitungszeit bestimmter Prozessaktivitäten haben. Diese Berücksichtigung der operativen Daten aus eigenständigen Geschäftsanwendungen führt letztendlich dazu, dass die Geschäftsprozesse in größerem Umfang optimiert und angepasst werden können und das Unternehmen wettbewerbsfähig bleibt.

1.2 Zyklus einer Business Impact Analysis

Ein einleitender Überblick zum Thema dieser Arbeit, der Business Impact Analysis, ist in [RML08] gegeben. Geschäftsprozesse stehen im Zentrum von BIA und des Lebenszyklus, um die Geschäftsfunktionalität zu verbessern. Das BIA-Projekt unterstützt die notwendigen Schritte in jedem Teil des Zyklus bis hin zur finalen Optimierung der Geschäftsprozesse. Die Phasen und Komponenten des Zyklus werden in Abb. 1.2 skizziert und in diesem Kapitel kurz erklärt.

1.2.1 Vorstellung der Schritte des BIA-Zyklus

Die Optimierung von Geschäftsprozessen wird in der Literatur oft Business Process Re-Engineering genannt [DvdAtH05]. Hier werden die Geschäftsprozesse aus Unternehmenssicht analysiert und der Kontrollfluss des Prozesses wird ausgewertet. Das BIA-Projekt fokussiert Aspekte für Prozessoptimierungen, die darüber hinausgehen, und stellt dabei

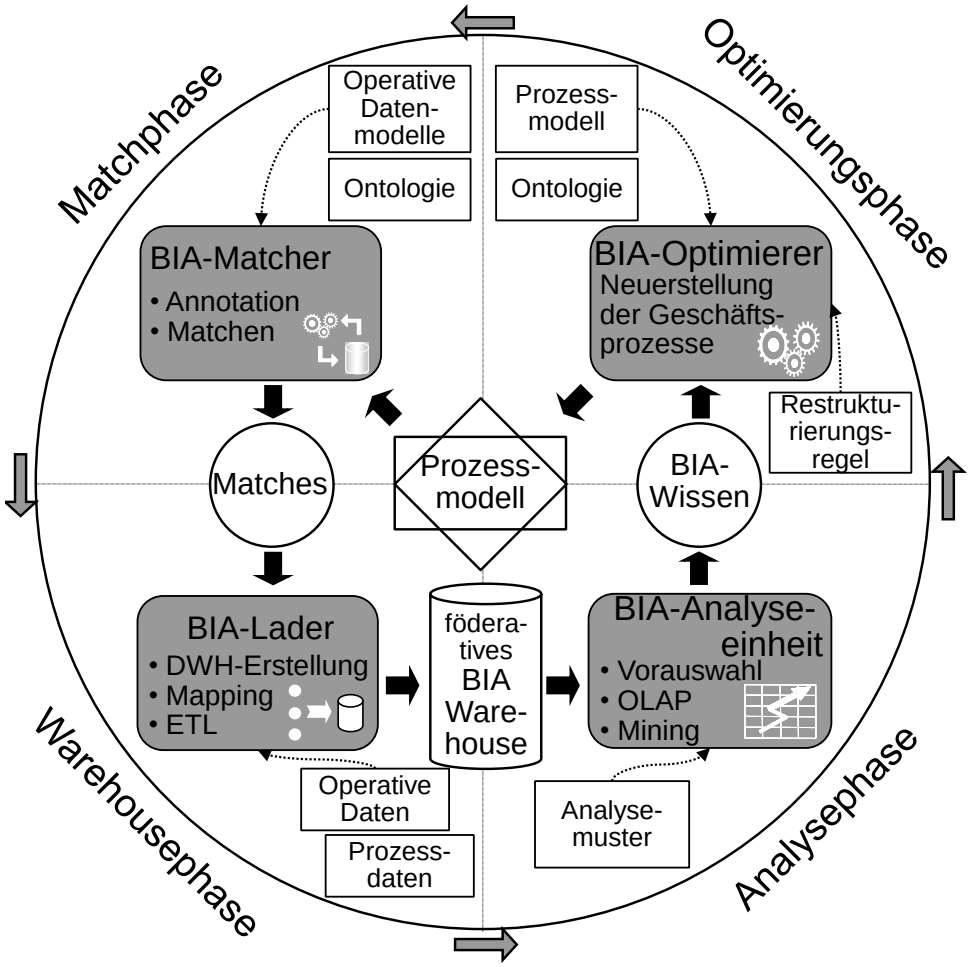


Abb. 1.2: Zyklus einer Business Impact Analysis

die operativen Daten als eine äußerst wichtige Informationsquelle für die Prozessoptimierung heraus. Um eine ganzheitliche Geschäftsanalyse und Optimierung durchführen zu können, durchläuft der Zyklus mehrere Phasen: Während der Matchphase werden die Prozessmodelle und operativen Datenmodelle kombiniert und gematcht. Die Matches werden in der Warehousephase für das ETL genutzt, um die Instanzdaten beider Modelle zu bereinigen und in einem BIA-Warehouse zu integrieren. Die anschließende ganzheitliche Analysephase deckt Hinweise über mögliche Optimierungen auf. Dieses Wissen ist die Grundlage für die Optimierungsphase, in der der Geschäftsprozess restrukturiert wird, um bessere Geschäftsergebnisse zu erzielen. Im Folgenden werden die Phasen genauer erläutert.

Matchphase

Die Aufgabe in der Matchphase ist es, die zugehörigen operativen Datenmodelle für alle relevanten Elemente im gegebenen Geschäftsprozess zu finden. Nachdem die Datenmodelle in den BIA-Matcher geladen wurden, werden sie auf die interne BIA-Matcher-Struktur

überführt. Wenn annotierte Modellelemente und ihre dazu verwendeten Ontologien vorhanden sind, die von manchen Regeln des BIA-Matchers verwendet werden, werden neben den Modellelementen auch ihre Annotationen und die referenzierten Ontologien in den BIA-Matcher geladen. Der Matcher ermöglicht es auch, solche semantischen Annotationen selbst durchzuführen. Schließlich wendet er verschiedene Regeln zum Matchen der Datenmodelle an. Die Matchphase mit dem BIA-Matcher und seine Komponenten, die Annotationsmechanismen und die Matchregeln, die für das Matchen der Datenmodelle verantwortlich sind, werden in den Kapiteln 3 und 4 dieser Arbeit vorgestellt.

Warehousephase

BIA muss große Mengen an Prozessausführungsdaten und operativen Stamm- und Transaktionsdaten bewältigen. Da die Datenquellen äußerst heterogen sind und sich nur in sehr wenigen Attributen überlappen und integrieren lassen, wird in dieser Phase kein konsolidiertes Warehouse gebildet, sondern die ganzen Quelldaten werden in den Quellsystemen belassen. Stattdessen wird eine föderative Warehouse-Architektur mit Mappings zwischen den Daten aufgebaut. Auf die Datenquellen kann über Standard-Wrapper zugegriffen werden, abhängig von den Audit Trails und den Speichersystemen der operativen Daten. Für die Erstellung des Mappings in der Föderationsebene unterstützt der BIA-Lader zwei Integrationsarten: das Mapping über eine einzige Matchtabelle, die die Matches aus der Matchphase enthält, oder über eine Bridgetabelle pro Match zwischen Prozessmodell und operativem Datenmodell. Diese zwei Warehouse-Architekturen werden in Kapitel 5 ausführlich diskutiert.

Die üblichen ETL-Schritte wie Datenextrahierung, Transformierung, Bereinigung (Cleansing) und Normalisierung werden vom BIA-Lader genutzt, um aufbereitete Eingabedaten für das BIA-Warehouse bereitzustellen. Basierend auf den Matches aus der Matchphase werden die betreffenden Attribute aus den Datenquellen im ETL integriert und im Warehouse gespeichert. Abhängig von der Wahl der Architektur sind verschiedene Transformationsschritte im ETL notwendig.

Analysephase

Das Ziel dieser Phase ist es, herauszufinden, welche Korrelationen für eine Prozessoptimierung wichtig sind. Dafür müssen alle Daten ganzheitlich analysiert werden. Prozessdaten wie z.B. die Ausführungszeiten, Wartezeiten und weitere Prozessattribute werden in Hinsicht des Prozessergebnisses analysiert, d.h. es wird beobachtet, welche Attribute für die Optimierung ausschlaggebend sind. Die BIA-Analyseeinheit berücksichtigt dabei auch die operativen Daten, die die Prozessattribute tiefer gehend beschreiben.

Da die Datenmengen im BIA-Warehouse gewaltig sind, nutzt die BIA-Analyseeinheit Analysemuster aus statistischen Auswertungen bereits durchgeführter Analysen, um bestimmte Attribute herauszufinden, die für die Optimierung eine wertvolle Rolle spielen könnten. Für BIA müssen viele Prozessdimensionen und entsprechend viele operative

Dimensionen beachtet werden. Um diese Dimensionalität ein wenig zu reduzieren, verwendet die BIA-Analyseeinheit diese Muster und entfernt Dimensionen und Attribute, die nicht in der Analyse betrachtet werden sollen. Einige Muster werden im BIA-Projekt als OLAP-Operatoren realisiert. Die Operatoren helfen dem Analysten bei der Formulierung von häufig gestellten Typen von Anfragen.

In der Analysephase ist es auch möglich, Standard-Mining-Techniken auf das BIA-Warehouse anzuwenden, um neue Einblicke in zusammenhängende Effekte zwischen Prozessen und operativen Daten zu erhalten. Die bereits erwähnten OLAP-Operatoren unterstützen die Mining-Algorithmen, um die integrierten Daten speziell für konkrete Mining-Techniken für BIA aufzubereiten. Die Analysephase wird neben der Warehousephase ebenso in Kapitel 5 erläutert.

Die Ergebnisse, die man durch die verschiedenen Analysen der BIA-Analyseeinheit erhält, müssen durch den Benutzer interpretiert werden. Dieses Wissen bildet die Grundlage für die nächste Phase.

Optimierungsphase

Die Optimierungsphase basiert auf dem Wissen, das durch die Interpretation der Ergebnisse der Business Impact Analysis erzielt wurde. Der BIA-Optimierer zielt darauf ab, dieses Wissen zu verwenden, um die Geschäftsprozesse so zu restrukturieren, dass ihre Ausführungszeit und ihr Ergebnis verbessert werden. Um das Prozessmodell erfolgreich umzuschreiben und dadurch zu verbessern, muss der BIA-Optimierer die richtigen Restrukturierungsregeln abhängig von den gegebenen Geschäftszielen herausfinden. Daher erhält der BIA-Optimierer das Optimierungsziel und alle Bedingungen, die während der Optimierung berücksichtigt werden sollen in Hinblick auf Kosten, Qualität, verwendeten Ressourcen und weiteren Geschäftsregeln. Der BIA-Optimierer identifiziert anwendbare Optimierungsmuster aus einem internen Katalog, um diese Geschäftsregeln zu erfüllen und entscheidet, welche Regel von diesen Mustern am besten passt und ausgeführt werden soll. Die Restrukturierungsregeln basieren auf den 'Best Practice'-Techniken für die Optimierung von Prozessen [DvdAtH05]. Die 'Best Practices' sind in diesen Regeln formalisiert, um ein hohes Maß an automatisierter Optimierung zu erreichen. Nach Bestätigung durch den Anwender restrukturiert der BIA-Optimierer automatisch den Geschäftsprozess bezüglich der gewählten Regel und erlaubt dem Anwender, zusätzliche Prozessänderungen vorzunehmen.

1.2.2 Ziele dieser Arbeit

Zur Realisierung der Phasen wurde in dieser Arbeit ein BIA-Framework entwickelt, das den Rahmen für die einzelnen Phasen des BIA-Zyklus stellt und die Schnittstellen zwischen den Phasen vorgibt. Im Framework werden durch diese Arbeit die ersten drei Phasen abgedeckt: die Match-, Warehouse- und Analysephase. Diese Arbeit soll die Grundlage dazu bilden, dass später die Umsetzung der gewonnenen Analyseerkenntnisse

in konkrete Optimierungsregeln erfolgen kann, um die Geschäftsprozesse neu zu strukturieren. Die letzte Optimierungsphase wird erst in zukünftigen Forschungsarbeiten [NRM10a] als neue Komponente hinzugefügt und ist nicht Bestandteil dieser Arbeit. Für die Umsetzung des Frameworks wurden verschiedene Ziele definiert. Die Anforderungen an diese Arbeit sehen nach Phasen gegliedert im Detail folgendermaßen aus:

Anforderungen an die Matchphase

- Eine Matchkomponente soll eingeführt werden, die neben der manuellen insbesondere die automatische Kombination zwischen Prozessdaten und operativen Daten unterstützt.
- Für die automatische Kombination von Geschäftsprozessen und operativen Daten sollen Matchregeln entwickelt werden.
- Diese Regelbasis soll u.a. neue Arten von Kombinationsregeln beinhalten, die in der Lage sind, operative Datenbankschemas mit Variablen aus Geschäftsprozessen zu integrieren. Die Standardtechniken aus dem Bereich Schema-Matching sind nicht ausreichend, da in dieser Arbeit nicht zwei ähnlich strukturierte Schemas gematcht werden sollen, sondern Schemaelemente mit Prozesselementen aus dem Audit-Schema. Der Inhalt und die Struktur der Schemas unterscheiden sich beträchtlich. Außerdem soll die Matchkomponente auch Informationen aus der Prozessstruktur für das Matching beachten. Daher müssen spezielle Regeln entwickelt werden, um diese Herausforderung zu meistern.
- Die Regelbasis soll außerdem semantische Informationen aus den zu matchenden Elementen interpretieren können, sodass die Regeln effizient und automatisiert Rückschlüsse auf verborgene Zusammenhänge ziehen können und eine gezielte Integration der Daten möglich wird. Für einen praktischen Einsatz unerlässlich sind ferner Methoden, um solche semantischen Annotationen möglichst aus bereits existierenden Beständen zu erhalten.
- Als Ergebnis der Matchphase soll u.a. eine Prozessbeschreibungssprache dBDD (deep Business Data Description) entstehen, mit der die Geschäftsobjekte mit weiteren Informationen aus operativen Daten angereichert werden können.
- Es sollen experimentelle Evaluierungen der Regeln durchgeführt werden, um ihre Effektivität zu beweisen.

Anforderungen an die Warehousephase

- Als Datenspeicher für die Prozessdaten aus dem Audit Trail sind Ausführungswarehouses notwendig und für die operativen Daten wiederum spezielle Warehouseschemas. Diese sind für BIA zusammenzuführen. Die Arbeit soll sich mit der Architektur des BIA-Warehouses beschäftigen.
- Das Warehouse soll mit Beispieldaten bestückt werden, um zu sehen, ob sich die Warehouse-Architektur für BIA eignet.

- Abschließend soll eine Evaluierung der Warehouse-Architektur bezüglich ihrer technischen Umsetzung und ihrer Benutzbarkeit erfolgen.

Anforderungen an die Analysephase

- Es sollen häufige Analysen (Muster) erdacht werden, die für BIA eine wichtige Rolle spielen.
- Die Muster sollen als OLAP-Operatoren konzipiert werden, um sie auf dem Warehouse ausführen zu können.
- In verschiedenen Fallbeispielen soll recherchiert werden, wie Mining-Techniken für BIA gewinnbringend eingesetzt werden können.

Die beschriebenen Anforderungen sollen nach ihrer Umsetzung in einer prototypischen Realisierung der Komponenten im BIA-Framework münden. Die Realisierung soll ein graphisches Endbenutzerwerkzeug umschließen, mit dem Geschäftsprozesse und operative Daten semantisch annotiert und gematcht werden können und der Match bei Bedarf in die Prozessbeschreibung mitaufgenommen werden kann. Außerdem sollen prototypisch eine für BIA angepasste Datenhaltung im BIA-Warehouse sowie die Analyseoperatoren für BIA umgesetzt werden.

Der Fokus der Arbeit liegt auf der Matchphase. Während die Konzipierung der Architektur des Warehouses ein weiterer wichtiger Punkt ist, sind sowohl die Entwicklung von neuartigen ETL-Prozessen, die auf die Anwendung in BIA spezialisiert sind, als auch neuartige Cleansing-Verfahren für die Datenversorgung des Warehouses nicht Bestandteil der Arbeit.

1.3 Beispielszenario

Das Beispielszenario zeigt einen Beispielprozess, der vereinfacht in Abb. 1.3 dargestellt ist, angelehnt an BPMN [OMG08]. Anhang A zeigt seine Umsetzung in ein BPEL-Workflowmodell. Die BPEL- und zugehörigen WSDL-Dateien sind dort ausschnittsweise dargestellt. Dieser Prozess soll optimiert werden, sodass teure und unnötig lang laufende oder oft abgebrochene Prozessteile identifiziert, analysiert und anschließend überarbeitet werden.

Der Prozess ist Teil eines Autovermietungsservices und beschreibt die Auswahl und Übergabe eines Leihautos. Er erhält die Eingabedaten durch die Aktivität *ReceiveCustomerData* mit Informationen über den Kunden und sein gewünschtes Automodell und seine Leihperiode und überprüft, ob ein solches Auto verfügbar ist, indem der Partnerdienst *RentalService* befragt wird. Wenn kein Auto im gewünschten Zeitraum verfügbar ist, führt ein Mitarbeiter den Human Task *ContractNegotiation* aus, um mit dem Kunden zu verhandeln, ob dieser nicht ein anderes Automodell oder einen anderen Zeitraum akzeptieren würde. Die Aufforderung, diesen Task auszuführen, wird nicht direkt irgendeinem Mitarbeiter zugeteilt, sondern einem, der die notwendige Rolle dafür hat. Das

bedeutet in diesem Prozess, dass *ContractNegotiation* von allen Angestellten der Abteilungen A, B und C ausgeführt werden darf. Wenn der Kunde keine Alternative akzeptiert bezüglich des Autos oder des Zeitraums, wird der Prozess abgebrochen. Ansonsten wird das Auto dem Kunden von einem Angestellten der Abteilung D in dem Human Task *CarHandOver* übergeben. Abb. 1.3 zeigt den Beispielprozess nur ausschnittsweise und natürlich besteht er noch aus mehr Komponenten und es gibt z.B. auch noch mehrere Prozesspfade zu den fehlerhaften oder fehlerfreien Endzuständen.

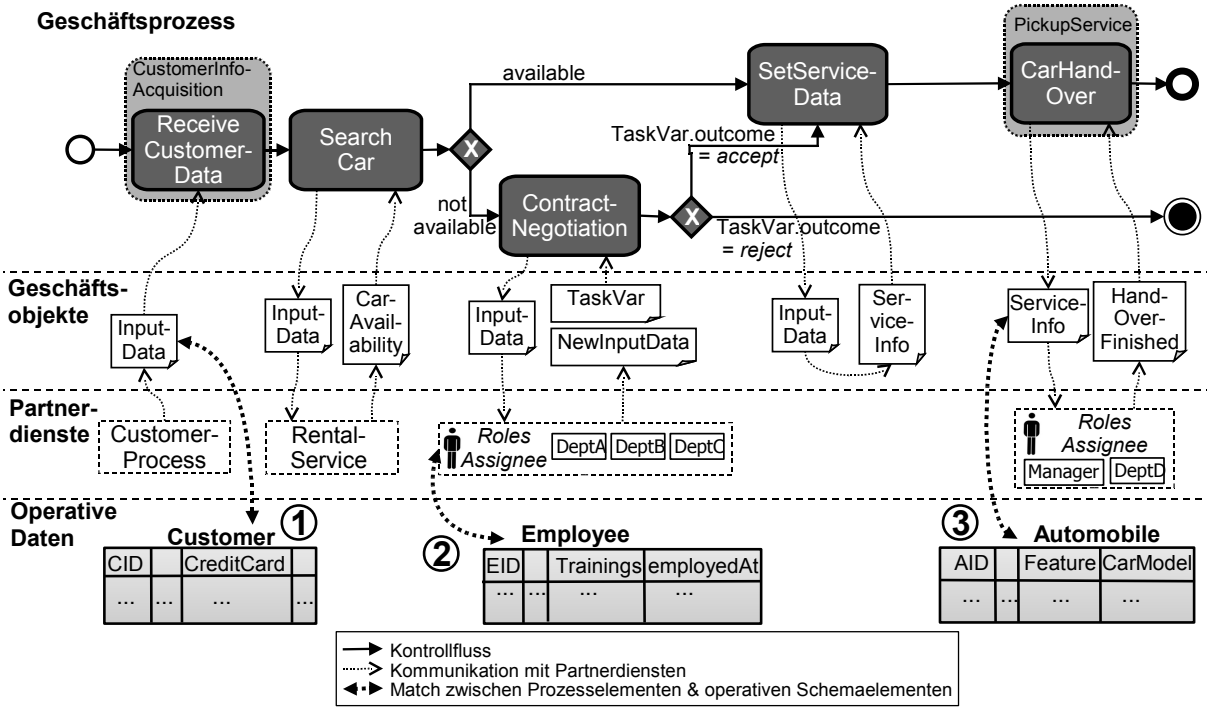


Abb. 1.3: Beispielszenario des Autovermietungsprozesses (Car Selection) in BPMN

Die operativen Daten in dem Szenario (hier beispielhaft drei Tabellen *Customer*, *Employee*, *Automobile*) beinhalten weitere nützliche Daten für die Optimierung. Sie enthalten Informationen über die Kunden, Angestellten oder Leihwägen. Elemente dieser Datenmodelle sollen mit den Elementen der Geschäftsobjekte der Prozessmodelle gematcht werden. Die Geschäftsobjekte werden im Prozess durch Variablen repräsentiert, die wiederum aus einzelnen Elementen bestehen. Die Elemente werden in Abb. 1.3 der Übersichtlichkeit wegen nicht dargestellt, können aber im Anhang A in Abb. A.4 eingesehen werden. Das Element *CustID* der Variable *inputData* wird so in (1) mit der Spalte *CID* der Tabelle *Customer* verbunden, die vorgegebenen Rollen der Aktivität *ContractNegotiation* mit der Spalte *employedAt* der Tabelle *Employee* in (2) bzw. der tatsächlich ausführende Angestellte der Aktivität *ContractNegotiation* aus dem Element *acquiredBy* der Variable *TaskVar* mit der Spalte *EID* der Tabelle *Employee* auch in (2) gematcht. Das Element *requestedCar* der Variable *ServiceInfo* wird in (3) mit der Spalte *CarModel* der Tabelle *Automobile* verbunden. In späteren Kapiteln wird beschrieben, wie das BIA-Framework

diese Matches herausfindet. Jeder Match ist das Ergebnis von der Anwendung spezieller Matchregeln, die u.a. Prozessinformation, wie z.B. Datenfluss und Kontrollfluss des Prozesses, berücksichtigen.

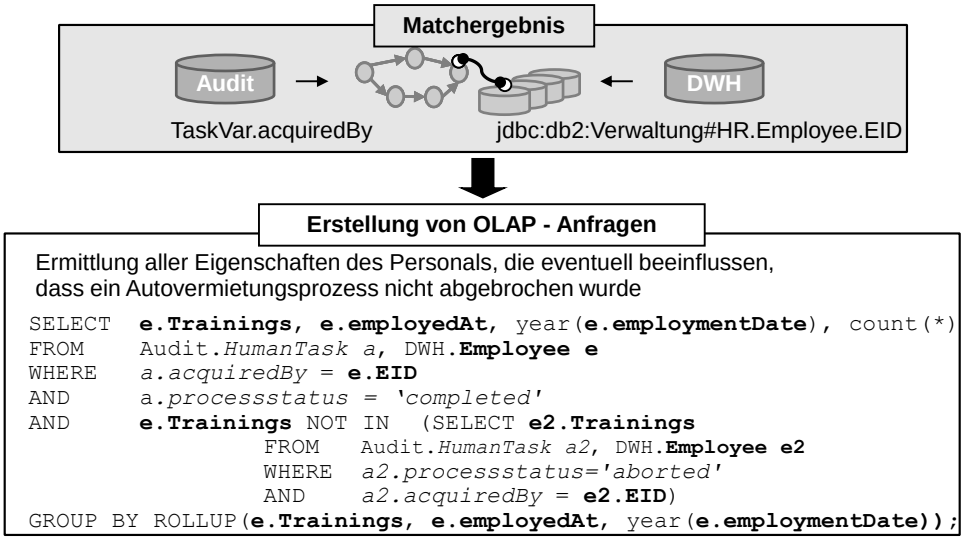


Abb. 1.4: Beispiel für eine OLAP-Analyse

In der Warehousephase werden diese Matches genutzt, um die Prozessdaten mit den operativen Instanzdaten für eine ganzheitliche Geschäftsanalyse zu integrieren. Ein Beispiel für einen Auszug einer solchen einfachen ganzheitlichen OLAP-Analyse ist in Abb. 1.4 dargestellt. Sie kann ausgeführt werden, da zuvor ein Match zwischen dem Element *AcquiredBy* im Human Task und der Spalte *EID* in der operativen Tabelle *Employee* gefunden wurde. Durch die OLAP-Analyse können z.B. die Fortbildungen (*Trainings*) der Angestellten herausgefunden werden, bei denen die Prozesse nicht abgebrochen, also profitabel für das Unternehmen sind.

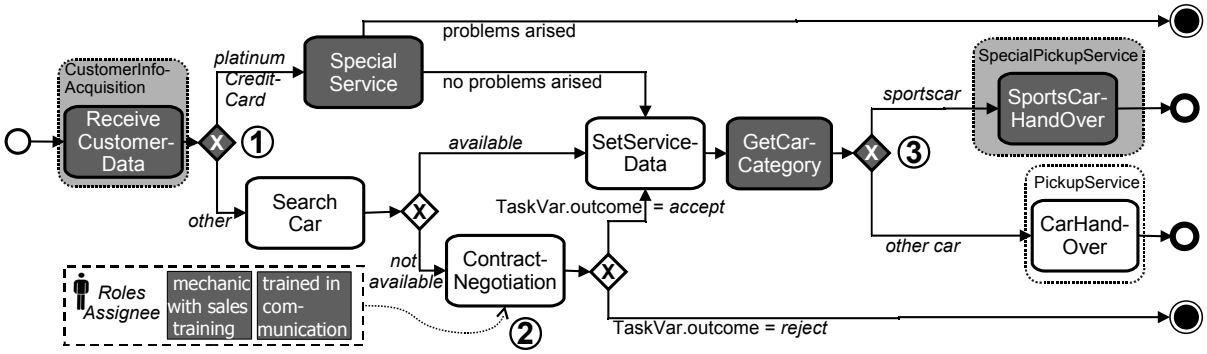


Abb. 1.5: Optimiertes Beispielszenario in BPMN

Die Analyse- und Optimierungsphase kann nützliche Zusammenhänge zwischen der Länge und dem Ergebnis eines Geschäftsprozesses und operativen Informationen eines

Geschäftsobjektes oder ausführender Ressourcen feststellen und so den Prozess optimieren, wie er beispielsweise in Abb. 1.5 angezeigt ist. Die neuen bzw. optimierten Komponenten sind in dunkelgrau dargestellt. In dem Beispielszenario könnte man in Abb. 1.3 (1) einen Zusammenhang zwischen dem Vermögen eines Kunden (Typ der Kreditkarte) und lang laufender Prozesse entdecken, d.h. dass bei Kunden mit Platinkreditkarte der Prozess oft eine überdurchschnittlich lange Ausführungszeit hat. Um diese wahrscheinlich wohlhabenden Kunden nicht zu verlieren, sollten sie in Abb. 1.5 (1) zu Services umgeleitet werden, die sich speziell ihren Bedürfnissen widmen. Das Ergebnis von *Contract-Negotiation* in Abb. 1.3 (2) hängt von dem jeweiligen Angestellten ab, der den Human Task ausführt, und seinen Fähigkeiten. Um die Anzahl der akzeptierten Aktivitäten zu erhöhen, ist eine Reorganisation dieser Ausführungsrollen in Abb. 1.5 (2) nach bestimmten Eigenschaften der Angestellten notwendig (*mechanics trained in sales*, *employees trained in communication*). In Abb. 1.3 (3) ergibt die Analyse eine Korrelation zwischen dem Modell des Leihwagens und der Ausführungszeit der Aktivität *CarHandOver*. Eine detaillierte Analyse zeigt, dass gewisse Automodelle bestimmtes Zubehör besitzen, das eine längere Einweisungszeit des Kunden benötigt, z.B. worauf man bei dem Fahren eines Sportwagens besonders achten muss. Diese Einführung sollte effizient ausgeführt werden mit einer Bereitstellung von geeigneten Angestellten (Ressourcen) in einer eigenen Aktivität wie in Abb. 1.5 (3) oder durch eine Ausgliederung von diesem Task *CarHandOver* an ein Subunternehmen.

1.4 Aufbau dieser Arbeit

Nach der Einführung in die Thematik einer Business Impact Analysis in diesem Kapitel wird in Kapitel 2 erläutert, welche aktuellen wissenschaftlichen oder kommerziellen Arbeiten für die einzelnen Phasen im BIA-Zyklus bereits bestehen und warum kein Ansatz existiert, der eine ganzheitliche Geschäftsanalyse ausführen kann. Außerdem werden die Begriffe Geschäftsprozesse und operative Daten klar definiert, auf deren Grundlage eine Business Impact Analysis durchgeführt wird.

Die folgenden drei Kapitel beschreiben detailliert die einzelnen Schritte der semantischen Annotation und der Kombination der Modelle, ihre Integration im Warehouse und die ganzheitliche Analyse der Daten. Kapitel 3 beschreibt die Strategie, die für die semantische Annotation der Geschäftsprozessmodelle und der operativen Schemamodelle gewählt wurde und welche Elemente der Modelle konkret damit annotiert werden. Des Weiteren wird gezeigt, wie die Annotation im Prototypen des Annotationseditors umgesetzt wird.

In Kapitel 4 werden verschiedene Arten der Kombination der Prozess- und operativen Datenmodelle definiert. Es wird das BIA-Framework mit Matchregeln vorgestellt, um diese Kombinationen effizient und effektiv durchzuführen. Die Matchregeln gliedern sich in drei Klassen: die semi-automatischen Regeln, die auf semantisch annotierten Modellen mit Hilfe einer Ontologie durchgeführt werden, und die automatischen Regeln, die keine Annotation voraussetzen. Die letzte Klasse enthält die Matchregeln, die die Struktur und den Kontext der Modelle verwenden und auf dieser Basis die bereits gefundenen

Matches noch feiner filtern und so den Korrektheitsgrad der Ergebnismenge steigern können. Zuletzt werden alle Matchregeln aus dem Kapitel evaluiert und ihre Umsetzung anhand des Prototyps im Matcheditor dargestellt.

Kapitel 5 zeigt, wie die Business Impact Analysis auf den integrierten Prozess- und operativen Datenmodellen durchgeführt werden kann. Zunächst wird das integrierte Warehouse vorgestellt, auf dessen Grundlage die Analyse ausgeführt wird. Anschließend folgen die Vorstellung der Analyseoperatoren, mit denen die ganzheitliche Geschäftsanalyse in OLAP unterstützt wird, und ihre Umsetzung im Prototypen des Analyseeditors sowie ihre Verwendung bei dem ganzheitlichen Prozess-Data-Mining.

Mit Kapitel 6 schließt die Arbeit. Es fasst noch einmal die wichtigsten Ergebnisse dieser Arbeit zusammen, bewertet das Einsetzen des BIA-Frameworks in einem Unternehmen und gibt einen Ausblick auf mögliche Erweiterungen des BIA-Frameworks. Des Weiteren werden im Ausblick Konzepte zur Prozessoptimierung vorgestellt, die durch die Ergebnisse dieser ganzheitlichen Geschäftsanalyse ermöglicht werden.

KAPITEL 2

Stand der Forschung

In diesem Kapitel wird der Stand der Wissenschaft dargestellt, soweit es für diese Promotion relevant ist. Zunächst werden grundlegende Begriffsdefinitionen für diese Arbeit bezüglich der Geschäftsprozesse und operativen Datenmodellen vorgestellt, die miteinander integriert werden sollen. Anschließend folgt die Abgrenzung zu existierenden Arbeiten, gegliedert in die Bereiche der verschiedenen Phasen des BIA-Zyklus.

2.1 Grundlagen

Die Integration und Analyse von Geschäftsprozessen mit Daten aus allen möglichen Anwendungen eines Unternehmens steht in dieser Arbeit im Vordergrund. Deshalb werden zunächst die Grundlagen der Geschäftsprozesse erläutert, bevor kurz auf die operativen Daten eingegangen wird. Die wesentlichen Punkte der Möglichkeiten für die Integration der Prozessmodelle und der operativen Modelle werden danach diskutiert.

2.1.1 Geschäftsprozessmodelle

Im Mittelpunkt des BIA-Zyklus (siehe Abb. 1.2) stehen die Geschäftsprozessmodelle, die optimiert werden sollen. Ein Geschäftsprozess beschreibt die Reihenfolge von einzelnen, logisch zusammenhängenden Aktivitäten, die ausgeführt werden, um ein betriebliches Ziel eines Unternehmens zu erreichen. Der Prozess kapselt die charakteristischen Aufgaben eines Unternehmens und wird somit normalerweise öfter ausgeführt. Ein wesentliches Merkmal eines Geschäftsprozesses ist die Bearbeitung von Material oder Informationen

(den Geschäftsobjekten), die in einer Ausgabe des Ergebnisses am Ende des Prozesses resultiert.

Die Geschäftsprozesse kapseln IT-Komponenten eines Unternehmens oder anderer Betriebe als Dienste und koordinieren diese, sodass sie als Anwendung dem Kunden oder anderen Organisationen zur Verfügung gestellt werden können. Diese Umsetzung der Geschäftsprozesse ist die Basis einer serviceorientierten Architektur (SOA) [WCL+05]. In der serviceorientierten Architektur sind typischerweise die einzelnen Dienste als Web Services realisiert, deren Operationen, Variablen und Austauschprotokolle zu anderen Diensten in einer WSDL-Datei beschrieben sind. Die Modellierung des Geschäftsprozesses selbst, der die Dienste koordiniert, erfolgt in einer Workflowsprache. Im BIA-Framework sind die Geschäftsprozesse in der Standardbeschreibungssprache BPEL (Business Process Execution Language) [OAS07] beschrieben. Die Geschäftsprozesse mit den einzelnen Diensten werden auf einem Laufzeitsystem, hier auf einem BPEL-WFMS, ausgeführt. Die Beschränkung auf BPEL ist sinnvoll, da BPEL sich über die letzten Jahre sowohl auf Anwender- als auch auf Herstellerseite als dominierender Standard für die Beschreibung von Workflows etabliert hat. Auszüge aus den Quelldateien (BPEL und WSDL) für das Autovermietungsszenario sind in Anhang A dargestellt. Neben ausführbaren Prozessen gibt es auch abstrakte Prozesse, die nur deren Verhalten beschreiben. Da die Analysephase jedoch auf den Ausführungsdaten abläuft, beschränkt sich diese Arbeit auf ausführbare Prozesse.

Ausgangsbasis für diese Arbeit sind die Daten, die während der Ausführung des Workflows in die so genannte Auditdatenbank geschrieben werden. Die Standardisierung der Datenformate in diesem Bereich (siehe [Wor98, LR00, zM04, NR04]) ist noch nicht sehr weit fortgeschritten. So muss für die sinnvolle Realisierung des Prototyps des BIA-Frameworks die Anzahl der BPEL-WFMS beschränkt werden. Dennoch enthalten sämtliche Auditdatenbanken der untersuchten BPEL-Engines (Oracle-Engine, IBM Process Server, ODE, Active BPEL) für jedes Ereignis im Prozess (z.B. Aufruf eines Web Services, Aktualisierung einer Variable) zumindest die folgenden Daten, die in den vier BIA-Phasen verwendet werden (siehe auch [Nie08]):

- ID des Workflowmodells
- ID der Workflowinstanz
- Typ und ID des Ereignisses
- Zeitstempel für den Zeitpunkt des Ereignisses
- Name und Typ der BPEL-Aktivität, welche das Ereignis ausgelöst hat (wenn die Aktivität einen Web Service aufgerufen hat, wird zusätzlich eine ID des Services angegeben)
- Name sowie der (neue) Wert der Variablen, wenn das Ereignis eine Variable verändert hat
- ID des Users oder interne ID des WFMS, das eine Aktion im Prozess angefordert hat

Neben den Daten im Audit Trail sind für die Matchphase jedoch erst einmal die Metadaten über das Prozessmodell wichtig, das mit den operativen Daten verbunden werden soll. Ein Prozessmodell kann als Tupel *P = (A, PV, CS, H, PD)* charakterisiert werden, das so zwar stark eingeschränkt ist, aber alle nötigen Informationen für das BIA-Framework enthält. *A* repräsentiert die Menge der Aktivitäten im Prozess, *PV* sind die Menge der Prozessvariablen, *CS* repräsentiert verwendete Correlation Sets, um Nachrichten von einer Instanz des einen Web Service zu der richtigen Instanz eines anderen Web Services weiterzuleiten. *H* sind Routinen, die bei einem Fehler als Ersatzprogramm bzw. bei einem speziellen Ereignis aufgerufen werden, um einen plötzlich auftretenden Fehler bzw. ein Ereignis behandeln zu können. *PD* deckt die Partnerdienste ab, mit denen der Prozess interagiert. Für diese Arbeit stehen in erster Linie die Prozessvariablen im Vordergrund, aber jede Phase benötigt zudem auch Kenntnisse über die anderen Komponenten aus dem Prozessmodelltupel. Daher werden im Folgenden basierend auf [OAS07, WCL+05] alle Teile des Tupels diskutiert. Tabelle 2.1 zeigt einen Überblick über die Komponenten und ihre Ausprägungen. Außerdem werden die Teile jeder Komponente diskutiert, die für die Phasen in BIA von Belang sind. Im Detail wird diese Verwendung in Kapitel 4 und 5 erläutert.

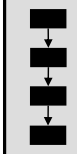
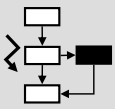
	Komponente	**Ausprägungen**	**Verwendung in BIA**
	Aktivität *A*	elementare Aktivitäten strukturierte Aktivitäten	Name und Typ der Aktivität, Kontroll- und Datenfluss mit der Art der Manipulation, Ausführungsdauer, Aktivitäten für Fehlerbehandlung
	Prozessvariable *PV*	Variable, <message>, <part>, XML-Schematyp, XML-Element	Name, Datentyp, Wert, interne Struktur, Annotation
	Correlation Set *CS*	<property>, <propertyAlias>, <correlationSet>	Name und Definition des CS, Name und Datentyp von <property>, Definition von <propertyAlias>
	Handler *H*	Compensation-, Fault- und Event-Handler	Semantik der Compensation-, Fault- und Event-Handler
	Partnerdienst *PD*	<portType>, <partnerlink> mit seinen Rollen und seinem Typ	Name aller Komponenten, Annotation von <portType>, Dauer und Fehler der PD

Tabelle 2.1: Wichtige Prozesskomponenten für BIA und ihre Ausprägungen

Aktivität A

Jede Aktivität wird durch einen lokal eindeutigen Namen beschrieben. BPEL-Aktivitäten sind in zwei Klassen unterteilt: elementare und strukturierte Aktivitäten. Elementare Aktivitäten beschreiben grundlegende Schritte des Prozessverhaltens und sind atomar, d.h.

sie sind nicht aus anderen Aktivitäten aufgebaut. Strukturelle Aktivitäten enthalten andere Aktivitäten und definieren den Kontrollfluss zwischen ihnen. Mit den elementaren Aktivitäten stehen verschiedene Aktivitätstypen zur Auswahl. Einige Aktivitäten modellieren die Interaktionen mit Web Services: <*receive*>, <*invoke*> und <*reply*> erlauben es dem Prozess, Nachrichten mit Web Services auszutauschen, die er koordiniert. <*pick*> wartet darauf, dass ein speziell definiertes Ereignis eintritt und führt die dafür modellierten Schritte aus. <*receive*>, <*reply*> und <*pick*> beschränken sich bei der Interaktion auf Operationen, die der Prozess selbst als Web Service anbietet. <*invoke*> dagegen referenziert Operationen, die der Partnerdienst dem Prozess anbietet. Die Datenverarbeitung, die in dem Partnerdienst stattfindet, und sein Kontrollfluss sind dabei für das Workflowmodell nicht sichtbar. Mehr dazu im Abschnitt, der die Partnerdienste erläutert. Die elementaren Aktivitäten dienen ferner zur Modellierung von Datenmanipulationen. Der Datenfluss wird nicht explizit modelliert, sondern die Daten werden als Variablenwerte von der <*assign*>-Aktivität bearbeitet. <*assign*> ist so definiert, dass sie Daten von einer Quelle zu einer Zielvariable kopiert. Die Quelle kann ein Literal, ein Ausdruck, ein Wert einer anderen Variable oder ein Wert eines Teils einer anderen Variable sein.

Zusätzlich gibt es noch einige weitere elementare Aktivitäten, die zwar keine matchbaren Geschäftsobjekte beinhalten, aber wertvolle Informationen für die Analyse- und Optimierungsphase liefern. <*wait*> erlaubt es, die Prozessausführung für ein gewisses Zeitintervall zu unterbrechen und <*exit*> beendet die Prozessausführung. <*throw*> wird im Zusammenhang von der Fehlerbehandlung verwendet und später bei den Handler-Komponenten erläutert. Der Vollständigkeit halber seien noch die Aktivitäten <*empty*> und <*extensionActivity*> genannt, die aber für die in BIA verwendeten Prozesse keine Relevanz haben. <*empty*> ist eine leere Aktivität und kann als Platzhalter für spätere Aktivitäten verwendet werden. Mit <*extensionActivity*> können eigene Aktivitäten definiert werden (auch mehrere, dann gehört sie allerdings zu den strukturierten Aktivitäten).

Die strukturierten Aktivitäten definieren den Kontrollfluss des Prozesses: <*sequence*> ist eine Sequenzierung von Aktivitäten, <*if*> stellt ein mehrzweigiges Entscheidungskonstrukt zur Verfügung, <*while*>, <*repeatUntil*>, <*foreach*> sind Schleifenkonstrukte. Die Strukturierung der Aktivitäten erfolgt entweder blockbasiert durch die Verschachtelung elementarer oder strukturierter Aktivitäten oder graphenbasiert durch das Verwenden der <*flow*>-Aktivität. <*flow*> verbindet die Aktivitäten über gerichtete Kontrollflusskanten. Die Aktivitäten werden durch eine <*scope*>-Aktivität strukturiert, die einen Ausführungsbereich definiert. In diesem Bereich können eigene Variablen, Partnerlinks, Correlation Sets und Handler deklariert werden, die nur innerhalb dieses Bereichs sichtbar sind.

Je nach Phase des BIA-Zyklus werden verschiedene Attribute und Merkmale der Aktivitäten verwendet. Für die Matchphase sind besonders folgende Teile interessant: Name und Typ aller elementaren und strukturierten Aktivitäten, insbesondere der strukturierte Aufbau der Namen der Aktivitäten, falls der Prozess blockbasiert strukturiert ist, um dadurch den Kontext der Variablen zu erfassen. Außerdem ist wichtig, wie der Datenfluss modelliert ist und dabei die Manipulation der Geschäftsobjekte konkret aussieht. Für die

Analysephase spielt dann neben diesen Merkmalen auch noch die Ausführungszeit der Aktivitäten und die Aktivitäten eine Rolle, die im Zusammenhang mit Fehlern auftreten.

Prozessvariable PV

Variablen werden je nach Aktivität gelesen, modifiziert und weitergegeben. Die Prozessvariablen speichern die Daten zu den Geschäftsobjekten, die im Prozess bzw. in den aufgerufenen Partnerdiensten bearbeitet werden. Sie beinhalten Eingabe- und Ausgabewerte der Benutzer, der aufgerufenen Dienste oder Anwendungen, aber auch Zwischenwerte für weitere Berechnungen. Außerdem können sie interne Informationen zur Steuerung des Prozesses speichern. Sie werden in der Orchestrierungsebene von BPEL definiert. Sie können aus komplexen oder elementaren XML Schematypen bestehen oder ihr Typ ist in einer WSDL-Datei des zugehörigen Prozesses oder Partnerdienstes definiert [Pap08]. Eine Variable mit komplexem Datentyp ist gemäß ihrem Datentyp hierarchisch geschachtelt. Deshalb kann sie als Baum dargestellt werden bestehend aus Knoten, die durch den Benutzer oder durch das jeweilige definierende Konstrukt benannt sind, wenn der Name fehlt. Die Konstrukte in einer BPEL bzw. WSDL-Datei sind in Abb. 2.1(a) beispielhaft dargestellt und im Folgenden aufgelistet:
(i) *Variablenknoten*: BPEL benutzt einen Message-Type als Variablendeklaration in Abb. 2.1 (nur in WSDL 1.1). Des Weiteren gibt es auch Deklarationen direkt mit einfachem oder komplexem XML-Schematyp oder XML-Schemaelement. Die Struktur der kompletten Variable hängt von dieser Deklaration ab. In Abb. 2.1 ist die Variable *inputData* durch den Message-Typen *input* deklariert.
(ii) *Message-Type-Knoten*: Die Variablen definieren Nachrichten, die von den Partnerdiensten des Prozesses ausgetauscht werden. Die Nachrichten bestehen aus einem oder mehreren Part-Knoten.
(iii) *Part-Knoten*: Jeder Teil der Nachricht ist mit einem XML-Schematyp oder XML-Schemaelement assoziiert. In Abb. 2.1 ist der Part *CustomerData* durch ein Element *CustComplexType* deklariert.
(iv) *XML-Schematyp*: Der Typ eines XML-Schemas kann komplex definiert sein und besteht aus einer Sequenz, Alternative (choice) oder Konjugation. Oder das Schema ist als einfacher Typ definiert und besteht aus einer Beschränkung, Auflistung oder Vereinigung von anderen einfachen Schematypen oder aus einer primitiven Schemadefinition mit einfachen Datentypen wie z.B. *Integer* des Elements *custID* in Abb. 2.1.
(v) *XML-Schemaelement*: Jedes Element der Prozessvariable hat einen Namen wie z.B. *custID* in Abb. 2.1 und ist als komplexer oder einfacher XML-Schematyp definiert.

Um die einzelnen Knoten des Prozessvariablenbaumes für die späteren Phasen des BIA-Zyklus zu identifizieren, werden die Namen jedes Knotens zusammen mit seinem Prozessnamen und ggf. Bereich im Prozess (<*scope*>, siehe Abschnitt für Aktivität) verwendet, wo die Variable deklariert wurde. Abb. 2.1(b) zeigt den Baum, der die Prozessvariable *inputData* aus Abb. 2.1(a) für BIA darstellt. Jeder Knoten wird dabei gemäß dem Preorder-Durchlauf des Variablenbaumes indiziert (erst Vater, dann linker Sohn, dann rechter Sohn) [AL91]. Formal lässt sich ein Element einer Prozessvariable mit Definition 1 beschreiben:

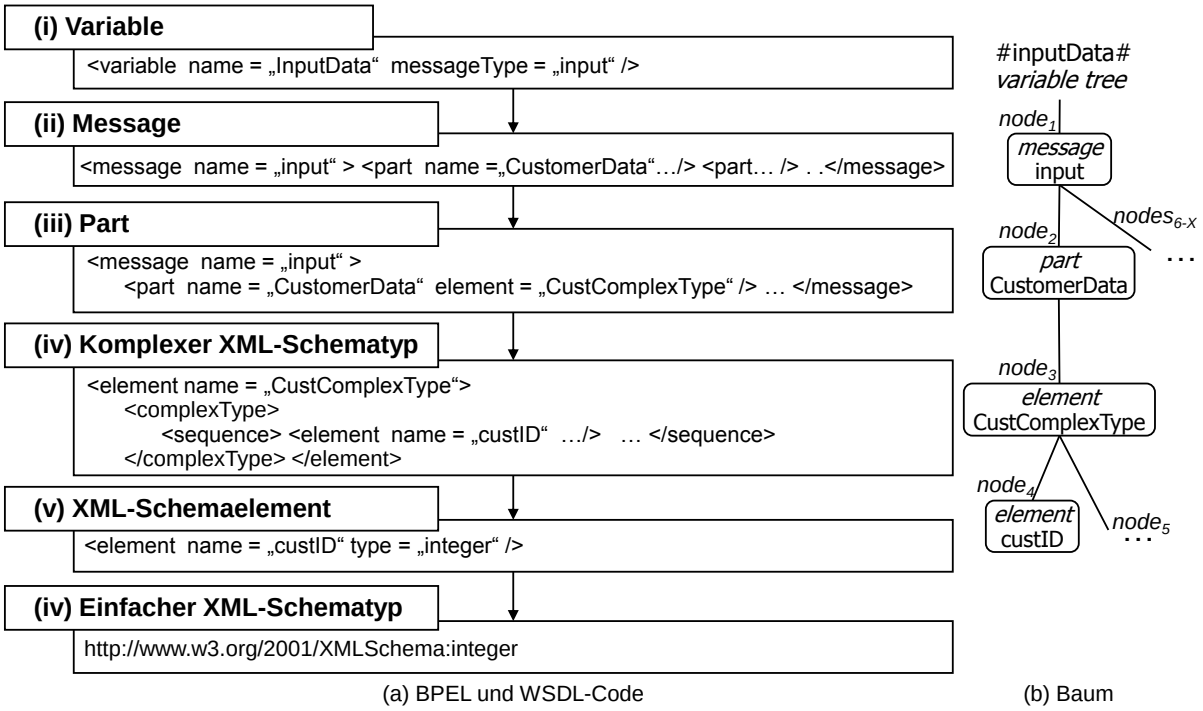


Abb. 2.1: Komponenten der Geschäftsprozessvariable

Definition 1 (Repräsentation eines Elements in einer Prozessvariablen) *$P_i.pv_j.node_m \in PV$ sei ein Element einer Variablen pv_j aus der Menge der Prozessvariablen PV und seinem Prozessbereich P_i, in dem es deklariert wurde, und seiner Position m, in der das Element als Knoten (node) im Variablenbaum gemäß der Suche nach dem Preorder-Durchlauf gefunden werden kann.*

Aus Gründen der Anschaulichkeit wird ein Prozesselement jedoch in allen folgenden Beispielszenarien durch seinen Prozessnamen und seinen Pfad von der Wurzel des Variablenbaumes der Variable, die das Element benutzt, bis hin zu dem Element selbst dargestellt. Das Element *custID* aus $CarSelection.inputData.node_4$ in Abb. 2.1 wird also durch $CarSelection.inputData.input.CustomerData.CustComplexType.custID$ beschrieben.

Im BIA-Zyklus sind die Variablen besonders wichtig. Neben dem Wert und möglichen semantischen Annotationen der Variablen (vgl. Kapitel 3), sind die Datentypen und die Struktur des Variablenbaumes für den Kontext der Variablenelemente für die Matchphase wichtig. Der Name für die Prozessvariable spielt dabei für ganz viele Matchregeln eine bedeutende Rolle. Dieser Name und auch der belegte Wert während der Ausführung werden auch in den anderen Phasen des BIA-Zyklus Verwendung finden.

Correlation Set CS

Ein Correlation Set ist eine Menge von *<property>*-Elementen, die insbesondere in asynchronen Prozessen verwendet werden, um den Nachrichtenaustausch zwischen einer konkreten Prozessinstanz und einer Partnerdienstinstanz zu steuern. In BPEL werden

die vielen Prozessinstanzen nicht per Identifikationsnummer adressiert, sondern es werden Nachrichten an die Instanzen der Partnerdienste zusammen mit den Werten der <*property*>-Elemente in den definierten Correlation Sets geschickt. Nach Beendigung des Dienstes wird der korrespondierende Prozess über die Correlation Sets und die Werte der <*property*>-Elemente identifiziert und an diesen ggf. eine Nachricht zurückgesendet.

<*property*>-Elemente sind Mittel, um auf Teile aus WSDL-Nachrichten zuzugreifen. Die Definition einer <*property*> besteht aus zwei Teilen: einer <*property*> selbst, die einen Namen sowie einen Message-Type oder XML-Schemadatentyp deklariert, und einem <*propertyAlias*>, welcher definiert, welchen Teil der WSDL-Nachricht <*property*> beschreibt (siehe Anhang A). Mit einem <*property*>-Element ist es möglich, die Prozesslogik von den Details der Variablendefinition zu trennen, sodass bei der Änderung der Datentypen im Prozess selbst nichts geändert werden muss.

Die Correlation Sets sind im BIA-Zyklus in der Matchphase von Bedeutung. Es werden dabei der Name und die Struktur des Correlation Sets, d.h. ihre zugewiesenen Propertys verwendet. Außerdem sind die Namen und Datentypen von <*property*> interessant sowie die Definition des <*propertyAlias*>. Diese Komponente beschreibt, auf welchen Teil des komplexen Variablentyps sich das <*property*>-Element bezieht und ob noch weitere Variablentypen mit der gleichen <*property*> referenziert werden.

Handler H

BPEL unterstützt Compensation-, Fault- und Event-Handler. Event-Handler reagieren auf Ereignisse, wann immer diese während der Ausführung des Scopes auftreten, in dem der Handler definiert ist. Zwei typische Ereignisse sind der Empfang einer neuen Nachricht oder der Ablauf eines Zeitstempels. Beide Ereignisse können den sofortigen Abbruch eines Prozesses oder die Ausführung anderer Aktivitäten nach sich ziehen.

Da Geschäftsprozesse oft eine lange Ausführungszeit besitzen und viele Variablen bearbeitet werden müssen, ist eine Fehlerbehandlung meist schwierig. Die Einhaltung von ACID-Eigenschaften (Atomarität, Konsistenz, Isoliertheit und Dauerhaftigkeit von Transaktionen im Prozess) ist nur auf lokale Update-Aktivitäten beschränkt, da die Sperrung und Isolation nicht solange aufrechterhalten werden sollte. Das bedeutet aber, dass nach dem Commit die Transaktion auf eine andere Weise rückgängig gemacht werden muss, falls später nach dem Commit ein Fehler im Prozess auftritt. In BPEL gibt es dafür den Compensation-Handler, der die spezifizierte Aktivitäten in <*compensate*> im Fehlerfall ausführt und so die ausgeführten Aktivitäten des betroffenen Scopes, wo der Fehler auftrat, rückgängig macht.

Der Fault-Handler behandelt Fehler, die durch Probleme bei der Prozessausführung oder explizit durch eine <*throw*>-Aktivität generiert wurden, die den Fehler genauer spezifiziert. Für die Fehlerbehandlung kann mit <*catch*> für jeden bestimmten Fehler eine Reaktion definiert werden. <*catchAll*> behandelt alle restlichen Fehler eines Scopes. Falls <*catchAll*> fehlt und ein Fehler nicht behandelt werden konnte, wird er an den Vaterscope propagiert.

Für die Matchphase im BIA-Zyklus sind die Handler weniger interessant, außer sie haben eigene Variablen definiert, denen Geschäftsobjekte aus dem Prozess zugewiesen werden und die das Matchen dieser Geschäftsobjekte unterstützen können. Für die Analysephase sind die Handler jedoch äußerst bedeutend, da manche Ereignisse und insbesondere Fehler analysiert werden sollten, um sie durch die Prozessoptimierung zu vermeiden.

Partnerdienste PD

BPEL kapselt Anwendungen u.a. zur Verarbeitung von Geschäftsobjekten in Partnerdienste, die in dem Prozess mit einer <*invoke*>-Aktivität aufgerufen werden. Auf genaue Informationen zu den ausgeführten Operationen zur Datenverarbeitung kann im Prozess dabei nicht zugegriffen werden und für die Matchphase also nicht verwendet werden. Die WSDL-Datei des Partnerdienstes beschreibt in der Komponente <*partnerlink*> jedoch die Schnittstellen. <*partnerLinkType*> gibt den Rollentyp der Interaktion an und mit <*portType*> (<*interface*> in WSDL 2.0), welche Operationen und Nachrichten der Prozess zur Verfügung stellt.

Für die Matchphase in BIA sind die Namen aller erwähnten Konstrukte und mögliche semantische Annotationen von <*portType*> wichtig, um das Matching zu unterstützen. In der Analysephase interessieren besonders die Dauer oder mögliche Fehler der Dienste, um den Prozess zu optimieren.

Eine spezielle Form von Partnerdiensten sind Human Tasks. Sie binden die menschliche Interaktion in den Geschäftsprozess ein. Ergänzungen für die Umsetzung des Human Tasks werden von der BPEL-Erweiterung WS-BPEL4People [KKP+05] zur Verfügung gestellt. Das Verwalten der Benutzer, ihre Zuweisung und ihre Rechte zu einer Aufgabe werden durch das WFMS geregelt und ist in einem Partnerdienst gekapselt. Die Bestimmung der Benutzergruppen und Rechte der einzelnen Benutzer erfolgt über ein Identifizierungsverzeichnis, in den meisten Fällen LDAP (Lightweight Directory Access Protocol), oder auch über andere Benutzerverzeichnisse wie z.B. über JAZN (Oracles Java AuthoriZatioN). Der Human Task stellt eine Komponente des Workflows dar, bei dem mit relativ wenig Matching- und Analyseaufwand große Optimierungssteigerungen erzielt werden können. Deshalb wird der Human Task im Rahmen dieser Arbeit immer wieder auftauchen.

2.1.2 Operative Datenmodelle

Unter operativen Datenmodellen werden Modelle von Daten verstanden, die aus Datenquellen der verschiedenen Unternehmensanwendungen stammen. Dabei ist eine Datenquelle ein beliebiger Datenbestand mit Inhalten, die für die Analysezwecke eines Unternehmens dienlich sind [BG04]. Die Daten stammen aus allgemeinen Anwendungsprogrammen für Operationen sowohl auf Stammdaten als auch auf Bewegungsdaten des Unternehmens. Streng genommen ist auch ein Geschäftsprozess eine Anwendung, die auf diesen Datenbeständen arbeitet. Zur Abgrenzung der operativen Daten in Geschäftsprozessen

wird in dieser Arbeit klar zwischen den Geschäftsobjekten in den Prozessvariablen (siehe Kapitel 2.1.1) der Prozesse, die optimiert werden sollen, und sonstigen operativen Datenmodellen im Unternehmen, die dafür als zusätzliche Informationsquelle dienen, unterschieden. Die Verwaltung dieser Datenbestände erfolgt zum einen durch ein Datenbankmanagementsystem (DBMS). Zum anderen können die Daten auch in Dateien oder anderen proprietären Datenformaten, wie z.B. flachen Dateien oder XML-Dokumenten, verwaltet werden.

Um den Zugriff und Austausch heterogener Daten im Hinblick auf das Warehousing und der Analyse zwischen den Software Tools und Repositorys zu erleichtern, wurde das Common Warehouse Metamodel [PCTM03] (CWM) von der Object Management Group entwickelt und als Standard eingeführt. Das CWM basiert auf einem UML-Modell und benutzt XMI (XML Metadata Interchange) als Metadatenaustauschformat. Um ein spezifisches Schema eines Anbieters zu beschrieben, reicht CWM jedoch oft nicht aus. Daher stellt CWM zwei Erweiterungsmöglichkeiten zur Verfügung:

- *Simple Extension*: Diese Erweiterung erlaubt es, Stereotypen und TaggedValues zu jedem Modellelement hinzuzufügen. Stereotypen sind eigene Benennungen für Kategorien, die eine zusätzliche Information über das jeweilige Element beinhalten. TaggedValues sind Bezeichner-Werte-Paare, die einem Element hinzugefügt werden und weitere Attribute zu dem Element enthalten. Das Attribut *Tag* enthält dabei den Namen von dem zusätzlichen Attribut und *Value* beschreibt den zugehörigen Wert. Der Datentyp des Wertes ist immer ein String.
- *Modeled Extension*: Diese Erweiterung geht über die vorherige hinaus. Hier wird die CWM-Definition durch objektorientierte Techniken erweitert. Entweder wird der Code, der beschreibt, wie die neue Erweiterung bearbeitet werden soll, direkt in die initiale Implementierung von CWM eingegliedert oder es wird ein Modellierungswerkzeug verwendet, um die Erweiterung graphisch in das UML-Modell einzubinden.

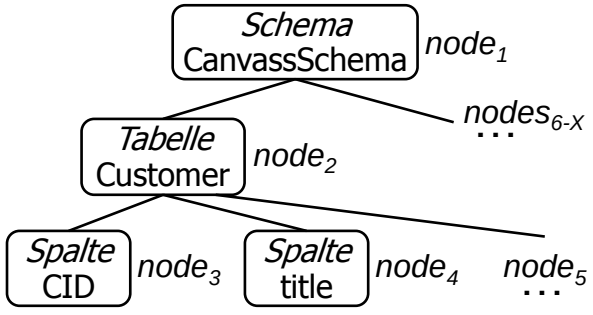


Abb. 2.2: Darstellung des operativen Schemas als Baum

Ein operatives Schema ist wie die Prozessvariablen hierarchisch geschachtelt und kann als Baum dargestellt werden. Um die einzelnen Knoten des Schemabaumes für die späteren Phasen des BIA-Zyklus zu identifizieren, werden die Namen jedes Knotens zusammen mit seinem Schemanamen verwendet, das das operative Element enthält. Abb. 2.2 zeigt den Baum, der die Tabelle *Customer* aus dem Schema *CanvassSchema* mit den Spalten

CID und *title* darstellt. Formal lassen sich die Elemente operativer Datenmodelle mit Definition 2 beschreiben:

Definition 2 (Repräsentation eines Schemaelements)
$S_s.node_r \in S$ sei eine Spalte einer Tabelle oder ein XML-Element eines operativen Schemas S. Die Spalte bzw. das Element sei durch seine Position r gekennzeichnet, in der die Spalte bzw. das Element als Knoten (node) im Schemabaum S_s gemäß der Suche nach der Preorder-Strategie gefunden werden kann.

Ach hier wird aus Gründen der Anschaulichkeit ein Schemaelement in allen folgenden Beispielszenarien durch seinen Schemanamen und seinen Pfad von der Wurzel des Schemabaumes bis hin zu dem Element selbst dargestellt. Das Element *CID* aus $CanvasSchema.node_3$ in Abb. 2.2 wird also durch $CanvassSchema.Customer.CID$ beschrieben.

2.1.3 Korrelation zwischen Prozessdaten und operativen Daten

Das Auftreten von Daten im Prozess, die sich auf das gleiche reale Objekt beziehen wie andere operative Schemamodelle im Unternehmen, kann in vier Korrelationsarten klassifiziert werden. Diese Klassifikationen sind in Tabelle 2.2 dargestellt. Allen Ansätzen gemein ist die Erkenntnis, dass diese Korrelationen in einer ganzheitlichen Geschäftsanalyse einen wertvollen Gewinn bringen können und die daraus entstehende Diskussion, ob und wie sie für BIA verwendet werden sollen. Die Tabelle zeigt, ob die Korrelation zwischen Geschäftsobjekt und operativen Elementen explizit im Prozess geschieht oder nicht modelliert ist, ob die notwendigen Komponenten für die Korrelation für den Anwender sichtbar sind und ob es überhaupt möglich ist, einen automatischen Match zwischen den Datenmodellen herzustellen. Im Folgenden werden die Klassifikationen und ihre Verwendung in BIA erläutert:

i. SQL in Partnerdienst: Strenggenommen erfolgen in einer serviceorientierten Architektur Datenzugriffe nur in Partnerdiensten. Da die Implementierung der einzelnen Partnerdienste nicht durch die Architektur vorgegeben ist, kann jeder einzelne Dienst durch eine eigene Datenbankanwendung umgesetzt werden. Die Korrelation zwischen den Prozessvariablen und den zugehörigen operativen Datenmodellen ist durch die Angabe des SQL-Statements zwar explizit, ist im allgemeinen aber für den Anwender des Prozesses nicht sichtbar, da das genaue SQL-Statement im externen Partnerdienst verborgen ist. Der Match in der Matchphase kann daher nicht über diese bestehende Korrelation hergeleitet werden, sondern die Prozessvariable, die das Ergebnis aus dem Partnerdienst erhält, muss wie eine Standardvariable gematcht werden, die in iii. beschrieben wird.

ii. BPEL/SQL: Es gibt neben der strikten Einbettung von SQL in einem Dienst auch Ansätze, die eine serviceorientierte Architektur so erweitern, dass Datenzugriffe auch direkt in der Prozessdefinition festgelegt werden können. In diesem Fall werden die SQL-Statements mit der exakten Angabe der Datenbank, Tabelle und Spalte in den Prozess eingebettet (z.B. durch das IBM BPEL/SQL Plugin [iib]) und das Ergebnis des Statements wird einer Prozessvariablen zugewiesen. Die Korrelation zu anderen

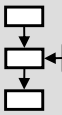
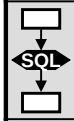
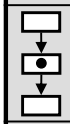
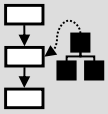
	modellierte Korrelation	sichtbar in Prozess	automatisch matchbar	Verwendung in BIA
 SQL in Partner-dienst	+	-	(+)	-
 BPEL/SQL	+	+	+	-
 Variable	-	+	+	+
 Human Task	-	(+)	(+)	(+)

Tabelle 2.2: Korrelation zwischen Prozesselement und operativem Element

operativen Datenquellen ist also explizit dargestellt und in diesem Fall auch im Prozess sichtbar. Wenn das SQL-Statement zudem komplex aufgebaut ist und mehrere Tabellen verbindet sowie mehrere Attribute zurück gibt, werden diese in einer komplexen Variable mit mehreren namenlosen Reihen zurückgegeben, die später im Prozess gegebenenfalls an entsprechende benannte Variablenelemente übergeben werden. Das bedeutet, dass weitere Parserschritte für das Matchen nötig sind. In diesem Fall ist außerdem keine einfache automatische Kombination der Elemente aufeinander möglich, sondern es sind komplexe Kombinationen, Bedingungen und Operationen zu beachten. Diese Matchvariante ist nicht Teil des Frameworks aufgrund der geringen Verbreitung von BPEL/SQL und der daraus resultierenden geringen Nutzung einer Implementierung der Variante. Die Möglichkeiten der SQL-Anfragen verschiedener Prozessserver werden in [VSRM08] detailliert beschrieben.

iii. Prozessvariable: Die relevanten Prozessvariablen für BIA speichern Informationen zu den bearbeiteten Geschäftsobjekten. Technische Prozessvariablen, die z.B. Daten zur Steuerung der Human Tasks speichern, sind für die Matchphase allerdings irrelevant. Prozessvariablen sind im Prozess zwar sichtbar, werden aber ohne explizite Korrelation zu einer operativen Datenquelle dargestellt. Geschäftsobjekte im Prozess beziehen sich jedoch meist auf reale Objekte oder Sachverhalte, auf die sich auch weitere operative Daten aus dem Unternehmen beziehen. Sie sind automatisch matchbar, wenn genügend Metainformationen für den Match vorhanden sind. Die Matchtechniken von BIA fokussieren diese Korrelation.

iv. Ressource in Human Task: Zusätzliche Informationen aus operativen Daten zu den Ressourcenattributen im Prozess versprechen ein großes Potenzial für Prozessoptimierungen. Die Ressourcen von Human Tasks sind nach bestimmten Eigenschaften organisiert und je nach dieser Organisationsstruktur und weiteren Richtlinien wird ih-

nen die Erlaubnis erteilt, einen bestimmten Human Task auszuführen. Generell findet die Verwaltung der Organisationsstruktur im Workflowmanagementsystem (WFMS) statt und eine explizite Korrelation zu anderen operativen Datenmodellen bezüglich dieser Ressourcen besteht nicht, sofern nicht alle Benutzer des Unternehmens über das gleiche Benutzerrepository verwaltet werden. Einzelne vom Prozess verwendete Informationen daraus werden in Prozessvariablen abgelegt. Je nach WFMS muss mehr oder weniger Aufwand betrieben werden, um die Organisationsstruktur eines Human Tasks sichtbar zu machen und die Organisationsstruktur ist in der Regel WFMS-spezifisch. Generell ist ein automatischer Match möglich. Um eine allgemein gültige Lösung zu haben, betrachtet BIA nur die Ressourceninformation in den Variablen. Die geforderte Gruppenzugehörigkeit der Benutzer wird z.B. in einer solchen Variable definiert. Die Prozessvariable für die Ressource wird in BIA wie eine Standardvariable gematcht, die in iii. beschrieben wird.

Die Korrelationen zwischen den Prozessvariablen und operativen Daten leitet das BIA-Framework mit Matchregeln ab. Das BIA-Framework ist also ein regelbasiertes System. Regelbasierte Systeme bestehen laut [BKI06] aus einer Faktenbasis (in BIA die Prozesselemente, operativen Daten und semantische Annotationskonzepte), einer Regelbasis (in BIA die Matchregeln) und einem Kontrollsystem, das die Regeln auf die Fakten anwendet, um neues Wissen zu generieren (in BIA das Herausfinden der Korrelationen). Eine Regel besteht aus einer Prämisse, die erfüllt sein muss, damit die Regel angewendet wird und die Konklusion ausgeführt wird. Die Regelbasis kann zu widersprüchlichen Ableitungen führen, falls mehrere Regeln für eine Prozessvariable und ein Schemaelement ausgeführt werden können und sie zu unterschiedlichen Ergebnissen kommen, ob nun eine Korrelation zwischen ihnen existiert oder nicht. Das BIA-Framework löst dies, indem es sich immer für das positive Ergebnis entscheidet. Mehr dazu in Kapitel 4.

2.2 Verwandte Arbeiten

Zu den vorgestellten Phasen für BIA aus Kapitel 1.2 wird in diesem Kapitel der Stand der Wissenschaft kurz betrachtet. Im Mittelpunkt stehen verwandte Arbeiten für das Matching und die Analyse von Daten im Warehouse. Ansätze der Prozessoptimierung werden nur am Rande erwähnt. Es wird gezeigt, warum bisherige Arbeiten dem ganzheitlichen Ansatz dieser wissenschaftlichen Abhandlung nicht gerecht werden.

2.2.1 Annotation und Matching der Daten

Für die semantische Annotation der Prozessmodelle und operativen Datenmodelle werden Ontologien verwendet, die mit einem Ontologieeditor benutzerfreundlich erstellt und betrachtet werden können. In [Den04] werden einige für die Ontologieerstellung geeignete Editoren präsentiert. Die Editoren, wie z.B. Protégé [KFNM04], sind aber generell nur zur Erstellung der Ontologien selbst geeignet, und nicht zur Annotation von Modellelementen mit dieser Ontologie. Eine Ausnahme bildet WSMO-Studio. Mit WSMO-Studio

[DSK+09, DSMK07] oder Radiant aus dem Meteor-S-Projekt der Universität Georgia [VGS+05] existieren Editoren, die die semantische Annotation von WSDL-Dokumenten unterstützen. Beide Werkzeuge sind an bestimmte Ontologien gekoppelt, entweder an WSML-Ontologien im ersten Editor oder an OWL, wie im Fall des zweiten Editors. Die jeweils andere Ontologie kann graphisch nicht eingebunden werden. Auch bei IBM ist ein semantisches Plugin [IBMb] für den WebSphere Integration Developer in Planung. Das Plugin ist speziell zur Auffindung ähnlicher Web Services gedacht und konzentriert sich nicht auf den Vorgang der Annotation selbst. Bei allen Editoren fehlt außerdem die Annotation der operativen Datenquellen.

Das Finden von Matches zwischen Prozessvariablen und operativen Datenmodellen ist eng verwandt mit vielen anderen Matchproblemen. Die größte Ähnlichkeit besteht zu dem Bereich der Schema- oder der Web-Service-Integration. In der Datenbank-Community gibt es sehr viele Ansätze für semi-automatisches Schema-Matching [RB01, LN07, DR02, DMR02, DMDH04, BH08]. In den Arbeiten werden verschiedene Methoden beschrieben, die versuchen Hinweise über die Semantik von Schemas zu erlangen und darauf basierend die Matches zwischen Schematabellen und ihren Attributen herauszufinden. Solche Methoden beinhalten linguistische Vergleiche, strukturelle Vergleiche, die Verwendung von Domänenwissen und Techniken zur gewinnbringenden Wiederverwendung gefundener Matches für das Entdecken neuer Matches. Jedoch unterscheiden sich die Schema-Matching-Ansätze in drei bedeutenden Aspekten von dem Matching von Schemaelementen mit Prozessvariablen:

- *unterschiedlicher Inhalt und andere Struktur der Datenquellen für das Matching*: Das Matching für BIA muss mit Variablen zurechtkommen, die in anderen nicht matchbaren Prozesskomponenten eingebettet sind wie z.B. Namen von Aktivitäten, welche im operativen Schema kein matchbares Gegenstück haben, während es das Ziel im reinen Schema-Matching ist, möglichst alle Elemente miteinander zu verbinden. Zudem unterscheidet sich der Audit Trail erheblich von der operativen Datenhaltung. So können die Attribute des operativen Schemas nicht auf die Attribute des Audit Trails gematcht werden, sondern nur auf einzelne Ausprägungen der Prozessvariablen und ihre belegten Werte.

- *nicht jede Variable soll mit einem operativen Schemaelement gematcht werden*: Einige Variablen speichern technische Routinginformationen und es macht keinen Sinn, diese Komponenten mit operativen Elementen zu kombinieren. Alleinige linguistische Verfahren würden hier irreführende Ergebnisse produzieren.

- *unterschiedlicher inhaltlicher Aufbau der Prozessvariablen und der Schemaelemente*: Die Variablen sind typischerweise inhaltlich ziemlich lose miteinander verbunden. Sie beinhalten verschiedene Eingabedaten mit notwendigen Informationen für den Geschäftsprozess, während die operativen Schemaelemente in einer Tabelle meist alle ein reales Objekt beschreiben. Strukturelle Schema-Matching-Ansätze sind daher nicht ausreichend. Mit alleiniger Anwendung dieser Ansätze würde die semantische Genauigkeit für BIA auf der Strecke bleiben.

Aktuelle Arbeiten für das Matchen von Web Services (z.B. [CS06, HLD+05, EKO07]) schlagen vor, Web Services manuell mit zusätzlichen semantischen Informationen aus Domänenontologien zu annotieren und dann diese Annotationen zu verwenden, um die Dienste automatisch zusammenzusetzen. In den Arbeiten von [CGBB08, DGBD09] und [DHM+04] werden hingegen nicht-annotierte Web Services miteinander kombiniert. Alle Ansätze beziehen sich jedoch nur auf die Kombination und Integration von Web Services. Das Ziel dieser Arbeit, durch das Matchen der Web Services mit externen operativen Daten weitere Informationen zu erhalten, um die Prozesse zu optimieren, wird nicht verfolgt. Eine Kombination von Geschäftsprozessen mit operativen Daten auf Basis semantischer Annotationen wurde bisher noch nicht vorgeschlagen. Ein weiterer Punkt ist, dass sich die Ansätze nur auf die Schnittstellen von Web Services und nicht auf den Geschäftsprozess selbst fokussieren. Diese Arbeit jedoch beachtet die Semantik des Geschäftsprozesses, indem sie die gesamte Struktur des Prozesses für das Matching heranzieht und neben den Daten in der WSDL-Datei auch den Kontroll- und Datenfluss im Prozess anschaut, um neue Matches zu finden.

Ansätze wie [CCDS07] und [zM04] haben zwar den Nutzen der Integration von Geschäftsprozessmodellen und operativen Datenmodellen erkannt. Bei ihnen erfolgt das notwendige Matchen jedoch von Hand. Dies ist sehr aufwändig und fehleranfällig für solch riesige Datenmengen sowie den komplexen und großen Schemas, wie sie im Audit Trail und auch in der operativen Datenhaltung vorliegen. Diese Arbeit geht davon aus, dass es sinnvoll ist, die Herausforderung anzunehmen und eine Technologie zu entwickeln, um die Daten automatisch zu matchen.

2.2.2 Warehousing und Analyse der Daten

Reine Prozessanalyse basiert auf Ablaufdaten, die im Audit Trail während der Laufzeit des Prozesses gespeichert werden. Hier sind Analysetechniken interessant wie z.B. das Business Activity Monitoring (BAM) [McC02], um noch während der Ausführung schnell auf Probleme und Zwischenfälle reagieren zu können, damit der Prozess möglichst schnell wieder fehlerfrei weiterlaufen kann. In diesem Zusammenhang sind die Bereitstellung und Definition von Key Performance Indikatioren (KPI) wie z.B. in [Pet04] wichtig, die die riesigen Datenmengen auf die für das Unternehmen wichtigsten Kerninformationen herunterbrechen. Diese Informationen sollen durch BAM im Blick behalten werden. Zu den KPIs der Autovermietungsfirma würden z.B. die Anzahl der begonnenen Vermietungsprozesse, Anzahl der getätigten Abschlüsse von Verträgen, getätigte Vermietungen pro Besucher der Website, durchschnittlicher ausgewiesener Betrag im Vertrag usw. gehören. Außerdem können die Audit Trails der Ablaufdaten in einem Warehouse integriert werden. Auf diesem Warehouse ist die Anwendung weiterer Analysetechniken möglich wie z.B. Analysen für das 'Business Performance Management' [SCDS02, BLS02], d.h. zu evaluieren, an welcher Stelle die Prozesse leistungsfähiger modelliert werden können. Workflow-Mining-Techniken [AGL98, RGvdA+07] liefern weitere Analyseergebnisse, mit deren Hilfe der Prozess optimiert werden kann. Im Fokus von 'Business Process Analytics' steht neben der Analyse von noch laufenden und abgeschlossenen Prozessereignissen auch

noch die Simulation des Verhaltens zukünftiger Prozessinstanzen auf Basis der Analyseergebnisse [zMS09]. Aber sowohl BAM als auch die genannten Analysen im Warehouse beziehen sich auf die Ablaufdaten im Audit Trail und ignorieren Effekte, die sich durch weitere operative Attribute erklären lassen würden.

Operative Daten im Unternehmen werden zur Analyse typischerweise auch in ein Warehouse [KRT+98] geladen. Es sind die verschiedensten Ansätze für ihre Analyse verfügbar. Zwei der wichtigsten Vertreter sind das Online Analytical Processing (OLAP) und das Data Mining. OLAP ermöglicht die mehrdimensionale Analyse von Daten in einem DWH sowie die graphische Darstellung der Analyseergebnisse [DSHB98, CD97]. Die Dimensionen entsprechen hierbei den Blickwinkeln, aus denen die Anwender vorhandene Daten analysieren. Für die OLAP-Operationen stellt SQL eigens dafür entwickelte Operatoren bereit, wie z.B. den CUBE-, ROLLUP- oder WINDOW-Operator [Iso03]. Sie erleichtern die Navigation, Aggregation und Selektion bei komplexen SQL-Anfragen. Während OLAP die Überprüfung von Hypothesen unterstützt, die durch den Anwender aufgestellt wurden, sollen Data-Mining-Werkzeuge neue Hypothesen möglichst selbständig aufstellen [Han05]. Bei Data Mining liegt der Schwerpunkt auf der Ableitung von Mustern und Regeln, die vorab nicht bekannt sind und die für eine bestimmte Anwendung von großem Nutzen sind. Diese Ansätze können jedoch nicht einfach auf das Szenario der ganzheitlichen Geschäftsprozessanalyse übertragen werden. BIA muss die Analysen auf einer riesigen Anzahl von Dimensionen gleichzeitig ausführen können, während im Warehouse normalerweise weniger Dimensionen pro Analyse berücksichtigt werden. Die Arbeit zeigt gewinnbringende Architekturansätze für das Warehouse und beschreibt, wie die OLAP- und Mining-Techniken aussehen müssen, damit die integrierten Daten effektiv und auf benutzerfreundliche Weise aufbereitet und analysiert werden können.

Auf dem Gebiet der ganzheitlichen Analyse für Prozessdaten und operativen Daten gibt es nicht viele verwandte Arbeiten. Die schon genannte Arbeit [CCDS07] erstellt ein Data Warehouse für ausgeführte Prozesse mit ihren Geschäftsobjekten und stellt ein Datenmodell für eine globale Analyse bereit. Im Gegensatz zu dem Schema des BIA-Warehouses fokussiert es jedoch nur die Prozessdimensionen und die operativen Dimensionen sind nicht ausgearbeitet, sondern einfach beliebig unter die Prozessdimensionen gemischt. Das Werkzeug in [zM04] verwendet in den Analysen neben den Ablaufdaten nur Prozessvariablen und operative Daten, die direkt durch Operationen in diesen Variablen im Prozess gespeichert wurden. Andere Attribute der operativen Daten werden nicht beachtet. Weiterhin sind seine Analysemöglichkeiten auf wenige einfache Operationen beschränkt.

Eine geeignete Warehouse-Architektur für BIA mit speziellen Analyseoperatoren wird in dieser Arbeit vorgestellt. Mit den föderativen Datenbanksystemen [Rah94, BS08, JMN03] existiert eine Technologie, die ein homogenes Datenbankschema für heterogene Datenquellen bereitstellt. Wie solch ein Föderationssystem für BIA verwendet werden kann und das Mapping in der Förderationsebene für eine effiziente Analyse der Daten aussehen muss, wird im Rahmen dieser Arbeit diskutiert.

2.2.3 Prozessoptimierung

Wie in Kapitel 1 bereits ausführlich erläutert, spielen Prozessoptimierungen eine immer wichtigere Rolle im Geschäftsleben. Die Analyseansätze für Geschäftsprozesse wie BAM oder 'Business Performance Management', die vorher erwähnt wurden, finden zwar Ergebnisse, die zeigen, wo im Prozess Probleme auftreten. Sie bieten dem Anwender jedoch wenig bis keine Unterstützung bei der Optimierung selbst.

Vorschläge für einzelne Optimierungsansätze in Geschäftsprozessen sind z.B. in [DvdAtH05] und [Cha95] dargestellt. Optimierungen auf Prozessebene werden hier oft als 'Business Process Re-engineering' terminiert. Bevor man mit dem Redesign des Prozesses startet, sollte klar sein, welche KPIs [DvdAtH05] berücksichtigt werden sollen. Es werden vier Hauptperformanzindikatoren unterschieden, bei denen nach Redesign des Prozesses Effekte gemessen werden können: Zeit (Laufzeit des Prozesses, benötigte Zeit der jeweiligen Ressourcen, Wartezeit), Kosten (Fixkosten z.B. für Infrastruktur, variable Kosten z.B. Anzahl der Verkäufe, Anzahl der Neuanstellungen), Qualität (Zufriedenheit der Kunden) und Flexibilität (z.B. um auf neue Tasks reagieren zu können). Der BIA-Zyklus hat zum Ziel, die Werte der analysierten KPI bei erneuter Prozessausführung zu verbessern und Prozessoptimierungen zu ermöglichen, die über diese bisherigen Ansätze hinausgehen. So sollen bei der Optimierung neue Hinweise, die durch zusätzliche operative Datenanalyse erlangt wurden, als genauso wichtig erachtet werden wie die Ablaufdaten des Prozesses.

2.3 Zusammenfassung

Dieses Kapitel hat einen Überblick über die Grundlagen und verwandten Arbeiten in dem Rahmen für BIA gegeben, wie er für das Verständnis dieser Arbeit nötig ist. Es wurden Komponenten eines Prozessmodells, nämlich Aktivitäten, Prozessvariablen, Correlation Sets, Handler und Partnerdienste, herausgegriffen, da ihre Verwendung, insbesondere ihrer Namen, für BIA beim Matchen oder der Analyse benötigt werden. In diesem Kapitel wurde auf verschiedene mögliche Korrelationen der Prozesselemente mit operativen Datenelementen eingegangen. Dabei erweist sich die Korrelation der Prozessvariablen mit den operativen Elementen als besonders sinnvoll, da sie automatisch durch BIA gematcht werden können.

Bezüglich den verwandten Ansätzen konnte in diesem Kapitel herausgestellt werden, dass in jeder Phase des BIA-Zyklus Arbeiten fehlen, die operative Elemente und Prozesselemente effizient zusammenbringen, um diese gemeinsam zu betrachten. Umso wichtiger sind die neuen Ansätze aus dieser Arbeit, durch die sich Prozesselemente und operative Elemente integrieren und analysieren lassen und die diese Erkenntnisse für eine neuartige, aus der BIA-Analyse resultierenden Geschäftsprozessoptimierung zur Verfügung stellen.

KAPITEL 3

Semantische Annotation der Prozess- und Datenmodelle

Eine Möglichkeit für das Matchen von Prozessmodellen und operativen Datenmodellen basiert auf der semantischen Annotation der Modellelemente. Dieses Kapitel beschreibt, welche Annotationsstrategie sich dafür eignet, welche Elemente beider Modelle dabei mit semantischer Information versehen werden und wie der Annotationseditor, der für diese Arbeit implementiert wurde, diese Strategie umsetzt.

3.1 Strategie für semantische Annotationen

Bei semantischen Annotationen werden die jeweiligen Modellelemente mit Hilfe von semantischen Begriffen aus einer Ontologie beschrieben. Dieses Kapitel erläutert zunächst den Begriff *Ontologie*, bevor konkrete Annotationsmechanismen gegenübergestellt und die Auswahl des Mechanismus für diese Arbeit begründet werden.

3.1.1 Ontologie für die Annotation

Eine Ontologie stellt die eindeutige Wissensrepräsentation eines formal definierten Systems von Begriffen und ihren Beziehungen dar [SS04]. Eine Ontologie kann demnach durch ein Tupel $O = (C, H_C, P_C, I, R, A)$ definiert werden. Die Menge C aus der Ontologie sind prägnante Begriffe, die die Terminologie der Domäne beschreiben. Die Begriffe sind in einer Hierarchie H_C angeordnet, die durch Subkonzeptbeziehungen dargestellt wird. Begriffe sind durch Attribute P_C definiert, die ihre Beziehung mit anderen Begriffen oder Begriffen in einem angegebenen Wertebereich angeben. Instanzen I repräsentieren

Objekte der Ontologie. Relationen in R beschreiben, welche Beziehungen zwischen den Begriffen bestehen. A beschreibt die Axiome. Das sind Regeln bestehend aus Begriffen, Attributen und Relationen, mit denen aus bereits definiertem Wissen neues Wissen abgeleitet werden kann.

Es gibt eine Reihe von Sprachen, mit denen eine Ontologie definiert werden kann. Zu den wichtigsten Vertretern gehören OWL (Web Ontology Language) und WSML (Web Service Modeling Language). OWL ist eine Spezifikation des W3C [BvHH+04]. Es stützt sich auf RDF und XML und erweitert es um eigene Konstrukte zur Beschreibung u.a. von Kardinalitätsbeschränkungen, speziellen Beziehungen und Charakteristika von Konzepten. OWL gibt es in drei Varianten mit wachsender Ausdrucksstärke: OWL Lite, OWL Description Logic (DL) und OWL Full. Diese Arbeit verwendet OWL DL, das die Verwendung von Schlussfolgerungstechnologien in der Matchphase von BIA erlaubt und gleichzeitig nur so ausdrucksstark ist, dass alle Schlussfolgerungen entscheidbar bleiben.

Auch WSML ist im W3C spezifiziert [dBFK+05]. Auch mit WSML können formale Sprachen für Ontologien (WSMO) definiert werden. WSML gibt es ebenso wie OWL in Varianten verschiedener Ausdrucksstärke: WSML-Core, - DL, -Flight, -Rule und -Full. Es basiert auf einer einfachen, benutzerfreundlichen Syntax. Daneben gibt es zum Austausch über das Web und mit anderen Anwendungen sowie zum Mapping mit OWL eine XML- und RDF-Syntax. Aus den gleichen Gründen wie bei OWL wird für die Arbeit WSML-DL verwendet.

3.1.2 Auswahl von SAWSDL als Annotationsmechanismus

Die Annotation von Web Services und ihrer Daten ist in den letzten Jahren im Rahmen des gestiegenen Interesses am Semantic Web immer wichtiger geworden. Die Grundlage des Semantic Web [BLHL01] ist das Anreichern von rein syntaktischen Informationen mit zusätzlichen eindeutigen semantischen Informationen, sodass Maschinen diese Information automatisch verarbeiten können. Für die semantische Annotation hat sich eine Reihe von Standards etabliert.

Auf der einen Seite stehen Ansätze, die neben der Beschreibung des Web Services selbst seine komplette semantische Repräsentation bereitstellen. Die prominentesten Vertreter dafür sind WSML (Web Service Modeling Language) [dBFK+05] und OWL-S (Web Ontology Language for Web Services) [MBH+04]. Beide Ansätze haben ihre eigene Ontologiesprache für die Beschreibung von Web Services und ihren eigenen Annotationsmechanismus. Beide Ansätze stellen die Verbindung, kurz Grounding [KRMF06], zwischen der semantischen und syntaktischen Ebene der Beschreibung des Web Services auf der semantischen Seite her. OWL-S basiert auf einer OWL-Ontologie. Mit den Klassen *Profile*, *Grounding* und *Process* definiert OWL-S die Funktionen des Web Services, wie man mit dem Web Service interagieren kann und wie er und seine Elemente auf WSDL abgebildet werden. In WSMO [ABK+04, dBBD+05] sind die Ontologien in WSML beschrieben und bestehen neben der WSML-Ontologie aus drei weiteren Komponenten: Ziele (Goals), Web-Service-Beschreibungen und Mediatoren. Goals spezifizieren die Ziele,

d.h. die von Aufrufenden erwartete Funktionalität eines Web Services, die Web-Service-Beschreibungen spezifizieren seine tatsächlichen Funktionen und nichtfunktionalen Eigenschaften sowie seine Schnittstellen. Mediatoren vermitteln zwischen verschiedenartigen WSMO-Elementen, um ihre Heterogenität in Bezug auf Daten, Protokoll oder den Prozessen aufzulösen. Sowohl OWL-S als auch WSML benötigen jedoch großen Aufwand um einen Dienst zu annotieren, weil der gesamte Web Service semantisch beschrieben wird. Einzelne Elemente, auf die in der Matchphase zugegriffen wird, müssen bei OWL-S auf komplizierte Weise extrahiert werden. Bei WSML werden einzelne Elemente nicht annotiert.

SAWSDL (Semantic Annotations for WSDL) [FL07] hingegen ist ein einfacher und flexibler Annotationsmechanismus. Er erweitert die Sprache WSDL oder XML lediglich um zusätzliche Elemente, anstatt einen eigenen Mechanismus zur Annotation einzuführen. Das Grounding findet so also in der WSDL-Datei selbst statt, die für das BIA-Framework schon Bestandteil der Eingabemenge ist und so keine anderen Beschreibungsdateien zusätzlich geladen werden müssen. Zur Annotation können zudem beliebige Ontologiesprachen verwendet werden. Die Annotationswerte können auf einfache Weise gespeichert und in der Matchphase wieder verwendet werden. SAWSDL kann in einer WSDL-Beschreibung eines Web Services folgende Komponenten annotieren: <*simpletype*>, <*complextype*>, <*element*>, <*attribute*>, <*portType*> (<*interface*> in WSDL 2.0), <*operation*> und <*fault*>. SAWSDL bietet drei Attribute an, die zur Annotation direkt dem betreffenden Element hinzugefügt werden:

- *modelReference*: Das Attribut wird verwendet, um die Komponenten mit Begriffen einer Ontologie zu verknüpfen. SAWSDL schreibt keine spezielle Struktur für diese Referenz vor, es hat sich aber die folgende Konvention eingebürgert, um den Begriff in der Ontologie zu adressieren: **OntologieID "#"BegriffID**.
- *liftingSchemaMapping* und *loweringSchemaMapping*: Diese Attribute ermöglichen es, eine Datei zu referenzieren, die die Transformation von komplexen XML-Komponenten in die Ontologiesprache (*liftingSchemaMapping*) bzw. in die andere Richtung (*loweringSchemaMapping*) realisiert.

Dieser Ansatz ist für das BIA-Framework besser geeignet als WSML oder OWL-S. Die Annotation soll fokussiert bestimmte Elemente einer WSDL- oder XML-Schema-Datei beschreiben und nicht die Semantik eines gesamten Web Services erfassen. Außerdem ist SAWSDL leichter um eigene für die Matchphase wichtige Annotationselemente zu erweitern und unabhängig von einer Ontologiesprache.

Neben der Annotation von Web Services gibt es auch verschiedene Ansätze, die Geschäftsprozessmodelle auf BPEL-Ebene zu annotieren, wie z.B. in [KVLL+08]. Die Bemühungen stehen jedoch erst am Beginn, sind noch nicht standardisiert und wurden daher für diese Arbeit nicht weiter beachtet.

3.2 Semantische Annotation der Geschäftsprozesse

SAWSDL ist, wie in Kapitel 3.1.2 erläutert, dafür ausgelegt, Web Services zu annotieren. Das Framework von BIA verwendet die drei Attribute von SAWSDL und ergänzt diese um ein viertes Attribut *modelType*. *modelType* gibt an, welche Ontologiesprache (WSML oder OWL) bei der Annotation verwendet wurde, um in der Matchphase das Parsen und die Verarbeitung der betreffenden Ontologie zu erleichtern.

In erster Linie sind bei den zu annotierenden Elementen die Prozessvariablen und ihre Elemente (siehe Kapitel 2.1.1, Prozessvariablen (i)-(v)) wichtig, da diese der Gegenstand des Matchens sind. Die Annotation einer Prozessvariablen ist formal in Definition 3 beschrieben.

Definition 3 (Annotation eines Elements in einer Prozessvariablen)
$P_i.pv_j.node_m(c_c) \in PV_{Ont}$ sei ein Prozesselement, das durch den Knoten m des Variablenbaumes pv_j eines Prozesses P_i repräsentiert ist. Die Prozessvariable sei durch den Begriff c_c der Ontologie Ont annotiert.

Allerdings sind auch die Annotationen der WSDL-Komponenten *<portType>* und *<operation>* für die BIA-Phasen nützlich, da sie Kontextinformationen zu den Variablen für das Matchverfahren liefern. Formal definiert dies Definition 4.

Definition 4 (Annotation einer WSDL-Komponente eines Prozesses)
$P_i.node_r(c_c) \in P_{Ont}$ sei eine Komponente des Geschäftsprozesses, dessen WSDL-Beschreibung durch den Begriff c_c einer Ontologie Ont annotiert ist.

Beispiele für die Annotation einiger Prozesselemente sind in Anhang A aufgezeigt.

3.3 Semantische Annotation der operativen Datenmodelle

Um das Matchverfahren im Prototyp möglichst einfach zu halten, sollen auch die operativen Schemaelemente mit SAWSDL annotiert werden. Deshalb wurden die SAWSDL-Attribute auch für diese Elemente bereitgestellt. Je nach Format der operativen Datenquelle sieht die Art, wie SAWSDL die Annotationsattribute einfügt, anders aus. Bei relationalen Datenbanken wird die Metainformation über die gesamten Tabellen und Spalten in einer Art separaten Systemtabelle mitgeführt, während bei XML-Dateien das XML-Schema direkt, ähnlich wie die Komponenten in WSDL, annotiert werden. Für operative Datenmodelle, die schon in einem Data Warehouse aggregiert sind, bietet es sich an, die Annotation auf dem CWM durchzuführen (vgl. Kapitel 2.1.2). Zur Erweiterung des CWM verwendet diese Arbeit *Simple Extensions*, einen einfachen Erweiterungsmechanismus des CWM. Da die semantische Annotation nur die Angabe der Attribute *modelReference*, *schemaLiftingMapping* und *schemaLoweringMapping* mit ihren Annotationswerten im Datentyp String verlangen, jedoch keine Funktionen dazu, um mit den Attributen zu arbeiten, sind *Simple Extensions* für die Annotation ausreichend. In *Simple Extensions* werden *Stereotypen* und *Tagged Values* definiert, die zu den Tabellen, Sichten oder Spalten hinzugefügt werden und ihre Annotation beinhalten:

- *Stereotypen*: Um effizient in der Matchphase des BIA-Zyklus zu erkennen, welche operativen Elemente semantisch annotiert sind, werden sie durch einen Stereotyp typisiert. Der Annotationseditor unterscheidet bisher zwischen zwei Stereotypen, die nach der verwendeten Ontologie benannt sind: Stereotyp *OWLAnnotation* für modeltype OWL und *WSMLAnnotation* für modeltype WSML.
- *Tagged Values*: Zu beiden Stereotypen gehört ein zwingend erforderliches Tag *ReferenceModel* und sein Wert, der eine URL als String speichert, um das zu der Annotation gehörige Ontologiekonzept zu referenzieren.

Formal wird ein annotiertes Schemaelement in Definition 5 beschrieben:
Definition 5 (Annotation eines Schemaelements) *$S_s.node_r(c_c) \in S_{Ont}$ sei ein Schemaelement, das durch den Knoten r des Schemabaumes eines Schemas S_s repräsentiert ist. Das Schemaelement sei durch den Begriff c_c der Ontologie Ont annotiert.*

3.4 Prototyp des Annotationseditors

Die Annotationsstrategie SAWSDL wurde für die Prozesselemente und operativen Elemente prototypisch in einem Annotationseditor umgesetzt. Dieses Kapitel zeigt seine Architektur und Funktionalität auf. Nähere Informationen zur Annotation der Prozesselemente findet man in [RNM08], während die Annotation der operativen Schemaelemente in [RM08] beschrieben wird.

3.4.1 Architektur des Annotationseditors

Die Architektur des Annotationseditors (siehe Abb. 3.1) gliedert sich im Wesentlichen in drei Teile: Modell-Transformation (MT), Benutzerschnittstelle und dem Kern des BIA-Editors. MT ist verantwortlich für das Laden der zu annotierenden Elemente, ebenso wie der Beschreibungsdaten für die Annotation. Über die Benutzerschnittstelle kann der Benutzer das Laden und Speichern der Modelle sowie die Annotation steuern. Der BIAEditor-Kern wiederum ist für die Ausführung der Funktionalitäten verantwortlich, die von der Benutzerschnittstelle zur Verfügung gestellt werden und steuert die MT. Im Folgenden werden die drei Komponenten näher ausgeführt.

Benutzerschnittstelle

Über die Benutzerschnittstelle führt der Benutzer die Annotationsstrategie der Prozess- und operativen Datenelemente durch. Ihm stehen folgende Funktionalitäten zur Verfügung, die in Kapitel 3.4.2 noch ausführlich diskutiert werden: Laden der annotierbaren Modelle - Prozesselemente und operative Daten - bzw. der Beschreibungsdaten, Darstellen der geladenen Elemente im Editor, Annotation der Elemente über Anklicken der jeweiligen Elemente und Konzepte, textuelle Annotation eines Elements über Eingabe der URL zu einem Beschreibungskonzept, Speichern der Annotation.

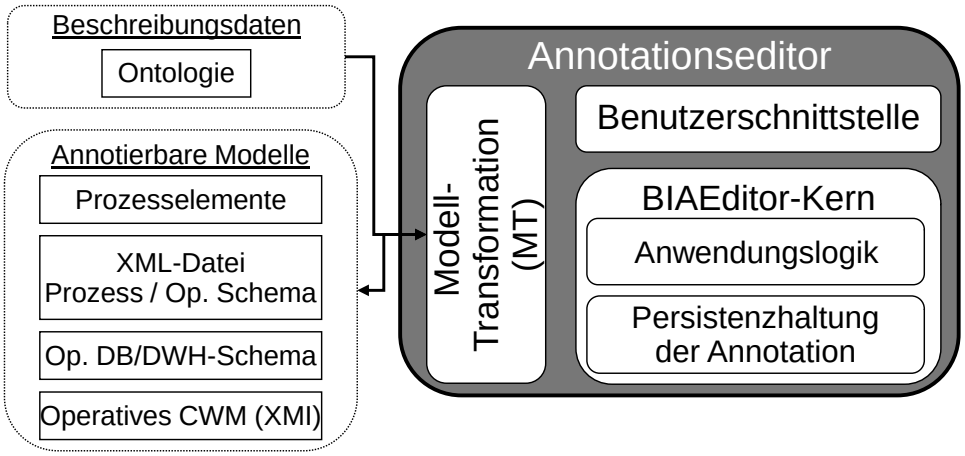


Abb. 3.1: Architektur des Annotationseditors

Modell-Transformation (MT)

Der Komponente zur Modell-Transformation (MT) werden von der Benutzerschnittstelle die annotierbaren Modelle zugewiesen. MT transformiert und mappt diese auf die entsprechende Repräsentation der internen Struktur des Annotationseditors. MT erlaubt es BPEL-Prozessvariablen direkt aus dem Audit-Trail zu laden. Die interne Struktur gleicht Abb. 2.1. Dabei werden XML-Schematypen, die von mehreren Nachrichten verwendet werden, für jede Message dargestellt. Außerdem können Prozesselemente aus der Prozessdatei statt aus dem Audit Trail geladen werden. Die interne Zielstruktur der Prozessvariablen ist in Abb. 3.2 (i) allgemein als XML-Datei dargestellt. Die Annotation der Elemente in der BPEL-Datei kann von Vorteil sein, falls die Elemente nicht aus dem Audit Trail geladen werden können, weil die Verbindung zum WFMS getrennt ist. Auch die operativen Elemente können als XML-Datei (i) vorliegen und durch MT geladen werden. Außerdem transformiert MT sowohl relationale Datenbankschemas (ii) auf die interne Struktur als auch Elemente im CWM-Format (iii). Beschreibungsdaten aus Ontologien (iv), die die Modelle semantisch annotieren sollen, werden von MT ebenso auf eine einheitliche interne Struktur abgebildet. MT erlaubt bisher, sowohl OWL- als auch WSML-Ontologien zu laden und zu transformieren.

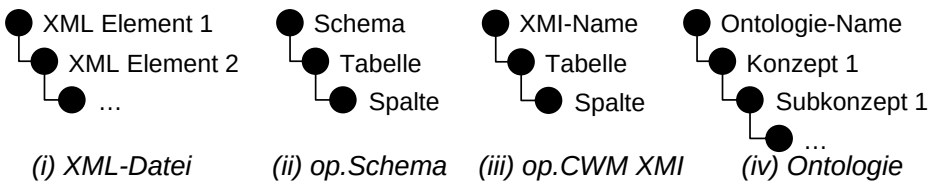


Abb. 3.2: Interne Zielstruktur der verschiedenen Modellkomponenten

Nach der Annotation ist es möglich, dass MT die annotierten Modelle wieder zurück in das ursprüngliche Format transformiert, um die Annotation zu speichern.

Kern des BIA-Editors

Im Kern wird die Anwendungslogik des Annotationseditors realisiert, die notwendig ist, um die MT für die Serialisierung, Traversierung und Annotation der annotierbaren Modelle und Ontologien zu steuern. Außerdem gewährleistet der BIAEditor-Kern die Persistenz der Annotationen von den Modellelementen. Je nach Format der Quelle des Prozessmodells werden die Annotationen in den Audit Trail in eine eigene Tabelle oder zu dem jeweiligen Element in der XML-Datei gespeichert. Für die Speicherung im Audit Trail muss berücksichtigt werden, dass der zugehörige Prozess bereits auf dem Prozessserver zur Ausführung gebracht wurde und dort vorliegt. Ansonsten ist das Modellelement, das die Annotationstabelle referenziert, noch nicht vorhanden und die Integrität würde verletzt werden. Wenn man das Annotationselement während der Bereitstellung des Prozesses auf dem Server gleichzeitig mit den Prozessdaten ablegen wollte, müsste der Deploy-Vorgang des Prozessservers angepasst werden, damit er das Annotationselement erkennt. Das ist jedoch für die hier verwendeten kommerziellen WFMS nicht möglich, da auf deren Anwendungslogik nicht zugegriffen werden kann.

3.4.2 Funktionalität des Annotationseditors

Die Umsetzung der Architekturkonzepte ist in dem Annotationseditor realisiert. Abb. 3.3 zeigt das Hauptfenster des Editors. Der Projektexplorer (1) zeigt die aktuell geladenen Prozesse, Dokumente und operativen Datenquellen an. Wie in der Abbildung zu sehen, ist aktuell nur der BPEL-Autovermietungsprozess *CarSelection* aus Kapitel 1.3 in den Arbeitsbereich des Editors geladen. Das zentrale Element des Editors ist der Annotationsexplorer (2), welcher den Variablenbaum bzw. den operativen Schemabaum visualisiert, d.h. die Komponenten der BPEL-Variablen sowie der WSDL- und XML-Schema-Dateien und operativen Daten. Die Struktur ist bis auf das letzte Element ohne Kindelement aufklappbar. In der Abbildung ist die BPEL-Variable *InputData* geöffnet und das Element *custID* ausgewählt. Der Benutzer kann einzelne Variablenkomponenten bzw. operative Komponenten auswählen und entweder über die Attributstabelle (4) oder den Listeneditor (3) annotieren. Die Attributstabelle zeigt die textuellen Werte der Annotationsattribute an. In der Tabelle kann man die Werte der Annotationsattribute manuell ändern. Meist ist es jedoch einfacher, dafür den Listeneditor zu verwenden. Über die Menüleiste (5) können alle Funktionen des Editors ausgeführt werden. Der Aufruf der wichtigsten Funktionen kann auch über die Toolbar (6) erfolgen. Am linken Rand der Toolbar beginnend sind Funktionen gezeigt, um Prozesse und Dokumente zu laden, zu modifizieren und zu löschen. Dann folgend die Funktionen, um Annotationen zu speichern und zu laden, operative Datenbankverbindungen bzw. Ontologie- und Audit-Trail-Verbindungen zu verwalten, sowie die Einstellungen des Editors zu ändern.

Um die Elemente für die Annotation zu laden, bedarf es der Durchführung mehrerer Schritte, von denen bei jeder Funktion jeweils die wichtigsten Schritte exemplarisch herausgegriffen sind:

- Laden des WFMS: Der Benutzer muss in Schritt 1 in Abb. 3.4 angeben, ob er eine Verbindung zu einem neuen WFMS herstellen möchte, oder eine existierende Verbindung

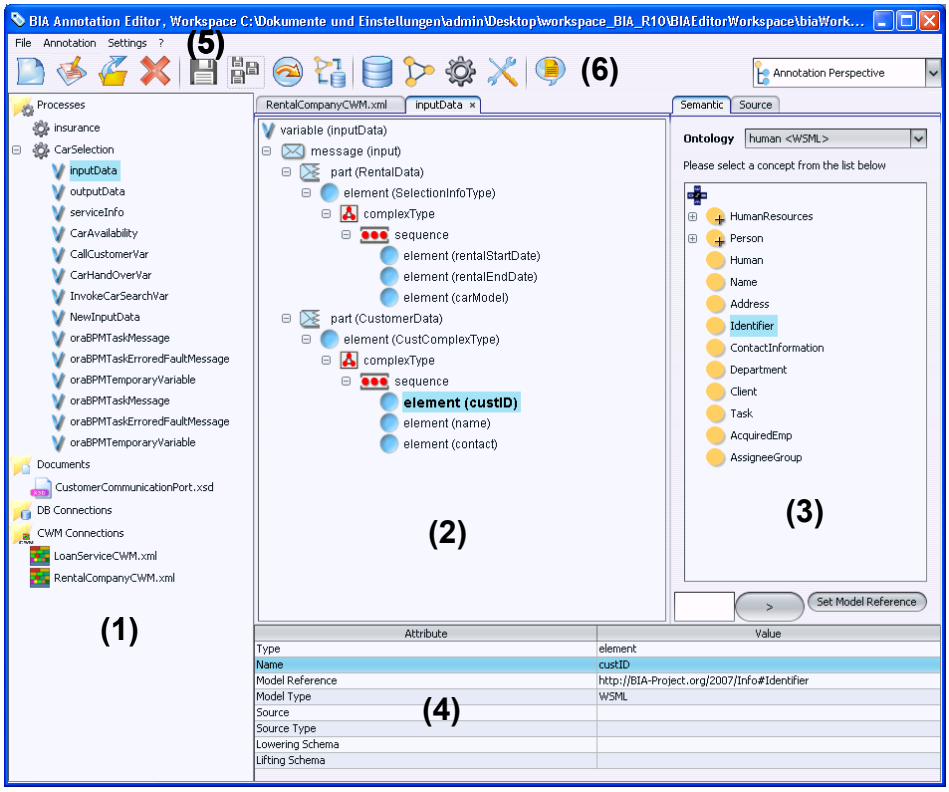


Abb. 3.3: Hauptfenster des Annotationseditors

ändern bzw. löschen möchte. In Schritt 2 gibt er dann den Namen an, unter welchem die Engine in dem Editor angezeigt werden soll, den Hersteller der Engine und welche Datenbank für den Audit Trail zu Grunde gelegt wird.

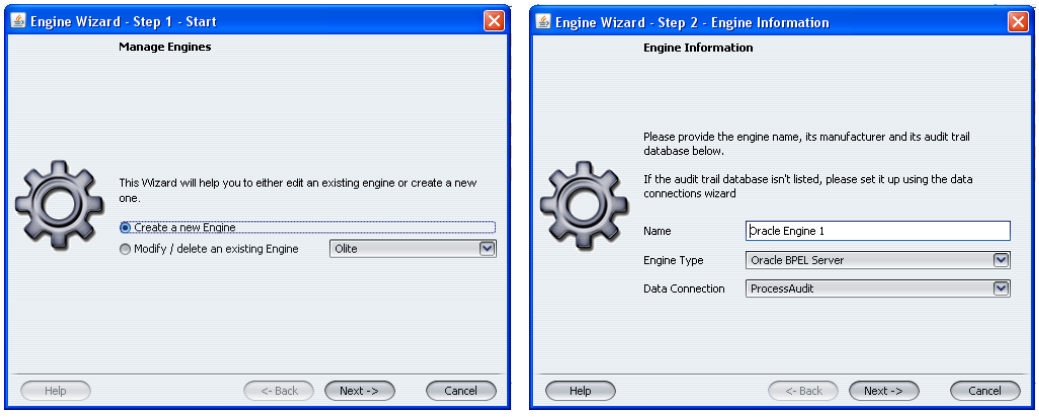


Abb. 3.4: Laden des WFMS

- Laden des Prozesses: Hier werden Screenshots aus dem Ladeprozess des Prozesses für die Annotation der Elemente im Audit Trail gezeigt. Dazu muss in Abb. 3.5 in Schritt

2 angegeben werden, für welches WFMS die Prozesse geladen werden. In Schritt 3 wird der Name des Prozesses für den Editor angegeben, sowie seine Domäne, Name des Prozesses im Audit Trail und seine Versionsnummer, unter der er im Audit Trail abgelegt wurde. Da sich die Informationen zu Variablenkomponenten meist in der WSDL-Beschreibung des Prozesses oder der aufgerufenen Partnerdienste befinden, muss der Pfad zu dem Deskriptor angegeben werden, der die Partnerdienste lokalisiert.

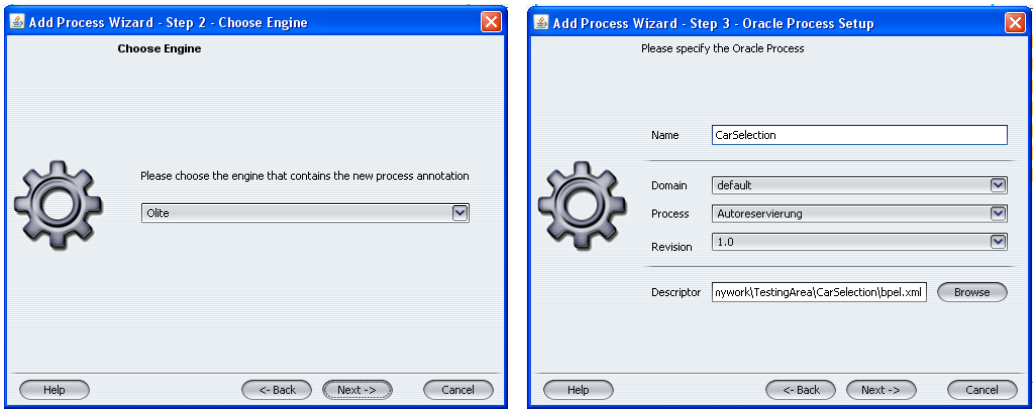


Abb. 3.5: Laden des Prozesses

- Laden der operativen Datenbankverbindung bzw. der Audit-Trail-Verbindung: In Abb. 3.6 in Schritt 2 gibt der Benutzer seinen gewünschten Namen für die Datenbankverbindung an und wählt aus, zu welchem Datenbankhersteller verbunden werden soll. In Schritt 3 erscheint dann abhängig vom Hersteller ein anderes Fenster. Hier wurde IBM ausgewählt. Der Benutzer muss nun noch den Treiber, den JDBC-Connection-String, seinen Benutzernamen und Passwort zu der Datenbank eingeben.

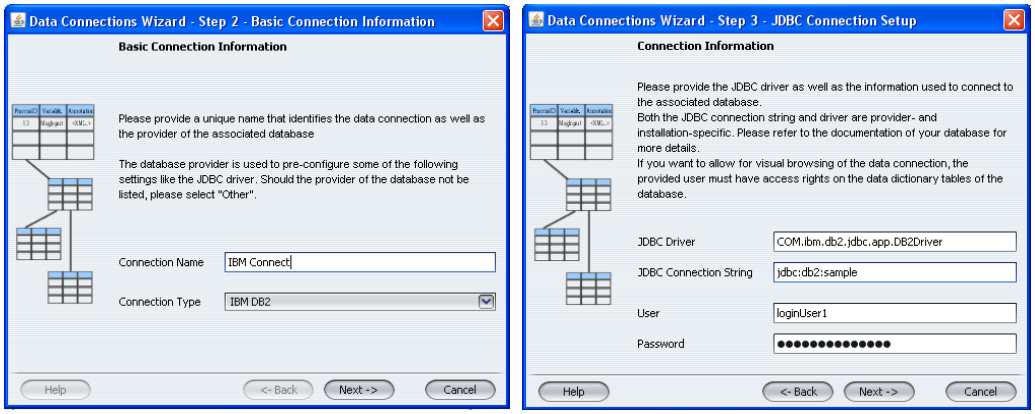


Abb. 3.6: Laden der operativen Datenbankverbindung bzw. der Audit-Trail-Verbindung

- Laden einer Ontologie: Um eine Ontologie zu laden, muss der Benutzer u.a. den Namen der Ontologie, unter welchem sie im Editor auftauchen soll, die Ontologiesprache und den Pfad angeben, wo die Ontologie gespeichert ist.
- Änderung der Einstellungen: Der Editor kann von dem Benutzer angepasst werden. Er wählt aus, ob der Zustand des Arbeitsplatzes und die getätigten Annotationen automatisch gespeichert werden, ob beim Start des Editors Ontologien und Datenbanken geladen werden sollen (resultiert evtl. in Verzögerung des Starts bei großen Datenmodellen) und welche Elemente annotierbar sein sollen (siehe Abb. 3.7).

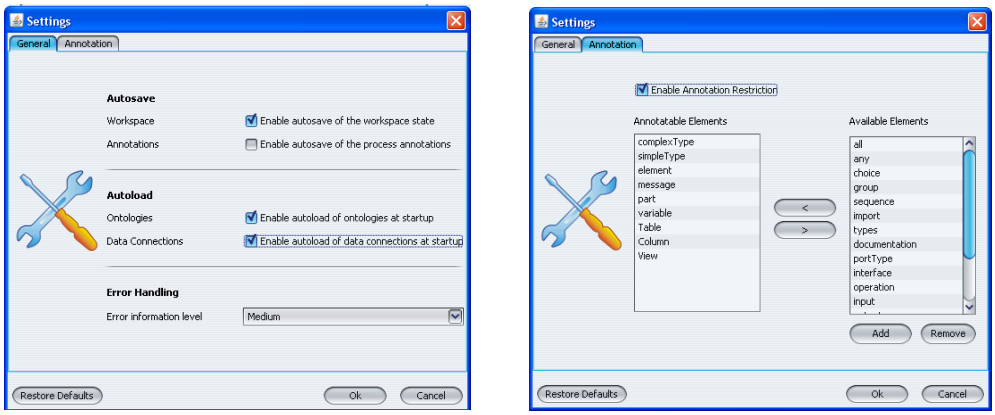


Abb. 3.7: Änderung der Einstellungen

3.4.3 Implementierungsaspekte des Annotationseditors

Der Editor ist unter Java Version 1.5 implementiert. Zur Erbringung seiner Funktionalität verwendet er eine Reihe von APIs wie z.B. WSDL4J[1] bzw. Apache Woden[2] für das Laden der WSDL1.1- bzw. der WSDL2.0-Beschreibungen oder Jena[3] bzw. WSMO4J[4] für das Laden und Visualisieren der OWL- bzw. WSML-Ontologien. Unterstützt wird das Laden von BPEL-Prozessen für IBM Websphere Process Server[5] und Oracle BPEL Process Manager[6]. Operative Datenbankschemas von führenden Datenbankherstellern IBM, Oracle, Microsoft Server und MySQL werden von dem Editor neben dem CWM unterstützt. Die Architektur des Editors bietet Schnittstellen, die die Unterstützung weiterer Engines, weiterer operativer Datenquellen oder anderer Ontologiesprachen relativ problemlos durch Hinzufügen der jeweiligen API und dem Mappen auf das interne Modell erlauben.

[1] http://sourceforge.net/projects/wsdl4j
[2] http://ws.apache.org/woden
[3] http://jena.sourceforge.net
[4] http://wsmo4j.sourceforge.net
[5] http://www-306.ibm.com/software/integration/wps/
[6] http://www.oracle.com/us/technologies/bpm/

Um die komplexe Abfolge der verschiedenen Ladevorgänge der MT zu unterstützen, sind Java Wizards [Kün06] implementiert worden, die diese Vorgänge in einzelne Schritte aufteilen und so ihre Handhabbarkeit durch den Benutzer vereinfachen.

Alle geladenen Modellelemente von MT werden auf die interne Struktur, auf JDOM-Bäume [7], abgebildet. Die Knoten der Bäume beinhalten je nach Art des Modellelementes oder Beschreibungskonzepts verschiedene Attribute. Variablenknoten beinhalten z.B. unter anderem je ein Attribut, das die Annotation und die Art der Variablenkomponente (z.B. WSDL-Message) speichert.

Um eine möglichst hohe Erweiterbarkeit des Editors an verschiedene Prozessserver und Datenbanken zu erreichen, wird die Annotation in allen Varianten in dem gleichen Tabellenschema *BIA_Annotation (AnnotationID, Elementname, Annotation)* gespeichert. Die Verbindung zu den konkreten Tabellen erfolgt je nach Art des Modells über entsprechende Bridgetabellen mit den Attributen aus dem Audit Trail des Prozessservers oder aus dem operativen Datenschema, die nötig sind, um das jeweilige zu annotierende Modellelement zu identifizieren und zu referenzieren (*BIA_Bridge (ModellelementID, AnnotationID)*). Für eine Referenzierung der Variable in einer Oracle-Engine sind dafür die BPEL-Domäne, die ProzessID und seine Versionsnummer sowie der Variablenname nötig, während bei der IBM-Engine der Variablenname und der Name ausreicht, unter dem der BPEL-Prozess auf dem Prozessserver gespeichert wurde. Für relationale operative Datenbanken ist zur Annotation der Datenbankname, Schemaname und Tabellenname notwendig.

3.5 Zusammenfassung

Die Annotationsstrategie, die in diesem Kapitel vorgestellt wurde, ist eine notwendige Voraussetzung für das semi-automatische Matchverfahren zwischen Prozesselementen und operativen Elementen. Nur wenn beide Elemente mit Hilfe von Ontologiekonzepten semantisch annotiert werden, können die semi-automatischen Matchregeln die Elemente miteinander in Verbindung bringen. Der dafür verwendete Standard SAWSDL erweist sich als einfacher und flexibler Annotationsmechanismus. Er erweitert die Prozesselemente um ein Attribut, das die URI zu dem jeweiligen Annotationskonzept anzeigt. Der Mechanismus wurde für die Annotation der operativen Elemente übernommen. Im Annotationseditor ist diese Annotationsstrategie prototypisch für BIA umgesetzt.

[7] http://www.jdom.org

KAPITEL 4

Kombination der Prozess- und Datenmodelle

Um eine effektive Optimierung der Geschäftsprozesse unter Berücksichtigung von zusätzlichen Informationen aus operativen Datenquellen durchführen zu können, ist es notwendig, diese Informationen mit den Geschäftsprozessdaten zu integrieren. Nur so können die Prozesse tiefer gehend analysiert werden und daraus Erkenntnisse für eine Optimierung gezogen werden. Dieses Kapitel stellt Konzepte für die Integration der Modelle vor und ihre Umsetzung im BIA-Framework.

Zunächst erfolgt ein Überblick über die verschiedenen Varianten der Integration der Prozessdaten und operativen Daten, die das Framework unterstützt. Anschließend erfolgt die genauere Betrachtung einiger Matchvarianten, bevor in den folgenden Kapiteln detailliert auf die einzelnen Matchregeln eingegangen wird. Die Kontrollstrategie für die Anwendung dieser Regeln wird danach diskutiert. Der Prototyp für das BIA-Framework, welcher die Matchregeln implementiert und in einem Editor ausführbar macht, wird in einem weiteren Kapitel präsentiert, bevor die Matchregeln in Kapitel 4.9 evaluiert werden. Die abschließende Zusammenfassung wiederholt anschaulich in der Tabelle 4.1 die vorgestellten Matchregeln mit den Formeln für die Berechnung ihrer Ähnlichkeitswerte. Einen guten Überblick über die Matchregeln erhält man auch in [RVSM11].

4.1 Überblick der Matchvarianten

Wie in Kapitel 1.1 erörtert, ist es das Ziel, die Prozessvariablen, bzw. einzelne Elemente davon, mit operativen Schemaelementen zu integrieren, die sich auf das gleiche reale Objekt beziehen. Wie in Kapitel 2.1.3 dargestellt wurde, gibt es verschiedene Möglichkeiten im Geschäftsprozess, wie die Variablen mit anderen operativen Daten zusammenhängen.

Das Framework realisiert das Matchen von Prozessvariablen mit operativen Daten des Unternehmens, die die gleiche Ressourcen beschreiben, die den Prozess ausführen; aber auch Prozessvariablen, zu denen es keine weitere Information der zugehörigen operativen Daten im Prozess gibt. Operative Informationen in einem externen Web Service, der durch den Prozess aufgerufen wird, sind nicht zugänglich und werden im Framework nicht beachtet. Unser Framework bietet für das Matchen manuelle, semi-automatische und vollautomatische Varianten an, die in den nächsten Unterkapiteln vorgestellt werden, und illustriert diese in Abb. 4.1. Der Benutzer wählt aus, welche Variante er ausführen möchte und stößt das Matchverfahren an. Nach Ablauf des Matchens wird das Ergebnis gefundener Matches und ihre Ähnlichkeitswerte in der Matchtabelle gespeichert.

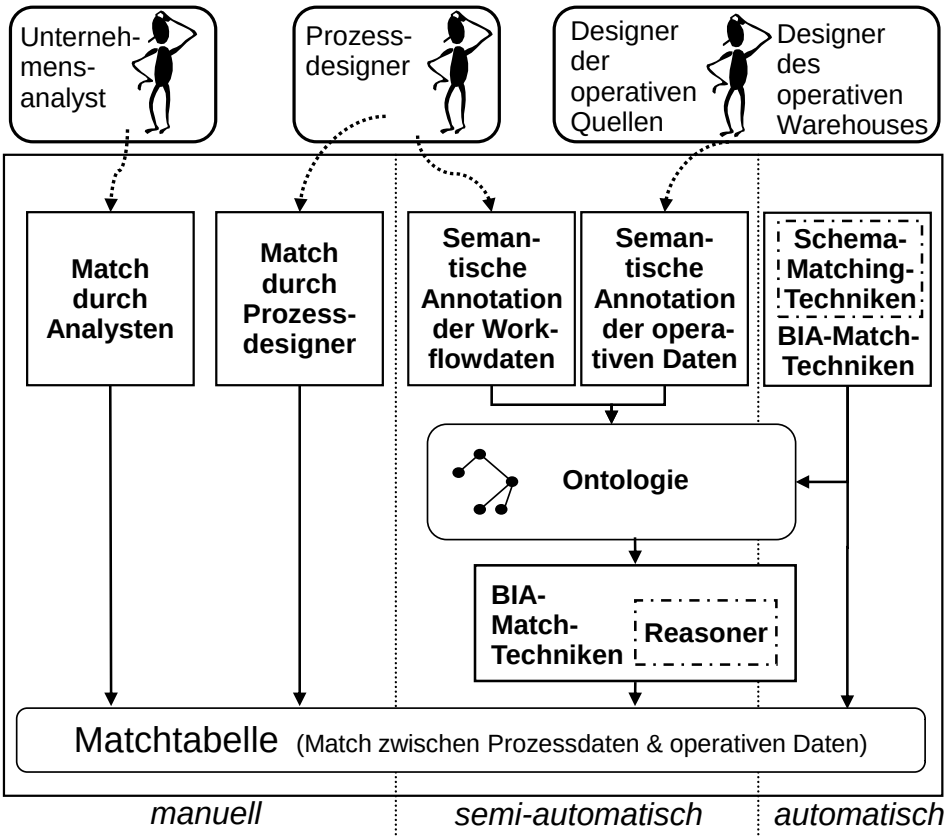


Abb. 4.1: Überblick über Matchvarianten des Frameworks

4.1.1 Manuelle Matchvariante

Die aktuelle Matchvariante für eine Geschäftsprozessanalyse ist es, die Prozessvariablen und operativen Schemaelemente manuell zu integrieren. Das verursacht jedoch eine Menge von Kosten und Aufwand, da die Person, die diese Integration durchführen soll, genau wissen muss, welche Daten für die ganzheitliche Analyse gebraucht werden, und sie sich dafür sehr genau sowohl in den Prozessdaten als auch in den operativen Daten auskennen

muss. Das kann sehr arbeitsaufwändig sein, wenn keine Dokumentation des Audit Trails oder des operativen Schemas vorhanden ist. Des Weiteren ist die manuelle Integration sehr subjektiv, d.h. abhängig von den Vorstellungen des Benutzers, und fehleranfällig, da er jedes einzelne Element betrachten muss. Es kann jedoch Matches geben, die nicht automatisch gefunden werden, da sie z.B. nicht annotiert sind oder keine ähnlichen Namen besitzen. Somit sollte das manuelle Matchen auch von dem BIA-Framework unterstützt werden. Da bei dem Design bzw. der Analyse des Geschäftsprozesses Personen in unterschiedlichen Positionen beteiligt sind, bietet das BIA-Framework verschiedene Ansätze für das manuelle Matchen an.

Match durch Analysten

Diese Matchmethode kann man in aktuellen Warehousesystemen finden. Hier muss der Analyst die Datenmodelle, die er in seiner Analyse verwenden will, direkt miteinander verknüpfen. Dieser Match bildet dann die Grundlage für den ETL-Fluss, um die Daten in ein integriertes Schema zu laden. Obwohl es das Ziel des BIA-Frameworks ist, diesen fehleranfälligen Weg mit Hilfe der folgenden Matchmethoden zu verbessern, gibt es einige Elemente, bei denen diese neuen Methoden nicht funktionieren und kein Matchergebnis gefunden wird. Deshalb muss dieser Ansatz Teil des Frameworks bleiben.

Match durch Prozessdesigner

In dieser Methode gibt der Prozessdesigner während oder nach dem Prozessdesign die Matches zwischen den Prozesselementen und den operativen Elementen an und speichert sie direkt ab. So können sie später von dem Analysten aus der Matchtabelle geholt und verwendet werden. Dieser Ansatz hat den Vorteil, dass sich der Prozessdesigner bestens bei den Prozesselementen auskennt. Das Gesamtergebnis hängt jedoch davon ab, wie gut er sich zusätzlich auch in den operativen Elementen auskennt und welche davon überhaupt mit den Prozesselementen ohne komplexe Transformationen integrierbar sind.

4.1.2 Semi-automatische Matchvariante

Semantische Beschreibungen der zu matchenden Elemente ermöglicht eine neue Art von Informationsintegration durch die Verwendung von Konzepten aus einer Ontologie kombiniert mit automatischen Schlussfolgerungsverfahren. Der Benutzer muss zwar Aufwand investieren, um eine Ontologie zu erstellen bzw. um eine geeignete, existierende Ontologie zu finden und um ein passendes Werkzeug, einen Reasoner, für das Schlussfolgern zu finden. Dieses Werkzeug soll die semantischen Informationen ausnutzen und neue Beziehungen zwischen den Elementen entdecken, die von den BIA-Matchtechniken im BIA-Framework verwendet werden können. Der Aufwand lohnt sich jedoch, da sowohl die Ontologie als auch die Werkzeuge immer wieder für Matchaufgaben des Frameworks verwendet werden können. Letztendlich arbeitet diese Methode durch das automatische Schlussfolgern effizienter und findet sehr wahrscheinlich auch neue Beziehungen zwischen

Konzepten, die durch rein manuelles Matching nicht gefunden worden wären. Weiterhin braucht der Benutzer nicht über Prozessdaten bzw. die operativen Daten Bescheid wissen, sondern der Prozessdesigner bzw. der operative Schemadesigner muss nur angeben, welche semantischen Konzepte seine Modelle, für die er verantwortlich ist, jeweils beschreiben. Die semantische Annotation des jeweiligen Elementes durch die Designer erfolgt über das Hinzufügen einer URI (Uniform Resource Identifier), die ein Konzept in einer Ontologie referenziert, wie es in Kapitel 3 beschrieben wurde.

4.1.3 Automatische Matchvariante

Weil das semi-automatische Integrationsverfahren immer noch eine Menge Aufwand für den Prozessdesigner bedeutet, ist das automatische Integrationsverfahren erstrebenswert, denn die automatische Integration benötigt keine semantische Annotation des Benutzers. Stattdessen stellt das Framework BIA-Matchtechniken zur Verfügung, die ohne Hilfe des Benutzers Zusammenhänge zwischen den Elementen erkennen. Dafür verwendet das BIA-Framework ausgesuchte Standardverfahren für Schema-Matching, die z.B. auf Grundlage der Namen der Prozesselemente und operativen Elemente arbeiten. Des Weiteren kommen neue Verfahren zum Einsatz, die die Struktur des Prozesses beachten. Dieses Verfahren wird von dem BIA-Framework auch angewendet, wenn eine Ontologie vorhanden ist und die operativen Schemaelemente vorher nur teilweise semantisch annotiert wurden. Dann werden sie durch die Matching-Verfahren auf die Konzepte der Ontologie gematcht, die schon zur Annotation der Prozesselemente verwendet wurde. Wenn dann alle Elemente annotiert sind, können die gleichen semi-automatischen Regeln aus 4.1.2 angewendet werden, um weitere neue Beziehungen zu entdecken.

4.2 Schritte des Matchverfahrens im Framework

Das Framework hat zum Ziel, mindestens ein matchendes Schemaelement für jedes Element der Prozessvariablen zu bestimmen und für die Kombination einen Ähnlichkeitswert zwischen 0 und 1 festzulegen. Die Kombinationen werden im Framework in zwei Schritten berechnet: zum einen werden die Elemente im Pairingschritt miteinander kombiniert und ein Ähnlichkeitswert für die Kombination berechnet. Zum anderen werden die gefundenen Matchergebnisse unter Berücksichtigung des Kontexts gefiltert. Abb. 4.2 illustriert die Pipeline, die eine Menge von Kombinationen, kurz COM, zwischen Prozessvariablen und operativen Schemaelementen als Ausgabetripel generiert.

Jede Matchregel in der Pipeline benötigt eine andere Eingabemenge: Prozessvariablen, Prozesskomponenten und Schemaelemente, entweder vollannotiert (PV_{Ont}, S_{Ont}) oder nur partiell annotiert (PV, S), eine Ontologie Ont, die für die Annotationen verwendet wurde, und den annotierten oder nicht-annotierten Kontext der Prozessvariablen (P_{Ont}, bzw. P). Alle Eingabemengen wurden bereits in den vorangegangenen Kapiteln beschrieben. PV_{Ont} ist die Menge der annotierten Prozessvariablen (siehe Definition 3 in Kapitel 3.2) und S_{Ont} die Menge der annotierten Schemaelemente (siehe Definition 5 in Kapitel 3.3).

PV und S beschreibt jeweils die Menge der partiell oder evtl. sogar nicht annotierterten Prozesselemente und Schemaelemente (siehe Definition 1 und 2 in Kapitel 2.1). Ont ist in Kapitel 3.1.1 erläutert und P_{Ont} in Definition 4 in Kapitel 3.2.

Über die Komponente der Schema-Annotation des Frameworks können als erster Pairingschritt teilweise annotierte Schemaelemente mithilfe von üblichen Schema-Matching-Verfahren semantisch annotiert werden, um auch für diese Elemente semi-automatische Matchregeln ausführen zu können. Abhängig davon ob die Eingabemengen annotiert sind oder teilweise bzw. gar nicht, wird nun im zweiten Pairingschritt mithilfe von semi-automatischen oder automatischen Regeln nach Kombinationen zwischen den Prozessvariablen und operativen Elementen gesucht. Man erhält die Menge von Matches COM_P. Diese werden im nächsten Schritt gefiltert. Der erste Filterschritt betrachtet den Kontext der an den Matches beteiligten Elemente in COM_P und berechnet ihren strukturellen Ähnlichkeitswert sim_{str}. Der letzte Filterschritt ergibt eine finale Ergebnismenge COM, optional unterstützt durch Benutzerinteraktion. Unter der Berücksichtigung der Tatsache, dass eine vollautomatische Lösung nicht möglich ist, ist eine benutzerfreundliche Oberfläche essentiell für die Verwendbarkeit eines Matching-Systems, über die Korrekturen der automatisch berechneten Kombinationen stattfinden können. Die graphische Benutzeroberfläche des Editors wird in Kapitel 4.8 erläutert. Sie erlaubt es, die erhaltenen Ergebnisse nach dem Matchen zu manipulieren. Die manuell hinzugefügten Matches werden automatisch mit dem höchsten Ähnlichkeitswert 1 versehen. Im letzten Schritt werden die finalen Matchergebnisse als Tripel in der Menge COM ausgegeben.

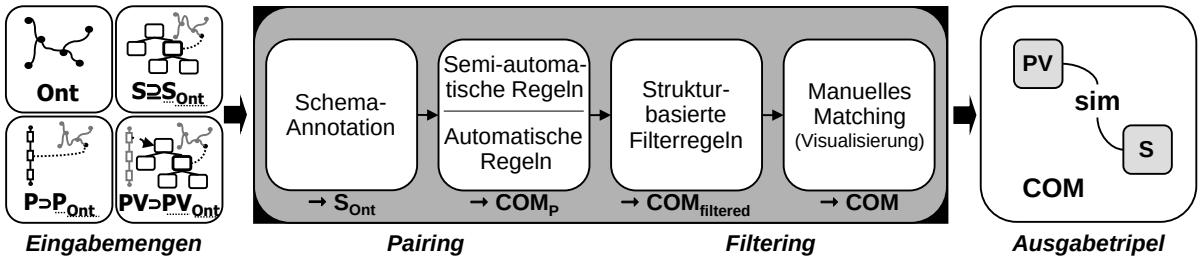


Abb. 4.2: Matching im BIA-Framework

Falls nötig, können einzelne Schritte der Pipeline auch übersprungen werden. Wenn manuelle Änderungen weggelassen werden möchten, dient die manuelle Filterung z.B. nur als visuelle Ausgabe oder wird ganz übersprungen. Auch im Pairingschritt können die Schema-Annotation bzw. die semi-automatischen oder automatischen Regeln weggelassen werden. Für das Pairing ergeben sich daraus folgende Möglichkeiten im Framework:

a. Pairing nur für vollannotierte Elemente

b. Pairing nur für partiell-annotierte Elemente

c. Pairing für vollannotierte und partiell-annotierte Elemente

d. Schema Annotation, danach vollannotiertes Pairing

e. Schema Annotation, danach partiell-annotiertes Pairing

f. Schema Annotation, danach vollannotiertes und partiell-annotiertes Pairing

g. Schema Annotation ohne Pairing

h. kein Pairingschritt

Die Möglichkeiten g. und h. machen für das Auffinden eines Matches keinen Sinn. Die Ausführung von Pairing auf vollannotierten Elementen in a. und d. erscheint unrealistisch, da normalerweise nicht alle Elemente annotiert sind und auch durch die Schema-Annotation nicht alle operativen Elemente mit einer Annotation versehen werden. Ebenso ist die Ausführung von partiell-annotiertem Pairing in b. und e. unwahrscheinlich, da der Trend zu semantisch annotierten Geschäftsprozessen geht [HLD+05] oder durch die Schema-Annotation zumindest ein paar Annotationen für operative Elemente gefunden werden, die über die semi-automatischen Regeln verwendet werden könnten. Daher konzentriert sich das Framework auf einen Mix aus Matchregeln für partiell- und vollannotierte Elemente, wie in c. und f. angegeben. Die Kontrollstrategie des Frameworks bietet aktuell die Variante c. an, kann aber durch ergänzende Schema-Matching-Verfahren, unterstützt durch den Framework-Charakter des BIA-Matching-Werkzeuges, unkompliziert erweitert werden.

Die annotationsbasierten und partiell-annotationsbasierten Regeln für semi- bzw. vollautomatisches Matching werden in den nächsten Kapiteln ausführlich erläutert. Das Framework wendet die Regeln nach einer wohldefinierten Kontrollstrategie an, die in Kapitel 4.7 erklärt wird. Um ein hochqualitatives Ergebnis zu erhalten, werden zuerst die annotierten Pairingregeln auf normalerweise wohlüberlegten Annotationen ausgeführt, bevor partiell-annotierte Regeln angewendet werden. Wenn verschiedene Regeln die gleichen Elemente kombinieren, wird das Ergebnis mit dem höchsten Ähnlichkeitswert gespeichert. Das Matchergebnis ist eine Menge von Matchtripeln, bestehend aus einem Element einer Prozessvariable, einem operativen Schemaelement und einem Ähnlichkeitswert zwischen 0 (verschieden) und 1 (sehr ähnlich), der die Plausibilität ihrer Zusammengehörigkeit angibt. In Definition 6 wird dieses Matchtripel formalisiert.

Definition 6 (Ergebnismenge COM der erstellten Kombinationen)
Sei $COM = PV \times S \times sim$ eine Menge von Matchtripeln. Ein Tripel daraus gibt an, dass sich das Element der Prozessvariable $P_i.pv_j.node_m \in PV$ und das operative Schemaelement $S_s.node_r \in S$ mit einer Wahrscheinlichkeit sim aus einem Intervall [0,1] der rationalen Zahlen auf das gleiche reale Objektattribut beziehen, ihre Korrelation also mit dem Ähnlichkeitswert sim gekennzeichnet werden kann.

Die Elemente werden sowohl in PV als auch in S so dargestellt, wie in Definition 1 und 2 in Kapitel 2.1 festgelegt. Das Ergebnis ist ein gerichteter Match, da das Ziel ist, alle Matchkandidaten für die Variablenelemente aus einem Prozess zu finden. Für manche operativen Schemaelemente wird dagegen kein Match gefunden. Das ist ein Unterschied und eine Vereinfachung zum Schema-Matching, wo alle Elemente aus beiden Quellen bidirektional Matchpartner finden sollten. Ein Match ist eine n:m-Beziehung, weil eine Variable auf viele Schemaelemente gemappt werden kann und ein Schemaelement zu mehreren Variablen passen könnte.

Da man im Pairingschritt Tripel aus Kombinationen von zu vielen Elementen erhält und die Ergebnismenge COM erst später im Filterschritt bereinigt wird, bestimmen wir folgende Teil- bzw. Obermenge (abhängig von dem Ergebnis der späteren Filterregeln) COM_P von COM mit $COM_P = PV \times S \times sim_p$, wobei gilt: $sim_P \in [0, 1]$. COM_P beschreibt dabei die Menge der Matchtripel, die aus dem Pairingschritt resultieren. Strukturbasierte Filterregeln werden danach auf COM_P angewendet. Sie verwenden den Kontext der Elemente, um ihre Ähnlichkeit sim_{str} bezüglich ihrer Struktur zu berechnen, d.h. für alle Kombinationen aus der Menge COM_P wird ein struktureller Ähnlichkeitswert $sim_{str} \in [0, 1]$ berechnet und mit sim_P kombiniert. Die Berechnung wird später in Definition 9 erläutert. Daraus entsteht $COM_{filtered}$. Es ist eine Teil- bzw. Obermenge (abhängig von dem Ergebnis der späteren manuellen Filterregeln) von COM mit $COM_{filtered} = PV \times S \times sim_i$, wobei gilt: $sim_i \in [0, 1]$. Falls bei den manuellen Filterregeln mehr neue Matches hinzukommen als in $COM_{filtered}$ reduziert wurden, gilt: $COM_{filtered} \subseteq COM$. Ohne manuelle Filterung gilt für die finale Tripelmenge $COM = COM_{filtered}$. Jeder Tripel mit einer Ähnlichkeit für eine Kombination zwischen $P_i.pv_j.node_m$ mit dem operativen Schemaelement $S_s.node_r$ über einer vorher festgelegten Schwelle wird vom Framework akzeptiert.

4.3 Semi-automatische Regeln für das Pairing annotierter Elemente

Dieses Kapitel stellt ausführlich die Regeln für die semi-automatische Integration vor. Voraussetzung bei allen Regeln ist die semantische Annotation beider Matchpartner durch die gleiche Ontologie. Falls unterschiedliche Ontologien verwendet wurden, müssen diese zuerst integriert werden. Für die Integration der Ontologien können Standard-Matching-Verfahren z.B. durch den COMA-Matcher [DR02] verwendet werden. Anschließend müssen die Annotationen der Elemente auf die Konzepte der neuen integrierten Ontologie angepasst werden. Danach können die Matchregeln des Frameworks für die Kombination der Variablen und der operativen Schemaelemente angewendet werden.

Im Folgenden werden fünf Regeln vorgestellt, die im Framework für das annotierte Pairing verantwortlich sind. Die Regeln basieren auf Reasoning-Verfahren, die in Ontologien normalerweise neue Zusammenhänge zwischen Konzepten und Instanzdaten aufdecken. Da die Regeln hier jedoch zum Ziel haben, Variablen aus Prozessmodellen und Spalten aus Schemamodellen zu matchen und die Integration ihrer Ausprägungen erst in späteren Schritten beachtet werden, konzentrieren sich die Regeln auf eine reine Modellintegration und lassen die Instanzen der Ontologiekonzepte zunächst außen vor. Bei zwei Regeln ist eine endgültige Prüfung des Matches jedoch unabdinglich und sollte in einem weiteren Schritt, ähnlich dem Cleansing in der Warehousephase, durchgeführt werden. Viele Ausprägungen sind sicherlich nicht als Instanzen in der Ontologie modelliert, deshalb kann diese Prüfung nur in manchen Fällen stattfinden. Alle Regeln, mit Ausnahme der Regel 4, stammen aus dem Bereich des Schema-Matchings und sind auf BIA angepasst worden. Regel 4 ist neu.

Die semi-automatischen Regeln haben alle die gleiche Eingabe- und Ausgabemenge, wie man in Abb. 4.3 sieht. Zur Eingabemenge gehören annotierte Prozesselemente PV_{Ont} von annotierten Prozessen P_{Ont}, Schemaelemente S_{Ont} und die Ontologie Ont selbst. Die Regeln suchen in jedem Prozessmodel $P_i \in P_{Ont}$ zu jedem Variablenelement $pv_j.node_m$ aus PV_{Ont} passende operative Elemente $S_s.node_r$ aus der Elementmenge S_{Ont}.

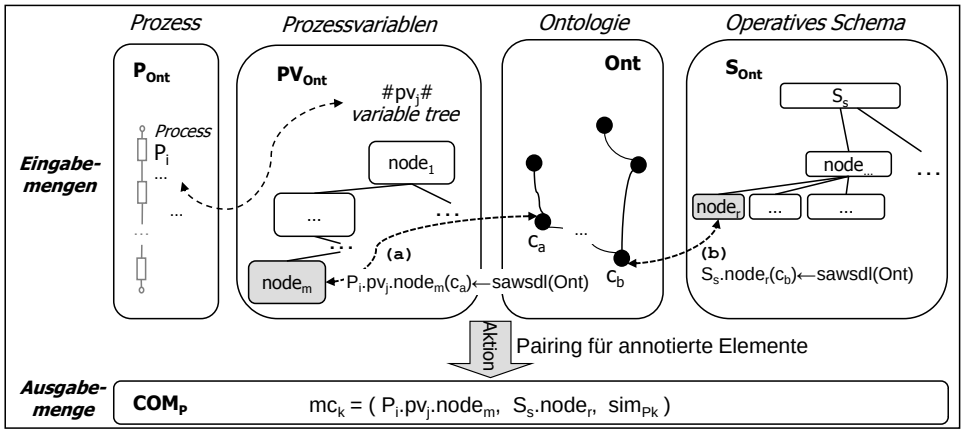


Abb. 4.3: Pairing annotierter Elemente

Die Prozessvariable pv_j ist in P_i deklariert und durch das Konzept c_a annotiert. Das operative Element aus S_s ist durch Konzept c_b annotiert. c_a und c_b sind aus der Menge der Konzepte $C = \{c_1, c_2, ..., c_c\}$, die in der Ontologie Ont definiert sind. Die Annotation erfolgte über die SAWSDL-Strategie aus Kapitel 3. Diese Vorbedingungen (Va) und (Vb) sind für alle semi-automatischen Regeln definiert.

(Va) $P_i.pv_j.node_m(c_a) \leftarrow sawsdl(Ont)$
(Vb) $S_s.node_r(c_b) \leftarrow sawsdl(Ont)$

Für jede Regel müssen zusätzliche Vorbedingungen (V1) - (V5) erfüllt sein, wie c_a und c_b zueinander in Ont in Beziehung stehen, damit die jeweilige Regel ausgeführt wird. Falls sie ausgeführt wird, berechnet sie einen neuen Matchkandidaten $mc_k = (P_i.pv_j.node_m,\ S_s.node_r,\ sim_{Pk})$ mit einem Ähnlichkeitswert sim_{Pk}, der jeweils in (E1) - (E5) für jede Regel definiert ist. Falls keine Regel ihre Vorbedingung erfüllt, erhält sim_{Pk} den Wert 0 für den jeweiligen Match bzw. es werden ggf. noch automatische Matchregeln ausgeführt. Die genaue Berechnung der Ähnlichkeitswerte bei mehreren Regelausführungen für eine Kombination ist in Kapitel 4.7.4 erläutert. Die Beschreibung der einzelnen Regeln und ihrer Vorbedingungen folgen hier im Anschluss.

4.3.1 Matchregel für gleiche Annotationen (R1)

Die Matchregel für gleiche Annotationen (R1) wird unter folgender Vorbedingung (V1) in Ont in Abb. 4.3 angewendet:

(V1) $c_a = c_b$

(E1) $sim_{Pk} = 1$

Das bedeutet, dass das Annotationskonzept c_a des Prozesselements $P_i.pv_j.node_m$ identisch zu dem Annotationskonzept c_b des Schemaelementes $S_s.node_r$ ist.

Durch Anwendung der Regel R1 werden diese gleich annotierten Matchpartner gesucht und als neuer Matchkandidat mc_k abgelegt. Der Matchkandidat mc_k hat den Ähnlichkeitswert $sim_{Pk} = 1$, da eine gleiche Annotation durch den Benutzer für eine hohe Wahrscheinlichkeit der Korrektheit des Matches spricht. Mögliche falsche Matches können durch die Filterregeln bereinigt werden.

Abb. 4.4 zeigt, wie die Regel R1 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.inputData.input.CustomerData...contact* und dem Schemaelement *CanvassSchema.Customer.Address* findet. R1 erstellt den Matchkandidaten mit $sim_p = 1$, da die Matchpartner mit gleichem Konzept *Address* (siehe Abb. 4.4 (a) + (b)) annotiert wurden.

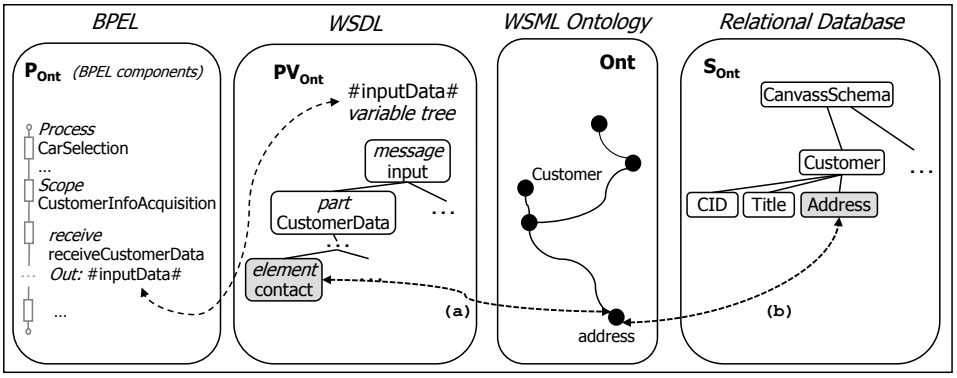


Abb. 4.4: Match im Beispielszenario mit Regel R1

Die Linien in *Ont* in dieser und den folgenden Abbildungen symbolisieren dabei die Beziehung, die die Elternkonzepte zu den Domänenkonzepte ihrer Attribute beschreiben.

4.3.2 Matchregel für Synonyme (R2)

Die Matchregel für Synonyme (R2) wird unter folgender Vorbedingung (V2) in *Ont* in Abb. 4.3 angewendet:

(V2) $c_a = synonymOf(c_b)$

(E2) $sim_{Pk} = 1$

Das bedeutet, das Annotationskonzept c_a des Prozesselements $P_i.pv_j.node_m$ ist ein Synonym zu dem Annotationskonzept c_b des Schemaelementes $S_s.node_r$.

Durch Anwendung der Regel R2 werden durch Synonyme annotierte Matchpartner gesucht und als neue Matchkandidaten abgelegt. Der neue Matchkandidat mc_k hat den Ähnlichkeitswert $sim_{Pk} = 1$, unter der Voraussetzung, dass die Synonymbeziehung in der Ontologie korrekt modelliert wurde. Eine Annotation durch den Benutzer durch synonyme Konzepte spricht genau wie eine gleiche Annotation für eine hohe Wahrscheinlichkeit der Korrektheit des Matches. Mögliche falsche Matches können durch die späteren Filterregeln bereinigt werden.

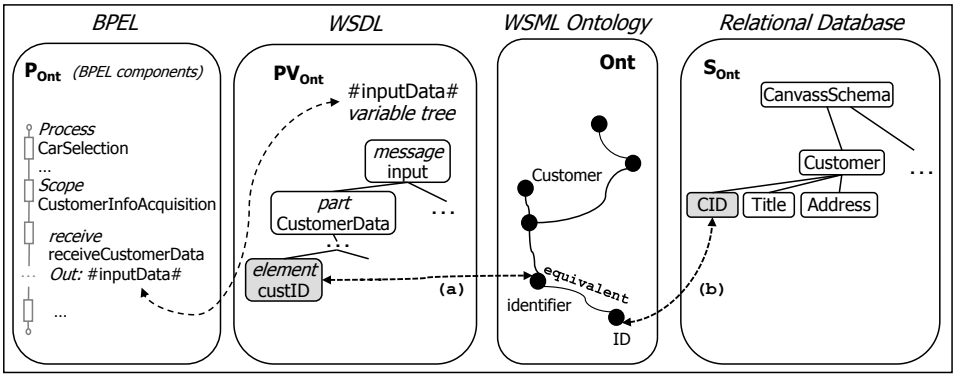


Abb. 4.5: Match im Beispielszenario mit Regel R2

Abb. 4.5 zeigt, wie die Regel R2 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.inputData.input.CustomerData...custID* und dem Schemaelement *CanvassSchema.Customer.CID* findet. R2 erstellt den Matchkandidaten mit $sim_P = 1$, da die Matchpartner mit synonymen Konzepten *Identifier* und *ID* (siehe Abb. 4.5 (a) + (b)) annotiert wurden.

4.3.3 Matchregel für Subkonzepte (R3)

Die Matchregel für Subkonzepte (R3) wird unter folgender Vorbedingung (V3a) in *Ont* in Abb. 4.3 angewendet:

(V3a) $c_a = [SubConceptOf(c_x[, c_y...]) = ... =] SubConceptOf(c_b)$
[(V3b) $c_a = ... = SubConceptOf(domainOfProperty = c_b)$]
[(V3c) $c_b = [SubConceptOf(c_x[, c_y...]) = ... =] SubConceptOf(c_a)$]
[(V3d) $c_b = ... = SubConceptOf(domainOfProperty = c_a)$]

(A3) $ReducedHierarchy\ AB$ = Ersetze für jede Hierarchieebene zwischen c_a bis

c_b die Subkonzepte durch ihre Superkonzepte

(E3) $sim_p = \frac{\#c_b \ in \ ReducedHierarchy \ AB}{\#Konzepte \ in \ ReducedHierarchy \ AB}$

Das bedeutet, das Annotationskonzept c_a des Prozesselements $P_i.pv_j.node_m$ ist ein Subkonzept von dem Annotationskonzept c_b des Schemaelementes $S_s.node_r$ (bzw. andersherum in (V3c)). Folgende Erläuterungen sowie (E3) sind für den Fall beschrieben, dass c_a ein Subkonzept von c_b ist. Sie können jedoch auch für die andere Richtung angewendet werden. *Ont* muss dann eine Beziehung zwischen den zwei Konzepten definieren, sodass c_a entweder eine unmittelbare Unterklasse von c_b beschreibt (ohne ein dazwischenliegendes Konzept c_x) oder über mehrere Vererbungen einige Unterkonzepte tiefer in der Vererbungshierarchie von c_b angeordnet ist. Aufgrund von evtl. Mehrfachvererbung kann c_a auch aus einer Menge von Konzepten c_x, c_y als Subkonzept definiert werden. Die Hierarchieebenen in einer Ontologie definieren sich durch die rekursiven Definitionen eines Konzepts als Subkonzept von wiederum anderen Konzepten aus der nächsten Hierarchiestufe. Das Beispiel in Abb. 4.6 besitzt drei Hierarchieebenen zwischen den Konzepten c_a und c_b, bei denen c_a als Subkonzept von c_b über die Konzepte c_3 und c_6 hervorgeht. Die Höhe des Teilhierarchiebaumes ist zwischen c_a und c_b also drei. Das Konzept c_a wird in Abb. 4.6 als Subkonzept der Menge c_1, c_2 und c_3 definiert. Falls eine Subkonzeptbeziehung zwischen c_a und c_b so komplex aufgebaut ist, muss später im ETL-Prozess überprüft werden, ob die konkreten Instanzdaten diese Einschränkungen in der Subkonzeptbeschreibung erfüllen und der evtl. gefundene Match zwischen den Elementen, die mit c_a und c_b annotiert wurden, für die jeweiligen Ausprägungen bestehen bleibt.

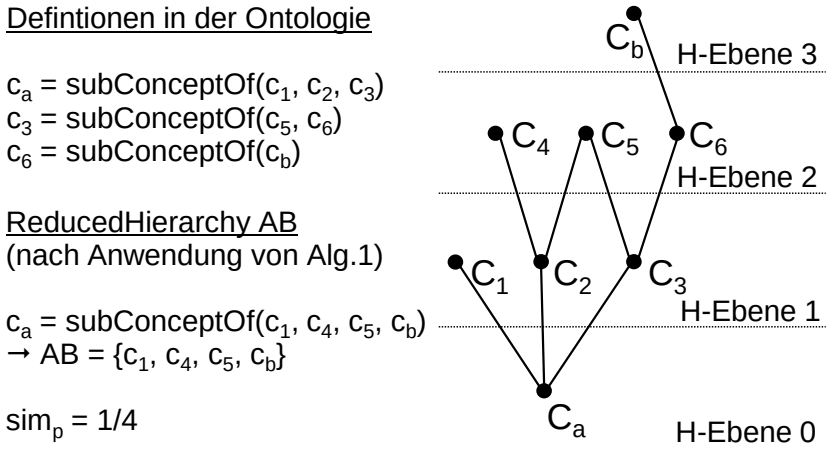


Abb. 4.6: Beispiel für Hierarchieebenen der Subkonzeptdefinitionen zwischen c_a und c_b

Die Subkonzeptbeziehung zwischen den zwei Konzepten c_a und c_b kann einfach sein wie in (V3a), sodass eine einfache Vererbungshierarchie aufgebaut wird, die besagt, dass c_b zu einer Menge von Konzepten gehört, die zusammen ein Konzept c_a definieren. Neben der Definition der Unterklasse als Konzept kann R3 stattdessen auch Eigenschaften zwischen

Algorithmus 1 *getReducedSubConceptHierarchy*

Input: c_a, $c_b \in Ont$, $SubConceptDefinitions \in Ont$
Output: ReducedHierarchy AB

1: $AB \leftarrow getSubConceptHierarchy(c_a, c_b, SubConceptDefinitions)$
2: $LevelNumber \leftarrow getNumberOfLevels(AB)$
3: **for** $Hierarchylevel\ h = 1$ to $LevelNumber$ **do**
4: replace each concept in level h by its subconcept definitions
5: store resulting hierarchy in AB
6: **end for**

den Konzepten für einen Match verwenden, wenn das Konzept c_b die Domäne einer Eigenschaft beschreibt und als Subkonzeptdefinition für c_a dient (V3b) (bzw. andersherum in (V3d)).

Diese Menge von verschiedenen Konzepten, aus der die Subkonzeptbeziehungen aufgebaut sein können, wird in der Ähnlichkeitsberechnung des Matches von R3 berücksichtigt. Der neue Matchkandidat $mc_k = (P_i.pv_j.node_m, S_s.node_r, sim_{Pk})$ hat einen Ähnlichkeitswert, der durch einen Ausdruck in (E3) berechnet wird. sim_{Pk} ist der Anteil von c_b an der Menge aller Konzepte, die für die Definition von dem Subkonzept c_a benötigt werden, bei der u.a. auch c_b enthalten ist. Dazu muss zunächst Aktion (A3) ausgeführt werden, um alle Subkonzepte in den Hierarchieebenen zwischen c_a und c_b durch ihre Subkonzeptdefinitionen zu ersetzen. A3 wird durch Algorithmus 1 ausgeführt. Algorithmus 1 (*getReducedSubConceptHierarchy*) bestimmt zunächst mit der Funktion *getSubConceptHierarchy* alle Subkonzeptbeziehungen in *Ont* zwischen den Konzepten c_a und c_b und speichert sie als Teilhierarchie *AB*. Für diese Teilhierarchie AB berechnet die Funktion *getNumberOfLevels* alle Ebenen wie in Abb. 4.6 angegeben. Dann werden die Konzepte jeder Hierarchieebene h in der Hierarchie AB gemäß ihrer Definition in der Ontologie durch ihre Subkonzepte ersetzt (Zeile 3 bis 6). In (E3) wird der Anteil der Anzahl der Konzepte von c_b an der Zahl der Konzepte in der durch den Algorithmus reduzierten und übrig gebliebenen Konzepte der Hierarchie AB berechnet. Mehr zu der Berechnung der Ähnlichkeitswerte folgt in Kapitel 4.7.3.

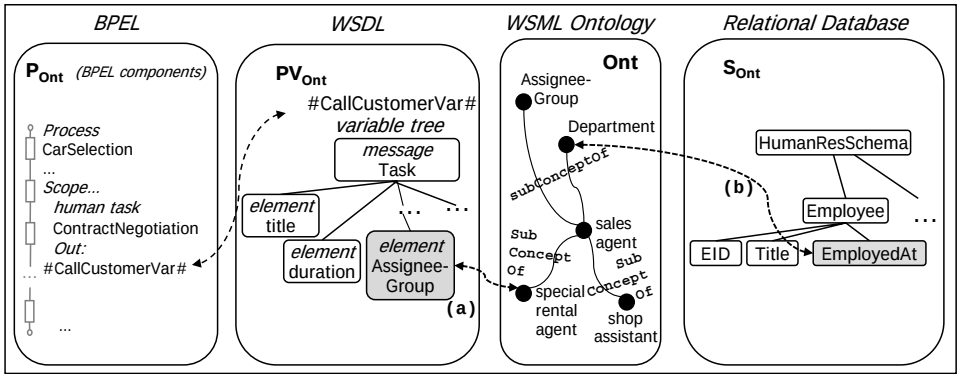


Abb. 4.7: Match im Beispielszenario mit Regel R3

Abb. 4.7 zeigt, wie die Regel R3 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.CallCustomerVar.Task.EmployeeTask.AssigneeGroup* und dem Schemaelement *HumanResSchema.Employee.EmployedAt* auf Basis der gegebenen Subkonzeptbeziehungen in *Ont* zwischen dem Konzept *SpecialRentalAgent* findet, mit dem das Element AssigneeGroup annotiert wurde (a) und dem Konzept *Department*, mit dem das operative Element *EmployedAt* (b) annotiert wurde. R3 erstellt den Matchkandidaten mit $sim_P = \frac{1}{2}$, da zwischen den Konzepten *SpecialRentalAgent* und *Department* Subklauseln mit zwei verschiedenen Wurzeln eingesetzt werden können.

4.3.4 Matchregel für ontologie-interne Axiome (R4)

Eine Ontologie kann neben den definierten Beziehungen zwischen zwei Konzepten wie z.B. Synonym oder Unterklasse, auch Regeln, sogenannte Axiome, beinhalten. Dadurch werden Konzepte miteinander in Beziehung gesetzt und neue Zusammenhänge zwischen den Elementen, die mit diesen Konzepten annotiert wurden, erkannt. Die Matchregel für ontologie-interne Axiome (R4) wird unter folgenden Vorbedingungen (V4a) bis (V4b) in *Ont* in Abb. 4.3 angewendet:

(V4a) $Axiom\ a_t \in Ont\ mit\ a_t : atom_i(?x, ...)\ and ... \rightarrow atom_j(?x, ...)\ and ...$
(V4b) $c_a = domain\ of\ atom_i, c_b = domain\ of\ atom_j$
[(V4c) $c_b = domain\ of\ atom_i, c_a = domain\ of\ atom_j$]

(E4) $sim_{Pk} = \frac{\#c_a\ und\ \#c_b\ in\ den\ Domains\ der\ Atome\ des\ Axioms\ a_t}{\#Atome\ in\ Axiom\ a_t}$

Das bedeutet, es ist in (V4a) ein Axiom a_t in *Ont* definiert mit $atom_i$ in der Prämisse und $atom_j$ in der Konklusion mit dem gleichen Parameter $?x$, das Annotationskonzept c_a des Prozesselements $P_i.pv_j.node_m$ beschreibt die Domäne von $atom_i$ und das Annotationskonzept c_b des Schemaelementes $S_s.node_r$ beschreibt die Domäne von $atom_j$ (V4b) (bzw. alternativ in (V4c) beide Domänen vertauscht).

In OWL wird für die Regelerstellung die Sprache Semantic Web Rule Language (SWRL) [HPSB+04] verwendet, in WSMO [dBBD+05] sind die Axiome schon Teil der Ontologiesprache. Ein Axiom besteht aus einer Prämisse und einer Konklusion in der Form $Prämisse \rightarrow Konklusion$, von denen beide aus einer Konjunktion oder Disjunktion von Atomen $(atom_1, ..., atom_n)$ bestehen können. Wenn die Prämisse erfüllt ist, ist auch die Konklusion wahr. Die Atome sind von der Form $atom_m(?x)$, $atom_p(?x, ?y)$ oder $builtIn(r, ?x, ...)$, wobei $atom_m$ ein Konzept oder ein Datenbereich ist, $atom_p$ eine Eigenschaft und r eine Standardfunktion wie z.B. numericEqual(?x, ?y). x und y sind Parameter für Instanzen oder Datenwerte der definierten Domäne von $atom_m$ oder $atom_p$ in *Ont*. Damit R4 ein Ergebnis findet, muss *Ont* also ein Axiom zwischen den zwei Konzepten c_a und c_b definieren, sodass c_a eine Domäne eines Atoms in der Prämisse oder Konklusion beschreibt und c_b eine Domäne eines Atoms in der entsprechend anderen Seite des Axioms. Es muss dabei eine neue Beziehung zwischen den zwei Konzepten

entstehen, die es entweder vorher noch nicht in *Ont* gab und daher bisher auch kein Matchkandidat dafür gefunden wurde. Oder es wird durch R4 ein höherer Ähnlichkeitswert für einen existierenden Match berechnet. Die Erfüllung des gesamten Axioms für die einzelnen Ausprägungen der Variablen und Schemaelemente muss später im ETL-Prozess überprüft werden. Dort wird überprüft, ob die konkreten Instanzdaten auch alle anderen Atome in der Bedingung erfüllen und der durch R4 gefundene Match zwischen den Elementen für die jeweiligen Ausprägungen bestehen bleibt.

Die Atome mit Domänen im Axiom, die andere Konzepte als c_a und c_b referenzieren, werden in der Ähnlichkeitsberechnung des Matches auch berücksichtigt. Der neue Matchkandidat mc_k hat den Ähnlichkeitswert in (E4), d.h. sim_{Pk} ist der Anteil der Konzepte c_a und c_b in den Domänenkonzepten der Atome an der Anzahl aller Atome, also inklusive aller anderen Domänenkonzepte, in dem Axiom a_t. Falls also das Axiom nur unter der Bedingung vieler anderer Atome mit anderen Domänen erfüllt wird, wird der Ähnlichkeitswert des gefundenen Matches sehr klein. sim_{Pk} bekommt den höchsten Wert 1 nur dann, wenn alle Domänen der Atome durch eines der betreffenden Konzepte c_a oder c_b definiert werden. Details zu der Berechnung von sim_{Pk} finden sich in Kapitel 4.7.3.

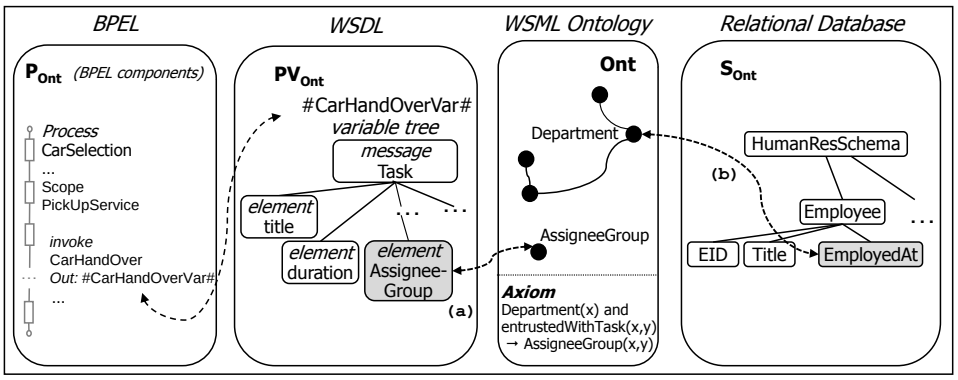


Abb. 4.8: Match im Beispielszenario mit Regel R4

Abb. 4.8 zeigt, wie die Regel R4 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.CarHandOverVar.Task.EmployeeTask.AssigneeGroup* und dem Schemaelement *HumanResSchema.Employee.EmployedAt* auf Basis des gegebenen Axioms in *Ont* findet. Das Axiom definiert eine Regel, die besagt, dass eine Abteilung, die mit einer Aufgabe vertraut wurde, als *AssigneeGroup* bezeichnet werden. R4 erstellt den Matchkandidaten mit $sim_P = \frac{3}{3}$, da die Atome alle durch eines der Konzepte *AssigneeGroup* oder *Department* annotiert werden (siehe Abb. 4.8 (a) + (b)).

4.3.5 Matchregel für Union (R5)

Die Matchregel für Union (R5) wird unter folgender Vorbedingung (V5a) in *Ont* in Abb. 4.3 angewendet:

(V5a) $c_a = union(c_b, ...)$
[(V5b) $c_b = union(c_a, ...)$]

(E5) $sim_{Pk} = 1$

Das bedeutet, das Annotationskonzept c_a des Prozesselements $P_i.pv_j.node_m$ ist eine Vereinigung in *Ont* von mehreren Konzepten darunter auch das Annotationskonzept c_b des Schemaelementes $S_s.node_r$ (bzw. in (V5b) alternativ auch in umgekehrter Reihenfolge).

Durch Anwendung der Regel R5 werden die Matchpartner gesucht, deren Annotationskonzepte über Union in Verbindung stehen, und als neuer Matchkandidat abgelegt. Der neue Matchkandidat mc_k hat den Ähnlichkeitswert in (E5). Die restlichen beteiligten Konzepte in der Vereinigung müssen dabei nicht erfüllt sein und das Matchergebnis von R5 daher später nicht mit den Instanzdaten geprüft werden. Schließlich ist die Vereinigung so definiert, dass nur eines der Konzepte in ihrer Klausel erfüllt sein muss.

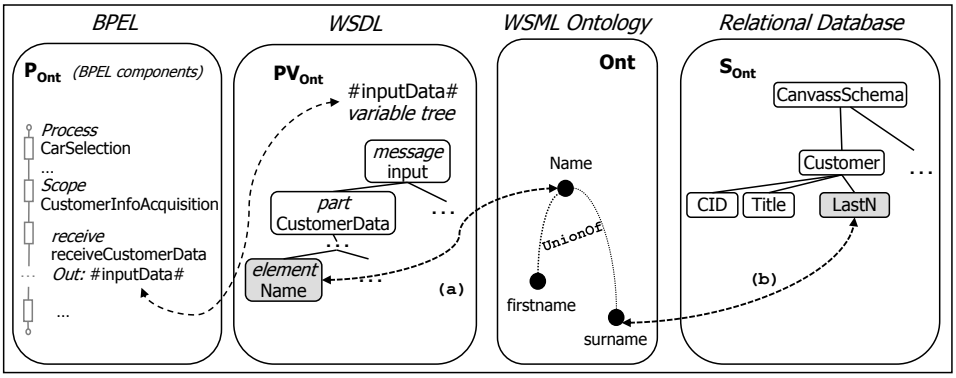


Abb. 4.9: Match im Beispielszenario mit Regel R5

Abb. 4.9 zeigt, wie die Regel R5 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.inputData.input.CustomerData...Name* und dem Schemaelement *CanvassSchema.Customer.LastN* findet. R5 erstellt den Matchkandidaten mit $sim_P = 1$, da die Elemente mit den Konzepten *Name* bzw. *Surname* (siehe Abb. 4.9 (a) + (b)) annotiert wurden und diese Konzepte über eine Union-Eigenschaft in der Ontologie zusammenhängen.

4.4 Automatische Regeln für das Pairing partiell-annotierter Elemente

Dieses Kapitel stellt ausführlich die Regeln für die automatische Integration vor. Diese Regeln werden hauptsächlich bei Prozessen und Schemas angewendet, deren Elemente nur

partiell semantisch annotiert sind. Die Regeln ziehen u.a. die linguistische Namensgebung der Elemente heran oder der Prozesskontext, in dem die Variablen verwendet werden.

Im Folgenden werden fünf Regeln (R6 bis R10) vorgestellt, die in dem Framework für die automatische Integration verantwortlich sind bzw. sie verhindern. Wie bei den semi-automatischen Verfahren konzentrieren sich die Regeln auf reine Modellintegration und die Integration ihrer Ausprägungen wird erst in der Warehousephase beachtet. Die Regeln 6, 7, 8 stammen aus dem Bereich des Schema-Matchings. Regel 6 ist genau so übernommen worden, Regeln 7 und 8 sind auf BIA angepasst worden. Regeln 9 und 10 sind Neuentwicklungen.

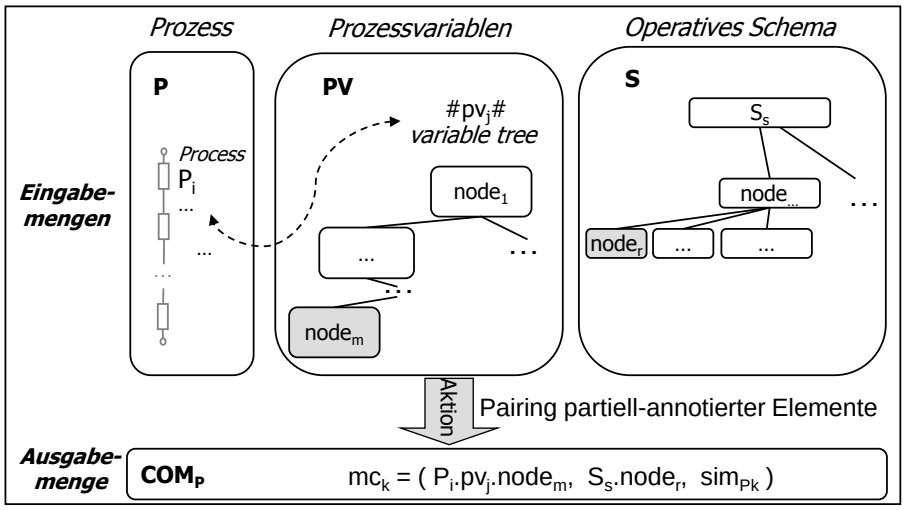


Abb. 4.10: Pairing partiell-annotierter Elemente

Die automatischen Regeln haben alle den gleichen Aufbau. Zur Eingabemenge gehören annotierte, teilweise annotierte und nicht-annotierte Prozesselemente PV von Prozessen P, die wiederum partiell annotiert sein können oder auch gar nicht. Außerdem gehören zur Eingabemenge sowohl annotierte als auch nicht-annotierte Schemaelemente S. Die Regeln suchen in jedem Prozessmodel $P_i \in P$ zu jedem Variablenelement $pv_j.node_m$ aus PV passende operative Elemente $S_s.node_r$ aus der Elementmenge S. Je nachdem, ob die Vorbedingungen der Regeln erfüllt sind (V6) - (V10), wird der Matchkandidat $mc_k = (P_i.pv_j.node_m, S_s.node_r, sim_{Pk})$ mit dem Ähnlichkeitswert sim_{Pk} in (E6) - (E10) berechnet.

4.4.1 Linguistische Matchregel (R6)

Für die linguistische Matchregel (R6) gibt es keine Vorbedingung, da jedes Element einen Namen hat. R6 berechnet den linguistischen Ähnlichkeitswert zwischen dem Namen des Prozesselements $P_i.pv_j.node_m$ und dem Namen des Schemaelementes $S_s.node_r$. Die Regel wird für jede Kombination aus PV und S ausgeführt. Damit es zu einem sinnvollen Ergebnis kommt, sollten die Elemente sprechende Namen besitzen.

(V6) $\emptyset$

(E6) $$sim_{Pk} = \frac{2 * \left(\sum_{t1 \in P_i.pv_j.node_m(label)} \max_{t2 \in S_s.node_r(label)} [sim_{lex}(t1,t2)] \right)}{\#token(P_i.pv_j.node_m(label)) + \#token(S_s.node_r(label))}$$

Durch Anwendung der Regel R6 wird für jede mögliche Kombination der Matchpartner die Ähnlichkeit berechnet und als neuer Matchkandidat abgelegt. Für die Regel R6 kann ggf. ein Lexikon als weiterer Teil der Eingabemenge aufgenommen werden, um die Ähnlichkeitswerte zu berechnen. Im Lexikon sind semantische Beziehungen (Synonyme, Hyponyme, Abkürzungen) aufgelistet. Falls kein Lexikon vorhanden ist, werden die Elementnamen nur bezüglich ihrer syntaktischen Namensgleichheit verglichen. R6 teilt die Namen (Label) der Variable und des jeweiligen operativen Schemaelements in einzelne Tokens auf, um von jedem einzelnen die linguistische Ähnlichkeit sim_{lex} zu berechnen. Die einzelnen Token werden dabei durch Großschreibung oder Punkt- bzw. Unterstrich getrennt und auf diese Weise von dem Tool erkannt. Wenn die Tokens syntaktisch gleich sind (oder bei Verwendung eines Lexikons auch synonym, hyponym (ein Unterbegriff) oder eine Abkürzung sind), dann beträgt $sim_{lex} = 1$, ansonsten 0. So ergibt sich insgesamt für die Kombination ein Wert sim_{Pk} wie in (E6) definiert (siehe auch [MBR01]). Zu jedem Token im Namen von $P_i.pv_j.node_m$ wird sim_{lex} zu jedem Token im Namen von $S_s.node_r$ berechnet und der höchste Wert davon aufaddiert. Der Quotient wird aus diesem Ergebnis durch die Summe der Tokenanzahl aus beiden Elementnamen gebildet.

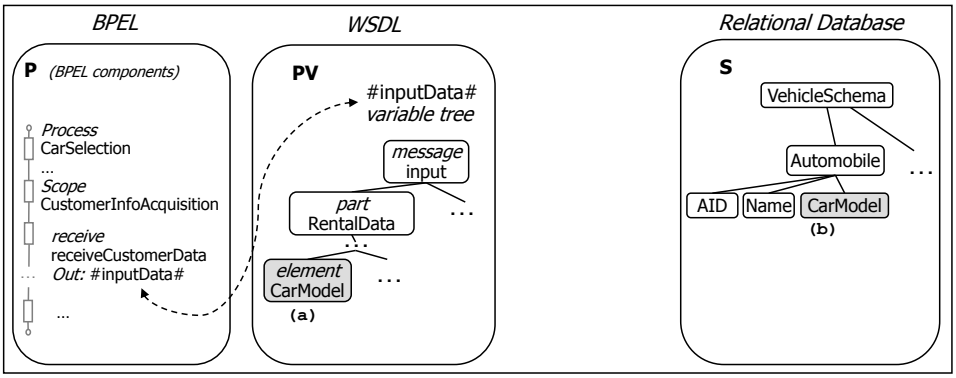


Abb. 4.11: Match im Beispielszenario mit Regel R6

Abb. 4.11 zeigt, wie die Regel R6 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.inputData.input.RentalData.CarModel* und dem Schemaelement *VehicleSchema.Automobile.CarModel* findet. Das Ergebnis von R6 basiert auf dem gleichen Label *CarModel* des Prozesselements und der operativen Spalte der Tabelle *Automobile*. R6 erstellt den Matchkandidaten mit $sim_p = \frac{2*(1+1)}{2+2} = \frac{4}{4}$, da das gematchte Label hier jeweils nur aus dem gleichen Token besteht (siehe Abb. 4.11 (a) + (b)).

4.4.2 Linguistische Matchregel für Ontologiekonzepte (R7)

Für die linguistische Matchregel für Ontologiekonzepte (R7) muss zusätzlich zu Abb. 4.10 die Vorbedingung (V7a) gelten.

(V7a) $P_i.pv_j.node_m(c_a) \leftarrow sawsdl(Ont)$ [(V7b) $S_s.node_r(c_a) \leftarrow sawsdl(Ont)$]

$$\text{(E7) } sim_{Pk} = \frac{2 * \left(\sum_{t1 \in c_a(label)} max_{t2 \in S_s.noder(label)}[sim_{lex}(t1,t2)] \right)}{\#token(c_a(label)) + \#token(S_s.node_r(label))}$$

Das bedeutet, das Prozesselement (oder alternativ das operative Schemaelement in (V7b)) muss mit dem Konzept c_a aus *Ont* annotiert und *Ont* in der Eingabemenge sein. Das jeweils andere Element kann, muss aber nicht annotiert sein.

Durch Anwendung der Regel R7 wird wie in Regel R6 die linguistische Ähnlichkeit der Elemente berechnet. Allerdings nimmt R7 vorhandene Ontologiekonzepte der annotierten Elemente zu Hilfe und vergleicht dieses Konzept mit dem Namen (Label) des anderen nicht annotierten Elementes. Im Folgenden wird die Funktionsweise von R7 mit der Annotation von $P_i.pv_j.node_m$ erläutert, kann aber ebenso mit der Annotation des Schemaelements verwendet werden. Die Verwendung eines Lexikons ist bei dieser Regel, wie bei R6, optional.

Durch Anwendung der Regel R7 wird für jeden Matchpartner die Ähnlichkeit berechnet und als neuer Matchkandidat abgelegt. Der Konzeptname und der zu matchende Name des nicht annotierten Elements $S_s.node_r$ werden von R7 in einzelne Tokens aufgeteilt, um von jedem einzelnen die linguistische Ähnlichkeit sim_{lex} zu berechnen. Wenn die Tokens gleich sind oder sie synonym, hyponym oder eine Abkürzung sind, wie es im Lexikon definiert wurde, dann beträgt $sim_{lex} = 1$, ansonsten 0. Zu jedem Token im Namen von c_a wird sim_{lex} zu jedem Token im nicht annotierten Elementnamen von $S_j.node_j$ berechnet und der höchste Wert aufaddiert. Der Quotient wird aus diesem Ergebnis durch die Summe der Tokenanzahl aus beiden verglichenen Labels gebildet. So ergibt sich sim_{Pk} aus (E7) analog zu E6.

Abb. 4.12 zeigt, wie die Regel R7 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.inputData.input.RentalData...rentalEndDate* und dem Schemaelement *CanvassSchema.CalendarData.CalendarDate* findet. Das Ergebnis von R7 basiert auf dem Annotationskonzept *CalendarDate* des Prozesselements und dem Namen *CalendarDate* der operativen Spalte der Tabelle Calendar (siehe Abb. 4.12 (a) + (b)). R7 erstellt den Matchkandidaten mit $sim_P = \frac{2*(1+1)}{2+2} = 1$, da die Token die identischen Namen besitzen.

Bei dieser Regel entstehen jedoch Abhängigkeiten zu Definitionen von Beziehungen in *Ont*, die in R1 - R5 untersucht werden (Synonyme, Subkonzepte, usw.). Diese Beziehungen sollten auch bei R7 nicht außer Acht gelassen werden, d.h. sobald ein Match zwischen c_a und einem Elementnamen gefunden wurde, müssen auch alle Elemente der Konzepte, die in einer Beziehung zu c_a stehen, die in R1 - R6 verwendet wurden, diesen Match

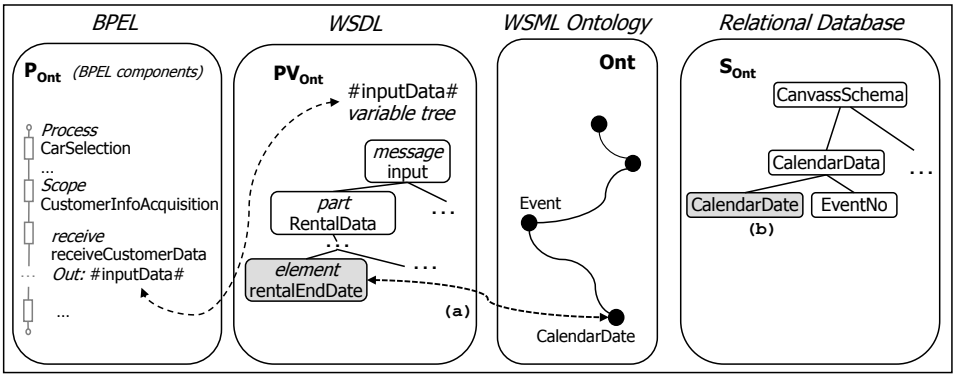


Abb. 4.12: Match im Beispielszenario mit Regel R7

mit dem gleichen nicht annotierten Matchpartner und Ähnlichkeitswert erhalten. Das geschieht unabhängig davon, ob die Elementnamen zwischen ihnen einen Match ergeben würde. Mehr dazu in Kapitel 4.7 bei der Kontrollstrategie des Frameworks.

4.4.3 Matchregel für Eliminierung (R8)

Durch Anwendung der Regel R8 werden bestimmte Elemente, die in der Eliminierungsliste angegeben sind, eliminiert, d.h. sie stehen als eventueller Matchpartner nicht mehr zur Verfügung. Das ist besonders wichtig für Variablenelemente aus bestimmten Aktivitäten, die zwar linguistisch mit operativen Daten matchbar wären, die allerdings semantisch ganz andere Sachverhalte beschreiben. In Prozessaktivitäten der *Human Tasks* gibt es z.B. viele Variablenelemente, die Steuerungsinformationen für den Ablauf dieser Aktivität speichern. Die Namen der Elemente sind zum Teil identisch mit einigen operativen Schemaelementnamen. Jedoch wäre ein Match, wie er mit R6 gefunden worden wäre, falsch.

Die Matchregel für Eliminierung (R8) wird unter folgenden Vorbedingungen (V8a) - (V8c) in Abb. 4.10 angewendet:

(V8a) $Aktivitat\ a_b(P_i.pv_j.node_m)$
(V8b) $a_b(type) = Human\ Task$
(V8c) $P_i.pv_j.node_m(label) \in Eliminierungsliste$

(E8) $sim_{Pk} = 0$

Das bedeutet, es wird geprüft, ob die Aktivitäten a_b im Prozess, die das Prozesselement $P_i.pv_j.node_m$ bearbeiten (V8a), vom Typ *Human Task* sind (V8b) und der Name (Label) des Prozesselements in der Eliminierungsliste (Ausschnitt in Abb. 4.13 für BIA) definiert ist (V8c).

In der Eliminierungsliste sind die Namen der Variablen von Aktivitäten aufgelistet, deren Elemente nicht als Matchpartner geeignet sind. In einer Human-Task-Aktivität macht es

z.B. nur Sinn, die durch den Task veränderbaren Daten sowie die verantwortlichen Angestellten für den Task (die Angestellten, die den Task ausführen dürfen und diejenigen, die ihn letztendlich ausgeführt haben) mit den operativen Daten zu matchen. Wenn das Label des Variablenelements $pv_i.node_m$ mit dem Aktivitätstyp in dieser Liste auftaucht, wird das Variablenelement für weitere Matchregeln ausgeschlossen. Die Regel wird vor allen anderen automatischen Matchregeln ausgeführt, damit es nicht zu unnötigen Regelanwendungen für diese Variablen kommt. Eliminierung heißt also in dem Framework, dass $P_i.pv_j.node_m$ in den anderen Matchregeln nicht mehr berücksichtigt wird. Die Liste kann bei Bedarf um weitere Prozesselemente erweitert und an andere Aktivitätstypen angepasst werden.

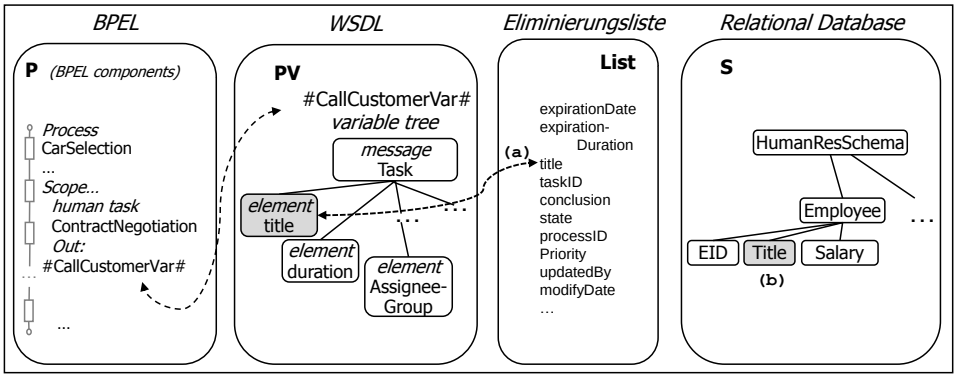


Abb. 4.13: Match im Beispielszenario mit Regel R8

Abb. 4.13 zeigt, wie die Regel R8 in unserem Beispielszenario aus Kapitel 1.3 ein Prozesselement *CarSelection.ContractNegotiation.Task.EmployeeTask.title* eliminiert. Das Ergebnis von R8 basiert auf der Eliminierungsliste, die das Prozesselements mit dem Namen *title* (siehe Abb. 4.13 (a)) von den Matchkandidaten ausschließt. *Title* ist für die interne Koordinierung des Tasks im Workflow zuständig und ein Match mit den nichtfunktionalen operativen Daten wäre falsch. R8 verhindert durch die Eliminierung, dass eine spätere Anwendung z.B. von R6 den falschen Matchkandidaten zwischen diesem Prozesselement und *HumanResSchema.Employee.Title* (siehe Abb. 4.13 (b)) findet, was die Genauigkeitsrate des Frameworks herabsetzen würde.

4.4.4 Matchregel für Datenfluss (R9)

Die Matchregel für Datenfluss (R9) wird unter folgenden Vorbedingungen (V9a) -(V9d) in Abb. 4.10 angewendet.

(V9a) $P_i.pv_j.node_m \leftarrow expression(P_i.pv_x.node_n, ...) \in Aktivitat\ a_b$
(V9b) $a_b(type) = assign$
(V9c) $expression = copy\ ohne\ Modifikationen$
(V9d) $mc_i = (P_i.pv_x.node_n, S_s.node_r, sim_i) \in COM$

(E9) $sim_{Pk} = sim_i$

Das bedeutet, es wird geprüft, ob mindestens eine Aktivität a_b im Prozess, die das Prozesselement $P_i.pv_x.node_n$ in ihrem Ausdruck verwendet (V9a), vom Typ *Assign* ist (V9b) und den Wert der Prozessvariable $P_i.pv_x.node_n$ auf die Prozessvariable $P_i.pv_j.node_m$ ohne Modifikationen kopiert (V9c). Außerdem muss mindestens ein Matchkandidat mc_i mit $P_i.pv_x.node_n$ existieren (V9d). Wenn die Vorbedingungen erfüllt sind, wird die Regel R9 ausgeführt. Sie erstellt für $P_j.pv_j.node_m$ einen neuen Matchkandidaten mc_k mit dem gleichen operativen Element $S_s.node_r$ aus mc_i als Matchpartner und dem gleichen Ähnlichkeitswert sim_i aus mc_i (E9). Sobald in der Assign-Aktivität jedoch Modifikationen auftreten, d.h. wenn die Variable durch arithmetische Operationen, Stringoperationen usw. verändert wird, kann keine semantische Übereinstimmung der Variablen gewährleistet werden. In solchen Fällen kann daher keine einfache Übertragung der Matchpartner und Ähnlichkeitswerte erfolgen und R9 wird nicht ausgeführt.

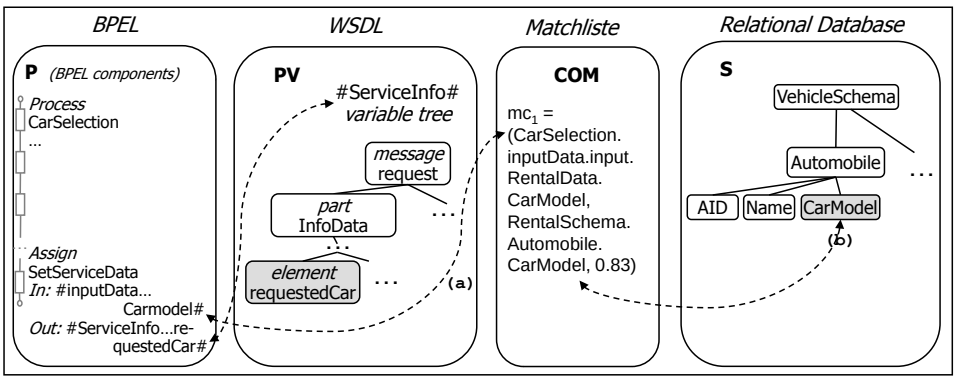


Abb. 4.14: Match im Beispielszenario mit Regel R9

Abb. 4.14 zeigt, wie die Regel R9 in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten zwischen dem Prozesselement *CarSelection.ServiceInfo.request.RentalData.requestedCar* und dem Schemaelement *RentalSchema.Automobile.CarModel* findet. Das Ergebnis von R9 basiert auf dem Matchergebnis mc_1 mit dem Prozesselement *CarSelection.inputData.input.RentalData.CarModel* (siehe (a) aus Abb. 4.14) aus dem Beispiel in Kapitel 4.4.1 von R6. Da die Assign-Aktivität *SetServiceData* die Variable *#inputData.input.RentalData.CarModel#* auf die Variable *#ServiceInfo.request.RentalData.requestedCar#* kopiert, wird ein neuer Matchkandidat erstellt durch Übernehmen des operativen Schemaelements *VehicleSchema.Automobile.CarModel* (siehe (b) aus Abb. 4.14) als neuen Matchpartner und des Ähnlichkeitswerts *0.83* aus mc_1.

4.4.5 Matchregel für Korrelationen (R10)

Die Komponente der Correlation Sets als Bestandteil der Beschreibung eines Geschäftsprozesses wurde bereits in Kapitel 2.1 erläutert. Wichtig für die Ausführung

der Regel R10 ist die Definition von einem <*property*>-Element, das als Alias für verschiedene Variablenelemente im Prozess dient. Ob die <*property*>-Elemente dann auch wirklich für die Definition eines Correlation Sets verwendet werden, ist für R10 nicht wichtig. Durch den Alias ist die Korrelation der beteiligten Variablenelemente bereits bestätigt.

R10 gliedert sich in drei Alternativen, die je nach Vorbedingungen bezüglich schon existierender Matchkandidaten auf die definierten Korrelationen ausgeführt werden. In der ersten Alternative (R10a) wird Regel R10 unter folgenden Vorbedingungen (V10a) - (V10d) in Abb. 4.10 angewendet.

(V10a) $Property\ p_n \in P_i$
(V10b) $Property\ p_n\ aliasFor\ P_i.pv_j.node_m$
(V10c) $Property\ p_n\ aliasFor\ P_i.pv_x.node_y$
(V10d) $mc_i = (P_i.pv_x.node_y, S_s.node_r, sim_i) \in COM$

(E10a) $sim_{Pk} = sim_i$

Das bedeutet, es wird geprüft, ob eine Property p_n im Prozess P_i definiert wird (V10a), die als Alias für das Prozesselement $P_i.pv_x.node_y$ dient (V10c), das als Matchpartner in einem Matchkandidaten mc_i verwendet wird (V10d). Außerdem muss diese Property p_n auch für ein weiteres Prozesselement $P_i.pv_j.node_m$ definiert sein (V10b). Wenn die Vorbedingungen erfüllt sind, wird die Regel R10 ausgeführt. Sie erstellt für $P_i.pv_j.node_m$ einen neuen Matchkandidaten mc_k mit dem gleichen operativen Element $S_s.node_r$ aus mc_i als Matchpartner und dem gleichen Ähnlichkeitswert sim_i aus mc_i (E10a).

In der zweiten Alternative (R10b) wird Regel R10 unter folgenden Vorbedingungen (V10e) - (V10h) in Abb. 4.10 angewendet.

(V10e) $Property\ p_n \in P_i$
(V10f) $Property\ p_n\ aliasFor\ P_i.pv_j.node_m$
[(V10g) $CorrelationSet\ cs_n \leftarrow p_n, ...$]
(V10h) $P_i.pv_j.node_m \notin COM$

(E10b) $$sim_{Pk} = \frac{2 * (\sum_{t1 \in p_n(label)} max_{t2 \in S_s.node_r(label)}[sim_{lex}(t1,t2)])}{\#token(p_n(label)) + \#token(S_s.node_r(label))}$$

Das bedeutet, es wird geprüft, ob eine Property p_n im Prozess definiert wird (V10e), die als Alias für das Prozesselement $P_i.pv_j.node_m$ dient (V10f), für das es dieses Mal anders als bei der Alternative (a) noch keinen Matchpartner gibt und es daher in keinem Matchkandidaten mc_i verwendet wird (V10h). Diese Property p_n kann für die Definition eines Correlation Sets cs_n verwendet werden (V10g). Diese Vorbedingung ist allerdings nur optional. Wenn die Vorbedingungen erfüllt sind, wird die Regel R10 ausgeführt. Sie benutzt die Label von p_n bzw. auch cs_n und versucht diese Namen mit operativen Schemaelementen mit Hilfe von linguistischem Namensmatching wie in R6

zusammenzuführen und den Ähnlichkeitswert sim_{Pk} in (E10b) zu berechnen. Falls ein Label von p_n oder cs_n auf ein operatives Element $S_s.node_r$ matcht, d.h. ihre Ähnlichkeit über einem festgesetzten Schwellwert liegt, wird diese Verknüpfung mit der berechneten Ähnlichkeit sim_{Pk} als neuer Matchkandidat $mc_k = (P_i.pv_j.node_m, S_s.node_r, sim_{Pk})$ abgelegt.

Die dritte Alternative (R10c) funktioniert ähnlich wie die zweite und hat die gleichen Vorbedingungen (V10e) - (V10h). Außerdem gilt noch die Vorbedingung (V10i).

(V10i) $S_s.node_r(c_a) \leftarrow sawsdl(Ont)$

(E10c) $$sim_{Pk} = \frac{2 * (\sum_{t1 \in p_n(label)} max_{t2 \in c_b(label)}[sim_{lex}(t1,t2)])}{\#token(p_n(label)) + \#token(c_b(label))}$$

Auch hier wird wieder linguistisches Namensmatching verwendet, dieses Mal ähnlich zu R7, um die Label von p_n bzw. cs_n mit den Konzeptlabeln der operativen Schemaelemente zusammenzuführen (E10c). Wird ein Match gefunden zwischen dem Namen von p_n bzw. cs_n und dem Namen des Konzeptes c_a, das $S_s.node_r$ annotiert, d.h. der Ähnlichkeitswert liegt über einem festgesetzten Schwellwert, wird der Matchkandidat $mc_k = (P_i.pv_j.node_m, S_s.node_r, sim_{Pk})$ erstellt. Hier können danach auch alle anderen semantischen Beziehungen zu c_b aus R1 - R5 für weitere Matchkandidaten mit $P_i.pv_j.node_m$ betrachtet werden.

Abb. 4.15 zeigt, wie die erste Alternative der Regel R10a in unserem Beispielszenario aus Kapitel 1.3 einen Matchkandidaten mc_2 zwischen dem Prozesselement *CarSelection.ServiceInfo.request.InfoData.ServiceRequest.Info* und dem Schemaelement *CanvassSchema.Customer.Address* findet. Das Ergebnis von R10 basiert auf dem bereits gefundenem Match mc_1 zwischen dem Prozesselement *CarSelection.inputData.input.CustomerData...contact* und dem Element *CanvassSchema.Customer.Address* (siehe Abb. 4.15 (a) und (b)), der in dem Beispiel mit R7 gefunden wurde. R10 verwendet neben diesem Match mc_1, die Property *CustomerAddress* und ihre Aliase, um den Match mc_2 zu finden. Da für das Prozesselement aus mc_1 das gleiche <*propertyAlias*>-Element *CustomerAddress* definiert ist wie für das Prozesselement *CarSelection...Info*, wird das operative Element aus mc_1 und der Ähnlichkeitswert übernommen und als Matchkandidat mc_2 abgelegt.

Die beiden anderen Alternativen von R10 funktionieren ähnlich, nur dass hier kein Match in der Matchliste mit einem Prozesselement aus dem <*propertyAlias*> existiert. Dagegen wird der Name der Property, also *CustomerAddress*, mit den operativen Elementen bzw. den Namen ihrer Annotationskonzepte verglichen. Im Falle eines Ähnlichkeitswertes über einem definierten Schwellwert wird er zusammen mit dem operativen Element und den Elementen im <*propertyAlias*> als zwei neue Matches gespeichert.

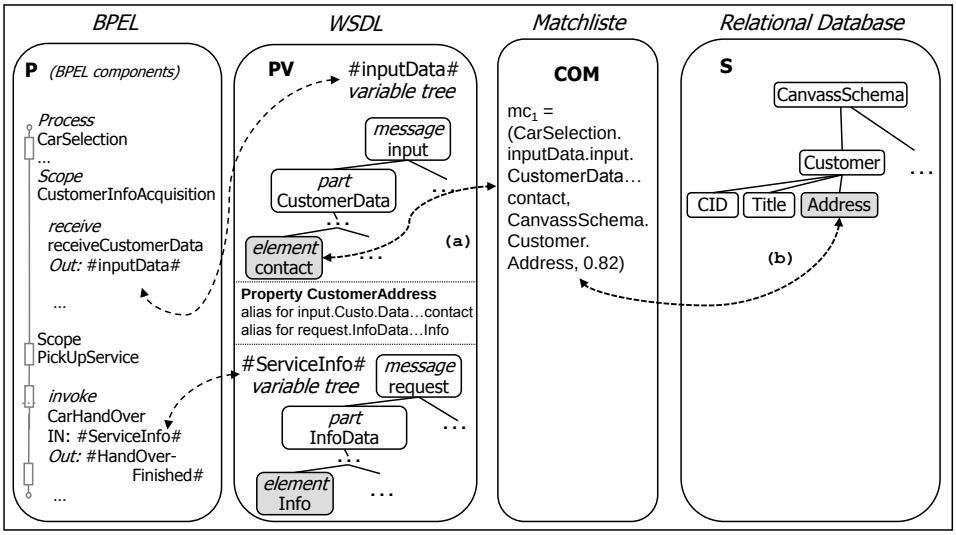


Abb. 4.15: Match im Beispielszenario mit Regel R10a

4.5 Regeln zur Filterung der Matchergebnisse

Dieses Kapitel stellt ausführlich die Regeln für die Filterung der Matchergebnisse vor, die durch das annotierte und partiell-annotierte Pairing gefunden wurden. Wenn die Pairingregeln ohne diese Filterung angewendet werden, werden neben den richtigen Ergebnissen zusätzlich sehr viele falsche Matchtripel gefunden. Durch die Beachtung des Kontexts der gematchten Elemente werden die Ergebnisse gefiltert und falsche Matches aus der Ergebnismenge entfernt.

Im Folgenden werden vier Regeln vorgestellt, die in dem BIA-Framework für die Filterung verantwortlich sind. Wie bei dem Pairing konzentrieren sich die Regeln auf reine Modellintegration und die Integration ihrer Ausprägungen wird erst in den ETL-Schritten später beachtet. Die Regeln 11 und 14 stammen aus dem Bereich des Schema-Matchings. Ihre Vorgehensweise ist so übernommen worden. Regeln 12 und 13 sind Neuentwicklungen.

Die Filterregeln haben alle den gleichen Aufbau, wie in Abb. 4.16 dargestellt. Sie haben als Eingabemenge voll-, partiell- oder nicht-annotierte Prozesselemente und Schemaelemente. Außerdem werden die gefunden Matchkandidaten COM_P von den Pairingregeln übergeben. Die Filterregeln berechnen unter Beachtung des Kontexts im Prozessmodel P_i zu jedem Variablenelement $pv_j.node_m$ und des Kontexts des gematchten operativen Elementes $S_s.node_r$ der Matchkandidaten mc_k aus der Matchliste COM_P den strukturellen Ähnlichkeitswert sim_{strk}. Für jede Regel kommen bei der Eingabemenge ggf. noch weitere Eingaben hinzu. Zudem müssen für jede Regel Vorbedingungen (V11) - (V14) erfüllt sein, damit die Regel ausgeführt wird. Da die Vorbedingungen und die weiteren Eingaben für jede Regel anders aussehen, werden sie in den Unterkapiteln bei den Regeln diskutiert. Falls die Regel ausgeführt wird, berechnet sie einen neuen Ähnlichkeitswert sim_{str} für jeden Matchkandidaten aus COM_P. Diese Regeln und ihre Ähnlichkeitsberechnungen (E11) - (E14) werden im Folgenden vorgestellt. Alle strukturellen Ähnlichkeitswerte der einzelnen Regeln werden zu einem finalen strukturellen Ähnlichkeitswert kombiniert, was

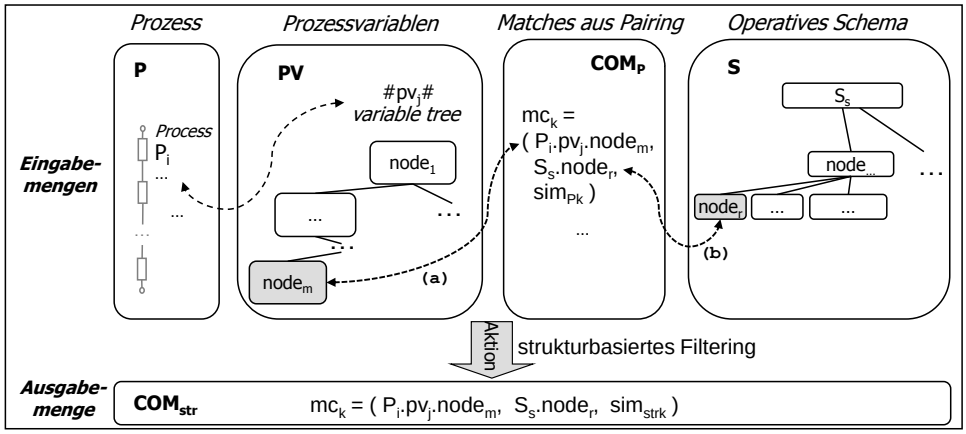


Abb. 4.16: Struktureller Filterschritt

später in Kapitel 4.7.4 erläutert wird. Außerdem wird dort aus sim_{str} zusammen mit dem Ähnlichkeitswert sim_P aus dem Pairing der finale Ähnlichkeitswert sim_i für jeden Match berechnet. Durch die Filterregeln werden so evtl. falsche Kombinationen aus dem Pairing herausgefiltert, indem der finale Ähnlichkeitswert durch sim_{str} abgeschwächt wird bzw. bei richtigen Kombinationen mit passendem Kontext wird der finale Ähnlichkeitswert erhöht.

4.5.1 Matchregel für gleiche Pfade (R11)

Die Regel R11 ist an die Schema-Matching-Verfahren angelehnt. Sie berechnet die Ähnlichkeit der Elternelemente von den zwei Elementen im gegebenen Matchkandidaten. Dabei vergleicht sie die Elternelemente in den zwei Pfaden von den Elementen $P_i.pv_j.node_m$ und $S_s.node_r$ aus dem Match bis hin zu deren jeweiligen Wurzel, d.h. von dem Knoten $node_m$ hoch bis zur Variablendeklaration pv_j bzw. von dem Knoten $node_r$ hoch bis zum Schemanamen S_s oder bis zur entsprechenden XML-Element-Wurzel. Damit es zu einem sinnvollen Ergebnis kommt, sollten die Elternelemente sprechende Namen besitzen. Die Regel hat keine Vorbedingung, da nur Elemente, die in Prozessvariablen verwendet wurden, gematcht wurden und daher auch immer ein Pfad von dem Element zur Wurzel existiert.

(V11) $\emptyset$

$$\text{(E11a) } sim_{R11} = \frac{\sum_{a=1}^{pl(S_s.node_r)} max_{b=1}^{pl(P_i.pv_j.node_m)} [sim_{rules}(parent_a(S_s.node_r), parent_b(P_i.pv_j.node_m))]}{pl(S_s.node_r)}$$

mit $pathlength(P_i.pv_j.node_m) > pathlength(S_s.node_r)$,
$pathlength = \text{pl}$,
rules $\in$ {R1, R2, R3, R4, R5, R6, R7}

$$[\text{(E11b) } sim_{R11} = \frac{\sum_{a=1}^{pl(P_i.pv_j.node_m)} max_{b=1}^{pl(S_s.node_r)}[sim_{rules}(parent_a(P_i.pv_j.node_m), parent_b(S_s.node_r))]}{pl(P_i.pv_j.node_m)}$$

mit $pathlength(P_i.pv_j.node_m) <= pathlength(S_s.node_r)$,
$pathlength$ = pl,
rules $\in$ {R1, R2, R3, R4, R5, R6, R7}]

Für jedes Elternelement im Pfad zur Wurzel von $P_i.pv_j.node_m$ wird die Ähnlichkeit mit maximalem Wert zu einem Element aus dem Pfad von $S_s.node_r$ berechnet und aufaddiert. Für die Ähnlichkeitsberechnung der einzelnen Elternelemente können die Regeln R1 - R7 angewendet werden. Für die Ausführung der einzelnen Regeln gelten die jeweils definierten Vorbedingungen und nötigen Eingabemengen. Letztlich ergibt sich der strukturelle Ähnlichkeitswert sim_{R11} in (E11a), falls die Pfadlänge von $S_s.node_r$ kürzer ist als die Pfadlänge von dem Variablenelement $P_i.pv_j.node_m$ zu seiner Wurzel. Ansonsten ergibt sich der Ähnlichkeitswert durch (E11b). sim_{R11} für (E11a) erhält man, indem man den Quotienten aus der Summe der kombinierten Ähnlichkeitswerte der Elemente auf den zwei Pfaden durch die Länge des minimalen Pfades berechnet.

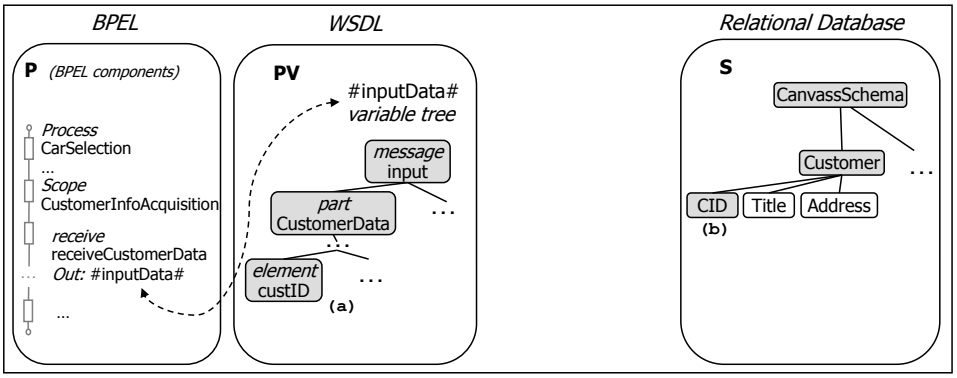


Abb. 4.17: Match im Beispielszenario mit Regel R11

Abb. 4.17 zeigt, wie die Regel R11 in unserem Beispielszenario aus Kapitel 1.3 einen Ähnlichkeitswert für den Matchkandidaten aus Kapitel 4.3.2 zwischen dem Prozesselement *CarSelection.inputData.input.CustomerData...custID* (a) und dem Schemaelement *CanvassSchema.Customer.CID* (b) berechnet. R11 bestimmt die zwei Stufen von *CID* zur Wurzel *CanvassSchema* als minimalen Pfad, d.h. pathlength = 2. Der Pfad im Prozesselement von *custID* bis hin zu *inputData* wäre mit mindestens drei Schritten nämlich länger. R11 summiert die höchsten Ähnlichkeitswerte zwischen dem Label jeder der 2 Knoten auf diesem Pfad (*Customer*, *CanvassSchema*) mit matchenden Labeln aus dem Pfad von dem Elternelement von *custID* bis zur Wurzel *inputData*. Die Labels werden hier mit R6 gematcht, da sie nicht annotiert sind. Nur *Customer* matcht mit dem Label des Partelements *CustomerData* mit einem Ähnlichkeitswert $\frac{2}{3}$, *CanvassSchema* matcht andere Elemente nur mit dem Wert 0. So erhält R11 einen Ähnlichkeitswert $\frac{\frac{2}{3}+0}{2} = \frac{1}{3}$.

4.5.2 Matchregel für Prozessstrukturnamen (R12)

Für die Regel R12 gibt es keine Vorbedingung, da jedes Prozesselement in mindestens einem Kontrollflusselement, nämlich dem BPEL-Element zur Prozessdefintion <process>, untergeordnet ist. Die Regel betrachtet die Namen der Strukturelemente des Prozesses genauer, d.h. von dem Kontrollfluss P_i, in den die Prozessvariable $P_i.pv_j.node_m$ aus gegebenem Matchkandidaten mc_k eingebettet ist. Die Regel sucht Hinweise aus den Namen der Operationen, Partner Links oder Aktivitäten, die das entsprechende Variablenelement $pv_j.node_m$ im Prozess P_i direkt verwenden oder wiederum diese Kontrollflusselemente verwenden (siehe Kapitel 2.1.1), und vergleicht sie mit Elementen aus dem Pfad des operativen Elements $S_s.node_r$ aus dem Matchkandidaten mc_k. Dabei vergleicht sie gegebenenfalls immer das gleiche Elternelement, falls die Höhe des operativen Elements bis zur Wurzel nur wenige Ebenen beträgt, mit den verschiedenen Namen der Strukturelemente.

(V12) $\emptyset$

(E12) $sim_{R12} = \frac{\sum_{a=1}^{|CFP|} max_{b=1}^{pathlength(S_s.node_r)}[sim_{rules}(CFP_a(P_i.pv_j.node_m), parent_b(S_s.node_r))]}{|CFP|}$
mit rules $\in$ {R6, R7} und $CFP = controlFlowParents(P_i.pv_j.node_m)$

Durch Anwendung der Regeln R6 und R7 wird für jedes Prozessstrukturelement, das $P_i.pv_j.node_m$ im Prozess benutzt, die *controlFlowParents* bzw. *CFP* berechnet und zu jedem Element aus dem Pfad aus CFP mit R6 oder R7 die Ähnlichkeit zu einem Element aus dem Pfad von $S_s.node_r$ mit dem maximalen Wert berechnet und aufaddiert (siehe (E12)). sim_{R12} erhält man, indem man den Quotienten aus der Summe der kombinierten Ähnlichkeitswerte durch die Anzahl der Elemente im Kontrollfluss von CFP berechnet, d.h. bis zu dem Element, das im Kontrollfluss an oberster Stelle steht (meist *process*). Algorithmus 2 (*getMaximumSimR12*) bestimmt sim_{R12}, indem er zunächst mit der Funktion *getDirectControlFlowParents* alle Prozessstrukturelemente bestimmt, die $P_i.pv_j.node_m$ direkt im Prozess P_i verwenden und speichert dies in $DCFP$. Für jedes Prozessstrukturelement aus $DCFP$ berechnet die Funktion *calculateSimR12* einen Ähnlichkeitswert unter Beachtung aller CFP, wie in (E12) beschrieben, ausgehend von dem jeweiligen $DCFP$. Falls $P_i.pv_j.node_m$ an mehreren Stellen im Kontrollfluss des Prozesses auftaucht, wird für jedes Auftauchen ein separater Ähnlichkeitswert sim'_{R12} berechnet und der höchste Ähnlichkeitswert davon für sim_{R12} in den Zeilen 4 bis 6 ausgewählt.

Abb. 4.18 zeigt, wie die Regel R12 in unserem Beispielszenario aus Kapitel 1.3 einen strukturellen Ähnlichkeitswert für den Matchkandidaten aus Kapitel 4.3.2 zwischen dem Prozesselement *CarSelection.inputData.input.CustomerData...custID* (a) und dem Schemaelement *CanvassSchema.Customer.CID* (b) berechnet. R12 zählt 3 CFP (Aktivität *receiveCustomerData*, Scope *CustomerInfoAcquistion*, Prozess *CarSelection*) des einen Kontrollflusselementes *receiveCustomerData*, das die Variable *inputData* direkt verwendet. R12 summiert die höchsten Ähnlichkeitswerte zwischen jedem der drei Namen mit

Algorithmus 2 $getMaximumSimR12$

Input: $P_i.pv_j.node_m \in PV$, P_i, $S_s.node_r \in S$
Output: sim_{R12}

```
1: DCFP ← getDirectControlFlowParents(P_i.pv_j.node_m, P_i)
2: for i = 1 to |DCFP| do
3:    sim'_R12 ← calculateSimR12(DCFP(i), P_i.pv_j.node_m, S_s.node_r)
4:    if (sim'_R12 > sim_R12) then
5:       sim_R12 ← sim'_R12
6:    end if
7: end for
```

matchenden Labeln aus dem Pfad des operativen Elements *CID*. Hier wird für das Matching nur R6 angewendet, da die anderen Elemente nicht annotiert sind. Das Label *receiveCustomerData* matcht die Tabelle *Customer* mit einem Ähnlichkeitswert $\frac{1}{2}$, das Label *CustomerInfoAquisition* mit *Customer* mit dem Ähnlichkeitswert $\frac{1}{2}$ und *CarSelection* matcht mit allen Elementen auf dem operativen Pfad mit der Ähnlichkeit 0. So erhält R12 einen Ähnlichkeitswert $\frac{\frac{1}{2}+\frac{1}{2}+0}{3} = \frac{1}{3}$.

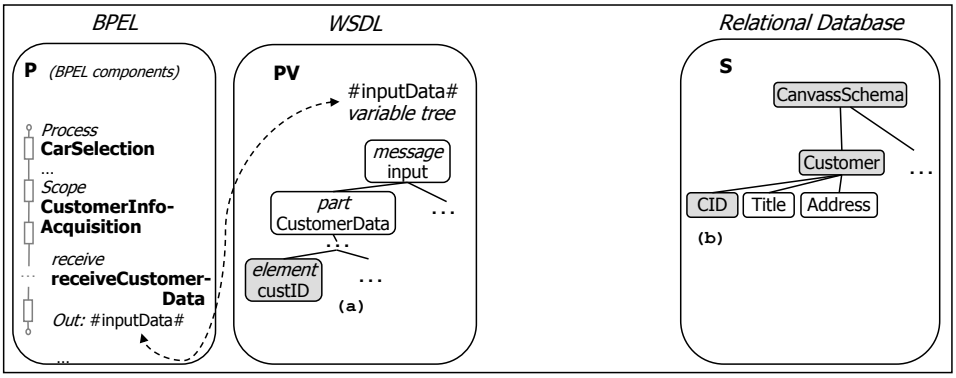


Abb. 4.18: Match im Beispielszenario mit Regel R12

Es scheint auf den ersten Blick so, dass für einige Elemente des Kontrollflusses Regel R12 keine Übereinstimmungen finden wird, da hier funktionale Daten mit den operativen Schemaelementen verglichen werden. Allerdings bilden auch die Schemaelemente oft Ergebnisse aus Vorgängen und Transaktionen ab und spiegeln dies in der Namensgebung ihrer Elemente wider. Wenn dann auch noch der gleiche Name in den Prozessstrukturelementen über mehrere Ebenen auftritt, dann liefert auch diese Regel einen wertvollen Beitrag zu der strukturellen Ähnlichkeitsberechnung der Matchpartner.

4.5.3 Matchregel für WSDL-Annotationen (R13)

Auch die WSDL-Annotationsregel (R13) betrachtet die Struktur des Prozesses, in den die gematchten Variablenelemente eingebettet sind. Sie kann nur bei annotierten Elementen angewendet werden. R13 prüft, wie groß die Distanz in der Ontologie zwischen

dem Annotationskonzept einer WSDL-Komponente einer Operation, die die gematchte Variable verwendet, bzw. deren <*portType*>-Definitionen, bis hin zu dem nächsten Annotationskonzept des Elternelements des gematchten operativen Schemaelements ist. Regel R13 wird unter folgenden zusätzlichen Vorbedingungen (V13a) und (V13b) in Abb. 4.16 angewendet.

(V13a) $controlFlowParent(P_i.pv_j.node_m)(c_p) \leftarrow sawsdl(Ont)$
(V13b) $parent(S_s.node_r)(c_s) \leftarrow sawsdl(Ont)$
[(V13c) $S_s.node_r(c_s) \leftarrow sawsdl(Ont)$]

(E13) $sim_{R13} = \frac{1}{minDistance(c_p,c_s)+1}$
mit Distanz = 1 pro Hierarchieebene über Subkonzept, Axiom, Eigenschaft
und Distanz = 0 pro Hierarchieebene über Synonym, Union

Das bedeutet, es wird geprüft, ob eine Prozesskomponente, die das Prozesselement $P_i.pv_j.node_m$ verwendet, mit dem Konzept c_p aus *Ont* annotiert ist (V13a). Außerdem muss mindestens ein Element im Pfad des operativen Schemaelements $S_s.node_r$ mit dem Konzept c_s annotiert sein (V13b), alternativ wenigstens das Element $S_s.node_r$ selbst (V13c). Falls c_s oder c_p nicht existieren, also die Vorbedingungen nicht erfüllt sind, wird R13 nicht ausgeführt und es wird $sim_{R13} = 0$ ausgegeben.

Um den strukturellen Ähnlichkeitswert zu erhalten, berechnet R13 die semantische Distanz der zwei Annotationskonzepte c_p und c_s. Dazu werden die Konzepte der Ontologie *Ont* als Graph für ihre Distanz dargestellt, wobei die Konzepte die Knoten darstellen und die Kanten verschiedene Beziehungen zwischen den Konzepten repräsentieren. Zum einen kann die Distanz der Konzepte betrachtet werden, die über Subkonzepte, Axiome, Synonyme oder Unions in *Ont* modelliert wurden. Anders als bei der Ähnlichkeitsberechnung der Regeln in Kapitel 4.3 wird bei der Distanzberechnung außer Acht gelassen, wenn ein Konzept als Subkonzept aus mehreren Subklauseln abgeleitet wird oder ein Axiom aus mehreren Atomen mit verschiedenen Domänenkonzepten besteht. Hier geht es um die reine Anzahl der Schritte, die von dem Konzept c_p bis zu dem Konzept c_s des Elternelementes von $S_s.node_r$ notwendig sind (z.B. ist *DataSelection* Subkonzept der Konzepte *DataAnalysis* und *InformationProcess* mit der Distanz 1 in Abb. 4.19). Zum anderen wird auch die Distanz der Konzepte zu anderen Konzepten betrachtet, die die Domäne ihrer Eigenschaften beschreiben. Die Distanz wird über die Anzahl der Kanten zwischen den Konzepten ermittelt. Solch eine Beziehung ist auch in Abb. 4.19 dargestellt. So beträgt die Distanz zwischen *DataSelection* und *Identifier* 2 Schritte, da *Identifier* eine Eigenschaft von *Customer* und *Customer* eine Eigenschaft von *DataSelection* ist. Synonyme und Union-Konzepte werden mit der Distanz 0 versehen (siehe *identifier* und *ID*).

Der strukturelle Ähnlichkeitswert von R13 für mc_k ist in (E13) dargestellt. R13 berechnet sim_{R13} als Anteil der Distanz zwischen c_p und c_s von dem optimalen Wert. Wenn also die Distanz 0 wäre zwischen dem Konzept c_p, das ein Kontrollflusselement von $P_i.pv_j.node_m$ semantisch beschreibt und einem Konzept c_s des operativen Elternelements von $S_s.node_r$ oder von $S_s.node_r$ selbst, dann ergibt sich daraus der Ähnlichkeitswert 1. Wenn mehrere Wege von c_p zu c_s in *Ont* und somit im Distanzgraphen abgebildet sind, wird der Weg

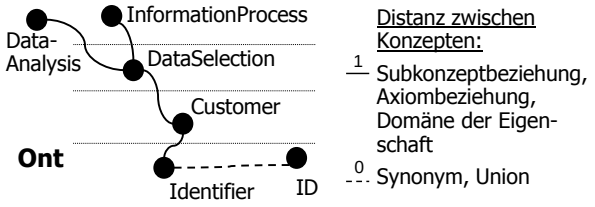


Abb. 4.19: Ausschnitt aus dem Distanzgraphen des Beispielszenarios

mit der kleinsten Distanz ausgewählt. Bei nicht existierenden Verbindungen in *Ont* zwischen den Annotationskonzepten, kann die Distanz nicht berechnet werden und sim_{R13} erhält den Wert 0. In dem Distanzgraphen können zudem Zyklen auftreten aufgrund der Präsentation der Distanzen aus verschiedenen Axiomen zusammen in einem Graphen. Sobald bei der Distanzberechnung ein Knoten erneut betrachtet wird, wird die Berechnung für diesen Zweig abgebrochen. Falls $P_i.pv_j.node_m$ an mehreren Stellen im Kontrollfluss des Prozesses in Operationen verwendet wird, die durch verschiedene Konzepte c'_p semantisch annotiert sind, wird für jedes c'_p ein sim'_{R13} berechnet und der höchste Ähnlichkeitswert für sim_{R13} ausgewählt.

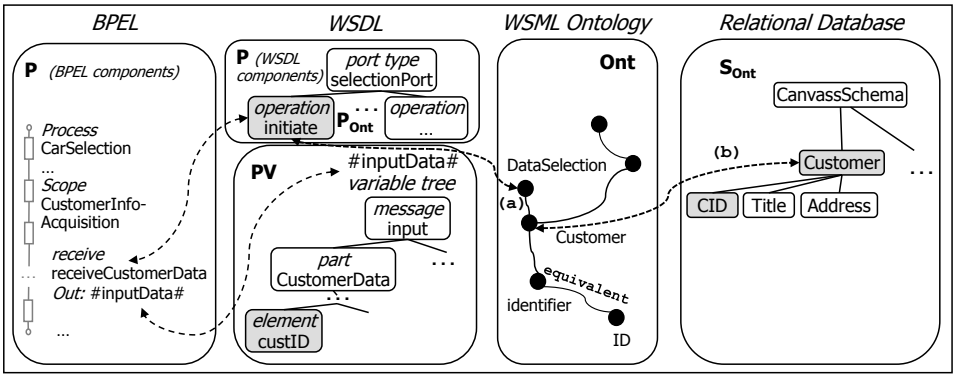


Abb. 4.20: Match im Beispielszenario mit Regel R13

Abb. 4.20 zeigt, wie die Regel R13 in unserem Beispielszenario aus Kapitel 1.3 einen strukturellen Ähnlichkeitswert für den Matchkandidaten aus Kapitel 4.3.2 zwischen dem Variablenelement *CarSelection.inputData.input.CustomerData...custID* und dem Schemaelement *CanvassSchema.Customer.CID* berechnet. Das Variablenelement wird in der receive-Aktivität *receiveCustomerData* verwendet, die in der WSDL-Datei über die Operation *initiate* definiert und mit dem Konzept *DataSelection* annotiert ist (siehe (a)). Auch das operative Elternelement *Customer* des Schemaelement *CID* ist annotiert, und zwar mit dem Konzept *Customer* (siehe (b)). Daher kann R13 die Distanz zwischen *DataSelection* und *Customer* und daraus den Ähnlichkeitswert $\frac{1}{1+1} = \frac{1}{2}$ berechnen.

4.5.4 Matchregel für gleiche Datentypen (R14)

Die Regel R14 betrachtet die Ähnlichkeit der gematchten Elemente bezüglich ihres Datentyps. Für R14 müssen keine weiteren Vorbedingungen erfüllt sein, da jedes Element durch einen Datentyp definiert ist. Für die Ausführung verwendet R14 eine Tabelle, die ähnliche Datentypen gruppiert. Hier werden die Datentypen der beiden Elemente $S_s.node_r$ und $P_i.pv_j.node_m$ des Matchkandidaten mc_k nachgeschlagen und ihre Ähnlichkeit überprüft. Falls sich ihre Datentypen in der gleichen Gruppe befinden, erhält der Ähnlichkeitswert den Wert 1, ansonsten wird er zu 0.

(V14) $\emptyset$

(E14a) $sim_{R14} = 1$ mit dataTypeGroup($P_i.pv_j.node_m$) = dataTypeGroup($S_s.node_r$)
[(E14b) $sim_{R14} = 0$ mit dataTypeGroup($P_i.pv_j.node_m$) <> dataTypeGroup($S_s.node_r$)

Die Gruppen für die konventionellen Datentypen ergeben sich wie folgt, die entsprechend auf die datenbank-spezifischen Typen abgebildet werden müssen:

- **Zeichen**: String, Character
- **Wahrheitswerte**: Boolean
- **Datum**: Date
- **Numerische Werte**: Integer, Float, Decimal, Numeric, Double, Int
- **Komplexe Strukturen**: Clob, Blob, XML

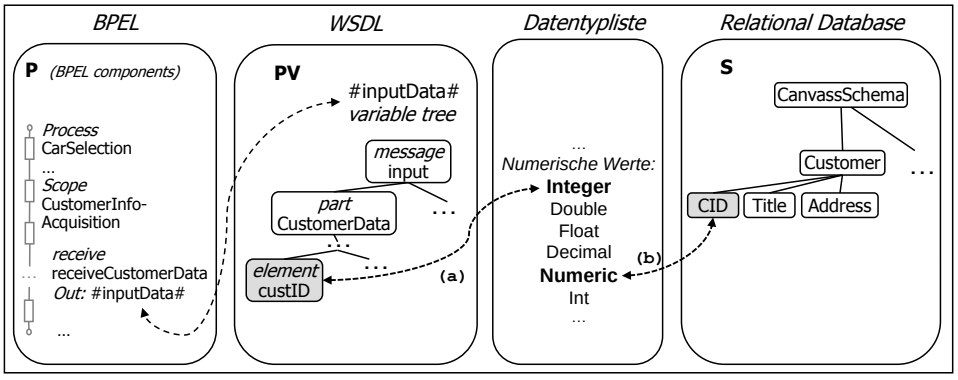


Abb. 4.21: Match im Beispielszenario mit Regel R14

Abb. 4.21 zeigt, wie die Regel R14 in unserem Beispielszenario aus Kapitel 1.3 einen strukturellen Ähnlichkeitswert für den Matchkandidaten aus Kapitel 4.3.2 zwischen dem Variablenelement *CarSelection.inputData.input.CustomerData...custID* und dem Schemaelement *CanvassSchema.Customer.CID* berechnet. Das Variablenelement ist als Integer definiert (siehe (a)). Das operative Schemaelement *CID* ist als Numeric definiert

(siehe (b)). Daher sind sie in der gleichen Datentypgruppe und R14 kann daraus den Ähnlichkeitswert 1 berechnen.

4.6 Regeln für manuelles Matchen

Das BIA-Framework unterstützt das manuelle Matching zu verschiedenen Zeitpunkten, also verschiedene Personen: den Prozessdesigner und den Analysten (siehe Kapitel 4.1.1). Das manuelle Matching durch den Analysten ist Teil der Matchpipeline und des Matcheditors. Es geschieht, wie im Framework in Abb. 4.2 angezeigt, optional nach dem strukturbasierten Filterschritt. Hier können die bereits gefundenen Matches korrigiert und fehlende Matches ergänzt werden. Das Matchergebnis wird direkt in der Matchtabelle abgelegt. Die Ähnlichkeitswerte der Matchergebnisse werden dabei unabhängig von evtl. bereits gefundenen Matchkandidaten auf 0 oder 1 gesetzt, je nachdem ob der Analyst den Match entfernt oder einen neuen hinzufügt.

Das manuelle Matching durch den Prozessdesigner ist hingegen streng genommen eine Erweiterung der Annotationsphase und wird daher als Teil des Annotationseditors realisiert. Durch die Spracherweiterung dBDD kann das manuelle Matching durch den Prozessdesigner realisiert werden. dPDD ergänzt BPEL und bietet zwei Attribute an, die zur Annotation direkt dem betreffenden Prozesselement im Annotationseditor hinzugefügt werden: *source* und *sourceType*. Das Attribut *source* ermöglicht es, ein XML-Element im Prozess direkt mit einer Entität in einer operativen Datenquelle zu verknüpfen. Es beinhaltet alle Infos, die nötig sind, um zu der operativen Datenquelle eine Verbindung herzustellen. Das Attribut hat die folgende Syntax:

source = <ID der Datenquelle> # <ID der Entität>

Für Fälle, in denen die Datenquelle eine relationale Datenbank ist, nimmt das Attribut die folgende Form an:

<JDBC Verbindungs-URL> # <Schemaname> . <Tabellenname> [.<Spaltenname>]

Für die Datenbank *Oracle thin* sieht die Belegung des Attributes für die JDBC Verbindungs-URL so aus: ***jdbc:oracle:thin:login1/password@*** *[<host>] [:<port>]* ***:***<*SID*> Die SID ist eine Abkürzung für ServiceID und ist eine Identifikation für eine spezifizierte Oracle-Datenbankinstanz. Für die Oracle-Instanz XE auf dem Localhost und dem Port 1521 sieht das Attribut für die JDBC-Verbindungs-URL dann so aus: ***jdbc:oracle:thin:login1/password@localhost:1521:XE***.

Analog zu *modelType* aus Kapitel 3.4 wird *sourceType* verwendet, um zwischen verschiedenen Arten von Datenquellen zu unterscheiden, z.B. relationale Spalten und Tabellen, XML-Daten oder Spalten in einer CSV-Textdatei (Character Separated Values). Dieses Attribut soll wie das Attribut *modelType* in erster Linie die Verarbeitung der Matches erleichtern. *sourcetype* nutzt derzeit die Werte *relCol* bzw. *relTable* für die Anzeige, dass

das Prozesselement mit einer operativen relationalen Spalte bzw. Tabelle gematcht wurde. Außerdem ist die Belegung *XMLData* möglich, wenn operative XML-Schemas referenziert wurden. Anhang A illustriert in Abb. A.4 einen manuellen Match, der im Beispielszenario zwischen dem Prozesselement *rentalStartDate* und operativen Elemente *CalendarDate* eingefügt wurde. Anhang C zeigt den Match als M_5.

Die Spracherweiterung dBDD wird jedoch nicht von den WFMS verstanden und wird daher beim Deployment von ihnen nicht berücksichtigt. Stattdessen speichert der Annotationseditor den Match direkt in der Matchtabelle.

4.7 Kontrollstrategie im Matchverfahren

Die Kontrollstrategie des BIA-Frameworks legt die Reihenfolge fest, in der die in den vorangegangenen Kapiteln vorgestellten Matchregeln auf die Prozessvariablenelemente und die operativen Elemente angewendet werden. Es werden Abhängigkeiten zwischen den Regeln betrachtet, um die Abfolge ihrer Ausführung festzulegen, damit unnötige oder mehrfache Regelausführungen verhindert werden. Außerdem garantiert die Kontrollstrategie, dass alle Möglichkeiten für Matches zwischen den Elementen untersucht werden, der höchste Ähnlichkeitswert pro Match berechnet wird und dass der Matchvorgang in endlicher Zeit zu einem Ergebnis kommt. Das heißt, dass sich die Kontrollstrategie des Frameworks bedeutend von der eines typischen Schema-Matchers wie z.B. des COMA-Matchers [DR02] unterscheidet. Bei diesem wird immer nur eine kleine, vom Benutzer getätigte Auswahl von Regeln auf die Eingabemenge angewendet.

In den nächsten Kapiteln wird die Kontrollstrategie des Matchverfahrens vorgestellt. Die Abhängigkeiten zwischen den Matchregeln, durch die die Reihenfolge der Regelausführung festgelegt wird, und die Erstellung von Konzeptgruppen, die diese Abhängigkeiten modellieren, werden zuerst genauer dargelegt. Danach wird in Kapitel 4.7.4 auf die Berechnung der Ähnlichkeitswerte, insbesondere bei der Kombination mehrerer Regelanwendungen eingegangen. In Kapitel 4.7.5 wird der Ablauf der Regelanwendungen erläutert und im anschließenden Kapitel beschrieben, wie die Kontrollstrategie terminiert.

4.7.1 Notwendigkeit von Konzeptgruppen

Die semi-automatischen Pairingregeln basieren auf den semantischen Beziehungen, die in einer Ontologie definiert wurden. Es können nun zwar die Regeln einzeln auf der Ontologie ausgeführt werden. Wenn allerdings Matches nur durch kombiniertes Anwenden mehrerer Regeln gefunden werden, z.B. ein Annotationskonzept mit dem Synonym eines seiner Subkonzepte verknüpft werden soll, dann müssten dafür sehr viele eigene Beziehungen oder Axiome in der Ontologie definiert werden. Mit Hilfe der Bildung von Konzeptgruppen, in die die Konzepte und ihre annotierten Elemente gemäß ihrer Matchregeln verschachtelt

werden (siehe Kapitel 4.7.3), kann das BIA-Framework die kombinierte Anwendung mehrere Regeln vereinfachen. Außerdem gewährleisten die Konzeptgruppen, dass die Matchregeln unabhängig voneinander anwendbar sind und es für das Matchergebnis keine Rolle spielt, welche der Regeln zuerst angewendet wurde.

4.7.2 Abhängigkeiten zwischen Matchregeln

Das Ergebnis in COM ist abhängig von der Ausführungsreihenfolge der Matchregeln und manche Regeln sind sogar nicht ohne die Ergebnisse der anderen Regeln anwendbar. Das sind Regeln, bei denen die Anwendung einer Regel die Anwendung einer anderen Regel erst ermöglicht. Dazu gehören die Regeln R9, R10a, R11, R12, R13 und R14. Sie sind abhängig von den Ergebnissen der anderen Regelausführungen. Die Regeln R7 und R10c basieren hingegen nicht auf dem Matchergebnis, sollen jedoch alle semantischen Beziehungen der Konzepte beachten. Daher sind sie abhängig von den annotierten Pairingregeln und können erst nach Erstellung aller verschachtelten Konzeptgruppen angewendet werden. Auch die Regeln R1, R2, R3, R4, R5 beachten die semantischen Beziehungen aus jeweils vorher ausgeführten Regeln. Sie sind aber mit Hilfe der Konzeptgruppen unabhängig voneinander anwendbar.

Die Matchregeln zur Erstellung der verschiedenen Kombinationen sind in drei Phasen aufgeteilt. Die Reihenfolge der Phasen selbst ist fest, aber die Regeln innerhalb der Phasen können in beliebiger Reihenfolge ausgeführt werden, da sie trotzdem immer die gleiche Matchmenge ergeben. Außerdem gibt es eine Filterphase, deren vier Filterregeln ebenso in beliebiger Reihenfolge ausgeführt werden können. Die Phasen lauten:

- 1. Pairingphase: R1, R2, R3, R4, R5
- 2. Pairingphase: R6, R7, R10b, R10c
- Filterphase: R11, R12, R13, R14
- 3. Pairingphase: R9, R10a

Die Regeln R1-R5 bauen die Konzeptgruppen auf und verschachteln diese ineinander. Da für jede verschachtelte Gruppe die Kombinierbarkeit aller möglichen Konzepte und ihrer annotierten Elemente zu den Elementen der inneren und umliegenden Gruppen geprüft wird, ist die Reihenfolge der Regeln unabhängig voneinander.

Die Regeln R7 und R10c beachten auch diese semantischen Beziehungen. Die Regeln selbst überprüfen zwar nur die syntaktischen Beziehungen zwischen dem Namen des Annotationskonzeptes und den Namen der Elemente (R7) bzw. der *<property>*- und *<corrlationSet>*-Komponenten (R10c). Allerdings sollen sie anschließend auch die zuvor gefundenen semantischen Beziehungen in die Bestimmung weiterer neuer Kombinationen miteinbeziehen. Sie können erst angewendet werden, wenn die semantischen Gruppen durch R1-R5 gebildet wurden. Deshalb werden sie erst in der zweiten Phase angewendet. Die Regeln R7 und R10c sind ebenfalls voneinander unabhängig, da sie unterschiedliche Komponenten (Elementname versus Propertyname) mit dem Konzeptlabel vergleichen. R6 und R10b beachten dagegen diese Gruppen und die semantischen Beziehungen gar

nicht, sondern konzentrieren sich auf die rein syntaktischen Beziehungen der Elementnamen. Daher sind sie von den anderen Regeln unabhängig. R6 und R10b sind auch voneinander unabhängig, da R6 die Namen der Elemente untereinander und R10b die Namen der Elemente mit den Namen der <*property*>- und <*correlationSet*>-Komponenten vergleicht. Natürlich könnten auch R6 und R10b mit den semantischen Beziehungen aus den vorherigen Regeln verbunden werden, um auch alle Erkenntnisse dieser Regeln auszunutzen. Diese zwei syntaktischen Regeln sollen in diesem Framework jedoch das Gegenstück zu den semantischen Regeln bilden. Sie lassen deshalb diese Beziehungen außen vor.

Auch die Ausführung der Filterregeln R11, R12, R13 und R14 basiert auf dem Vorhandensein bereits gefundener Matches in COM_P, von denen nun der Kontext beider beteiligten Matchpartner des jeweiligen Matches analysiert wird. Das bedeutet hier, dass mindestens eine der Regeln R1 bis R10 ausgeführt sein muss, sodass mindestens ein Tripel gefunden wurde, auf das alle Filterregeln unabhängig voneinander angewandt werden.

In der letzten Phase können die Regeln R9 und R10a angewendet werden. Sie basieren auf den Kombinationsergebnissen, die durch die vorherigen Regeln und nach Anwendung der Filterregeln gefunden wurden. Daher sind sie erst jetzt anwendbar. Das bedeutet, dass mindestens eine der Regeln R1 bis R7 ausgeführt worden sein muss, um ein Tripel für COM_P zu finden. Ebenso müssen die Filterregeln R11 bis R14 auf diese Tripel angewendet und in COM gespeichert worden sein, sodass der finale Ähnlichkeitswert des Tripels feststeht, bevor es von der Regel R9 oder R10a verwendet wird. R9 und R10a können auch rekursiv angewendet werden, d.h., dass Matches in COM, die durch R9 und R10 gefunden wurden, wiederum die Ausführung von R9 bzw. R10a für weitere Variablenelemente im Prozess ermöglichen. Beide Regeln sind abhängig voneinander, da sie beide die Ergebnismenge erweitern, die sie wiederum als ihre Eingabemenge verwenden. Es ergeben sich daraus vier Möglichkeiten, in welcher Reihenfolge die zwei Regeln angewendet werden sollen, um herauszufinden, mit welchen Schemaelementen das Prozesselement $P_i.pv_j.node_m$ kombiniert werden kann unter der Berücksichtigung der Assign-Aktivitäten a_i, a_t mit $0 < i,t < Activityanzahl\ in\ P_i$ und a_i bzw. a_t: $source \rightarrow target$ sowie der Property p_x, der u.a. das Element $P_i.pv_j.node_m$ angehört:

- $\forall a_i \;\; P_i.pv_j.node_m \notin source$ und $\exists a_t \; P_i.pv_j.node_m \in target \rightarrow$
  zuerst R9, dann R10a, um neue Matches aus R9 auch auf andere Prozesselemente in p_x übertragen zu können
- $\exists a_i \;\; P_i.pv_j.node_m \in source$ und $\forall a_t \; P_i.pv_j.node_m \notin target \rightarrow$
  zuerst R10a, dann R9, um neue Matches aus R10a auch auf andere Prozesselemente in a_t übertragen zu können
- $\forall a_i \;\; P_i.pv_j.node_m \notin source$ und $\forall a_t \; P_i.pv_j.node_m \notin target \rightarrow$
  keine Möglichkeit der Anwendung von R9 auf $P_i.pv_j.node_m$
- $\exists a_i \;\; P_i.pv_j.node_m \in source$ und $\exists a_t \; P_i.pv_j.node_m \in target \rightarrow$
  Regelreihenfolge nicht definierbar, R9 und R10a müssen abwechselnd wiederholt werden, bis sich COM nicht mehr ändert

4.7.3 Erstellung von Konzeptgruppen

Die Matchregeln R1 bis R5 benutzen verschiedene semantische Beziehungen aus *Ont* zwischen vollannotierten Elementen bzw. auch teilweise annotierten Elementen wie in R7 und R10c. Um bei dem Matchverfahren alle bereits gefundenen semantischen Beziehungen auch bei anderen Matches zu berücksichtigen (siehe auch Kapitel 4.7.1), arbeitet das Framework auf Konzeptgruppen. Zu Beginn wird für jedes Konzept der Ontologie, das mindestens ein Prozesselement oder operatives Element annotiert, eine eigene Konzeptgruppe erstellt (siehe Abb. 4.22 (a)). Gruppe G_0 beinhaltet dagegen alle nicht-annotierten Elemente mit einem Konzept c_0 ohne Label (Abb. 4.22 (b)), um später auch bequem mit Hilfe der Konzeptgruppen die partiell-annotierten Pairingregeln R7 und R10c auf Elemente und ihre definierten Beziehungen in *Ont* anwenden zu können. Abb. 4.24 (a) zeigt zwei Konzeptgruppen, die für die Konzepte *identifier* und *ID* aus dem Beispielszenario aus Kapitel 4.3.2 erstellt wurden. Die Konzepte annotieren jeweils ein Prozesselement und ein operatives Schemaelement.

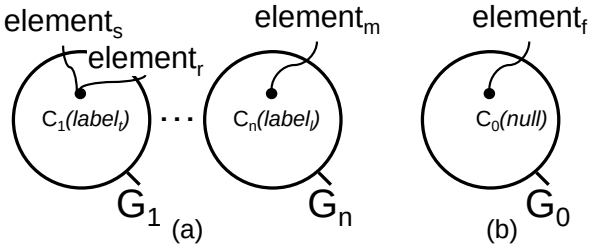


Abb. 4.22: Konzeptgruppe

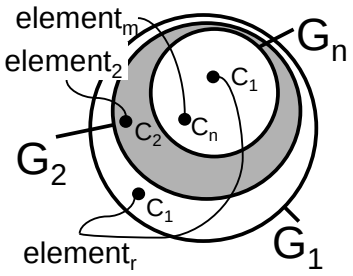


Abb. 4.23: Gruppenblock

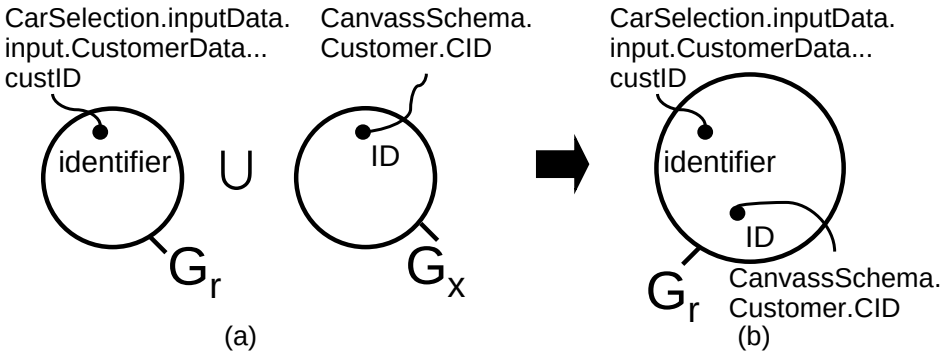


Abb. 4.24: Konzeptgruppen im Beispielszenario

Je nach Regel führt die Kontrollstrategie eine Verschachtelung, eine Vereinigung oder eine Duplizierung der Gruppen durch. Durch Anwendung der verschiedenen Regeln ist eine Mehrfachverschachtelung in einen Block von Gruppen möglich (siehe Abb. 4.23). Dabei kann es passieren, dass auch gleiche Konzepte, wie hier z.B. c_1, ineinander verschachtelt

werden. Dies wird jedoch später bei der Berechnung der Ähnlichkeitswerte berücksichtigt. Bei dem annotierten Pairing können grundsätzlich nur Elemente aus gleichen Gruppen oder verschachtelten Gruppen miteinander kombiniert werden, und nicht aus separat liegenden Gruppen. Eine Ausnahme bilden die Gruppe G_0 und die partiell-annotierten Pairingregeln, die jedes Element, auch aus anderen Gruppen miteinander kombinieren.

Nach der initialen Gruppenbildung (siehe Abb. 4.22 oder 4.24 (a)) befindet sich jedes Konzept in einer eigenen Gruppe, die je nach Regelanwendung miteinander integriert werden. Im Folgenden wird nun auf diese Integration der Gruppen eingegangen. Dies betrifft die Pairingphasen I und II, da nur deren Regeln die semantischen Beziehungen betrachten. In der Filterphase betrachtet zwar auch R13 semantische Beziehungen. Diese betrachtet jedoch Distanzen und wird über den Distanzgraphen hergeleitet, im Gegensatz zu den Konzeptgruppen, die die Ähnlichkeitswerte zwischen den Konzepten betrachten. Für die Erstellung der Konzeptgruppen sind jedoch nur die Regeln der Phase I anzuwenden, da neue Beziehungen aus R7 und R10c einfachheitshalber nicht rekursiv von R7 und R10c berücksichtigt werden sollen. Von den Regeln müssen R2, R3, R4, R5 betrachtet werden, da nur sie eine Integration der Konzeptgruppen bewirken können. Regeln R2 und R5 haben gleiche Auswirkungen. Sie vereinigen die Konzeptgruppen und werden deshalb beide durch Abb. 4.25 dargestellt. Sind zwei Konzepte c_r und c_x aus verschiedenen Gruppen G_r und G_x synonym zueinander oder die Ontologie spezifiziert eine Union-Beziehung wie z.B. c_r *unionOF* ..., c_x in OWL bzw. c_r $definedBy$ ... *or* ... *or* c_x als Axiom in WSML, dann werden die Konzepte samt allen annotierten Elementen in eine Gruppe übernommen. In Abb. 4.25 werden die Konzepte aus G_x in G_r gespeichert. Danach wird G_x gelöscht. Überträgt man dieses Vorgehen auf das Beispielszenario aus Kapitel 4.3.2, bleibt aus den zwei ursprünglichen Konzeptgruppen mit den synonymen Konzepten in Abb. 4.24 (b) am Ende eine Konzeptgruppe übrig.

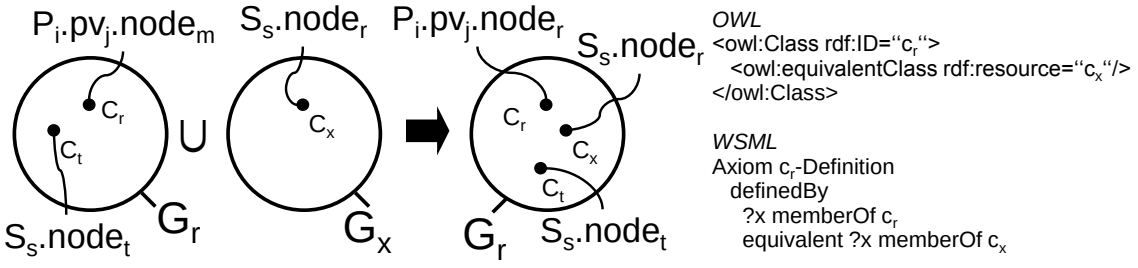


Abb. 4.25: Ausführung von R2 zum Aufbau der Konzeptgruppen (R5 gleich)

Als optimale Kontrollstrategie ergibt sich aus den Verschachtelungen der Konzeptgruppen, dass R2 und R5 vor R3 angewendet werden. Dadurch ist gewährleistet, dass noch keine verschachtelten Gruppen mit unterschiedlichen Ähnlichkeitswerten existieren und so die Konzeptgruppenblöcke bei der Vereinigung nicht zu groß werden.

Die Regel R3 arbeitet auf Subkonzeptbeziehungen, die in der Ontologie definiert sind. Über diese Definitionen erstellt R3 einen Graphen mit Hierarchieebenen wie in Abb. 4.26 links. 4.26 zeigt außerdem einen Ausschnitt der Subkonzeptdefinitionen aus der OWL-

und WSML Ontologie. Wie man hier sieht, kann ein Subkonzept c_x aus mehreren Konzepten zusammengesetzt sein (c_1, c_2 und c_r), von denen jedes einzelne Konzept wiederum Subkonzept von anderen Konzepten sein kann. Die Konzepte, die Elemente annotieren, liegen in Konzeptgruppen. Es werden gemäß der Regel R3 alle Ähnlichkeitswerte der Konzepte, die sich in den Gruppen befinden, berechnet. Mit der Formel aus Kapitel 4.3.3 $\frac{\#c_b \text{ in } ReducedHierarchy\ AB}{\#Konzepte \text{ in } ReducedHierarchy\ AB}$ beträgt der Ähnlichkeitswert hier zwischen Annotationselementen von c_x und c_f $\frac{1}{4}$ (da c_x subConceptOf c_1, c_3, c_4, c_f), zwischen c_x und c_r $\frac{1}{3}$ (da c_x subConceptOf c_1, c_2, c_r) und zwischen c_f und c_r $\frac{1}{2}$ (da c_r subConceptOf c_4, c_f). Die Gruppen werden hierarchisch ineinander geschachtelt, sodass am Ende jede Gruppe A mit ihren Konzepten in die Gruppe B verschachtelt wurde, die Konzepte enthält, die ein Konzept aus Gruppe A für seine Subkonzeptdefinitionen verwendet hat. Gruppe $G_{f''}$ wird daher in Gruppe $G_{f'}$ und danach mit Gruppe $G_{f'}$ in Gruppe G_f verschachtelt. Jede Gruppe wird dabei mit so vielen Ähnlichkeitswerten gekennzeichnet, abhängig davon, in wie viele Gruppen sie verschachtelt wurde. So kann später bei der Kombination und Ähnlichkeitsberechnung der Matches von Elementen aus verschachtelten Konzeptgruppen auf diesen Wert zugegriffen werden. Falls ein Konzept aus Konzepten verschiedener Gruppen zu einem Subkonzept zusammengesetzt ist, wird die Gruppe vervielfältigt und in alle verschiedenen Elternkonzeptgruppen verschachtelt. Abb. 4.26 zeigt die verschachtelten Gruppen $G_{f''}$, $G_{f'}$ und G_f mit den zuvor berechneten Ähnlichkeitswerten.

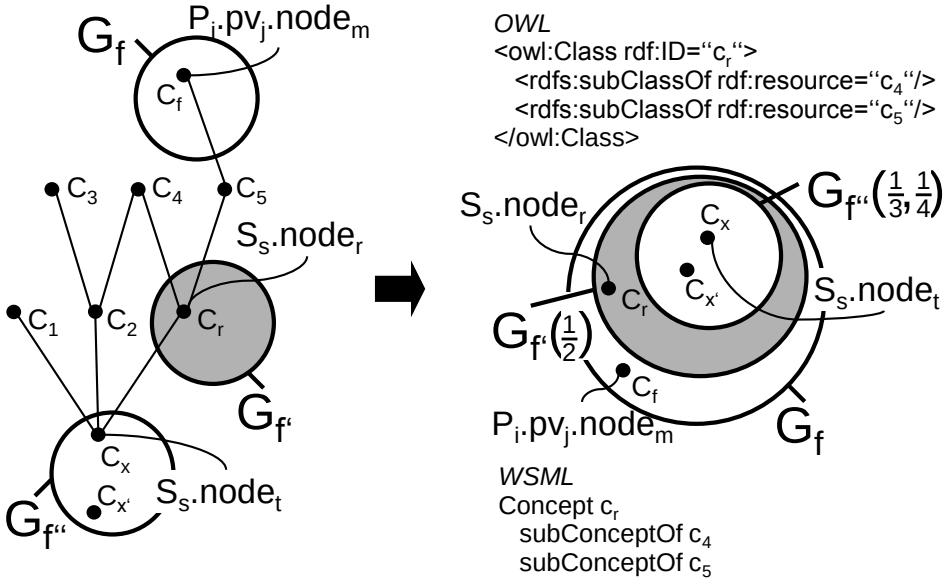
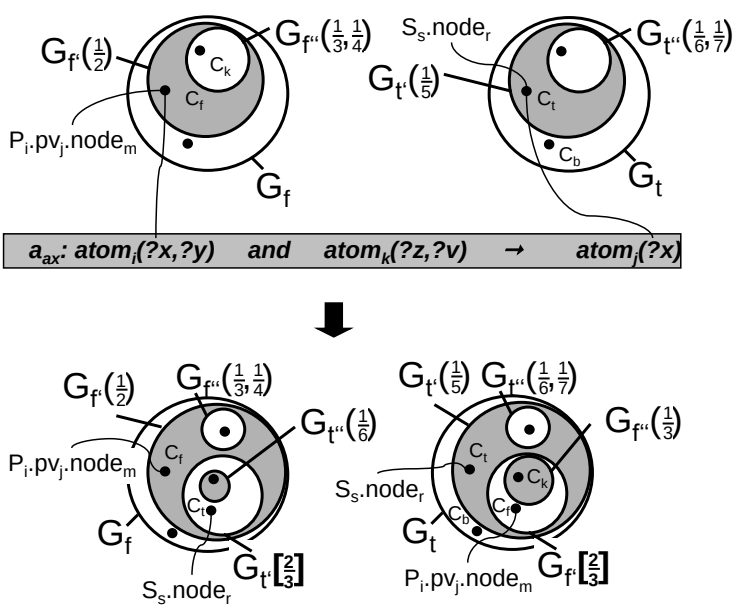


Abb. 4.26: Ausführung der Regel R3 zur Verschachtelung der Konzeptgruppen

Abb. 4.27 zeigt das Ergebnis der Verschachtelung der Regel R4 aus Axiomen, die rechts in SWRL bzw. WSML dargestellt sind. Es werden alle Konzeptgruppen integriert, deren Konzepte den gleichen Parameter der Domäne des Atoms i und j beschreiben. Im Beispiel entspricht die Domäne von $atom_j$ dem Konzept c_t und die Domäne von $atom_i$ dem Konzept c_f. Hier werden durch R4 sowohl $G_{f'}$ in $G_{t'}$ geschachtelt und $G_{t'}$ in $G_{f'}$. So werden zusätzlich sowohl Matches zwischen den Elementen in G_t

mit Elementen in $G_{f'}$ und $G_{f''}$ als auch von G_f mit $G_{t'}$ und $G_{t''}$ ermöglicht. Auch hier wird die eingeschachtelte Gruppe mit einem Ähnlichkeitswert gemäß der Formel $\frac{\#c_a \text{ und } \#c_b \text{ in den Domains der Atome des Axioms } a_t}{\#Atome \text{ in } Axiom \ a_t}$ versehen, die in Kapitel 4.3.4 beschrieben wurde. Der Ähnlichkeitswert zwischen den Annotationselementen von c_f und c_t beträgt also $\frac{2}{3}$ und kennzeichnet sowohl die Gruppe $G_{t'}$ als auch $G_{f'}$ nach der Verschachtelung. Falls eine Konzeptgruppe G selbst keine Untergruppe ist, wird sie in die Gruppe mit dem gematchten Konzept nur eingeschachtelt ohne auch diese Gruppe in sich zu verschachteln, da alle Beziehungen bereits dadurch abgedeckt sind. Falls die eingeschachtelte Konzeptgruppe n in sich verschachtelte Untergruppen aus R3 hat und vor der R4-Verschachtelung selbst auch eine Untergruppe mit k Obergruppen aus R3 war, werden nach der Verschachtelung bei den Untergruppen jeweils die k letzten Ähnlichkeitswerte gestrichen. Demnach wurde im Ergebnis in Abb. 4.27 im ersten Gruppenblock bei $G_{t''}$ der Wert $\frac{1}{7}$ und im zweiten Gruppenblock bei $G_{f''}$ der Wert $\frac{1}{4}$ gestrichen. Existieren gekennzeichnete Gruppen aus R4 bereits als Obergruppe oder Untergruppe, die zuvor verschachtelt wurde, wird k bzw. n nur ab bzw. bis zu dieser R4-Gruppe bestimmt und die Streichung der Ähnlichkeitswerte dementsprechend vorgenommen. In Abb. 4.27 sind alle Konzepte vor der Verschachtelung nur einmal und deshalb auch nur jeweils eine Konzeptgruppe vorhanden. Durch Anwenden von R4 können die Konzeptgruppen vervielfältigt werden wie im Ergebnis von Abb. 4.27 mit doppelten Gruppen $G_{t'}$, $G_{t''}$, $G_{f'}$ und $G_{f''}$. Bei Existenz mehrerer Axiome mit gleichen Domänenkonzepten muss die Verschachtelung durch R4 daher immer auf alle vorhandenen Konzeptgruppen des betreffenden Konzeptes angewendet werden. Die Kontrollstrategie des Matching ist so ausgelegt, dass R3 vor R4 ausgeführt wird. Die Reihenfolge der Regelanwendungen wäre generell aber auch andersherum möglich und der Vorgang zur Konzeptgruppenbestimmung müsste nur entsprechend angepasst werden.



OWL

```
<ruleml:imp> <ruleml:_rlab ruleml:href="a_ax"/>
<ruleml:_body>
  <swrlx:individualPropertyAtom
            swrlx:property="atom_i">
    <ruleml:var>x</ruleml:var>
    <ruleml:var>y</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
            swrlx:property="atom_k">
    <ruleml:var>z</ruleml:var>
    <ruleml:var>v</ruleml:var>
  </swrlx:individualPropertyAtom>
</ruleml:_body>
<ruleml:_head> <swrlx:classAtom>
    <owlx:Class owlx:name="atom_j">
    <ruleml:var>x</ruleml:var>
 </swrlx:classAtom> <ruleml:_head>
</ruleml:imp>
```

WSML

```
axiom a_x definedBy
?x [atom_i hasValue ?y] memberOf c_f and
                    ?z [atom_k hasValue ?v] ...
implies ?x memberOf c_t
```

Abb. 4.27: Ausführung der Regel R4 zur Verschachtelung der Konzeptgruppen

Auch die Regeln der Phase II nutzen diese kombinierte Ähnlichkeitsberechnung der verschachtelten Konzeptgruppen aus. Mehr dazu im nächsten Kapitel.

4.7.4 Berechnung des Ähnlichkeitswertes zwischen Konzeptgruppen

Für jede Kombination zwischen Prozesselementen und operativen Schemaelementen werden Ähnlichkeitswerte berechnet. Gemäß der Kontrollstrategie (siehe Kapitel 4.7.5) werden Matches erstellt und für diese auch in Abhängigkeit der ausgeführten Regeln die Ähnlichkeitswerte für die Matches bestimmt. Die meisten Regeln haben Vorbedingungen und werden nur ausgeführt, wenn diese Vorbedingungen erfüllt sind und bestimmen auch nur dann einen Match und einen Ähnlichkeitswert. Damit auch alle semantischen Beziehungen bei der Matcherstellung und Ähnlichkeitsberechnung sim_p berücksichtigt werden können, wurden Konzeptgruppen aufgebaut, die bei der Ähnlichkeitsberechnung in Phase I und II berücksichtigt werden.

Abb. 4.28 zeigt die Ausführung der einfachsten Regel R1 auf eine Gruppe G_n, die ein Konzept c_f beinhaltet, das sowohl Prozessvariablen $P_i.pv_j.node_m$ als auch Schemaelemente $S_s.node_r$ und $S_s.node_t$ annotiert. Durch R1 ergeben sich alle Kombinationsmöglichkeiten zwischen Elementen aus PV_{Ont} und S_{Ont}, die durch c_f annotiert sind. R1 resultiert in diesem Beispiel in den zwei Tripeln com_{P1} und com_{P2}. Angewendet auf das Beispielszenario aus Kapitel 4.3.1 ergibt sich der Matchkandidat $com_{Pk} = (CarSelection.inputData.input.CustomerData...contact,\ CanvassSchema.Customer.Address, 1)$, da die Elemente mit dem gleichen Konzept $c_f = address$ in der Ontologie annotiert sind und sich deshalb in der gleichen Konzeptgruppe befinden. Außerdem ergeben sich Matchkandidaten zwischen allen weiteren Prozesselementen und Schemaelementen, die mit diesem Konzept annotiert sind.

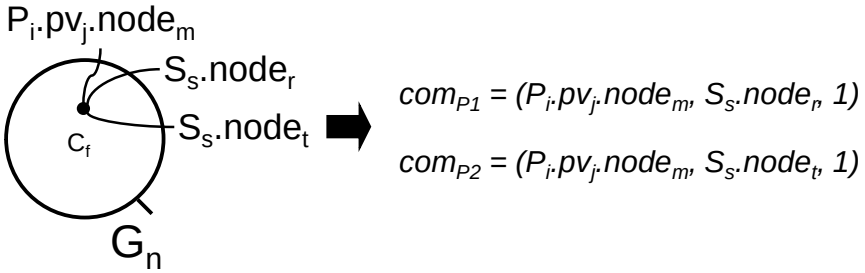


Abb. 4.28: Ausführung der Regel R1 in Konzeptgruppe

Auch die Kombinationen durch R2 und R5 sind mit Hilfe der Konzeptgruppen leicht zu bewerkstelligen. In Abb. 4.25 ergeben sich durch R2 bzw. R5 alle Kombinationsmöglichkeiten zwischen Elementen aus PV_{Ont} und S_{Ont}, die durch c_r, c_x oder c_t annotiert sind. R2 bzw. R5 resultieren in den zwei Tripeln $com_{P1} = (P_i.pv_j.node_m, S_s.node_r, 1)$ und $com_{P2} = (P_i.pv_j.node_m, S_s.node_t, 1)$.

R3 und R4 verwenden die verschachtelten Konzeptgruppen und ihre gekennzeichneten Ähnlichkeitswerte. Falls nur Elemente aus verschachtelten Gruppen mit R3 kombiniert werden, wird der Ähnlichkeitswert entsprechend der Schachtelungstiefe, d.h. der Anzahl der durchquerten Konzeptgruppen ausgewählt. In Abb. 4.26 werden somit die Tripel $com_{P1} = (P_i.pv_j.node_m, S_s.node_r, \frac{1}{2})$ und $com_{P2} = (P_i.pv_j.node_m, S_s.node_t, \frac{1}{4})$ erstellt. Natürlich können auch in diesem Gruppenblock Kombinationen durch R1, R2 oder R5 entstehen oder weitere Kombinationen durch R3 mit Elementen, die mit $c_{x'}$ annotiert wurden.

Die Berechnung des Ähnlichkeitswertes in Gruppenblöcken mit Gruppen, von denen einige mit R3 und andere mit R4 gekennzeichnet sind wie in Abb. 4.27, ist komplexer. Sobald Elemente aus solchen verschiedenen Konzeptgruppen kombiniert werden, die nicht nur durch R3 gekennzeichnet sind, wird der Ähnlichkeitswert sim_P aus den Werten der Konzeptgruppen kombiniert (siehe Definition 7). Falls dagegen nur Elemente einer oder zweier Konzeptgruppen am Match beteiligt sind, erhält sim_P den jeweiligen Wert der Pairingregel, die angewendet wurde, bzw. der Konzeptgruppe, deren Grenze bei dem Match überschritten wurde.

Definition 7 (Berechnung des Ähnlichkeitswertes sim_p)
Die Berechnung des Ähnlichkeitswertes sim_p einer Kombination von Elementen, die mit Konzepten aus verschachtelten Konzeptgruppen annotiert sind, ist definiert als: sim_p = (Produkt der Ähnlichkeitswerte der Konzeptgruppen bzw. Regeln).

Der kombinierte Ähnlichkeitswert in Definition 7 besteht aus den Werten der Konzeptgruppen, die durch die Matchregeln vorher unabhängig voneinander erstellt wurden. Daher ist der Ähnlichkeitswert mit dem Produkt der einzelnen Ähnlichkeitswerte gemäß der allgemeinen Definition der Wahrscheinlichkeit stochastisch unabhängiger Ereignisse berechnet worden ($P(A \cap B) = P(A) * P(B)$). So ergibt sich als Beispiel für die verschachtelte Kombination für R4 bei einem Match zwischen Elementen aus c_b und c_k z.B. der Wert $sim_P = \frac{1}{5} * \frac{2}{3} * \frac{1}{3}$. Kombinationen sind dabei nur von Elementen möglich, deren Konzepte in untergeordneten Gruppen liegen. Konzepte aus Geschwistergruppen wie z.B. in Abb. 4.27 von c_k mit den Konzeptelementen aus der Gruppe $G_{t''}$ sind nicht möglich. Wenn man die Matches und ihre Ähnlichkeitswerte mit Hilfe der Definition 7 in dem leicht abgewandelten Beispielszenario aus Kapitel 4.3.4 in Abb. 4.29 berechnet, erhält man mit R3 und R4 u.a. einen verschachtelten Gruppenblock wie in Abb. 4.29 rechts dargestellt. Daraus ergibt sich ein Wert von $sim_P = \frac{1}{1} * \frac{3}{3} = 1$ für den Match der dargestellten Elemente.

Abb. 4.30 zeigt die Anwendung der Regeln R7 und R10b. Hier müssen schon die Regeln aus Phase I angewendet worden sein, sodass die semantischen Beziehungen in den Konzeptgruppen modelliert sind. Nacheinander werden nun durch R7 alle Namen der Schemaelemente mit den Namen der Konzepte, die Prozessvariablen annotieren, verglichen (siehe Abb. 4.30 (a)). Es entsteht der Match com_{P1}. Abb. 4.30 (b) zeigt die Anwendung von R7 und R10b auf das Konzept c_n, das Schemaelemente annotiert und dessen Konzeptlabel deshalb mit den Namen der Prozessvariablen aus G_0 oder der <*property*>- bzw. <*correlationSet*>-Elemente verglichen werden. Hier erhält man die Matchkandidaten com_{P1} und com_{P3}. Hierzu wurden die Regeln R7 und R10b angewendet wie in Kapitel 4.4.2 und Kapitel 4.4.5 beschrieben. Mit Hilfe der Konzeptgruppen und der De-

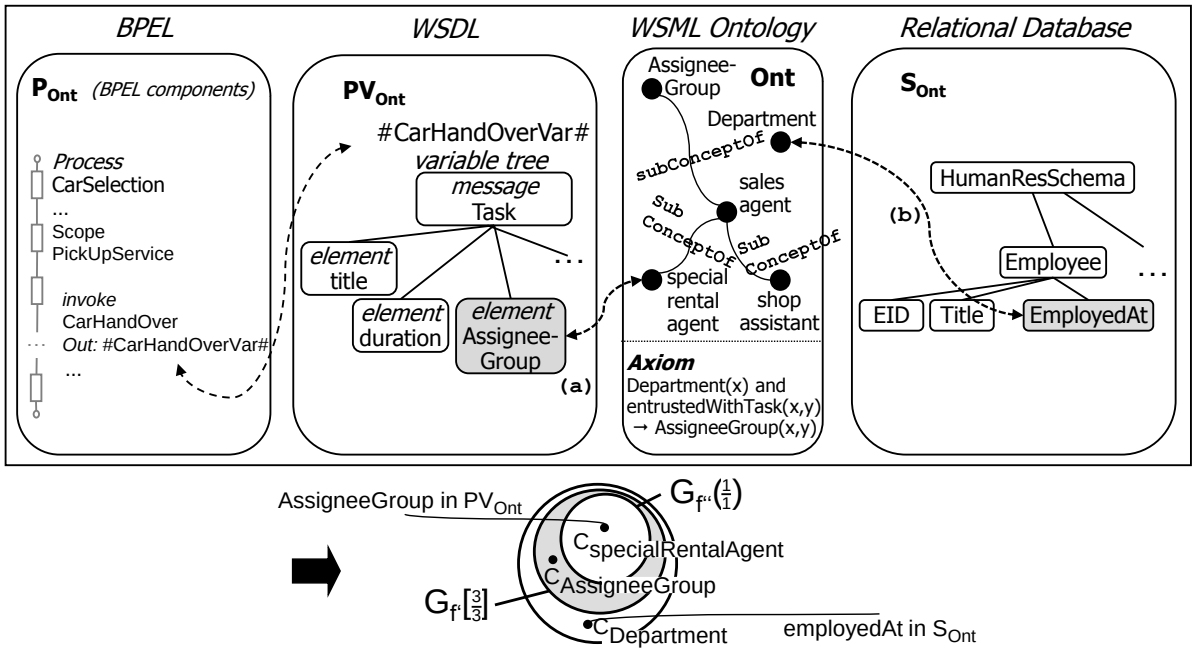


Abb. 4.29: Ausführung der Regel R3 und R4 in Konzeptgruppe

finition 7 können nun auch noch weitere Matchkandidaten in Abb. 4.30 als Kombination erstellt werden als com_{P2} in (a) und com_{P2} und com_{P4} in (b). So werden zuvor gefundene semantische Beziehungen auch für diese Matchkandidaten berücksichtigt.

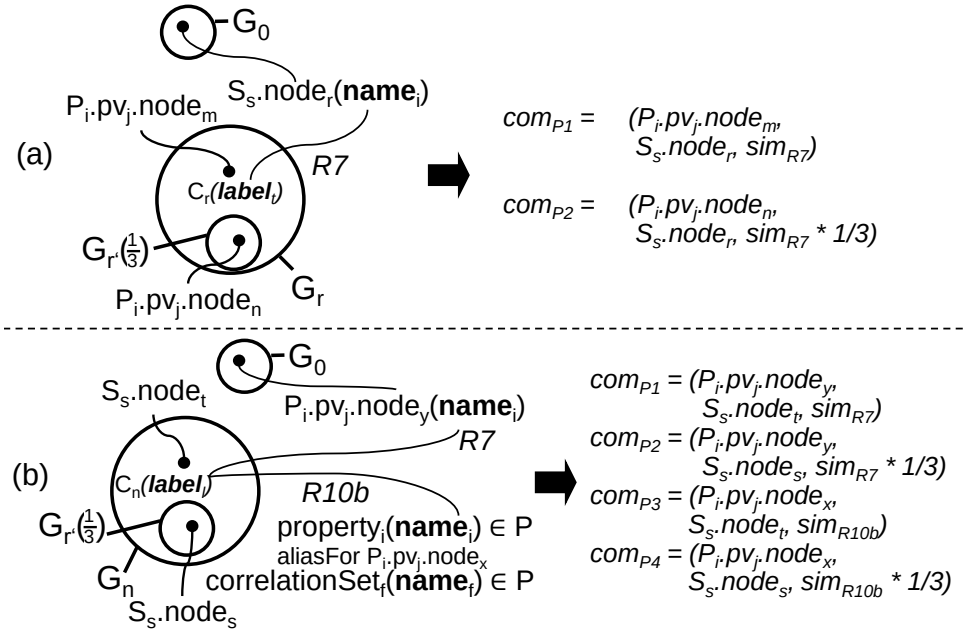


Abb. 4.30: Ausführung der Regel R7 und R10b in Konzeptgruppe

Bei den Regeln R6 und R10b werden alle Elemente der Prozessvariablen mit Schemaelementen kombiniert und der Ähnlichkeitswert durch das Matchen ihrer Namen berechnet. Die syntaktischen Regeln sollten nicht auf den Konzeptgruppen, sondern direkt auf den Elementen durchgeführt werden, damit keine redundanten Prüfungen der Namen von zu matchenden Elementen in evtl. mehrfach verschachtelten Konzeptgruppen nötig werden.

Für jede Kombination, die durch die Regeln R1 bis R10 gefunden wurde (d.h. bei der betreffenden Kombination ist $sim_P > 0$), wird zusätzlich der Kontext der Matchpartner beachtet, um den strukturellen Ähnlichkeitswert sim_{str} zu erhalten. Dafür werden für jede Kombination alle Regeln R11 bis R14 ausgeführt, um sim_{R11}, sim_{R12}, sim_{R13} und sim_{R14} zu bestimmen wie in Kapitel 4.5 beschrieben. sim_{str} setzt die Ähnlichkeitswerte dann zusammen wie in Definition 8 beschrieben:

Definition 8 (Berechnung des Ähnlichkeitswertes sim_{str})
Sei der Wert sim_{str} der strukturelle Ähnlichkeitswert, der sich aus den berechneten Ähnlichkeitswerten der Regeln R11, R12, R13 und R14 folgendermaßen ergibt: $sim_{str} = \alpha * sim_{R11} + \beta * sim_{R12} + \gamma * sim_{R13} + \delta * sim_{R14}$. Dabei gilt $\alpha + \beta + \gamma + \delta = 1$.

Jeder einzelne berechnete Ähnlichkeitswert liegt zwischen 0 und 1. Dabei können die Ähnlichkeitswerte unterschiedlich stark gewichtet werden. Die Gewichtung einzelner Regeln ist 0, wenn die Voraussetzung der Regel nicht erfüllt ist oder sie vom Analysten als unwichtig eingestuft wird. Falls z.B. keine Ontologie vorhanden ist und damit auch keine annotierten Matchelemente, kann Regel R13 nicht ausgeführt werden, d.h. in diesem Fall wird $\gamma = 0$.

Zuletzt wird für jede Kombination $com_i \in COM$ der finale Ähnlichkeitswert sim_i für ein Tripel $com_i \in COM$ mit der Formel berechnet, in der ihr Wert sim_{Pi} mit dem Wert aus sim_{str_i} verknüpft wird (siehe Definition 9 und [MBR01]):

Definition 9 (Berechnung der Ähnlichkeit als gewichteter Durchschnittswert)
Sei der Wert sim_i ein gewichteter Durchschnittswert der Ähnlichkeitswerte sim_P und sim_{str} eines Matchtripels com_i, dann gilt: $sim_i = w_{weight} * sim_P + (1 - w_{weight}) * sim_{str}$ mit $w_{weight} \in \mathbb{Q}$ zwischen 0 und 1.

Falls für eine Kombination mit gleichen Matchpartnern mehrere sim_P-Werte existieren, wählt die Kontrollstrategie des Frameworks einfachheitshalber den höchsten Ähnlichkeitswert aus. Eine Ausnahme ist die Regel R8, die den Ähnlichkeitswert auf 0 setzt und keine anderen Regelausführungen zulässt. Jeder Tripel mit einer finalen Ähnlichkeit für eine Kombination über einer durch den Benutzer festgelegten Schwelle ist akzeptabel ($sim_i > Schwellwert$) und wird vom Framework gespeichert.

4.7.5 Definition der Kontrollstrategie

In Abb. 4.31 ist die Kontrollflussstrategie des Matchprozesses dargestellt. In der Abbildung sind der erste Pairingschritt (Schema-Annotation) und der letzte Filterschritt (manuelles Matching) nicht illustriert, da dieses Kapitel den jeweils anderen Pairing- und Filterschritt fokussieren soll. Wie in Kapitel 4.2 erwähnt, erhält der BIA-Matcher die Prozessvariablen, ihre Elemente und die Kontrollflusselemente der Prozesse, in die sie

eingebettet sind, sowie die operativen Schemaelemente und wenn vorhanden die Ontologie *Ont*. Welche Teile aus der Eingabemenge in den verschiedenen Schritten benötigt werden, ist oben in Abb. 4.31 durch die Pfeile dargestellt.

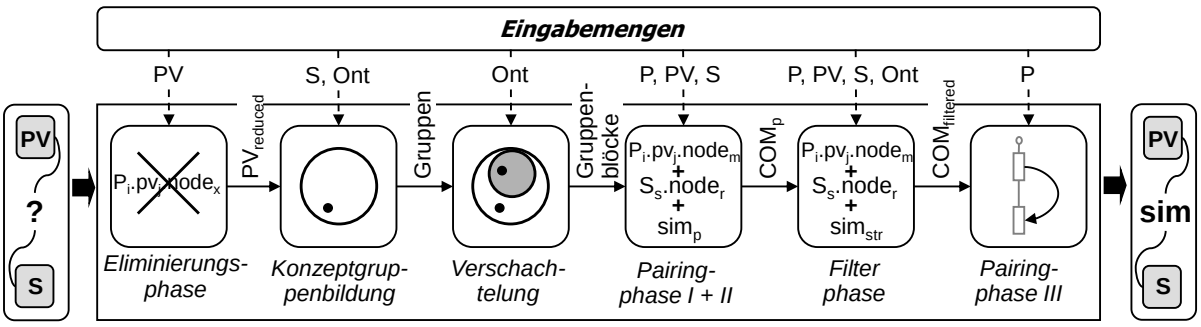


Abb. 4.31: Kontrollstrategie zum Erstellen der Matches

Zuerst wird die Regel R8 angewendet und eliminiert die nicht matchbaren Prozesselemente. Danach werden für die Anwendung der Pairingregeln die Konzeptgruppen initialisiert. Jedes Konzept in *Ont*, das für eine semantische Annotation der Prozessvariablenelemente und der operativen Elemente verwendet wurde, wird in eine eigene Konzeptgruppe G_i mit $i > 0$ mitsamt seinen Annotationselementen gespeichert. Alle restlichen nicht annotierten Variablenelemente und Schemaelemente werden mit einem leeren Konzept c_0 annotiert und in die Gruppe G_0 gespeichert. Danach beginnt die Verschachtelungsphase. In dieser Phase werden nacheinander die Regeln R2 bis R5 ausgeführt. Eine Regel wird nacheinander für alle Konzeptgruppen und ihre enthaltenen Konzepte überprüft und die Aktion bei Erfüllung der Vorbedingungen ausgeführt. Durch die Aktion werden die Konzeptgruppen ineinander als Blöcke verschachtelt und evtl. dupliziert und manche Gruppen mit einem Ähnlichkeitswert versehen (R3, R4).

Die verschachtelten Gruppenblöcke werden von den Regeln R1 bis R5 und R7, R10c in den anschließenden Pairingphasen I und II verwendet, die in Abb. 4.32 detaillierter dargestellt sind. Zuerst wird das nächste Konzept in der aktuellen Konzeptgruppe gemäß der Tiefensuchstrategie [AL91] ermittelt (A). Für dieses Konzept bzw. seine annotierten Elemente werden die Kombinationsmöglichkeiten durch die Matchregeln der Phasen bestimmt. Dafür wird die nächste Regel herausgesucht (B). Bei der Reihenfolge der Anwendung sind die Abhängigkeiten zwischen den einzelnen Regeln zu berücksichtigen (siehe Kapitel 4.7.2). Im nächsten Schritt wird die Regel angewendet (C). Dazu wird ein Matchpartner herausgesucht, überprüft, ob die Vorbedingungen erfüllt sind und dann ggf. Kombinationen erstellt. Dieser Schritt wird für alle Kombinationsmöglichkeiten mit den Elementen des Konzepts und der aktuellen Regel wiederholt. Danach wird für alle Kombinationen überprüft, ob in COM_P schon ein Match mit den gleichen Matchpartnern existiert (D), da dann nur der Ähnlichkeitswert angepasst werden muss, falls der gespeicherte Wert niedriger ist. Ansonsten werden die Kombinationen in COM_P aufgenommen (E). Außerdem wird in (D) noch für jeden Ähnlichkeitswert sim_i der neuen Kombination überprüft, ob die nächste Kombinationsmöglichkeit für das aktuelle Kon-

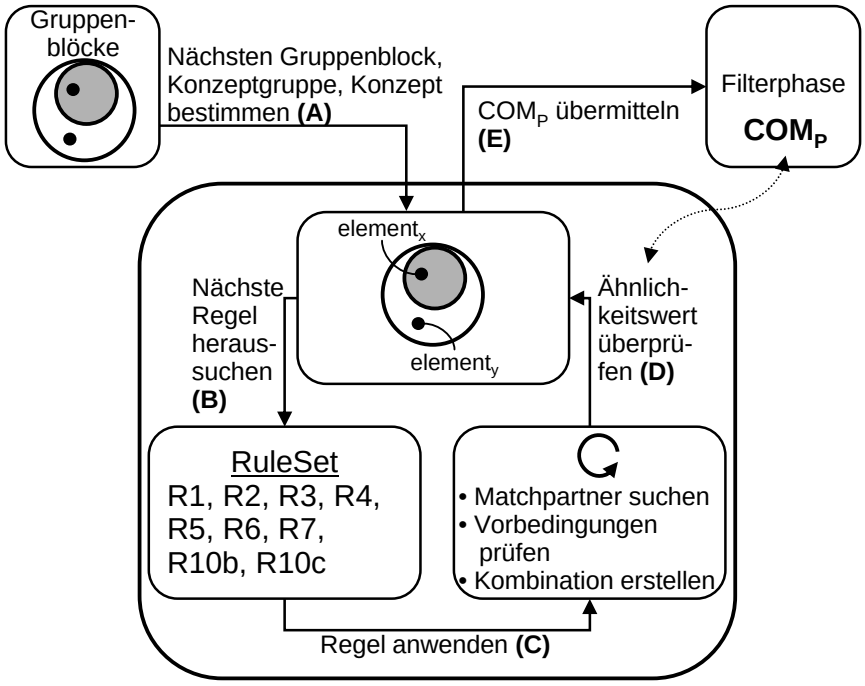


Abb. 4.32: Kontrollstrategie der Pairingphasen I + II

zept mit der nächsten Regel bestimmt werden soll oder für das nächste Konzept weiter gesucht werden soll, wenn für das aktuelle Konzept schon alle Regeln durchgeführt wurden. Nach Umformung der Formeln in Definition 7 und Definition 9 muss nämlich gelten: $sim_i * sim_{nextConceptGroup} > \frac{threshold-(1-w_{weight})}{w_{weight}}$. Kann eine Regel auf kein Konzept in einem Gruppenblock mehr angewendet werden, wird das nächste Konzept im nächsten Gruppenblock für die Kombination herangezogen. Auf diese Weise werden alle Konzeptgruppen, ihre verschachtelten Konzeptgruppen und ihre enthaltenen Elemente in den Matchprozess miteinbezogen, um die besten Kombinationen zu finden.

In der Kontrollstrategie in Abb. 4.31 in der anschließenden Filterphase werden die Filterregeln R11 bis R14 auf die Matchergebnisse in COM_P angewendet. Der Ablauf ist genauer in Abb. 4.33 dargestellt. Hier wird nacheinander jedes com_i aus COM_P in (A) bestimmt, in (B) alle Regeln geladen und angewendet (C). Für die Ausführung muss der Kontext der Matchpartner bestimmt und überprüft werden, ob die Vorbedingungen erfüllt sind. Es wird jeweils der Ähnlichkeitswert sim_{str} und der finale Ähnlichkeitswert sim_i berechnet mit den Formeln aus Definition 8 und Definition 9 und in com_i angepasst (E). Auch hier muss in (D) gelten: $sim_i > threshold$. Der Ablauf der Filterphase setzt sich so lange fort, bis alle Matchergebnisse aus COM_P durch die Filterregeln vollständig überprüft und angepasst worden sind.

Danach werden in der Pairingphase III in Abb. 4.31 die Regeln R9 und R10a auf $COM_{filtered}$ angewendet. Es wird nach Elementen in $COM_{filtered}$ gesucht, deren Prozesskontext die Bedingung der Regeln erfüllen. Wird ein Match gefunden, werden die in

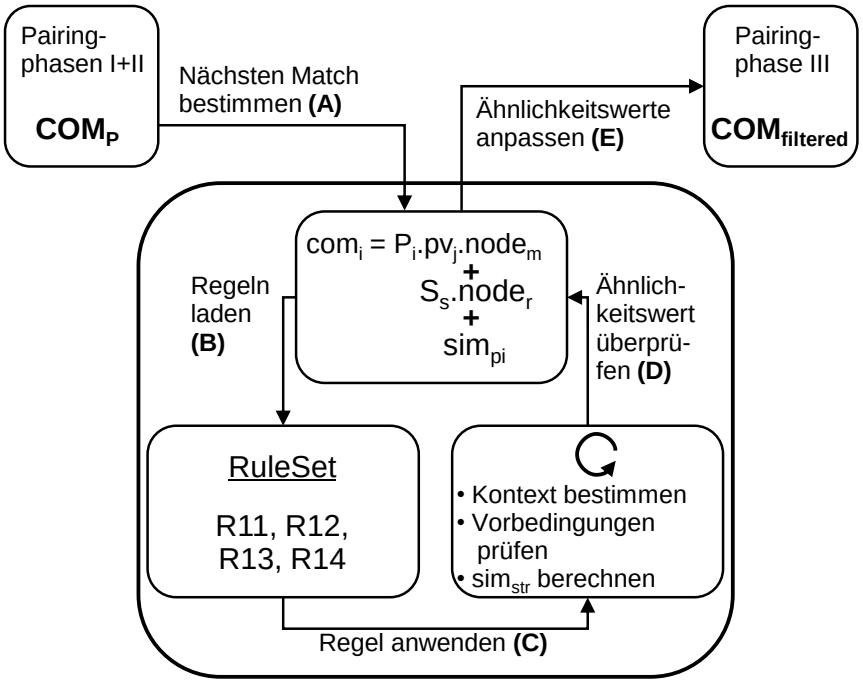


Abb. 4.33: Kontrollstrategie der Filterphase

der Regel definierten Berechnungen angewendet, um den Ähnlichkeitswert für die neue Kombination com_i zu bestimmen bzw. anzupassen.

Verschiedene Algorithmen realisieren diesen Ablauf. Im Folgenden werden die Algorithmen herausgegriffen, die die für diese Arbeit interessantesten Phasen, die Pairingphasen I+II und die Filterphase, aufzeigen. Die Kontrollstrategie für die Pairingphase III ist ähnlich zu der Strategie der Filterphase aufgebaut. Zunächst folgen die Algorithmen 3, 4 und 5 für den Ablauf der Pairingphasen. Die Methode *receiveCombinations* (Algorithmus 3) traversiert eine gegebene Menge von Konzeptgruppenblöcken *gbSet*, die enthaltenen Gruppenblöcke *gb* und Gruppen *g* (auch doppelt enthaltene Gruppen) sowie ihre enthaltenden Konzepte *c*, um die Matchkandidaten für COM_P zu bestimmen. Dazu wird mit den Methoden *getNextGroupingBlock*, *getNextGroup* und *getNextConcept* jeweils immer der nächste Gruppenblock bzw. die nächste Gruppe oder das nächste Konzept herausgegriffen. Für jedes Konzept *c* werden alle Regeln *r* im RuleSet (R1 - R7, R10b, R10c) mit der Methode *getNextRule* nacheinander bestimmt und die Methode *applyCombinationRules* (Algorithmus 4) aufgerufen.

Der Algorithmus 4 erstellt mit der Regel *r* die Kombinationen. Für die linguistischen Regeln (R6, R7, R10b, R10c) sollen durch die Methode $findCombinationViaNames$ alle Namen der Elemente und betreffenden Annotationskonzepte verglichen und Matches mit Ähnlichkeitswerten über einem vorher festgelegtem Schwellwert in $CAND$ abgelegt werden. Die anschließende Methode $getMatchedElementsAndSim$ speichert die gematchten, nicht annotierten Partnerelemente und ihre Ähnlichkeitswerte für die Elemente, die mit *c* annotiert sind, in *partnerSet* für spätere abhängige Kombinationen mit der Regel R7

Algorithmus 3 *receiveCombinations*

Input: Set of Grouping Blocks *gbSet*, *RuleSet*
Output: Set of Combinations COM_P

```
 1: COM_P ← ∅
 2: while gbSet is not fully traversed do
 3:    gb ← getNextGroupingBlock(gbSet)
 4:    while gb is not fully traversed do
 5:       g ← getNextGroup(gb)
 6:       while g is not fully traversed do
 7:          c ← getNextConcept(g)
 8:          while RuleSet is not fully traversed do
 9:             r ← getNextRule(RuleSet)
10:             CAND ← applyCombinationRule(r, c, gb)
11:             COM_P ← adjustCombinations(CAND, COM_Pold)
12:          end while
13:       end while
14:    end while
15: end while
```

oder R10c im Gruppenblock. Danach wird die Regel r angewendet, wenn sie nicht Regel R6 oder R10b ist und das Konzept c Teil der Ontologie ist, also nicht aus Gruppe G_0 ist. Der Algorithmus soll weitere Matches über die semantischen Beziehungen herauszufinden. Für jedes Konzept c sucht die Methode $findRelatedConcepts$ gemäß der aktuellen Matchregel r in dem gleichen Gruppenblock gb nach passenden Konzepten. Die gefundenen Konzepte werden in $relSet$ abgelegt und für jedes Konzept rel daraus (bestimmt mit Methode $getNextConcept$ die Methode $findSemanticCombinations$ (siehe Algorithmus 5) aufgerufen, um die Elemente der Konzepte rel und c zu matchen.

Algorithmus 4 *applyCombinationRule*

Input: Rule r, Concept c, Grouping Block gb
Output: Set of Combination Triples $CAND$

```
 1: CAND, partnerSet ← ∅
 2: if (r ≥ R6) then
 3:    CAND ← findCombinationsViaNames(P, PV, S, c, r)
 4:    partnerSet ← getMatchedElementsAndSim(CAND, c, r)
 5: end if
 6: if ((r ≠ R6) and (r ≠ R10b)) and g ≠ G_0 then
 7:    relSet ← findRelatedConcepts(c, r, gb)
 8:    while relSet is not fully traversed do
 9:       rel ← getNextConcept(relSet)
10:       CAND ← findSemanticCombinations(rel, c, r, gb, partnerSet)
11:    end while
12: end if
```

In Algorithmus 5 werden die Elemente der Konzepte rel und c zunächst durch die Funktionen *getElements* in Zeile 2 und 3 bestimmt. Falls das *partnerSet* nicht leer ist, wird gerade R7 oder R10c ausgeführt und dementsprechend in *cElements* statt den eigentlichen Elementen von c die gematchten nicht annotierten Elemente aus *partnerSet* mit der Funktion *getPartnerElements* bestimmt und zurückgegeben, da sie nun mit den Elementen des Konzeptes rel gematcht werden sollen. In der While-Schleife von Zeile 4 bis 13

werden alle Prozesselemente mit Schemaelementen kombiniert. Gemäß der verwendeten Regel r und dem beteiligten Konzeptgruppenblock gb wird mit der Methode *calculateSimilarity* die Ähnlichkeit berechnet (Zeile 9) und der Kombinationstripel als neuer Matchkandidat mit der Methode *addCandidate* in $CAND$ aufgenommen. R7 oder R10c übernehmen dabei einfach den Ähnlichkeitswert aus dem *partnerSet* und kombinieren ihn ggf. gemäß Definition 7 mit den anderen berechneten Werten.

Algorithmus 5 *findSemanticCombinations*

Input: RelatedConcept rel, Concept c, Rule r, Grouping Block gb, MatchedElements $partnerSet$
Output: Set of Combination Triples $CAND$

```
1: CAND ← ∅
2: relElements ← getElements(rel)
3: cElements ← getElements(c, getPartnerElements(partnerSet))
4: while cElements is not fully traversed do
5:    cElem ← getNextElement(cElements)
6:    while relElements is not fully traversed do
7:       relElem ← getNextElement(relElements)
8:       if (cElem ∈ PV and relElem ∈ S) or (cElem ∈ S and relElem ∈ PV) then
9:          cand ← calculateSimilarity(cElem, relElem, r, gb, getSim(partnerSet))
10:         CAND ← addCandidate(cand)
11:      end if
12:   end while
13: end while
```

Diese neue Matchkandidatenmenge $CAND$ wird über die Funktion *adjustCombinations* (Zeile 11 in Algorithmus 3) mit den schon existierenden Kombinationen in COM_P verglichen und aufgenommen bzw. ersetzt, wenn es schon eine Kombination zwischen den gleichen Elementen mit einem niedrigeren Ähnlichkeitswert gibt.

Die Filterphase wird durch die Algorithmen 6 und 7 aufgezeigt. Der Algorithmus 6 ($receiveFilteredCombinations$) traversiert eine gegebene Menge von Kombinationen aus COM_P, bestimmt mit der Funktion *getNextCombination* jeweils den nächsten Matchkandidaten *cand* aus COM_P und ruft für jeden Matchkandidaten die Funktion $applyFiltering$ (Algorithmus 7) auf. Der Algorithmus 7 ist für Berechnung des strukturellen Ähnlichkeitswertes sim_{str} für jeden Matchkandidaten verantwortlich und erstellt einen neuen Matchkandidaten $cand_{new}$ mit angepasstem finalem Ähnlichkeitswert.

Algorithmus 6 $receiveFilteredCombinations$

Input: Set of Combinations COM_P
Output: Set of Filtered Combinations COM

```
1: while COM_P not fully traversed do
2:    cand ← getNextCombination(COM_P)
3:    cand_new ← applyFiltering(cand)
4:    COM ← adjustCombinationSet(cand_new)
5: end while
```

Die Methode $applyFiltering$ traversiert alle existierenden Filterregeln (R11 bis R14) im $FilterRuleSet$ und die Funktion $getNextRule$ ermittelt die nächste Matchregel r. Wenn

die Vorbedingungen der Regel für den zu filternden Matchkandidaten erfüllt sind, wird für diesen Kandidaten der strukturelle Ähnlichkeitswert mit dieser Regel r berechnet, wie in Kapitel 4.7.4 beschrieben und zu dem bisherigen strukturellen Ähnlichkeitswert der vorher ausgeführten Filterregeln dazuaddiert (siehe Zeile 4 in Algorithmus 7). Abhängig von der jeweiligen Regel werden dazu Elemente aus dem Prozess P, der Variablen PV, dem operativen Schema S oder der Ontologie Ont benötigt. Werden dagegen die Vorbedingungen der Regel nicht erfüllt, wird diese Funktion übersprungen und die nächste Filterregel betrachtet. Schließlich erstellt die Funktion *calculateFinalSimilarity* aus sim_{str} und sim_P einen Matchkandidaten mit einem neuen Ähnlichkeitswert gemäß Definition 9.

Abschließend wird die Ergebnismenge COM in Zeile 4 des Algorithmus 6 durch Aufnahme des veränderten Matchkandidaten mit Hilfe der Funktion *adjustCombinationSet* angepasst, falls noch kein gleicher Match existiert oder nur mit niedrigerem Ähnlichkeitswert.

Algorithmus 7 *applyFiltering*

Input: Combination Triple *cand*
Output: filtered Combination Triple $cand_{new}$

1: $sim_{str} \leftarrow \emptyset$
2: **while** *FilterRuleSet* is not fully traversed **do**
3: $r \leftarrow getNextRule(FilterRuleSet)$
4: $sim_{str} \leftarrow sim_{str} + calculateSimilarity(r, cand, P, PV, S, Ont)$
5: **end while**
6: $cand_{new} \leftarrow calculateFinalSimilarity(cand, sim_{str})$

4.7.6 Terminierung des Matchalgorithmus

Die Terminierung der vorgestellten Kontrollstrategie des Matchalgorithmus wird in folgenden Punkten erläutert:

- *keine zyklischen Anwendungen der Matchregeln*: Normalerweise werden die Regeln immer auf ein Paar von Konzepten oder Elementen angewendet, um sie zu kombinieren und zyklische Anwendungen treten nicht auf. Bei der Verschachtelung der Konzeptgruppen durch R3 oder R4 könnte man jedoch vermuten, zyklische Verschachtelungen zu finden. Unter der Annahme, dass die verwendete Ontologie korrekt modelliert wurde, ist der Vorgang der Verschachtelung der Konzeptgruppen, die Subkonzepte enthält, durch Regel R3 nicht zyklisch, sondern absteigend modelliert. Ist das Konzept ein kombiniertes Subkonzept von mehreren Konzepten aus verschiedenen Konzeptgruppen, wird es vervielfältigt und in beide Konzeptgruppen aufgenommen.

  Auch die Regel R4 verschachtelt je nach Art der Axiome und vorher ausgeführten Matchregeln schon vorhandene Konzepte im Gruppenblock erneut in diesen. Diese Verschachtelung wird jedoch nur einmal pro Axiom für jede Konzeptgruppe durchgeführt und nicht für die neu verschachtelten Gruppen bezüglich des gleichen Axioms. R4 führt daher zu keiner zyklischen Anwendung. Für das Pairing werden alle Konzepte in den Gruppenblöcken miteinander verglichen und ihr Ähnlichkeitswert gemäß den

Regeln berechnet. Folglich endet die Pairingphase, wenn alle Gruppen begutachtet wurden.

- *keine oszillierenden Regelanwendungen*: Die Kontrollstrategie legt die Reihenfolge der verschiedenen Phasen fest. So existieren keine Regeln, welche Matches anderer Regeln wieder rückgängig machen. Es werden nur die Ähnlichkeitswerte der Matches angepasst. Einzelne Elemente werden nur bei der Regel R8 gelöscht, d.h. aus der verfügbaren Menge der möglichen Matchpartner herausgenommen. Jede dieser Phasen und ihre Regeln werden gemäß der Kontrollstrategie sequentiell ausgeführt und nach Anwendung der nächsten Phase nicht mehr. Deshalb sind immer wieder abwechselnde Regelanwendungen in zyklisch abwechselnden Phasen, die eine Terminierung der Kontrollstrategie in endlichen Schritten verhindern, nicht möglich.
- *endlich große Eingabemengen*: Beim Aufbau der Konzeptgruppen wird jedes Konzept für jede Regel nur einmal betrachtet. Auch die Traversierung beim Matchen innerhalb einer Gruppe für eine Regel betrachtet jedes Konzept der Gruppe bzw. jedes Prozesselement und operatives Schemaelement genau einmal, wenn überprüft wird, ob zwei Matchpartner gemäß der jeweiligen Regel als Match abgespeichert werden können. Da die Anzahl von Konzepten, Konzeptgruppen und Regeln endlich ist, wird gewährleistet, dass der ganze Matchprozess nach einer endlichen Anzahl von Schritten terminiert.

4.8 Prototyp des Matcheditors

Die Matchregeln aus den vorherigen Kapiteln wurden prototypisch in einem Matcheditor umgesetzt. Dieses Kapitel zeigt seine Architektur und Funktionalität auf. Details finden sich in [RNB10].

4.8.1 Architektur des Matcheditors

Die Architektur des Matcheditors (siehe Abb. 4.34) gliedert sich im Wesentlichen in vier Teile: Modell-Transformation (MT), Benutzerschnittstelle, Linker-Engine und der Kern des BIA-Editors. MT ist verantwortlich für das Laden der zu matchenden Elemente, ebenso wie der Beschreibungsdaten, die zur semantischen Annotation benutzt wurden. MT ist die gleiche Engine, die auch zum Laden der zu annotierenden Modelle und ihrer Transformation auf die interne Struktur des Matcheditors in Kapitel 3.4 verwendet wurde. Früher schon getätigte Annotationen für die semi-automatische bzw. manuelle Kombination werden auch von MT in den Editor geladen. Die automatische Schema-Annotation aus Abb. 4.2 ist bisher nicht Teil des BIA-Editors. Über die Benutzerschnittstelle kann der Benutzer das Laden der Modelle und Speichern der Matches sowie den Matchprozess selbst steuern. Der BIAEditor-Kern wiederum ist für die Ausführung der Funktionalitäten verantwortlich, die von der Benutzerschnittstelle zur Verfügung gestellt werden und steuert die MT und die Linker-Engine. Im Folgenden werden die Komponenten außer der bereits bekannten MT näher ausgeführt.

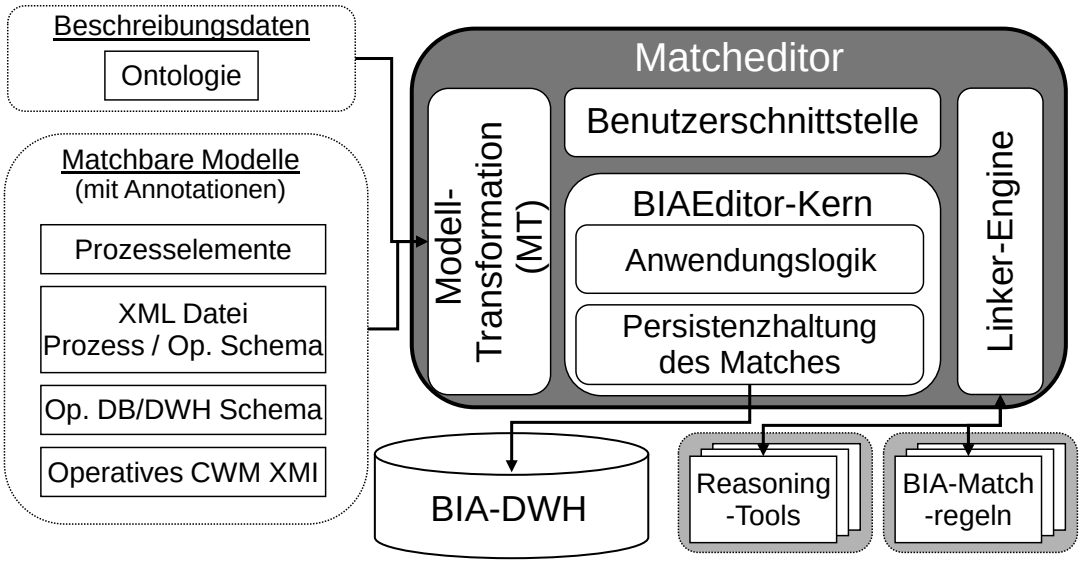


Abb. 4.34: Architektur des Matcheditors

Benutzerschnittstelle

Über die Benutzerschnittstelle führt der Benutzer die Kombination der annotierten bzw. der partiell-annotierten Prozess- und operativen Datenelemente durch. Ihm stehen folgende Funktionalitäten zur Verfügung, die in Kapitel 4.8.2 noch ausführlich diskutiert werden: Laden der matchbaren Modelle - Prozesselemente und operative Daten - bzw. ggf. ihrer Annotations- und Kontextdaten, Matchen der Elemente durch die BIA-Matchregeln unter der Verwendung von Reasoning-Tools bei annotierten Elementen, manuelles Matchen der Elemente, Darstellen der gematchten Elemente im Editor, Speichern der Matches.

Linker-Engine

Die Engine steuert die Ausführung der Matchregeln. Sie führt die Regeln gemäß der beschriebenen Kontrollstrategie aus, um Konzeptgruppen aufzubauen und Matches zwischen Prozesselementen und operativen Schemaelementen mit möglichst hohem Ähnlichkeitswert zu finden. Um die semantischen Beziehungen in der Ontologie zwischen den annotierten Elementen zu erkennen, verwendet die Engine Reasoning-Tools, die die Ausführung der semi-automatischen Regeln unterstützen.

Kern des BIA-Editors

Der Kern des Matcheditors realisiert seine Anwendungslogik, die notwendig ist, um die zu matchenden Modelle und Ontologien zu serialisieren, zu traversieren und zu kombinieren. Für diese Realisierung steuert der BIAEditor-Kern die MT und die Linker-Engine. Außerdem gewährleistet er die Persistenz der Matches zwischen den Modellelementen. Je

nach Matchvariante werden die Matches in den Audit Trail in eine eigene Tabelle (gilt für alle Varianten) oder in eine separate Datei (manuelle Kombination durch den Prozessdesigner) gespeichert ähnlich wie die Annotationen in Kapitel 3. Wie für die Annotationen muss auch für die Speicherung der Matches im Audit Trail berücksichtigt werden, dass der referenzierte Prozess bereits nach dem Prozessdesign in das WFMS überführt wurde und dort vorliegt.

4.8.2 Funktionalität des Matcheditors

Die Umsetzung der Architekturkonzepte ist in dem Matcheditor realisiert. Abb. 4.35 zeigt das Hauptfenster des Editors. Man kann es im Annotationseditor auswählen oder es öffnet sich automatisch nach dem Starten des Matchvorgangs. Das zentrale Element des Editors ist der Prozesselementexplorer (1), welcher die Struktur von BPEL-Variablen sowie WSDL- und XML-Schema-Dateien visualisiert. Die Struktur ist bis auf das letzte Element ohne Kindelement aufklappbar. In der Abbildung ist die BPEL-Variable *CarSelection.inputData.input.CustomerData.CustComplexType.custID* geöffnet. Rechts daneben befindet sich der operative Elementexplorer (2), der die operativen Elemente visualisiert. Die Matches zwischen den Elementen werden mit Verbindungslinien angezeigt. Der Benutzer kann zu den (semi-)automatisch gefundenen Matches neue Matches dazuklicken. Diese erhalten den Ähnlichkeitswert 1. Außerdem kann er manuell Matches löschen, die dann den Wert 0 erhalten und nicht mehr angezeigt werden. Über die Attributstabelle (3) können die Annotationen des gerade angeklickten Elementes angezeigt werden, falls eine Annotation vorhanden ist. Über die Menüleiste (4) können alle Funktionen des Editors ausgeführt werden. Der Aufruf der wichtigsten Funktionen kann auch über die Toolbar (5) erfolgen. Sie startet mit den Funktionen, partiell-annotierte Regeln auszuführen, und fährt mit Funktionen fort, um Matches zu speichern sowie das manuelle Zufügen und Löschen von Matches durchzuführen.

Für den Matchprozess bedarf es der Durchführung mehrerer Schritte, von denen die Wichtigsten exemplarisch dargestellt sind:

- Semi-automatisches Matchen: Im Annotationseditor kann der semi-automatische Matchvorgang der geladenen Elemente gestartet werden. Durch Klicken auf das Symbol (zweites von rechts in der Toolbar in Abb. 3.3) oder durch Auswahl in der Menüleiste wird der Benutzer über mehrere Schritte, in denen er Angaben zu den Quelldaten geben muss, zum Ergebnis des semi-automatischen Matchings geleitet. Abb. 4.36 zeigt Schritt 2, in dem der Benutzer den Prozess, und Schritt 3, in dem er die operative Quelle auswählen muss. Der Editor schaltet automatisch auf die Matchperspektive nach Beendigung des Matchvorgangs.
- Automatisches Matchen: Es wird in der Matchperspektive gestartet und es ist nur ein Zwischenschritt nötig, bevor das Matching beginnen kann. In dem Schritt (siehe Abb. 4.37) wählt der Benutzer die Prozessvariablen aus, deren Elemente durch die Matchregeln mit den operativen Daten gematcht werden sollen. Als Ergebnis werden die Linien zwischen den gematchten Elementen dargestellt, wie in Abb. 4.35 gezeigt.

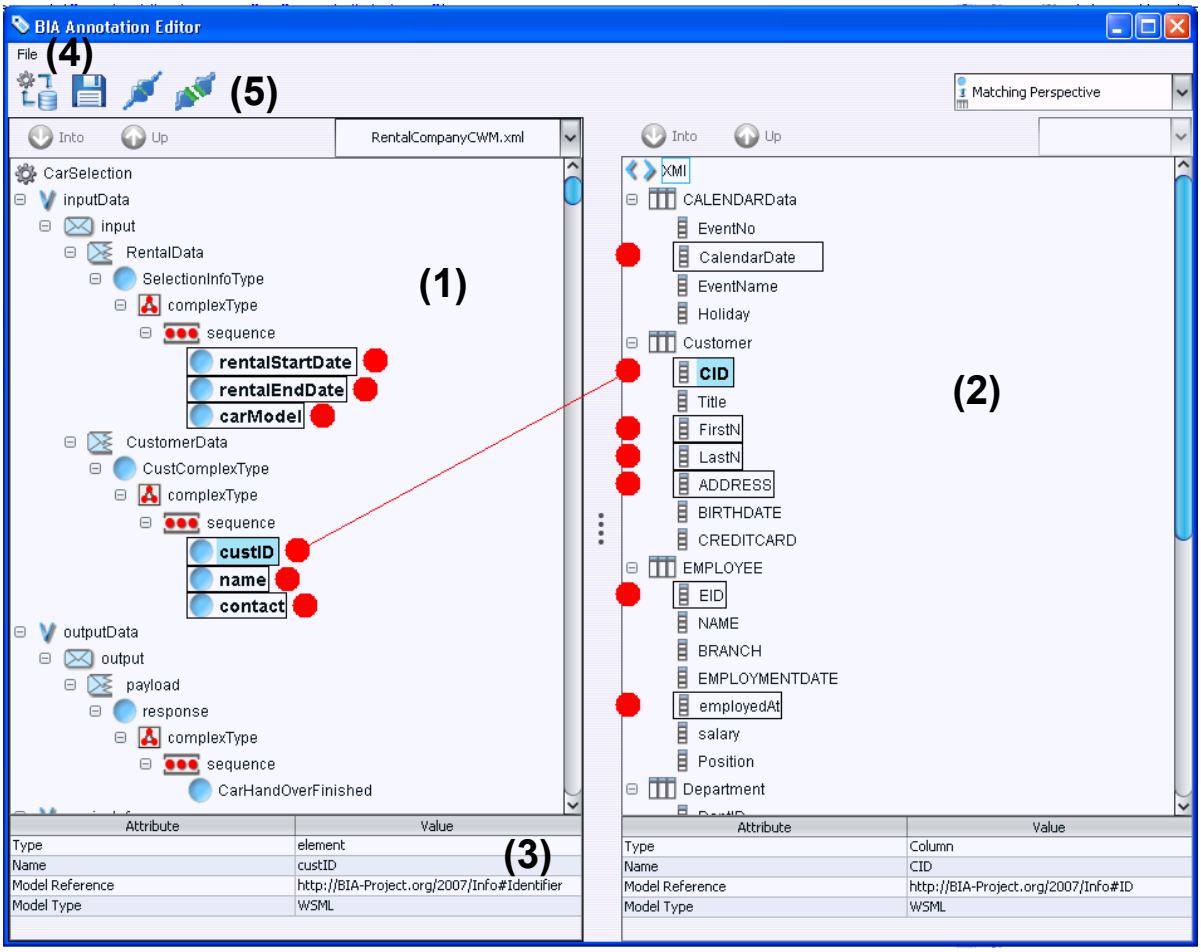


Abb. 4.35: Hauptfenster des Matcheditors

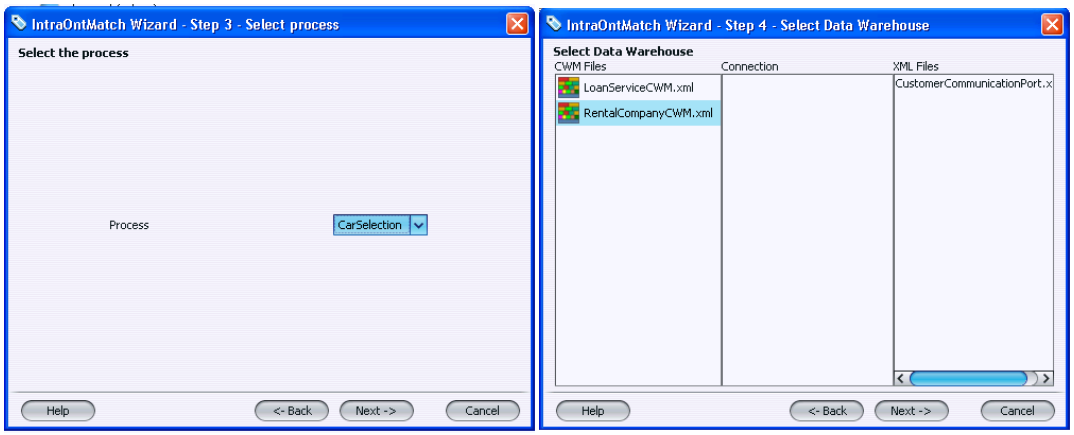


Abb. 4.36: Schritte für das semi-automatische Matchen

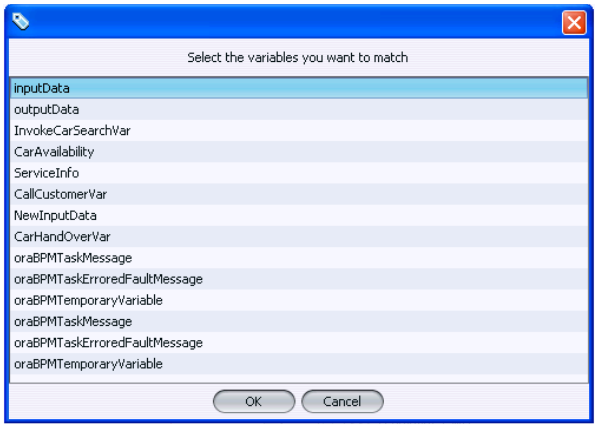


Abb. 4.37: Schritte für das automatische Matchen

- Manuelles Matchen: Der Benutzer kann die dargestellten Matches löschen oder neue Matches mit dem Ähnlichkeitswert 1 hinzufügen, jeweils durch Anklicken der zwei beteiligten Elemente und der zwei Symbole in der Toolbar, das sich am weitesten rechts befinden. Auf dieselbe Art funktioniert das manuelle Löschen eines Matches.
- Speichern: Hier wird die gesamte Menge der angezeigten Matches in eine Matchtabelle gespeichert. Ein Auszug der gefundenen Matches aus dem Beispielszenario ist im Anhang C dargestellt.

4.8.3 Implementierungsaspekte des Matcheditors

Auch der Matcheditor ist wie der Annotationseditor unter Java Version 1.5 implementiert, da sie miteinander zusammenhängen und als ein Werkzeug verwendet werden. Zur Erbringung seiner Funktionalität verwendet der Matcheditor daher den Annotationseditor, um die Elemente zu laden. Die Reasoner Jena [HPO08] bzw. Pellet [SPG+07] werden vom Matcheditor verwendet, um die semantischen Matchregeln zu unterstützen. Mit diesen Reasoning-Tools können u.a. Synonyme und Subkonzepte in den Ontologien aufgedeckt werden, die dann vom Editor weiterverarbeitet werden. Je nachdem, ob WSML oder OWL als Ontologiesprache verwendet wurde, müssen die Matchregeln auf andere Schlüsselwörter in der Definition der Konzepte und Axiome achten. Details zu den Unterschieden sind in [SdBF08] beschrieben.

Die Matchregeln R6, R11 und R14 basieren auf Matchverfahren, die in Werkzeugen bereits umgesetzt wurden (wie z.B. in [DR02]). Da der Quellcode jedoch nicht frei verfügbar war und keine API existierte, wurden die Techniken für den BIA-Ansatz ebenfalls implementiert.

Wie es aus der Funktionalität des Matcheditors ersichtlich wird, wurde die Kontrollstrategie aus Abb. 4.31 für den Prototypen leicht modifiziert. Dafür wurden die semiautomatischen und automatischen Matchregeln separat ausgeführt, d.h. die Pairingphase I + II wurden als separate Schritte implementiert. Nach der Filterphase in Abb. 4.31

folgt also ein Schnitt, nach dem erneut eine Benutzereingabe wie in Abb. 4.37 nötig ist. Daraufhin können Pairingphase II, Filterphase und Pairingphase III erfolgen. Dies wurde aus Gründen einer einfachen Evaluation der Pairingregeln so implementiert, um eine Trennung bei der Bewertung der semi-automatischen und automatischen Pairingregeln zu ermöglichen (siehe auch Kapitel 4.9). In zukünftigen Arbeiten am Matcheditor kann die Kontrollstrategie jedoch bei Bedarf auf Abb. 4.31 angepasst werden.

Um eine möglichst hohe Erweiterbarkeit des Editors an verschiedene Engines und Datenbanken zu erreichen, wird, wie die Annotation, auch der Match in einem gleichen Tabellenschema gespeichert. Dabei kommt es darauf an, ob der Match manuell durch den Prozessdesigner im Annotationseditor gefunden wurde oder später in der Matchphase. Im ersten Fall wird der Match wie die Annotation im Annotationseditor in die Tabelle *BIA_Annotation* gespeichert. Andernfalls wird der Match in einer Matchtabelle mit dem Namen *BIA_Match (ProcessID, Variable, Element, Database, Schema, Table, Attribute)* angelegt. Das Attribut *ProcessID* beinhaltet auch hier die ID und die Version des Prozesses. Der Ähnlichkeitswert wird nicht gespeichert, da nur die Matches über einem gewissen Schwellwert gespeichert werden und in der nächsten Phase, dem Warehousing, weiterverwendet werden.

4.9 Evaluierung der Matchregeln

Dieser Abschnitt analysiert die Effektivität der Matchregeln und den Frameworkansatz in zwei getrennten Szenarien. Das erste Szenario verwendet annotierte Prozess- und Schemamodelle, auf die semi-automatische Regeln und strukturbasierte Filterregeln angewendet werden. Das zweite Szenario wendet automatische Matchregeln und strukturbasierte Filterregeln auf Prozesse und Schemas an, die nicht oder nur partiell annotiert sind. Des Weiteren werden die berechneten Ähnlichkeitswerte in beiden Szenarien evaluiert, da das Matchergebnis entscheidend davon abhängt. Durch die Aufteilung in zwei Szenarien können die Matchregeln jeweils für sich evaluiert werden, ohne die unterschiedliche Ausgangsbedingung bzgl. des Vorhandenseins einer Annotation beachten zu müssen.

Die Dauer des Matchvorgangs ist abhängig von der Anzahl der Prozessvariablen und ihrer Elemente sowie der operativen Schemaelemente. Des Weiteren ist die Ausführungslänge aller semi-automatischen Regeln und der automatischen Regeln R7 und R10c abhängig von der Anzahl der Annotationen und der Größe der Ontologie. Ebenso spielt bei R12 die Verschachtelungstiefe der Aktivitäten und die Häufigkeit des Auftretens der Variablen im Prozess eine Rolle, da dies bei der Berechnung des Ähnlichkeitswertes jeweils getrennt berechnet wird, um den maximalen Ähnlichkeitswert herauszufinden. Da das Matchverfahren des BIA-Frameworks jedoch auf der Workflowmodellierungsebene und nicht zur Laufzeit durchgeführt wird, ist die Dauer des Matchverfahrens keine kritische Größe und wird hier nicht gemessen. Durch den Matchvorgang wird die Ausführung des Workflows nicht beeinträchtigt. Würde der Matchprozess z.B. als Bestandteil erweiterter Monitoringverfahren angewendet, müsste hingegen auch die Dauer der Regelausführung genau evaluiert werden.

Da bisher für dieses Integrationsproblem, wie es in BIA betrachtet wird, kein anderes automatisches Integrationswerkzeug existiert, gibt es kein Benchmark und keine Datenmenge, mit der die Ergebnismenge des BIA-Frameworks verglichen werden kann. Daher wurden die notwendigen Eingabemengen bestehend aus Ontologie, BPEL-Prozessen und einer operativen relationalen Datenbank selbst erstellt. Trotzdem erlaubt uns die Evaluierung, die Qualität des Frameworks zu diskutieren und weitere Verbesserungsvorschläge anzubringen.

4.9.1 Konfiguration für die Experimente

Die experimentelle Konfiguration besteht aus drei Eingabemengen: den Modellen der Geschäftsprozesse, den operativen Datenbankschemas und der Ontologie für die semantische Annotation. Für die Evaluierung werden 20 verschiedene BPEL-Prozesse mit durchschnittlich 7 Variablen verwendet, von denen jede zwei bis fünf, manchmal auch bis zu 32 Elemente enthält, und ein Schema mit rund 400 Attributen. Prozesse und Schemas stammen aus der Autovermietungsdomäne. Die Ontologie ist in WSML modelliert und enthält alle Konzepte für die Annotationen. Die abgeleiteten Matchtripel hängen von der Qualität der verwendeten Ontologie und Annotationen ab, der Ausdrucksstärke der Elementnamen sowie der Namen der Kontextkomponenten. Für das erste Matchszenario wurden alle matchbaren Elemente annotiert, während im zweiten Szenario kein Schemaelement und nur jedes zehnte Prozesselement eine Annotation hatte. Um die Qualität der Regeln zu evaluieren, wurden die wirklich erwünschten Matches R manuell bestimmt und mit den Matches T verglichen, die durch die BIA-Matchregeln gefunden werden. Die Prozesse sind so modelliert, dass sie für ca. 31% aller Variablenelemente einen Match ergeben sollten. Aus R $\cup$ T können die korrekt ermittelten Matches C bestimmt werden. Basierend auf den Kardinalitäten der zwei Mengen, wurden die zwei Maßeinheiten für die Qualität berechnet (vgl. [DR02]):

- Genauigkeitswert = $\frac{|C|}{|T|}$, gibt die Verlässlichkeit der Vorhersagen für die Matches an
- Trefferquote = $\frac{|C|}{|R|}$, spezifiziert den Anteil der realen Matches, der von den Regeln gefunden werden

Im Idealfall, d.h. wenn alle und somit nur die realen Matches gefunden werden, erreichen die Messwerte den höchsten Wert, d.h. 1.

Aufgrund der spärlichen Beziehungen zwischen den Elementen innerhalb einer Variable, sind die strukturellen Ähnlichkeitswerte sim_{str} niedriger gewichtet als sim_P, d.h. w_{weight} aus Definition 9 wird auf 0.6 gesetzt. T wird durch Anwendung der Matchregeln und Filterregeln bestimmt und nimmt diejenigen Kombinationen in die Ergebnismenge auf, deren finaler Ähnlichkeitswert einen Schwellwert von 0.7 überschreitet. Dieser Wert basiert auf dem Ergebnis einer signifikanten Anzahl von Testläufen, kann aber natürlich für andere Prozesse oder Schemaelemente bei Bedarf variiert werden.

4.9.2 Ergebnisse für Trefferquote und Genauigkeitswert

Abb. 4.38 (a) zeigt die Matchergebnisse der semi-automatischen Regeln und der Filterregeln, die auf die annotierten Elemente angewendet wurden. Wie erwartet ist die Trefferquote sehr hoch (0.95), weil alle Modellelemente korrekt annotiert wurden. Eine detaillierte Analyse deckt auf, dass die fehlenden 5% der Matches, die das Framework nicht findet, zwar über die Ontologie gefunden werden könnten, aber die semantischen Beziehungen zwischen ihren Annotationskonzepten nur schwer zu erkennen ist. So wurden z.B. manche Beziehungen über Axiome modelliert, die jedoch fünf und mehr verschiedene Domänenkonzepte beinhalten, was nach Anwendung von R4 zu einem Ähnlichkeitswert unter dem Schwellwert führt. Auch R3 findet kein Ergebnis, wenn die Definition eines Konzeptes aus zu vielen verschiedenen Subkonzepten besteht. Durch die Ähnlichkeitsberechnung (E3) wird der Nenner durch die hohe Anzahl sehr groß und dadurch der Ähnlichkeitswert sehr klein, sodass er insgesamt unter dem Schwellwert liegt. Bei R13 kommt noch hinzu, dass in einigen Fällen keine Distanz zwischen zwei Konzepten berechnet werden konnte, da in der Ontologie keine Verbindung zwischen ihnen existiert, und sim_{R13} dann auf 0 gesetzt wurde. In einigen Fällen findet auch die Matchregel für gleiche Pfade (R11) keine strukturellen Matches, u.a. auch weil es in den flachen Hierarchien der relationalen operativen Datenmodelle nur drei Ebenen im Pfad gibt (Schema, Tabelle und Spalte) und R11 bei operativen XML-Schemas wohl mehr Erfolg hätte.

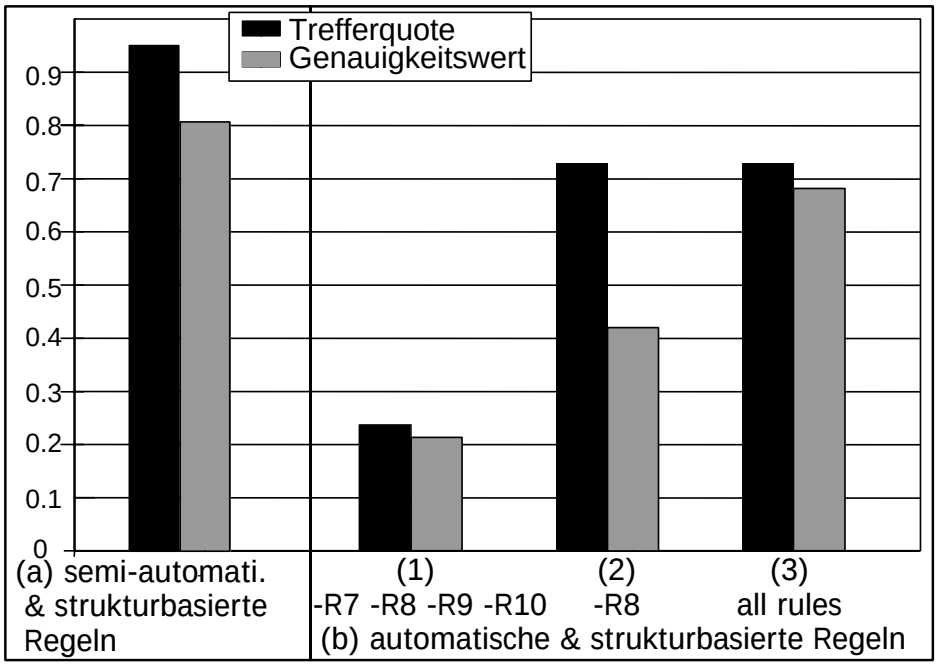


Abb. 4.38: Evaluierung der Matchregeln

Die meisten annotierten Pairingregeln, die bei dem Szenario zu einem Matcherfolg führten, waren die Regeln R1 und R2. Abb. 4.39 stellt dies in (a) dar und zeigt, wel-

che Regeln mit welcher Häufigkeit bei der Bestimmung der korrekten Matches in diesem Experiment angewendet werden konnten. Das Ergebnis ist natürlich abhängig von der Häufigkeit und der Art der semantischen Beziehung der Konzepte, die in der Ontologie modelliert wurden, und ebenso davon, welche Konzepte für die Annotationen verwendet wurden. Im Evaluierungsbeispiel bestand der Hauptanteil aus gleichen oder synonymen Konzepten, die für die Annotation der zu matchenden Prozesselemente und operativen Elemente herangezogen wurden. R5 spielt eher eine untergeordnete Rolle, da nur wenige Annotationskonzepte als Unionkonzept modelliert wurden.

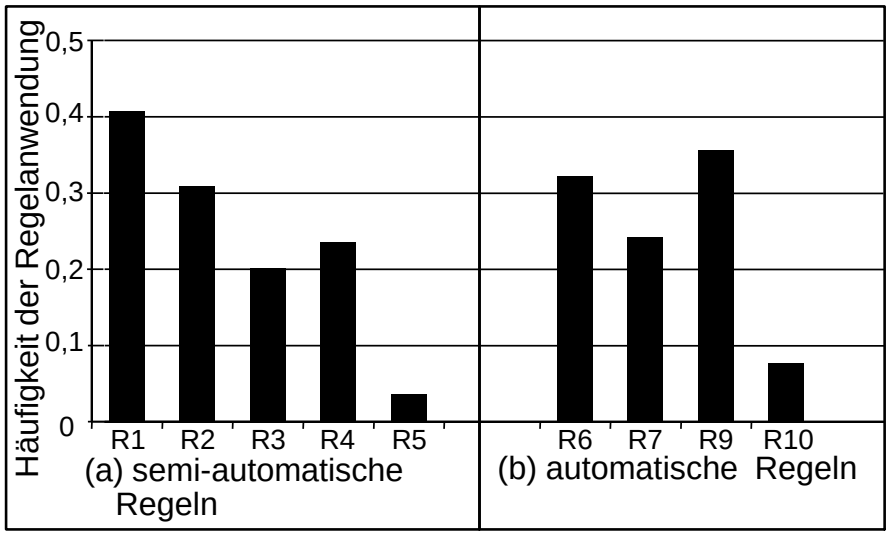


Abb. 4.39: Anteil ausschlaggebender Pairingregeln an korrekt bestimmten Matches

Das Ergebnis für die Genauigkeit des semi-automatischen Matchvorgangs in Abb. 4.38 (a) ist fast genauso gut. Einen positiven Einfluss auf die Genauigkeitsrate haben die Filterregeln, da die Pairingregeln zu viele Matches finden, darunter auch viele falsche. So sind z.B. *CarSelection.inputData.input.CustomerData.CustComplexType.custID* und *HumanResourceSchema.Employee.EID* mit dem gleichen Konzept *Identifier* annotiert und werden deshalb durch R1 gefunden. Sie werden aber in der Ergebnismenge durch die Filterregeln wieder eliminiert. Dabei werden immer alle Regeln ausgeführt, da alle Elemente annotiert sind und daher die einzigen Vorbedingungen für R13 erfüllt sind. Aber es wurden trotzdem auch einige falsche Matches in der Ergebnismenge behalten. Aufgrund der fehlenden Überprüfung durch Instanzdaten finden R3, R4 und R5 Matches, die nicht mit konkreten Datenwerten validiert wurden. So kann es vorkommen, dass in dem Prozessmodell nicht die Subkonzepte, Axiomkonzepte oder Unionkonzepte mit dem operativen Annotationskonzept gematcht werden sollten, da in dem operativen Schema zusätzliche passendere Attribute existieren und nur diese als Matchpartner korrekt wären. Eine anschließende Cleansingphase, die genau die Instanzdaten vergleicht, könnte das Ergebnis verbessern. So können auch Filterregeln in einzelnen Fällen die zusätzlichen falschen Matches nicht eliminieren, da die Variablen bzw. die Prozessstruktur oder die Datentypen der operativen Schemastruktur entspricht. Auch bei Ergebnissen durch R1

und R2 kann das z.B. bei Fremdschlüsseln auftreten, die dann fälschlicherweise mit dem gleichen Prozesselement wie der Primärschlüssel der Tabelle gematcht werden, da die Filterregeln die gleiche operative Struktur vorfinden.

Das zweite Experiment adressiert insbesondere die Bedeutung der Regeln R8 bis R10, die das Prozesswissen ausnutzen. Hier werden die automatischen Regeln und Filterregeln auf partiell-annotierte Elemente angewendet. Abb. 4.38 (b) zeigt die Qualität der Matchergebnisse, die durch diese Regeln erreicht wurde anhand von drei verschiedenen Szenarien: (1) zeigt die Ergebnisse, die nur durch R6 und die Filterregeln R11 und R14 erreicht wurden (also durch alle automatischen Regeln ohne R7, R8, R9 und R10). Dies bedeutet, das Szenario (1) ähnelt in gewisser Weise den Standard-Matching-Verfahren, auch wenn diese noch andere Pairingregeln bzw. Filterregeln berücksichtigen. Szenario (2) zeigt die Matchergebnisse, die mit den Regeln R6, R7, R9, R10a-R10c und den Filterregeln erreicht wurden, und in dem letzten Szenario (3) werden alle automatischen Matchregeln für partiell-annotierte Modelle, also auch noch R8, angewendet. Diese Auftrennung zeigt den Einfluss der neuen Regeln.

Wenn man die Ergebnisse der Trefferquote von (1) und (2) miteinander vergleicht, sieht man eine große Verbesserung in der Trefferquote. In Szenario (3) bleibt die Trefferquote gleich wie in (2), da R8 streng genommen nur für die Filterung der Matches zuständig ist. Dass die Trefferquote in (2) ansteigt, liegt insbesondere an der Regel R9. Abb. 4.39 (b) zeigt, welche partiell-annotierten Pairingregeln bei der Bestimmung der korrekten Matches am häufigsten zu diesem Treffer führten. R9 gibt z.B. Matches an Prozesselemente weiter, bei denen R6 aufgrund anderer Schreibweise oder eines fehlenden Lexikons für Synonyme keinen Match fand. Auch R7 findet zusätzlich Matches für Elemente, bei denen wenigstens das Annotationskonzept eines Prozesselementes die gleiche Schreibweise besitzt wie ein operatives Element. Mit R10 könnten im gleichen Ausmaß wie mit R9 Matchergebnisse gefunden werden, allerdings sind in den verwendeten Prozessmodellen nur wenige Variablen in einem <*property*>-Element modelliert worden, sodass R10 hier nur begrenzt angewendet wurde. Die linguistisch-basierten Matchregeln R6, R7, R10b, R10c, R11 und R12 zeigen ein paar Probleme in (2) und (3), da sie nur Tokens mit klaren Hinweisen zur Aufsplittung im Wort finden, z.B. durch Unterstriche oder Großbuchstaben, und kein Lexikon verwendet wurde. Die fehlenden 26% bei der Trefferquote betreffen also Prozesselemente, für die aufgrund ihrer Namensgebung bei R6 und R7 oder aufgrund ihrer Verwendung in komplexen Assign-Aktivitäten bei R9 keine Matchpartner gefunden wurden.

Von (1) zu (2) steigt der Genauigkeitswert. Dies resultiert jedoch hauptsächlich davon, dass die Ausschussmenge der zuviel gefundenen Matches zwar konstant bleibt, aber die Pairingregeln mehr richtige Matches finden, sodass der Wert des Quotienten steigt. In (3) steigt der Genauigkeitswert wegen der Anwendung von R8 an. Falsche Kombinationen mit Elementen von Variablen aus Human Tasks, z.B. die *Titel* der Human Tasks, werden hier aus der Ergebnismenge gelöscht. Ein paar andere falsche Matches stammen von Tokens, die sehr häufig in Variablen der Autovermietungsdomäne auftauchen, z.B. das englische Token *Car*. Das führt zu falschen Kombinationen von Elementen, die dieses Token neben anderen Tokens beinhalten, z.B. *CarInsuranceID* und *CarStationID*. Ein

Workaround könnte es sein, das Token *Car* und andere häufig vorkommende Worte als besondere Wörter zu deklarieren, die gesondert für den Matchvorgang analysiert werden müssen. Zudem filtern auch hier wie zuvor in Abb. 4.38 (a) die Filterregeln manche falschen Matches aufgrund der gleichen Struktur nicht heraus.

Wenn die Elemente der fehlenden Matches in Abb. 4.38 (b3) entsprechend semantisch annotiert wären und man dann alle Pairing- und Filterregeln anwendet, würde man eine Trefferquote von fast 100% erreichen. Die Genauigkeit würde jedoch immer noch bei 68% liegen, da sich an der Anwendung der Filterregeln nichts ändert. Ein solches Szenario würde die Annotation von genau diesen fehlenden Elementen voraussetzen. Aufgrund dieser recht strikten Vorbedingung ist es nicht in Abb. 4.38 aufgenommen worden.

4.9.3 Ergebnisse zur Bewertung des Ähnlichkeitswertes

Falsch berechnete Ähnlichkeitswerte haben auch Auswirkungen auf die Genauigkeitsrate und die Trefferquote. Das Framework findet zwar korrekte Matches, aber schließt diese in einigen Fällen aus der Ergebnismenge wegen einem niedrigen, aber falsch berechneten Ähnlichkeitswert aus. Deshalb wird im Folgenden die ganze Ergebnismenge aller Matches im Hinblick auf ihren Ähnlichkeitswert und ihre Genauigkeit evaluiert. Abb. 4.40 (a) zeigt die Genauigkeit aller Matches, die durch Anwendung der semi-automatischen Matchregeln gefunden wurden, ohne sie durch den Schwellwert zu eliminieren (also $\frac{|C|}{|T'|}$), sortiert nach ihren Ähnlichkeitswerten. Die Trefferquote ist bei dieser Evaluierung uninteressant, da nur die Qualität des Ähnlichkeitswertes von den durch die Matchregeln gefundenen Matches evaluiert werden kann. Abb. 4.40 (b) illustriert die gleichen Berechnungen für die automatischen Regeln. Die Ergebnisse der Genauigkeitswerte unter 0.7 sind sehr niedrig in beiden Abbildungen. Für den Ähnlichkeitswert 0.2 werden von den annotierten Regeln z.B. 25 Matches gefunden, von denen jedoch nur 2 Stück korrekte Matches wären. Daraus ergibt sich für den Ähnlichkeitswert 0.2 der Genauigkeitswert 0.08 in Abb. 4.40 (a). Im Gegensatz dazu sind die meisten Matches, die von dem Framework mit einem Ähnlichkeitswert über 0.7 entdeckt wurden, korrekte Kombinationen (hohe Genauigkeit). Das bestätigt die Einstellung des Schwellwerts des Frameworks auf 0.7. So vermeidet man, dass zu viele falsche Matches anerkannt werden und insgesamt die Genauigkeitsrate zu niedrig wird. Unter anderem hängt die korrekte Einstellung des Schwellwertes von der Komplexität der Regeln ab. Weitere Betrachtungen zeigten, dass der Schwellwert auf einen niedrigeren Wert gesetzt werden sollte, wenn die Matches z.B. hauptsächlich durch die Matchregel R3 abgeleitet werden und dabei komplexe Subkonzeptbeziehungen der Elemente beachtet werden müssen.

4.9.4 Vergleich zu Standard-Schema-Matching-Verfahren

Werkzeuge für reines Schema-Matching würden nur auf einem kleinen Teil der Variablenmenge des Prozesses arbeiten können, nämlich auf dem Teil, der durch ein XML-Schema

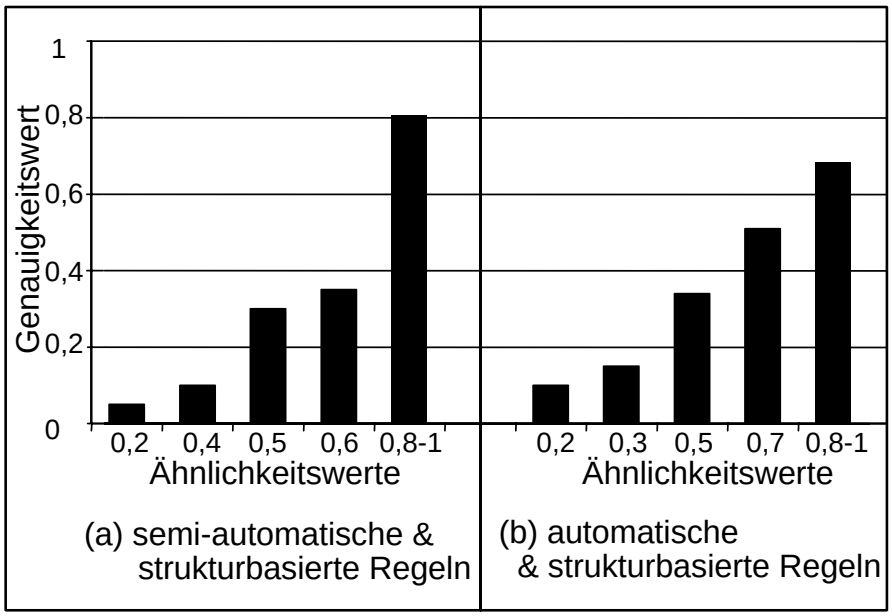


Abb. 4.40: Evaluierung der Ähnlichkeitswerte

definiert ist. Sie würden für die Prozesse und operativen Elemente keine Matches finden, wenn wir sie auf die rohen Prozessdaten anwenden würden, ohne vorher die XML-Variablen aus dem BPEL-Prozess zu extrahieren, wie dies in BIA gemacht wird. Nur dann wäre es für Standardverfahren wahrscheinlich möglich, Matches auf Basis von Annotationen wie M_1 zwischen *CustID* und *CID* aus Anhang C zu finden, wenn sie in der Lage sind, auf den semantischen Annotationen zu arbeiten. Matches auf Grundlage von Axiomen (R4) oder der Prozessstruktur (R9, R10, R12, R13) wie z.B. M_{12}, M_{13} oder M_{14} würden jedoch auch dann nicht gefunden werden, da benutzerdefinierte Axiome in Ontologien oder der Prozesskontext kein Bestandteil von Schema-Matching-Verfahren ist.

Dagegen würde sicherlich der falsche Match M_{22} für die Human-Task-Variable *title* in der Ergebnismenge der Standardtechniken sein, da der Name *Titel* für sie kein Grund ist, den Match zu eliminieren. Die BIA-Matchtechniken eliminieren den Match dagegen, da ein Match zwischen technischen Variablen mit den operativen Elementen keinen Sinn macht.

4.10 Zusammenfassung

Die in diesem Kapitel behandelten Matchverfahren dienen dazu, Korrelationen zwischen den Elementen der Prozessvariablen und der operativen Schemas aus anderen Anwendungen im Unternehmen zu finden. Das Framework unterstützt manuelle, semi-automatische und vollautomatische Verfahren, um die Matches zu finden. Bei den semi-automatischen Verfahren müssen die Elemente vorher annotiert worden sein, während bei den vollautomatischen Techniken die Matches ohne Unterstützung des Benutzers gefunden werden

sollten. Das manuelle Verfahren erfordert dagegen die genaue Eingabe (bzw. Anklicken) der zu matchenden Elemente im Editor.

Im Fokus dieses Kapitels stand die Präsentation der semi- und der vollautomatischen Verfahren und der Matchregeln, mit denen diese Verfahren realisiert wurden: Die Pairingregeln und die Filterregeln. Die Pairingregeln unterstützen die Kombination der mit einer Ontologie annotierten Elemente und der partiell-annotierten bzw. nicht annotierten Elemente. Die Filterregeln reduzieren die Ergebnisse nach dem Pairingschritt um falsche Matches oder bestätigen das Ergebnis durch Erhöhung des Ähnlichkeitswertes. Das Endergebnis ist eine Menge aus Tripeln ($PV \times S \times sim$), bestehend aus jeweils einem Variablenelement PV aus einem Prozess, einem operativen Schemaelement S und ihrem Ähnlichkeitswert sim. Die Ähnlichkeitswerte werden durch jede ausgeführte Regel berechnet und der höchste Ähnlichkeitswert der Pairingregeln (sim_P) gemäß der Kontrollstrategie für BIA mit dem Wert der Filterregeln (sim_{str}) für das Tripel zusammengesetzt ($sim_P * w_{weight} + sim_{str} * (1 - w_{weight})$ mit w_{weight} als festgelegte Gewichtung). Nur die Matches mit einem Ähnlichkeitswert über einem vom Benutzer festgesetzten Schwellwert werden in die Ergebnismenge übernommen. Die Tabelle 4.1 gibt einen Überblick darüber, durch welche Matchregeln sich die vorgestellten Matchverfahren charakterisieren, welchen Ähnlichkeitswert sie berechnen, ob die Elemente, auf die die Regeln angewendet werden, annotiert sein müssen und zu welchem Zeitpunkt in der Matchphase sie ausführbar sind:

- *annotierte Pairingregeln*: Diese Regeln sind nur auf annotierten Elementen ausführbar (++ der Spalte *Ont* in Tabelle 4.1). Sie kombinieren Elemente, deren Annotationen auf das gleiche Konzept (R1) einer Ontologie verweisen oder auch auf synonyme Konzepte (R2), Subkonzepte (R3), Domänenkonzepte unterschiedlicher Atome eines Axioms (R4) oder Konzepte, von denen eines als Vereinigung durch das andere Konzept definiert ist (R5). Gematchte Elemente mit R1, R2 und R5 erhalten den Ähnlichkeitswert 1. R3 bzw. R4 berechnen den Anteil der zwei Annotationskonzepte an der gesamten Konzeptanzahl in der Definition des Subkonzeptes bzw. in dem Axiom.

  Außer R1 werden alle Regeln verwendet, um vor dem Matchprozess alle Konzepte gemäß ihrer semantischen Beziehung miteinander in Konzeptgruppen zu verschachteln. R2 und R5 vereinigen die Konzepte in eine Gruppe, während R3 und R4 die Konzeptgruppen ineinander verschachteln und mit ihrem berechneten Ähnlichkeitswert versehen. Auf diese Weise ist es möglich, alle semantischen Beziehungen bei der Erstellung der Matches der Elemente zu berücksichtigen, die mit Konzepten aus dem gleichen (verschachtelten) Gruppenblock annotiert sind. Die Regeln werden zuerst, also in Phase 1, ausgeführt.

- *partiell-annotierte Pairingregeln*: Diese Regeln kombinieren Elemente, deren Namen gleich sind (zumindest einiger Tokens) (R6) oder von denen bei einem der Name dem Namen des Annotationskonzepts des anderen Elements gleicht (R7). Außerdem vergleichen sie die Namen der <*property*>-Elemente mit den Namen der Schemaelemente (R10b) bzw. den Namen des Annotationskonzepts der Schemaelemente (R10c) und matcht das Schemaelement dann mit den Elementen der Prozessvariablen, wenn diese in dem jeweiligen <*property*>-Element verwendet wurden. Weitere Regeln kopieren Matchpartner aus bereits existierenden Matches auf Prozesselemente, die über Assign-

Aktivitäten (R9) oder <*property*>-Definitionen (R10a) mit dem Prozesselement aus den existierenden Matches verbunden sind. Prozesselemente, die sich in der Eliminierungsliste befinden, werden nicht gematcht (R8). Nur bei den Regeln R7 und R10c ist es notwendig, dass mindestens ein Element annotiert ist. Und auch nur diese Regeln arbeiten auf den Konzeptgruppen, die zuvor erstellt wurden, um alle weiteren semantischen Beziehungen des gematchten Elements zu berücksichtigen. Bei den anderen Regeln müssen die Elemente nicht annotiert sein. In Phase 2 werden die Regeln R6, R7, R10b, R10c ausgeführt, während R9 und R10a erst nach Erstellung der Matches in Phase 3 ausgeführt werden können. R8 ist ein Sonderfall und wird noch vor allen Regeln in Phase 0 ausgeführt.

- *strukturbasierte Filterregeln*: Diese Regeln betrachten gefundene Matches, die im Pairingschritt erstellt wurden, in Bezug auf ihren Kontext und ihre Struktur. Matches werden nur in der Ergebnismenge behalten, wenn ihre Elemente eine ähnliche Struktur besitzen (R11), die Namen des Schemaelements in den Komponentennamen des Kontrollflusses auftauchen (R12) oder die gematchten Elemente als ähnlicher Datentyp definiert sind (R14). Bei annotierten Elementen wird zudem geprüft, ob eine semantische Beziehung der annotierten Kontrollflusselemente zu den Elternkonzepten der Schemaelemente besteht (R13). Die Regeln werden in der Filterphase (F) nach der Pairingphase 2 und noch vor der Pairingphase 3 durchgeführt.

Die Kontrollstrategie des Frameworks setzt diese Regeln um und berücksichtigt dabei Abhängigkeiten zwischen den Regeln. Durch die Erstellung von Konzeptgruppen wird erreicht, dass alle semantischen Beziehungen der Elemente in den Matchprozess einfließen. Die Ausführung der Regeln in verschiedenen Phasen ermöglicht zudem das effektive Auffinden der meisten Matches. Ebenso ist gewährleistet, dass die Algorithmen bei der Umsetzung der Matchregeln in endlicher Zeit die Ergebnismenge erstellen. Im Matcheditor des BIA-Frameworks sind die Matchverfahren prototypisch umgesetzt.

Zuletzt wurde in diesem Kapitel die Effektivität der Matchregeln untersucht. Dazu wurden Variablenelemente aus verschiedenen Prozessen teilweise mit Ontologiekonzepten annotiert, mit den verschiedenen operativen Schemaelementen gematcht und das Ergebnis analysiert. Die wichtigsten Resultate dieser Evaluierung kann man wie folgt zusammenfassen:

- Die guten Ergebnisse für die Trefferquote und für die Genauigkeitswerte der Experimente bestätigen die Effektivität der Regeln zur Matcherstellung.
- Wenn man die Ergebnisse im Abb. 4.38 (a) und Abb. 4.38 (b3) mit den Ergebnissen in Abb. 4.38 (b1) vergleicht, die hauptsächlich durch die bekannte Matchregel R6 (linguistische Matchregel) gefunden wurden, zeigt sich eine große Lücke zwischen den Werten für die Qualität der Ergebnisse. Das bekräftigt die Aussage, dass die neuen Matchregeln die Trefferquote und die Genauigkeit der Matchergebnisse erhöhen. Diese neuen Regeln fokussieren sich auf semantische Annotationen (Abb. 4.38 (a)) und auf prozessspezifische Aspekte (Eliminierung, Datenfluss, Korrelation in Abb. 4.38 (b3)), die von den bekannten Schema-Matching-Ansätzen nicht berücksichtigt werden.

- Durch die Experimente konnte ebenso gezeigt werden, dass die Einstellung des Frameworks bei einer Schwelle von 0.7 für den Ähnlichkeitswert sinnvoll ist. Werte unter dieser Schwelle haben laut Abb. 4.40 eine zu niedrige Genauigkeit.

Abschließend kann festgehalten werden, dass das BIA-Framework einen wertvollen Beitrag bei der (semi-)automatischen Kombination der Prozesselemente mit den operativen Schemaelementen leistet. Falsche oder fehlende Matchtripel können auf einfache und benutzerfreundliche Weise manuell über den Editor korrigiert werden.

Regel	Regelname	Ähnlichkeitswert	*Ont*	Phase
R1	gleiche Annotationen	1	++	1
R2	Synonym	1	++	1
R3	Subkonzept	$\frac{\#c_b \text{ in } ReducedHierarchy \text{ } AB}{\#Konzepte \text{ in } ReducedHierarchy}$	++	1
R4	ont.-interne Axiome	$\frac{\#c_a \text{ und } \#c_b \text{ in den } Domains \text{ der } Atome \text{ des } Axioms \text{ } a_t}{\#Atome \text{ in } Axiom \text{ } a_t}$	++	1
R5	Union	1	++	1
R6	linguistisch	$\frac{2 * \sum_{t1 \in P_i.pv_j.node_m(label)} max_{t2 \in S_s.node_r(label)}[sim_{lex}(t1,t2)]}{\#token(P_i.pv_j.node_m(label)) + \#token(S_s.node_r(label))}$	0	2
R7	ling. für Konzepte	$\frac{2 * \sum_{t1 \in c_a(label)} max_{t2 \in S_s.node_r(label)}[sim_{lex}(t1,t2)]}{\#token(c_a(label)) + \#token(S_s.node_r(label))}$	+	2
R8	Eliminierung	0	0	0
R9	Datenfluss	*sim aus Match in COM*	0	3
R10a	Korrelationen	*sim aus Match in COM*	0	3
R10b	Korrelationen	*via R6*	0	2
R10c	Korrelationen	*via R7*	+	2
R11	gleiche Pfadstruktur	$\frac{\sum_{a=1}^{pathlength(S_s.node_r)} max_{b=1}^{pathlength(P_i.pv_j.node_m)}[sim_{rules}(parent_a(S_s.node_r),parent_b(P_i.pv_j.node_m))]}{pathlength(S_s.node_r)}$	0	F
R12	Prozessstruktur	$\frac{\sum_{a=1}^{\lvert CFP\rvert} max_{b=1}^{pathlength(S_s.node_r)}[sim_{rules}(CFP_a(P_i.pv_j.node_m),parent_b(S_s.node_r))]}{\lvert CFP\rvert}$	0	F
R13	WSDL-Annotationen	$\frac{1}{minDistance(c_p,c_s)+1}$ oder 0 (ohne Annotation oder Verbindung in Ontologie)	++	F
R14	gleicher Datentyp	1 (gleiche Datentypgruppe) oder 0 (ungleicher Datentyp)	0	F

Tabelle 4.1: BIA-Matchregeln

KAPITEL 5

Ganzheitliche Geschäftsanalyse

Nach der Zusammenführung der Informationen aus verschiedenen operativen Datenquellen des Unternehmens mit den Ausführungsdaten aus den Geschäftsprozessen, können die Prozesse tiefer gehend analysiert werden, sodass sie anschließend in umfassendem Maße optimiert werden können. Dieses Kapitel beschäftigt sich mit der Struktur des integrierten BIA-Warehouses und seinen Operatoren, die für die BIA-Analyse verwendet werden können, und ihre Umsetzung im BIA-Framework.

Zunächst wird die Architektur des Warehouses vorgestellt, in das die Informationen integriert werden und auf dessen Basis die ganzheitlichen Analysen ausgeführt werden. Es werden zwei Möglichkeiten herausgestellt, wie die Verbindung zwischen den Prozessdaten und den operativen Daten im Warehouse realisiert werden kann und die Ansätze miteinander verglichen. Abschließend erfolgt die genauere Betrachtung eines Beispielwarehouses, bevor in den folgenden Kapiteln detailliert auf die einzelnen OLAP-Operatoren eingegangen wird, deren Analysemethoden mit Hilfe der Daten aus dem Beispielwarehouse erläutert werden. Der Prototyp für das BIA-Framework, das die Operatoren auf dem Warehouse implementiert und in einem Editor ausführbar macht, wird in einem weiteren Kapitel präsentiert. Kapitel 5.4 schließt mit der Darstellung von Anwendungsfällen der Operatoren im Data Mining für BIA ab.

5.1 Integriertes BIA-Warehouse

Ein integriertes Warehouse sowohl mit Prozessdaten als auch operativen Daten bildet die Grundlage für BIA. Hier werden zwei verschiedene Architekturen für das Warehouse präsentiert und es wird diskutiert, wie sie BIA unterstützen. Den Kern der beiden

Warehousesysteme bilden verschiedene Datenwürfel für die Analyse, von denen jeder eine Faktentabelle besitzt (siehe auch [BG04]). Das Schema der Faktentabellen und ihrer zugehörigen Dimensionen hängen davon ab, welche Attribute für BIA interessant sind. Daher wird nun zuerst ein Überblick über diese Analysedaten gegeben, bevor die Architekturen erläutert werden.

5.1.1 Analysedaten für BIA

Die für BIA interessanten Analysedaten können anhand folgender Kategorien [CCDS07] klassifiziert werden:

- Prozessdaten: Diese Fakten basieren auf den Ablaufdaten eines Prozesses, z.B. der Dauer zwischen Aktivierung und Beenden einer Aktivität oder den Zeitabständen zwischen dem Beenden einer Aktivität und dem Start einer neuen. Außerdem gehört zu diesen Informationen u.a. auch ein Hinweis, ob die Aktivität fehlerfrei beendet oder abgebrochen werden musste.
- Geschäftsobjektdaten: Sie beinhalten alle Werte der Geschäftsdaten, die im Geschäftsprozess verwendet werden.
- Ressourcendaten: Diese Fakten betrachten Daten, die sich auf menschliche und maschinelle Ressourcen beziehen, z.B. ihre Leistungsfähigkeit bei der Ausführung von Aktivitäten bezüglich Schnelligkeit und Erfolg der Abwicklung. Sie werden im BIA-Warehouse wie die Geschäftsobjektdaten behandelt, da bisher nur Informationen in BIA verwendet werden, die in Prozessvariablen gespeichert sind.
- Operative Daten: Sie betrachten die Werte der operativen Daten, die zum Teil bestimmte Prozessattribute weiter gehend beschreiben.

Diese Daten sollen für BIA gemeinsam hinsichtlich ihres Einflusses auf die Prozessausführung analysiert werden. Deshalb sollten sie in der Faktentabelle einer integrierten Warehouse-Architektur gespeichert werden, die im nächsten Kapitel beschrieben ist.

5.1.2 Architektur des BIA-Warehouses

Sowohl die operativen Datenquellen als auch die Prozessdatenquellen enthalten riesige Mengen von operativen Daten (operative Transaktionsdaten und Stammdaten) bzw. Prozessausführungsdaten, die sich jedoch beide nur in vergleichsweise wenig Attributen überschneiden. Für eine ganzheitliche Analyse verwendet das BIA-Framework daher eine föderierte Warehouse-Architektur basierend auf einem föderierten Datenbanksystem [SL90, Rah94], wie sie in Abb. 5.1 dargestellt ist und lässt alle Daten in seinen Quellsystemen, anstatt ein konsolidiertes Warehouse aufzubauen.

Die Datenquellen der operativen Daten und der Prozessdaten sind dieser Föderationsarchitektur über Standard-Wrapper zugänglich, deren Aufbau vom Datenbanksystem des Audit Trails und von der Speicherung der operativen Daten abhängt.

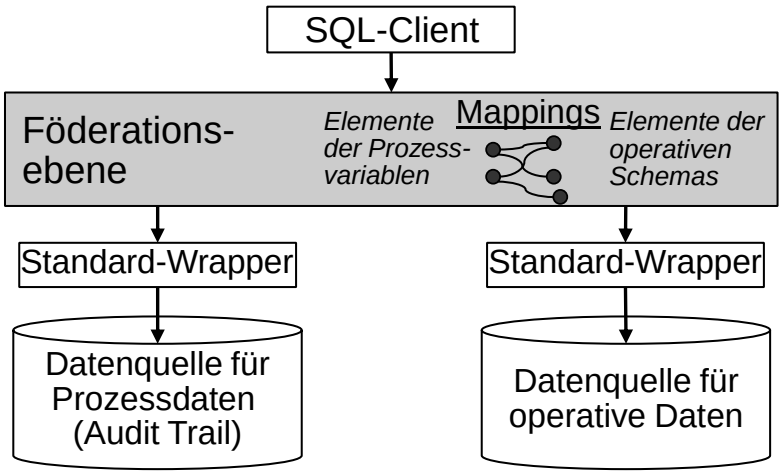


Abb. 5.1: Föderatives BIA-Warehouse

Beide Datenquellen können dann von dem Analysten über einen SQL-Client für BIA verwendet werden. Die Föderationsebene ist verantwortlich für die Integration der Datenquellen und enthält die Mappings, die von den Matches resultieren, welche in der Matchphase zwischen den Prozessvariablen des Audit Trails und den operativen Schemaelementen identifiziert wurden. Diese Mappings sind die einzigen integrierbaren Elemente zwischen den Prozessdaten und den operativen Daten, weil der Audit Trail in erster Linie Attribute speichert, die während der Ausführung der Prozesse relevant sind, z.B. Zeitstempel, die kein Gegenstück in den operativen Daten besitzen. Im Folgenden werden zwei Varianten vorgestellt, wie die Mappings gespeichert werden: via einer Matchtabelle, die alle Matches enthält, oder via einer Bridgetabelle für jeden Match zwischen einem Prozessattribut und einem operativen Schemaattribut, die alle Instanzdaten des Matches enthält. Zunächst wird jedoch ein allgemeiner Überblick über die Architektur eines typischen Datenwürfels auf Grundlage dieses BIA-Warehouses hinsichtlich der diskutierten Analysedaten gegeben.

Abb. 5.2 zeigt einen allgemeinen konzeptionellen Ausschnitt eines BIA-Datenwürfels. Der Würfel ist sehr allgemein gehalten, um darzustellen, dass er auf verschiedene Analysesituationen anwendbar ist, d.h. auf verschiedene Geschäftsprozesse und verschiedene operative Datenmodelle, und dass sich mit ihm verschiedene Fakten für BIA analysieren lassen. Abb. 5.2 zeigt die wichtigsten Komponenten. Die Fakten in der Faktentabelle p_{fact} beschreiben die Analysedaten, die im vorherigen Kapitel diskutiert wurden. Die Fakten sind durch die Prozessdimensionen p_x, p_y, p_n, p_m, usw. beschrieben, da die Analyse der Prozesse im Vordergrund steht. Die Dimensionen enthalten die Prozessausführungsdaten und Daten über die verwendeten Prozessmodelle. Im Gegensatz zu Warehouses nur mit Ausführungsdaten sind im BIA-Warehouse zusätzlich operative Subdimensionen o_n bis o_m hinzugefügt. Sie reichern die Prozessdimensionen p_n bis p_m mit Informationen aus den operativen Daten an. Jedes Attribut in einer Prozessdimension kann durch keine, genau eine oder mehrere operative Subdimensionen beschrieben sein. Diese Korrelationen sind in Abb. 5.2 nur als Pfeile dargestellt, weil sie für jede Variable in jedem Prozessmodell

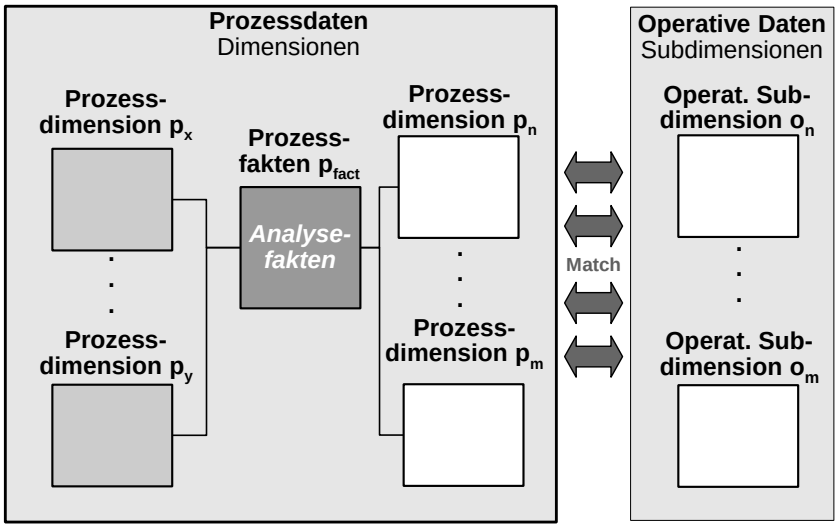


Abb. 5.2: Konzeptionelle Sicht auf einen BIA-Würfel im BIA-Warehouse

anders aussehen. Für jedes matchende Prozessattribut sieht das Schema der operativen Zieltabellen und die Anzahl der Korrelationen zu diesen Zieltabellen unterschiedlich aus.

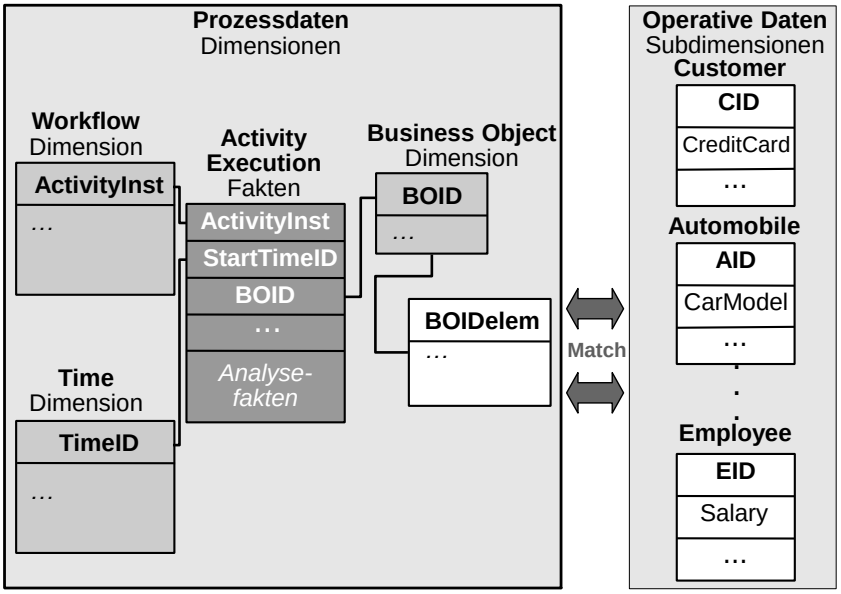


Abb. 5.3: Detaillierte konzeptionelle Sicht auf einen BIA-Würfel im BIA-Warehouse

Ein detaillierter konzeptioneller Ausschnitt eines Beispielwürfels ist in Abb. 5.3 dargestellt. Hier stehen die Fakten über Aktivitätsausführungen für die Analyse im Mittelpunkt. Die Fakten zu den Aktivitätsausführungen werden durch drei Prozessdimensionen beschrieben: *Workflow*, *Time* und *Business Objects*. Die Workflow-Dimension speichert die Daten über eine Aktivität, ihre InstanzID (*ActivityInst*), ProzessID und weitere Details über die Aktivität und den Prozess, z.B. den Namen, ausgeführte Version auf dem

Prozessserver, usw. In der Zeit-Dimension werden die Startzeitpunkte der ausgeführten Aktivitäten gespeichert und in Zeiteinheiten wie Tage, Monate, Jahre usw. aufgefächert. In der Geschäftsobjekt-Dimension werden die Workflow-Variablen und ihre Elemente aufgelistet, die bei der Aktivitätsausführung bearbeitet werden oder Informationen zu Ressourcen beinhalten. Das BIA-Warehouse referenziert diese durch die künstlichen Schlüssel *BOIDelem* und *BOID* (Business Object Identifier). Sobald in einer neuen Aktivität im gleichen Prozess sich der Wert eines gleichen Variablenelementes ändert, wird sie unter einer neuen *BOIDelem*-Identifikationsnummer gespeichert.

Die Elemente aus Geschäftsobjekten können durch Attribute aus operativen Subdimensionen zusätzlich beschrieben werden. Abb. 5.3 zeigt drei operative Tabellen: *Customer*, *Employee* und *Automobile*. Sie enthalten Attribute wie z.B. Gehalt der Angestellten (*salary*) oder Informationen über die Automobilklassen (*CarModel*). Die Korrelationen zwischen den operativen Attributen und den Prozessdimensionen stammen aus der Matchphase. In Abb. 5.3 könnten z.B. die Elemente eines Geschäftsobjekts aus einer Aktivität Attribute in den Subdimensionstabellen *Customer*, *Automobile* oder *Employee* referenzieren. Für jede Aktivität in einem Prozessmodell mit seinen Variablen werden zugehörige operative Subdimensionen zu dem BIA-Würfel hinzugefügt und dann für die Analyse verwendet. In dem föderativen Warehouse sind diese Mappings in der Föderationsebene gespeichert. Da jede Variable mit einem anderen operativen Attribut eines evtl. anderen Datentyps gematcht ist, ist diese Integration komplex. Im Folgenden sind zwei Ansätze dargestellt, um die Mappings in der Föderationsebene zu realisieren. Beide Ansätze verwenden eindeutige Kurznamen, die im föderativen System definiert werden müssen. Damit wird hier der Zugriff auf externe Daten benannt, den ein föderatives System anbietet. Hier werden Kurznamen verwendet, um auf die Prozesstabellen und operativen Schematabellen in den Quelldaten zugreifen zu können.

Integration über eine Matchtabelle

Abb. 5.4 zeigt einen Ansatz, wie die Föderationsebene im BIA-Warehouse modelliert werden kann. Er beschreibt die Integration auf Schemaebene. Die Architektur verwendet genau eine Matchtabelle, um die Prozessdimensionen mit den operativen Subdimensionen zu verbinden. Diese Matchtabelle resultiert aus der Matchphase und wird ins Föderationssystem übernommen.

Die Matchtabelle enthält eine genaue Definition der Korrelationen zwischen den Dimensionen. Sie speichert das Mapping zwischen einem Prozessattribut eines Geschäftsobjektelements und dem operativen Schemaattribut, das das gleiche reale Objekt beschreibt. Das Element in der Prozessvariable wird hier durch den Kurznamen auf seinen Namen (*Var-nickname*) und seinen Variablennamen (*varElem-nickname*) identifiziert und auf die Identifikation und Version seines Prozesses (*process-nickname*). In einer weiteren Spalte befindet sich der Pfad (*path*) von der Variable zu dem Element, was bei Variablen mit mehreren gleichen Elementnamen zur Auflösung des Matches wichtig ist (siehe auch Definition 1 einer Variable in Kapitel 2.1.1). Das matchende operative Attribut wird durch die letzte Spalte der Matchtabelle beschrieben. Sie speichert den

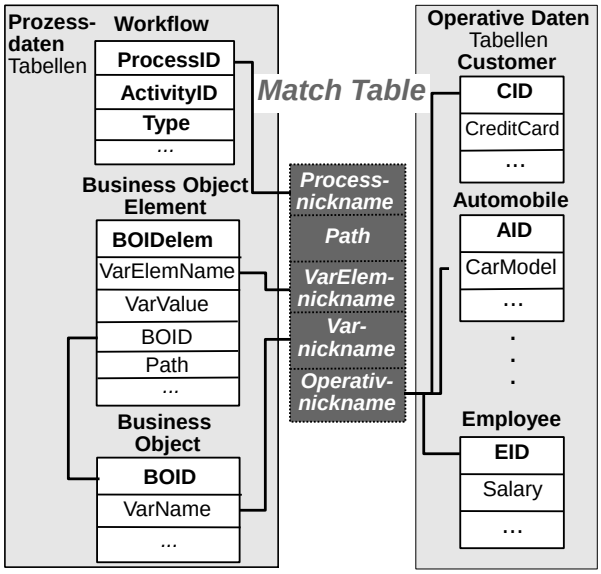


Abb. 5.4: Integriertes Warehouse mit Matchtabelle

Kurznamen *Operativ – Nickname* des operativen Attributs der zugehörigen operativen Tabelle eines bestimmten Datenbankschemas und Datenbankservers.

Integration über Bridgetabellen

Ein anderer Ansatz für die Integration von Prozessdaten und operativen Daten wird in Abb. 5.5 dargestellt. Diese Architektur löst die Matchtabelle auf und erzeugt für jeden Match, also für jedes Tupel aus der Matchtabelle, eine eigene Bridgetabelle. Demzufolge können auch die zugehörigen Datenbelegungen in den Bridgetabellen gespeichert werden und die Integration findet somit auf Instanzdatenebene statt.

Um eine Bridgetabelle zu erzeugen, geht das BIA-Framework folgendermaßen vor: Für jeden Match, der in der Matchtabelle der Matchphase aufgelistet ist, bestimmt das BIA-Framework alle Tupel aus der Element-Tabelle für Geschäftsobjekte, die Variablenelemente mit dem gleichen Namen (*VarElemName*) beinhalten. Davon nimmt das Framework nur die Tupel, die sich auf die gleiche *ProcessID* beziehen, die in der Matchtabelle beschrieben ist. Das wird dadurch erreicht, indem die Faktentabelle mit der Workflow-Tabelle und der Tabelle *BusinessObject* und *BusinessObjectElement* gejoint werden. Von den erhaltenen Tupeln werden nur diese weiterverwendet, die den gleichen Wert in der Tabelle *BusinessObjectElement* haben wie in dem gematchten Attribut der operativen Tabellen.

Die Namen der Bridgetabellen tragen den Namen der ProzessID, der Variable und des Variablenelements aus dem Match (spielt erst eine Rolle bei der späteren OLAP-Analyse (vgl. Kapitel 5.2)), durchnummeriert nach dem Auftreten des ursprünglichen Matches in der Matchtabelle, da es mehrere Variablen mit dem gleich benannten Element im Prozess geben kann, die sich nur durch ihren Pfad unterscheiden. Das Attribut *path* ist in der Tabelle *BusinessObjectElement* gespeichert. Einfacher für die späteren Operatoren wäre es

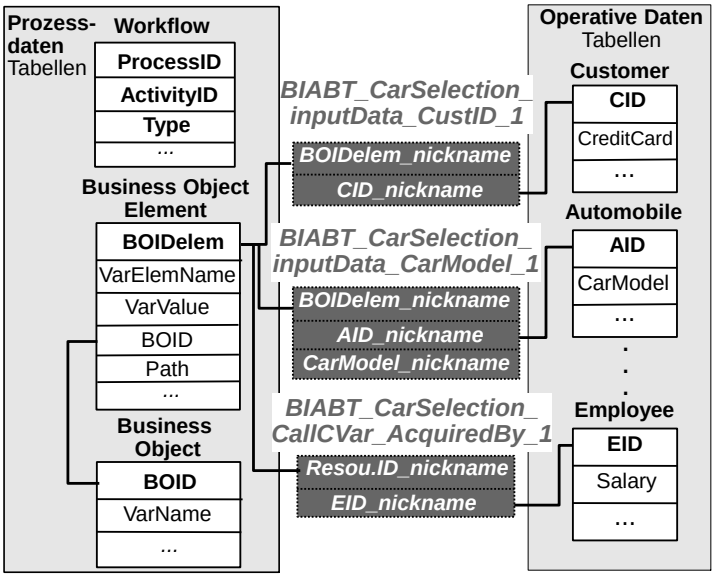


Abb. 5.5: Integriertes Warehouse mit Bridgetabellen

natürlich, den Pfad auch als Teil des Namens der Bridgetabelle aufzunehmen. Bei langen Pfaden könnte jedoch der Platz, der bei Tabellennamen z.B. im IBM Federation Server auf 128 Character beschränkt ist, nicht ausreichen [IBMa]. Entsprechend dieser Architekturbeschreibung wird eine neue Bridgetabelle *BIABT_CarSelection_inputData_CustId_1* in Abb. 5.5 dargestellt für einen Match zwischen dem Element *CustID* der Variable *InputData* und *CID* der operativen Tabelle *Customer* aus dem Beispielszenario aus Abb. 1.3. Eine andere Bridgetabelle *CarModel_1* adressiert den Match zwischen dem Element *CarModel* der Variable *inputData* und der Spalte *CarModel* der Tabelle *Automobile*. Da das Attribut *CarModel* kein Primärschlüssel in der Tabelle *Automobile* ist, benötigt die Bridgetabelle *BIABT_CarSelection_inputData_CarModel_1* das operative Attribut *AID* als Fremdschlüssel und zusätzlichen Primärschlüssel, um alle Tupel mit dieser *CarModel* der Tabelle *Automobile* eindeutig zu identifizieren. Die Bridgetabelle *AcquiredBy_1* stellt den Match der ausführenden Ressource mit der Spalte *EID* der *Employee*-Tabelle dar.

Die Attribute in allen Bridgetabellen sind Kurznamen auf die Quelltabellen im Föderationssystem, um die Attribute in den Quelldaten zu identifizieren. Zuletzt sind noch standardisierte Cleansing-Verfahren nötig, wenn die gematchten Attribute z.B. unterschiedliche Formate besitzen. Beispiele mit Instanzdaten für diese Warehouse-Architektur können in Kapitel 5.1.4 betrachtet werden.

5.1.3 Vergleich der Warehouse-Architekturen

Eine Übersicht beider Architekturen aus Abb. 5.4 und Abb. 5.5 kann in Tabelle 5.1 betrachtet werden. Dieses Kapitel evaluiert die Schemas der beiden Warehouses und verschiedene Kriterien, um die Entscheidung für eine der beiden Architekturen zu erleichtern. Besonders achtet es dabei auf ihre Benutzbarkeit und gibt eine Einschätzung der benötigten Anfragezeit für Analysen und des benötigten Speicherplatzes. Da beide

Warehouse-Architekturen Vor- und Nachteile aufweisen, muss man sich für die Architektur entscheiden, die am besten den eigenen Anforderungen entspricht.

Matchtabelle	**Bridgetabellen**
Integration auf Schemaebene	Integration auf Instanzebene
keine Schemaanpassung, sondern reines Einfügen der neuen Matches	Schemaanpassung mit neuen Bridgetabellen für neue Matches
ein Tupel pro Match	eine Tabelle pro Match
Cleansing während der Analysephase	Cleansing während ETL

Tabelle 5.1: Vergleich der Warehouse-Architekturen

Evaluation im Hinblick auf den ETL-Prozess

Ein positiver Aspekt des Warehouses mit einer Matchtabelle ist wie in Tabelle 5.1 beschrieben, dass es nur eine Tabelle enthält, nämlich die Matchtabelle. Daraus ergeben sich zwei Vorteile. Zum einen ist der ETL-Prozess in der Warehousingphase dadurch sehr einfach, weil er einfach nur die Matches aus der Matchphase in die Föderationsebene des Warehouses kopieren muss. Es ist dabei keine Schemaanpassung nötig. Zum anderen benötigt das BIA-Warehouse nur zusätzlichen Speicherplatz, der von der Anzahl der gefundenen Matches (n) abhängt, da jedes Tupel in der Matchtabelle nur einen Match zwischen den Elementen der Modelle enthält. Dieser Speicherplatz kann durch O(n) abgeschätzt werden. Der Match zwischen den zugehörigen Instanzdaten wird nicht gespeichert, sondern erst zur Anfragezeit bestimmt.

Im Gegensatz dazu ist es für das Warehouse mit Bridgetabellen erforderlich, alle Matches auf Instanzebene zu extrahieren und in die Bridgetabellen zu laden. Für jeden Match wird eine eigene Bridgetabelle erzeugt, die die Prozess- und operativen Schematabellen miteinander verknüpft und dabei erhebliche Speicherplatzanforderungen erzeugt. Aufgrund der riesigen operativen Datenmengen (beinhaltet u.a. alle ausgeführten Transaktionen) und den riesigen Audit Trails (beinhaltet alle ausgeführten Prozesse) wird der Speicherplatzbedarf sehr hoch. Um eine Vorstellung des benötigten Speicherplatzes zu erhalten, gibt diese Arbeit eine Abschätzung der benötigten Anzahl von Bridgetabellen BT und von ihren gespeicherten Tupel S_{BT}:

Sei
- p die Anzahl der Prozessmodelle
- v die durchschnittliche Anzahl ihrer Variablen
- e die durchschnittliche Anzahl der Variablenelemente
- o die durchschnittliche Anzahl der matchenden operativen Elemente

Dann lässt ich die Anzahl der Bridgetabellen wie folgt berechnen durch $BT = p * v * e * o$.

Sei
- i die Anzahl der Prozessinstanzen
- ot die Anzahl der operativen Tupel für jedes gematchte Variablenelement
Daraus erhält man einen Speicherplatzbedarf für die Anzahl $S_{BT} = p * v * e * i * o * ot$ von Tupeln, die in den Bridgetabellen in der Föderationsebene gespeichert sind.

Dieser beträchtliche Speicherplatzverbrauch könnte unnötig sein, da viele Korrelationen und folglich auch viele der Bridgetabellen nicht für die Analyse verwendet werden, wenn die zugehörige Aktivität nicht optimiert werden soll. Ein anderer Nachteil ist die erforderliche Anpassung der Warehouse-Architektur während des ETL-Prozesses. Anstatt nur die neuen Matches in die Matchtabelle zu laden, muss hier jedes Mal, wenn die erste Version eines neuen Prozessmodells ausgeführt wurde, eine neue Bridgetabelle für das Warehouse erzeugt werden. Daher muss nicht nur das Warehouse, sondern auch der ETL-Prozess dynamisch angepasst werden. Trotz all dieser Nachteile der Warehouse-Architektur mit den Bridgetabellen im Vergleich zu der Architektur mit der Matchtabelle (siehe auch Tabelle 5.2, Zeilen 1 und 2) zeigt der folgende Abschnitt, wie man von dieser Architektur während der Analyse profitieren kann.

	Matchtabelle	Bridgetabelle
Zeitbedarf für ETL	+	-
Speicherplatzbedarf	+	-
Verwendung durch OLAP-Tools	-	+
adhoc SQL-Anfrage (Query i)	-	+
Anfrage mit Operator (Query ii)	0	0
Benutzbarkeit	0	0

Tabelle 5.2: Übersicht der Evaluationen der Warehouse-Architekturen für BIA

Verwendung durch existierende Analyseanwendungen

Die Architektur mit Bridgetabellen kann für Analysen ohne Einschränkungen direkt verwendet werden (Zeile 3 in Tabelle 5.2). Es ist möglich, darauf alle standardisierten OLAP- und Data-Mining-Operationen auszuführen und dabei die Daten aus Prozessen mit den operativen Daten zu kombinieren. In dem Warehouse mit der Matchtabelle muss diese Integration während der Analyse geschehen. Im nächsten Kapitel werden Operatoren für BIA eingeführt, die diese Integration als Teil ihrer Anfrageausführung durchführen können. Wenn wir annehmen, dass die Operatoren die häufigsten Anfragen widerspiegeln, die für eine ganzheitliche Analyse von Prozess- und operativen Schemadaten benötigt werden, dann kann diese Warehouse-Architektur ebenso direkt ohne weitere Anpassungen verwendet werden. Allerdings kann sie nicht von beliebigen bereits existierenden Analyseanwendungen verwendet werden.

Experimentelle Evaluation der Anfrageperformanz

Tabelle 5.3 zeigt den Vergleich der Ausführungszeit bestimmter Anfragen für die zwei verschiedenen Warehouse-Architekturen. Für die Evaluation wurde die Föderationsebene auf dem IBM Infosphere Federation Server Version 9.7 [IBMa] mit einer Puffergröße von 30 KB umgesetzt. Die Puffergröße wurde so klein gewählt, um ein Laden aller Daten in den Puffer zu verhindern. So haben alle Anfragen die gleichen Ausgangsbedingungen. Der Server läuft auf einem Windows XP System mit zwei 2.4 GHz Prozessoren und einem Hauptspeicher von 4 GB. Für die Evaluation wurde das Datenvolumen in den Tabellen, die die Prozessinstanzen und operativen Tupel speichern, unabhängig voneinander von 10k bis zu 10000k (k = 1000) variiert. Die Matchtabelle wurde mit konstant 100 Matches gefüllt. In der Bridgetabellen-Architektur ergab das 100 Bridgetabellen mit einer unterschiedlichen Anzahl von Tupeln. In dieser Evaluation reichte die Anzahl der Matchinstanzen von 10k bis zu 100k Zeilen. Sie hängt von dem Datenvolumen der Prozess- und der operativen Schemadaten ab, d.h. es ist unwahrscheinlich eine hohe Anzahl von operativen Daten und Prozesstupeln zu haben und gleichzeitig eine niedrige Anzahl von Tupeln in den Bridgetabellen. Daraus ergeben sich neun Messpunkte für die Anfragen auf den Warehouse-Architekturen durch die Kombinationsmöglichkeiten, die durch die variable Größe des Datenvolumens der Prozessinstanzen und der operativen Schemainstanzen bestimmt werden. Das Datenvolumen der Matchinstanzen in den Bridgetabellen wird hierbei, wie erwähnt, der Größe dieser zwei Datenmengen angepasst.

Messpunkte	①	②	③	④	⑤	⑥	⑦	⑧	⑨
Datenvolumen für Prozessinstanzen (Anzahl der Tupel)	100k	100k	100k	1000k	1000k	1000k	10000k	10000k	10000k
Datenvolumen für operative Tupel (Anzahl der Tupel)	10k	100k	1000k	10k	100k	1000k	10k	100k	1000k
Datenvolumen für Matchinstanzen (Anzahl der Tupel)	10k	100k	100k	10k	100k	100k	10k	100k	100k
Query i WH MT / WH BT *	2,6	2,1	2	10,3	3,9	2,2	100,1	50	11,3
Query ii WH MT / WH BT *	1,6	1,1	1,4	1,5	1,1	1,5	1,3	1,4	1,3

*Ausführungszeit einer Anfrage auf dem Warehouse mit Matchtabelle geteilt durch Ausführungszeit der Anfrage auf dem Warehouse mit Bridgetabellen

Tabelle 5.3: Vergleich der Ausführungszeit von Anfragen auf den Warehouse-Architekturen

Für die Experimente wurden zwei Klassen von Analysen evaluiert: Analyseanfragen, die als eine Art zufälliges SQL-Statement formuliert wurden (Query i) und Analyseanfragen,

die BIA-Operatoren verwenden (Query ii). Um beide Klassen evaluieren zu können, wurde jeweils eine typische repräsentative Anfrage auf jeder Warehouse-Architektur ausgeführt. In den Experimenten sollten beide BIA-Warehouse-Architekturen miteinander verglichen werden. Daher zeigt Tabelle 5.3 nur das Verhältnis zwischen den Ausführungszeiten der Anfragen auf beiden Architekturen anstatt der konkreten Zeitdauer für die Anfrageausführung. Jeder Tabelleneintrag basiert auf dem Durchschnitt von 10 Messungen für jede Architektur, jede Anfrageklasse und jeden Messpunkt.

Für Query i muss das Schema aller Warehouse-Tabellen wohlbekannt sein. In dem Warehouse mit Bridgetabellen werden für die Anfrage nur zwei Joins zwischen der Bridgetabelle, der referenzierten operativen Tabelle und der Tabelle Geschäftsobjektelement benötigt, um alle Tupel für ein Variablenelement und alle Attribute in der operativen Tabelle des zugehörigen operativen Elements zu erhalten (vgl. Abb. 5.5). Da jede operative Tabelle die gleiche Anzahl von Tupeln in unserem Beispiel hat, ist es unwichtig, welcher Match evaluiert wird. In dieser Evaluation wird Tabelle *Automobile* herausgegriffen mit dem Variablenelement *CarSelection.NewInputData.ChangeDate.payload.SelectionInfoType.CarModel* und deshalb hier die Bridgetabelle *CarModel_1*. Einfachheitshalber sind hier alle Kurznamen für die Quellattribute gleich benannt wie ihre Originale. In der Matchphase wurde der Match pro Variable nur einmal pro Prozess erstellt. Die Instanzen für die gleiche Prozessvariable ändern sich für jede Aktivität eines Prozesses üblicherweise. Durch den Natural-Join werden die korrekten Geschäftsobjektelemente der betreffenden Aktivitäten mit den gleichen Werten wie die operativen Attributswerte herausgefiltert, und es spielt bei Query i keine Rolle, welcher Aktivität sie angehören. Query i lautet also für die Architektur mit Bridgetabellen:

```
select a.aid, a.CarModel, a.color, a.plate
from BusinessObjectElement e
     natural join BIABT_CarSelection_NewInputData_CarModel_1 b
     natural join automobile a
```

In der Architektur mit der Matchtabelle muss Query i zusätzlich einen Join mit den Tabellen *BusinessObject*, *Workflow* und *ActivityExecution* enthalten, um die matchenden Instanzdaten zu erhalten (vgl. Abb. 5.3 und 5.4):

```
select a.aid, a.CarModel, a.color, a.plate
from BusinessObjectElement e
     natural join BusinessObject b
     natural join ActivityExecution f
     natural join workflow p
     natural join automobile a
where a.CarModel = e.value
     and e.VarElemName = 'CarModel'
     and p.processId = 'CarSelection'
     and b.VarName = 'NewInputData'
```

and E.PATH = 'CHANGEDATE.PAYLOAD.SELECTIONINFOTYPE'

Wie man in der Tabelle 5.3 in der Zeile für die Evaluation für Query i entnehmen kann, braucht die Anfrage in der Architektur mit der Matchtabelle immer länger als in der Lösung mit den Bridgetabellen. Wenn das Datenvolumen in den Prozesstabellen viel höher ist als in den operativen Tabellen und den Bridgetabellen, ist dieser Unterschied besonders groß (Ergebnisse in Spalten 4, 7, 8, 9). Dieses Ergebnis war zu erwarten, da Query i bei dem Warehouse mit der Matchtabelle drei zusätzliche Joins zwischen den riesigen Prozesstabellen (*BusinessObject*, *ActivityExecution*, *Workflow*) benötigt. Es würde sogar noch länger dauern, wenn Cleansingschritte notwendig wären, da in dieser Architektur das Cleansing Teil der Analysephase ist. Bei diesen Messungen liegen die Daten jedoch schon bereinigt vor und Cleansing ist nicht nötig. Aus den Messergebnissen ergibt sich, dass zufällige Anfragen in der Lösung mit der Matchtabelle in Tabelle 5.2, Zeile 4 weniger positiv bewertet werden als in dem Warehouse mit den Bridgetabellen.

Die Operatoren im nächsten Kapitel unterstützen die Analyse für die Prozessoptimierung. Ein typischer Repräsentant für Query ii ist ein Operator, der die Prozessseite analysiert und die zugehörigen operativen Daten ausgibt, da es das Ziel der Evaluation ist, die zwei Warehouse-Architekturen mit ihren verschiedenen Mappings zu vergleichen. In Query ii suchen die Operatoren die operativen Attribute für das gegebene Prozessvariablenelement *CarModel* einer bestimmten Aktivität (*CarHandOver* in Query ii). Dabei ist dem Benutzer nicht bekannt, dass das zugehörige operative Element auch *CarModel* heißt. Der Operator erhält das Variablenelement *CarModel* als Eingabe und ebenso den Wert der *ActivityID* für ihre Aktivität und den Wert der *ProcessID* für ihren Prozess. Der Operator bestimmt alle operativen Attribute, die in der operativen Tabelle für das matchende Prozessattribut aus der Matchtabelle vorhanden sind. Die Ergebnistupel sind gleich bei der Ausführung auf der Architektur mit der Matchtabelle und mit den Bridgetabellen. Dies ist hier der Fall, da die Datenwerte nicht durch Cleansingschritte verändert wurden. In beiden Architekturen muss der Operator in Query ii die gleichen Prozesstabellen und operativen Tabellen joinen, da der Operator immer bei den Prozesstabellen beginnt und nach den korrekten Variableninstanzen sucht, bevor er die matchenden operativen Tabellen mit dem matchenden Attribut und den gleichen Attributswerten herausfinden kann, die gejoint werden müssen. Deshalb ist die Ausführungszeit für Query ii in beiden Architekturen ähnlich und wird in Tabelle 5.2, Zeile 5 als 0 dargestellt. Die Ergebnisse von Query ii in Tabelle 5.3 werden detailliert in Kapitel 5.2.4 nach der Vorstellung der BIA-Operatoren erläutert.

Diskussion über die Benutzbarkeit der Architekturen

Ein anderer Aspekt bei dem Vergleich der zwei Warehouse-Architekturen ist ihre Benutzbarkeit durch den Analysten. Die Bridgetabellen sind besser verwendbar, um ad-hoc SQL-Anfragen über die Mappings zwischen den Prozessdaten und operativen Schemadaten zu erstellen, da sie auf Bridgetabellen mit einer gewohnten Tabellenstruktur aus dem Bereich des Warehousing basieren. Die Anfragen können so mit Hilfe von Standardwissen über SQL formuliert werden. Im Gegensatz dazu enthält eine Warehouse-Architektur mit

einer Matchtabelle viel weniger Tabellen aufgrund der Tatsache, dass alle Matches in dieser einen Matchtabelle gespeichert sind, anstatt in einer eigenen Bridgetabelle pro Match. Das macht es für den Analysten viel einfacher, den Überblick über die verfügbaren Tabellen zu behalten. Insgesamt balancieren sich die Vor- und Nachteile beider Architekturen im Hinblick auf ihre Benutzbarkeit aus (vgl. Tabelle 5.2, Zeile 6). Letzten Endes muss der Analyst selbst entscheiden, welche Aspekte besser seine Anforderungen erfüllen.

5.1.4 BIA-Warehouse des Beispielszenarios

Der Beispieldatenwürfel, mit dessen Hilfe im nächsten Kapitel auch die BIA-Operatoren illustriert werden sollen, basiert auf dem Beispielszenario und dem Prozessmodel aus Abb. 1.3. Er kombiniert Prozessdaten des Autovermietungsprozesses mit zugehörigen operativen Daten und wird hier in einem Warehouse mit Bridgetabellen gespeichert.

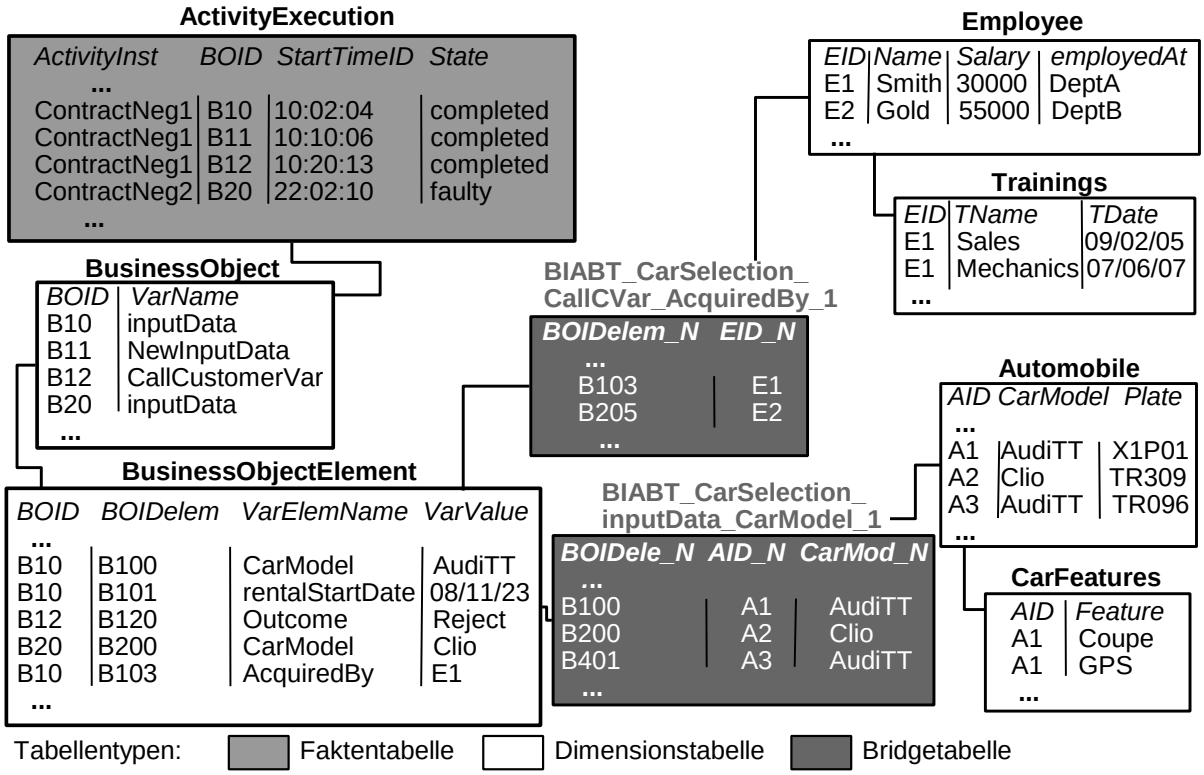


Abb. 5.6: Beispielszenario im BIA-Warehouse

Abb. 5.6 zeigt einen Ausschnitt aus diesem Warehouse, gefüllt mit Beispieldaten. Die Faktentabelle enthält alle Ausführungen verschiedener Aktivitäten, z.B. der Aktivität *ContractNegotiation*. Die Variablen und ihre Details sind in den Tabellen *BusinessObject* und *BusinessObjectElement* gespeichert. Die Bridgetabelle *BIABT_CarSelection_inputData_CarModel_1*, kurz *CarModel_1*, verbindet das Element *CarModel*, das u.a. in der Aktivität *ContractNegotiation* in dem Autovermietungsprozess

CarSelection verwendet wird, mit der Spalte *CarModel* der operativen Tabelle *Automobile* und die Bridgetabelle *BIABT_CarSelection_CallCustomerVar_AcquiredBy_1*, kurz *AcquiredBy_1*, verbindet die Tabelle *BusinessObjectElement* mit der operativen Tabelle *Employee*. Die operativen Tabellen *Training* und *CarFeatures* stehen mit den Tabellen *Employee* und *Automobile* über Fremdschlüssel in Beziehung. Aus Platzgründen sind die Tabellen *CarFeatures* und *Trainings* nicht entsprechend dem Schema aus Anhang B wiedergegeben, sondern gleich mit *AID* bzw. *EID* aus der Tabelle *CarCharacteristics* bzw. *TrainedEmployee* integriert dargestellt. Weiterhin ist auch die Tabelle *Department* aus dem *HumanResourceSchema* hier nicht dargestellt.

5.2 BIA-Operatoren

Um Geschäftsprozesse auf Basis des BIA-Warehouses zu optimieren, ist es hilfreich eine Unterstützung für OLAP zu haben, die über die gewöhnlichen SQL-Befehle für OLAP wie z.B. ROLLUP oder WINDOW [Iso03, Pet07] hinausgehen. Wie in dem vorherigen Kapitel erläutert wurde, ist die operative Dimension des BIA-Warehouses in mehrere Subdimensionen unterteilt. Um diese Subdimensionen effizient in der OLAP-Analyse und im Data Mining verwenden zu können, werden hier neue Operatoren eingeführt. Wird zudem die Warehouse-Architektur mit der Matchtabelle verwendet, kann die OLAP-Anfrage nicht aus reinem SQL bestehen. Stattdessen muss der BIA-Operator sogar verwendet werden, um über die Matchtabelle die operativen Daten zu gewünschten Prozessvariablen zu extrahieren. Ein weiteres Ziel der Operatoren ist es, häufig zu stellende Anfragen in BIA als eine Art von Makro zu vereinfachen, z.B. als ein Operator, der nach den Aktivitäten mit der durchschnittlich längsten Ausführungszeit oder mit den meisten Fehlern sucht, sodass diese Teile in einer anderen Anfrage benutzt und nicht mehr ausformuliert werden müssen. Die Operatoren helfen dabei, die drei Hauptperformanzindikatoren bei der Ausführung von Geschäftsprozessen zu analysieren: Kosten, Zeit und Qualität der Prozesse (siehe [DvdAtH05]). Ein anderer Zweck der Operatoren ist es, die Warehouse-Dimensionen für die Mining-Algorithmen aufzubereiten, sodass die verschiedenen Techniken auf die Daten angewendet werden können.

Einen ersten Überblick über die BIA-Operatoren gibt folgende Auflistung. Diese Operatoren werden in den nächsten Kapiteln genauer erläutert:

- *BIASUBALL, BIASUBNUM, BIASUBLABEL*: Diese Operatoren geben zu den angegebenen Variablen in Instanzen einer bestimmten Aktivität eines bestimmten Prozesses alle matchenden operativen Attribute und ihre Daten sowie alle weiteren operativen Attribute und ihre Daten in der Tabelle aus, in der sich das gematchte Attribut befindet. Außerdem werden alle operativen Attribute aus Tabellen ausgegeben, zu denen von der gematchten Tabelle eine Fremdschlüsselbeziehung egal in welche Richtung existiert.
- *BIAHT, BIAPL, BIAFLUCT*: Diese Operatoren geben alle Geschäftsobjekte und ihre Werte von Prozessen aus, in denen Human Tasks oder Partnerdienste existieren, die großes Optimierungspotenzial bergen. BIAFLUCT hingegen gibt Geschäftsobjekte

von Prozessinstanzen aus, bei denen Aktivitätsausführungen zeitlichen Schwankungen unterliegen.

- *BIAERROR, BIAEXCEPT*: Diese Operatoren geben alle Geschäftsobjekte und ihre Werte von Prozessinstanzen aus, bei denen ein bestimmer Prozentanteil entweder in einem fehlerhaften Zustand endet oder durch Compensation-Handler abgefangen wird.

Alle BIA-Operatoren werden durch die Syntax in Abb. 5.7 in Backus-Naur-Form definiert. Die Syntax ist in beiden Architekturen (Bridgetabelle vs. Matchtabelle) die gleiche, aber die Operatoren unterscheiden sich in ihrer Ausführung. Die formale Definition der Operatoren findet man im Anhang D.

(a) <from clause> ::= **FROM** <table reference>

(b) <table reference> ::= <table primary> | <joined table> | <bia clause> [**,** <table reference>]

(c) <bia clause> ::= <biasub clause> |
<biaht clause> |
<biapl clause> |
<biafluct clause> |
<biaerror clause> |
<biaexcept clause>

(d) <biasub clause> ::= <biasubAll clause> | <biasubLabel clause> | <biasubNum clause>

(e) <biasubAll clause> ::= **BIASUBALL(** <var> **,** <processid> **,** <activityid> **)**

(f) <biasubLabel clause> ::= **BIASUBLABEL(** <var> **,** <processid> **,** <activityid> **)**

(g) <biasubNum clause> ::= **BIASUBNUM(** <var> **,** <processid> **,** <activityid> **)**

(h) <biaht clause> ::= **BIAHT(** <processid> **)**

(i) <biapl clause> ::= **BIAPL(** <processid> **)**

(j) <biafluct clause> ::= **BIAFLUCT(** <processid> **,** <duration> **,** <percent> **)**

(k) <biaerror clause> ::= **BIAERROR(** <processid> **,** <percent> **)**

(l) <biaexcept clause> ::= **BIAEXCEPT(** <processid> **,** <percent> **)**

Backus-Naur-Form
<symbol> : Nicht-Terminalsymbol **fettg. Symbol** : Terminalsymbol symbol1 | symbol2 : Auswahl
[symbol] : optionales Symbol

Abb. 5.7: Syntax der BIA-Operatoren

Die Operatoren unterscheiden sich hauptsächlich in der Bearbeitung der unterschiedlichen Mappings und darin, dass die Operatoren in dem Warehouse mit der Matchtabelle das Cleansing während der Ausführung durchführen müssen. Daher müssen diese Cleansingschritte automatisch ausführbar sein. Wie es die Syntax in Abb. 5.7 definiert, sind alle BIA-Operatoren eine Komponente der Tabellenreferenz in der <from>-Klausel des jeweiligen Tabellenausdrucks im SQL-Statements (siehe (a) und (b)). Die BIA-Klausel in (c) ist also eine alternative Tabellenreferenz, die wiederum aus sechs alternativen

Klauseln besteht, die die BIA-Operatoren repräsentieren. Die automatische Ausführung der BIA-Operatoren erfordert die strenge Einhaltung der Namensgebung und Fremdschlüsselbeziehungen verschiedener Attribute und Tabellen der Prozessdimensionen. Die Operatoren arbeiten auf einem Schema mit Kurznamen, wie es in Abb. 5.3 gezeigt wird, um die Tabellen *Workflow*, *Time*, *ActivityExecution*, *BusinessObject*, *BusinessObjectElement* und *Resource* zu identifizieren. Zumindest für die Attribute *ProcessID*, *ActivityID* und *ActivityType* in der Tabelle *Workflow* sollten auch Kurznamen definiert sein, außerdem für das Attribut *Name* in der Tabelle *BusinessObject*, für *VarElemName* und *VarValue* in Tabelle *BusinessObjectElement*, für *Starttime* in der Tabelle *Time* und für alle Attribute in der Faktentabelle.

5.2.1 Operatoren für die Analyse der Subdimensionen

Diese erste Gruppe von Operatoren kann verwendet werden, um eine erste Übersicht der zugehörigen operativen Attribute für eine gegebene Aktivität und ihre Variablen zu erhalten. Abhängig von der verwendeten Warehouse-Architektur, evaluieren sie die Matchtabelle (vgl. Abb. 5.4) oder die Bridgetabellen (vgl. Abb. 5.5), um alle zu tätigenden Joins zwischen den operativen Dimensionen und der Dimension des Geschäftsobjekts zu finden. Die Operatoren sind als alternative Klauseln für <biasub clause> in Abb. 5.7 (d) dargestellt. Es wird ein Operator BIASUBALL angeboten, der einfach alle zugehörigen subdimensionalen Attribute zurückgibt. Die Anzahl und der Datentyp der Ergebnisspalten sind dabei abhängig von der matchenden Variable des angegebenen Prozess- und Aktivitätsmodells, auf die der Operator angewendet wird. Des Weiteren werden zwei Operatoren BIASUBLABEL und BIASUBNUM angeboten, um die Ergebnistabelle in Hinblick auf verschiedene Formatbedingungen einzuschränken. Diese Operatoren sind besonders interessant, um den passenden Input für die Mining-Verfahren zu liefern.

BIASUBALL-Operator

Der Operator BIASUBALL (Abb. 5.7 (e)) hat drei Eingabeparameter: <var>, <processid> und <activityid>. BIASUBALL gibt alle Spalten und ihre Werte aus genau den operativen Tabellen aus, die gematchte Attribute zu dem spezifizierten Variablenelement in der spezifizierten Aktivität und Prozess enthalten. Das Variablenelement <var> ist durch einen eindeutigen Elementnamen definiert, <processid> enthält die Kennung des verlangten Prozesses und <activityid> die Kennung der zu untersuchenden Aktivität, welche die gegebenen Variablenelemente bearbeitet.

Der Operator geht folgendermaßen vor: Er wählt diejenigen Tupel aus den Prozessdimensionen aus, die die gegebene ProzessID, ActivityID und Variable enthalten und alle Attribute der zugehörigen operativen Tabellen. Auch alle Attribute derjenigen operativen Tabellen werden ausgegeben, die das matchende operative Attribut durch eine Fremdschlüsselbeziehung referenzieren. Die zugehörigen operativen Attribute erhält der Operator, indem er entweder die Matchtabelle bzw. die Metadaten des Warehouses durchsucht, um nach dem gematchten Variablenelement bzw. den Bridgetabellen zu suchen, die nach

dem zugehörigen Variablenelement benannt sind. Die Ergebnistabelle enthält die Aktivitätsinstanzen und alle zugehörigen operativen Attribute mit ihren Werten. Durch die Analyse der Ergebnistupel kann man neue Optimierungshinweise erhalten.

Beispiel Im Folgenden wird der Operator BIASUBALL dargestellt, wie er auf das Schema aus Abb. 5.6 angewendet werden könnte. Die Anfrage findet Anhaltspunkte dafür heraus, ob und wie die Attribute der Ressource aus dem Variablenelement *AcquiredBy* den Erfolg oder Misserfolg der Ausführung der Aktivität *ContractNegotiation* in dem Autovermietungsprozess bestimmen:

select BIA.*, BOE.VARELEMNAME, BOE.VARVALUE
from *BIASUBALL*(CALLCUSTOMERVAR.TASK.EMPLOYEETASK.ACQUIREDBY,
CARSELECTION, CONTRACTNEGOTIATION) AS BIA, BUSINESSOBJECT BO,
BUSINESSOBJECTELEMENT BOE, ACTIVITYEXECUTION A
where BIA.ACTIVITYINST = A.ACTIVITYINST
and BO.BOID = A.BOID **and** BO.BOID = BOE.BOID
and BO.NAME = 'CALLCUSTOMERVAR'
and BOE.VARELEMNAME = 'OUTCOME'

Alle operativen Attribute zu dem Element *AcquiredBy* in der Aktivität *ContractNegotiation* werden durch BIASUBALL selektiert und mit den Tabellen *ActivityExecution* und *BusinessObject(Element)* gejoint, um auch noch die Werte des Prozessattributs *Outcome* zu erhalten, die das Ergebnis der Aktivität speichern. Wenn diese Anfrage auf dem Beispielwarehouse ausgeführt wird, führt dies zu einer Ergebnistabelle, von der ein Auszug in Tabelle 5.4 dargestellt ist. Attribute aus der Tabelle *Department* fehlen in der Abbildung aus Platzgründen. Auch sind nicht alle Attribute aus *Employee* dargestellt und aus der Tabelle *Trainings* fehlt das Attribut *minimumAttendance*. Diese Tabelle zeigt, dass die Mehrheit der Fälle mit *Outcome = 'reject'* in Beziehung mit dem Training der Agenten steht und dass sie meistens von Angestellten ausgeführt werden, die in Verkaufsstrategien (*Sales*) fortgebildet sind. Hier sieht man das durch bloßes Betrachten der Tabelle, aber

ActivityInst	EID	Name	Salary	employedAt	Tname	Tdate	VarEle.Name	VarValue
ContractNeg1	E1	Smith	30000	DeptA	sales	09\05\05	outcome	reject
ContractNeg1	E1	Smith	30000	DeptA	tooling	07\06\07	outcome	reject
ContractNeg2	E2	Gold	55000	DeptB	leading	15\05\08	outcome	accept
ContractNeg3	E3	Wild	34000	DeptA	-	-	outcome	accept
ContractNeg4	E4	Howly	50000	DeptB	sales	01\06\08	outcome	reject
ContractNeg4	E4	Howly	50000	DeptB	leading	15\05\08	outcome	reject
ContractNeg5	E5	Stix	35000	DeptC	sales	27\10\07	outcome	reject

Tabelle 5.4: Ergebnis des Operators BIASUBALL

gewöhnlicherweise sind weitere Analysen notwendig, wie z.B. Mining-Techniken. Auch für Optimierungen der Aktivität müssen diese Ergebnisse und ihr Zusammenhang mit anderen Aktivitätsinstanzen des Prozesses weiter analysiert werden.

BIASUBNUM und BIASUBLABEL-Operator

Da manche Mining-Verfahren sich auf bestimmte Kategorien von Eingabewerten beschränken, wurden im Rahmen dieser Arbeit zwei zusätzliche Operatoren entwickelt. Der Operator BIASUBLABEL (f) in Abb. 5.7 filtert alle Attribute aus der Ergebnistabelle von BIASUBALL, die einen nominalen Datentyp (string, character,...) haben. Dieser Operator ist besonders für Mining-Verfahren wie z.B. Classification notwendig, wo alle Attribute nach ihren nominalen Kategorien gruppiert werden. In den Fällen, wo eine Umwandlung von numerischen Attributen zu solchen Kategorien nicht möglich oder ihre Betrachtung in der Analyse nicht erwünscht ist, ist dieser Operator sehr wichtig. Seine Syntax ist gleich wie die des Operators BIASUBALL und auch seine Ergebnistupel, wobei nur die nominalen operativen Attribute und der Primärschlüssel der Tabelle des gematchten Attributs ausgegeben werden. Der Operator vergleicht die Datentypen der Attribute mit einer Liste, in der alle nominalen Datentypen aufgelistet werden. Im Gegensatz dazu gibt der Operator BIASUBNUM (g) alle numerischen, operativen Attribute und den Primärschlüssel der Tabelle des gematchten Attributs zurück, indem er die Datentypen mit einer ähnlichen numerischen Liste vergleicht (integer, float, decimal, numeric,...). Das ist notwendig für Mining-Verfahren, wie z.B. Clustering oder Regression, die auf kontinuierlichen Werten arbeiten. Auch die Syntax dieses Operators gleicht dem Operator BIASUBALL.

Beispiel Wenn BIASUBNUM anstatt von BIASUBALL in der vorherigen Beispielanfrage verwendet wird, sieht die Ergebnistabelle ähnlich wie in Tabelle 5.4 aus. Dieses Mal erhalten wir jedoch nur die numerischen Spalten. Die kategorischen Attribute werden aus der operativen Ergebnistabelle eliminiert. Das bedeutet, dass die operativen Spalten der Tabelle *Employee* (in Tabelle 5.4 *Name, employedAt, Tname und Tdate*) ausgeschlossen werden, da ihr Datentyp nicht numerisch ist. Alle anderen Spalten werden angezeigt, da sie numerisch sind (in Tabelle 5.4 *Salary*) oder zu gejointen Prozesstabellen (*ActivityInst*, *VarElemName*, *VarValue*) gehören bzw. operative Primärschlüssel (*EID*) sind. BIASUBLABEL würde analog arbeiten und stattdessen die nominalen Attribute zurückgeben.

5.2.2 Operatoren für die Analyse der Ausführungszeit

Da die Länge der Ausführungszeit von Geschäftsprozessen eine wichtige Rolle bei der Optimierung der Prozesse spielt, bietet das BIA-Framework drei Operatoren an, um die Analyse der Ausführungszeit zu unterstützen: BIAHT, BIAPL und BIAFLUCT. Ihre Syntax ist in Abb. 5.7 (h), (i) und (j) dargestellt.

BIAHT- und BIAPL-Operator

Die Aktivitäten vom Typ *Human Task*, oder allgemeiner vom Typ *invoke*, die auf die Antwort von Partnerservices warten, stellen ein geeignetes Ziel für Optimierungen dar, weil sie oft länger als nötig dauern. Die Operatoren selektieren diese Aktivitätstypen in den gegebenen Prozessausführungen und geben die zugehörigen Tupel zusammen mit ihren Variablen und ihrer Ausführungsdauer zurück. Der BIAHT-Operator (h) sucht nach allen Human Tasks für ein gegebenes Prozessmodell <processid>, d.h. nach Aktivitäten, die den TaskManager aufrufen. Er joint dafür die Faktentabelle mit der Workflow-Tabelle (um alle zugehörigen Aktivitätsinstanzen bzgl. *Type* und *ProcessID* zu selektieren) und mit der Zeit-Tabelle (um die Dauer der Aktivitätsausführungen zu berechnen). Die Workflow- und die Zeit-Tabelle sind nicht in Abb. 5.6 dargestellt, sondern nur in Abb. 5.3 angedeutet.

Der Ansatz für den Operator BIAPL (i) ist ähnlich, allerdings sucht er nach allen Invoke-Aktivitäten im Gegensatz zu dem BIAHT-Operator, der nur nach Invoke-Aktivitäten sucht, die einen Human Task umsetzen.

Beispiel In der folgenden Beispielanfrage wird BIAHT verwendet, um die Ausführungszeit von Human Tasks zu untersuchen, die im Autovermietungsprozess benutzt wurden:

select * **from** *BIAHT*(CarSelection)

Über diese Anfrage erhält man die ausgeführten Aktivitäten (*ActivityInst*) und ihre Dauer (*Duration*) zusammen mit den Geschäftsobjekten des Human Tasks. Wenn man die Anfrage auf die Daten im Beispielwarehouse anwendet, erhält man eine Ergebnistabelle, wie sie ausschnittsweise in Tabelle 5.5 dargestellt ist. Auch hier fehlen aus Platzgründen einige Variablenelemente, die neben den angezeigten Elementen zu den Geschäftsobjekten der angezeigten Aktivitätsinstanzen gehören. Die Tabelle zeigt die Dauer, die über die Startzeiten der Aktivitäten in der Zeit-Tabelle berechnet wurde, und die Attribute *CarModel*, *requestedCar*, *acquiredBy* und *rentalStartDate* der Tabelle der Geschäftsobjektelemente. Wenn man die Ergebnistupel analysiert, könnte man entdecken, dass Prozessinstanzen in der Aktivität *ContractNegotiation* mit dem Wert *AudiTT* für das Element *CarModel* oder *requestedCar* langsamer ablaufen als Instanzen mit anderen Werten für die gleichen Elemente und dass sie oft mehr als 30 Minuten brauchen. Diese Schwankungen in der Ausführungsdauer können mit dem nächsten Operator genauer analysiert werden.

BIAFLUCT-Operator

Große Unterschiede in der Ausführungszeit für die gleiche Aktivität bei verschiedenen Prozessinstanzen geben Hinweise auf Prozessaktivitäten, die eventuell optimiert werden könnten. Der Operator BIAFLUCT (Abb. 5.7 (j)) zielt darauf ab, diese Aktivitäten zu finden. Der Operator sucht nach allen Aktivitäten für einen gegebenen Prozess, ähnlich

ActivityInst	Duration	acquiredBy	CarModel	requestedCar	rentalStartdate
ContractNeg1	000:00:39:020	E1	AudiTT	-	08/11/23
ContractNeg2	000:00:01:060	E2	Clio	-	08/12/25
CarHandOver2	000:00:44:120	E4	-	Clio	-
ContractNeg3	000:00:52:010	E3	AudiTT	-	08/11/30
CarHandOver3	000:00:31:340	E5	-	AudiTT	-

Tabelle 5.5: Ergebnis des Operators BIAHT

wie die zwei vorherigen Operatoren. Er sucht jedoch nach Tupeln mit Aktivitäten, deren Ausführungsdauer um den im Operator angegebenen Wert schwankt. Das bedeutet, dass er prüft, ob der Anteil von <percent> Ausführungen dieser Aktivität in dem angegebenen Wert <duration> schwankt, d.h. ob diese Anzahl der Ausführungen langsamer oder schneller ist als die durchschnittliche Zeit der Instanzen für das gleiche Aktivitätsmodel. Der Wert für die Dauer muss dabei in dem Format 000:00:00:000 (Tage:Stunden:Minuten:Sekunden) angegeben werden.

Beispiel Im Folgenden werden alle Aktivitäten im Autovermietungsprozess *CarSelection* näher untersucht. Die Arbeit zeigt eine Beispielanfrage mit BIAFLUCT, um alle Aktivitäten zu suchen, von denen mindestens 20% länger oder weniger als 30 Minuten dauern als die durchschnittliche Ausführungszeit dieser Aktivität. Der Operator selektiert die Aktivitätsausführungen zusammen mit Informationen über ihre verwendeten Variablenelemente:

select *
from *BIAFLUCT*(CarSelection, 000:00:30:000, 20)

Die Ergebnistabelle ähnelt Tabelle 5.5. Sie zeigt die gleichen Attribute, jedoch nur für die Aktivitäten, von denen mindestens 20% mehr als 30 Minuten von der Durchschnittszeit der Aktivität abweichen. Zudem kann die Tabelle 5.5 neben Human Tasks auch noch andere Aktivitätsinstanzen enthalten, z.B. Instanzen der Aktivität *SearchCar* mit anderen Variablenelementen, da der Operator alle Aktivitätstypen bearbeitet. Man erhält bei diesem Operator auf dem Beispielwarehouse auch wieder Instanzen der Aktivität *ContractNegotiation* als Ergebnis. Durch Verwendung des BIASUBALL-Operators auf die Ergebnistupel, kann der Grund für die längere Verhandlungsdauer bei bestimmten Automodellen festgestellt werden: viele Wahlmöglichkeiten für Extraleistungen bei manchen Automodellen führen zu einer hohen Rate von notwendigen Rückrufen bei den Kunden, um ihre Bestätigungen einzuholen. Für eine Prozessoptimierung könnte man das Prozessmodell ändern und eine zusätzliche Aktivität vor der Aktivität *ContractNegotiation* einfügen, die automatisch zusätzliche Informationen von den Kunden anfordert, die an bestimmten Automodellen interessiert sind.

5.2.3 Operatoren für die Analyse von Fehlern

Ein wichtiges Ziel von Prozessoptimierungen ist das Verhindern von Fehlern während der Prozessausführung. Das Framework bietet zwei Operatoren für die Fehleranalyse der Prozessausführungen an: BIAERROR und BIAEXCEPT.

BIAERROR-Operator

Der Operator BIAERROR (Abb. 5.7 (k)) vergleicht fehlerhafte mit fehlerfreien, vollständig ausgeführten Prozessen und gibt die Tupel von kritischen Aktivitätsausführungen mit ihren zugehörigen Geschäftsobjekten zurück. Der Operator sucht nach allen Aktivitäten für den angegebenen Prozess <processid> und gibt die Aktivitätsinstanzen zurück, welche bei einer bestimmten Anzahl (<percent>) von Ausführungen nicht terminiert haben. Dazu müssen in den Tabellen die korrekten Tupel bzgl. ihrer ProzessID und Zustand der Aktivitätsausführung selektiert werden, bevor die Tabellen *ActivityExecution*, *BusinessObject* und *BusinessObjectElement* mit ihr gejoint werden. Die Ergebnistabelle enthält die zugehörigen Variablen der Aktivität.

Beispiel Im Folgenden werden wieder die Aktivitäten des Autovermietungsprozesses untersucht. Die Beispielanfrage verwendet BIAERROR, um nach allen Aktivitäten zu suchen, die in mehr als 30% der Ausführungen in einem fehlerhaften Zustand enden. Diese Aktivitätsausführungen werden zusammen mit ihren Variablen selektiert:

select * **from** *BIAERROR*(CarSelection, 30)

In dem Ausschnitt aus der Ergebnistabelle in Tabelle 5.6 entdeckt man, dass Instanzen der Aktivität *CarHandOver* sehr oft fehlerhaft sind. Für eine Prozessoptimierung müssen jedoch weitere Analysen durchgeführt werden, um die zugehörigen operativen Attribute zu den gefundenen Variablen und den vorhergehenden Aktivitätsvariablen zu untersuchen.

ActivityInst	State	acquiredBy	CarModel	requestedCar	rentalStartDate
SearchCar7	faulty	-	Clio	-	08/12/09
CarHandOver2	completed	E4	-	Clio	-
CarHandOver3	faulty	E5	-	AudiTT	-
CarHandOver9	faulty	E11	-	MazdaMX5	-

Tabelle 5.6: Ergebnis des Operators BIAERROR

BIAEXCEPT-Operator

Der Operator BIAEXCEPT in Abb. 5.7 (l) betrachtet die Behandlung von Ausnahmen im Prozess genauer, die verwendet wird, um mögliche Fehler während der Prozessausführung abzufangen und zu bearbeiten. Dieser Operator evaluiert die Ausführungsdaten also im Gegensatz zu BIAERROR im Hinblick auf Fehler, die nicht so offensichtlich sind, weil der Prozesszustand korrekt beendet ist. Auch wenn die Fehler abgefangen wurden, signalisiert die Fehlerbehandlung ein Problem im Prozess. Das Problem wurde zwar im Prozess gelöst, aber trotzdem erfordert es in der Analyse eine nähere Betrachtung, um den Prozess zu ändern und später optimal ausführen zu können. Der Operator vergleicht reguläre Prozessausführungen mit solchen, bei denen eine Fehlerbehandlung durch einen Compensation-Handler stattfand. Dies ist im Audit Log gespeichert. Er gibt die Aktivitätsinstanzen mit ihren Variablen zurück.

5.2.4 Bewertung der Operatoren

Diese Arbeit möchte ein besonderes Augenmerk auf den Vergleich der zwei vorgestellten Warehouse-Architekturen legen. Da sich die Architekturen nur in der Art und Weise unterscheiden, wie sie die Mappings zwischen den Prozessdaten und den operativen Daten speichern, werden hier nur die Operatoren evaluiert, die auf diesen Mappings arbeiten. Das sind eigentlich die drei BIASUB-Operatoren. Hier werden jedoch nur Anfragen mit BIASUBALL evaluiert, da sich die anderen zwei Operatoren bis auf die Reduzierung der zurückgegebenen operativen Attribute gleich verhalten. Für die Evaluierungen werden die gleichen Beispieldaten wie in Kapitel 5.1.3 verwendet. Das Ergebnis ist in Tabelle 5.3 in der Zeile Query ii dargestellt. Für beide Architekturen wurde die Anfrage auf die gleiche Weise formuliert:

select *
from *BIASUBALL*(ServiceInfo.request.InfoData.ServiceRequest.requestedCar, CarSelection, CarHandOver)

Die Anfrageergebnisse in den Ausgabetabellen zeigen jede Aktivitätsinstanz der Aktivität *CarHandOver* im Autovermietungsprozess mit dem Wert des Elements *ServiceInfo.request.InfoData.ServiceRequest.requestedCar* und die matchenden operativen Attribute, ebenso wie die anderen Attribute in diesen operativen Tabellen. Wie das Ergebnis des Vergleichs in Tabelle 5.3 zeigt, ist der Unterschied in der Ausführungszeit zwischen den zwei Architekturen nicht sehr hoch, sondern bei den meisten Messungen fast gleich. Das wäre anders, wenn man nicht die Bridgetabellen nach dem Muster *BIABT_ProzessID_Variablenname_Matchelement_Nummer* benennen würde (siehe Kapitel 5.1.2). Folglich muss Query ii nur in den Metadaten des DWH die Namen der Bridgetabellen mit den Namen der geforderten Elemente vergleichen und bei mehreren Bridgetabellen zusätzlich den Wert des betreffenden Attributes *Path* in der Tabelle *BusinessObjectElement* vergleichen, um die korrekten Bridgetabellen für die Joins zu finden. Ansonsten wäre es viel zeitaufwändiger, die richtige Bridgetabelle zu finden.

Insgesamt kann man sagen, dass die Operatoren eine große Hilfe sind, häufige Anfragen an das Warehouse zu formulieren. Wenn der Analyst die matchenden operativen Tabellen und Spalten zu dem Element *ServiceInfo.request.InfoData.ServiceRequest.requestedCar* zudem nicht kennt, ist eine Standardanfrage wie in Kapitel 5.1.3 nicht möglich, weil in beiden BIA-Architekturen ein dynamisches Einfügen der zugehörigen operativen Spalten in die <from>-Klausel der SQL-Anfrage notwendig wäre. Es hängt von dem Optimierungsziel des Unternehmens ab, ob die BIA-Operatoren für die Analyseaufgabe ausreichend sind. Sie dienen jedoch als guter Ausgangspunkt für weitere Analysen. Des Weiteren können die Operatoren im Data Mining für BIA wieder verwendet werden, was im Kapitel 5.4 näher beschrieben ist.

5.3 Prototyp des Analyseeditors

Die beschriebenen Analysen für BIA wurden auf Grundlage beider Warehouse-Architekturen prototypisch in einem Analyseeditor umgesetzt. Dieses Kapitel zeigt seine Architektur und Funktionalität auf.

5.3.1 Architektur des Analyseeditors

Die Architektur des Analyseeditors (siehe Abb. 5.8) gliedert sich im Wesentlichen in drei Teile: Benutzerschnittstelle, Analyse-Engine und der BIAAnalyse-Kern. Über die Benutzerschnittstelle kann der Benutzer die Auswahl der Analyseoperatoren steuern, die von der Analyse-Engine zur Verfügung gestellt werden. Der BIAAnalyse-Kern wiederum ist für die Ausführung der Funktionalitäten verantwortlich, die von der Benutzerschnittstelle zur Verfügung gestellt werden, d.h. er steuert die Anfragen der Analyse-Engine über die Datenbankschnittstelle an das BIA-Warehouse. Im Folgenden werden die drei Komponenten näher ausgeführt.

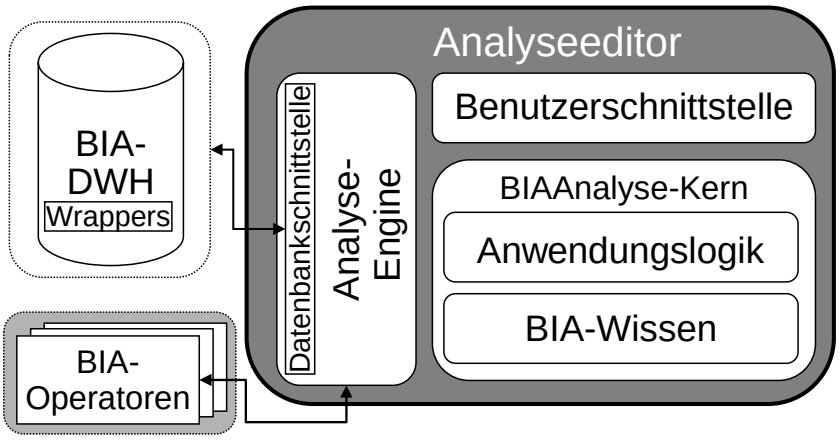


Abb. 5.8: Architektur des Analyseeditors

Benutzerschnittstelle

Über die Benutzerschnittstelle führt der Benutzer die Analyse durch. Ihm stehen folgende Funktionalitäten zur Verfügung, die in Kapitel 5.3.2 noch ausführlich diskutiert werden: Auswahl des BIA-Warehouse für die Analyse, Auswahl des BIA-Operators, Betrachten der Analyseausgabe.

Analyse-Engine

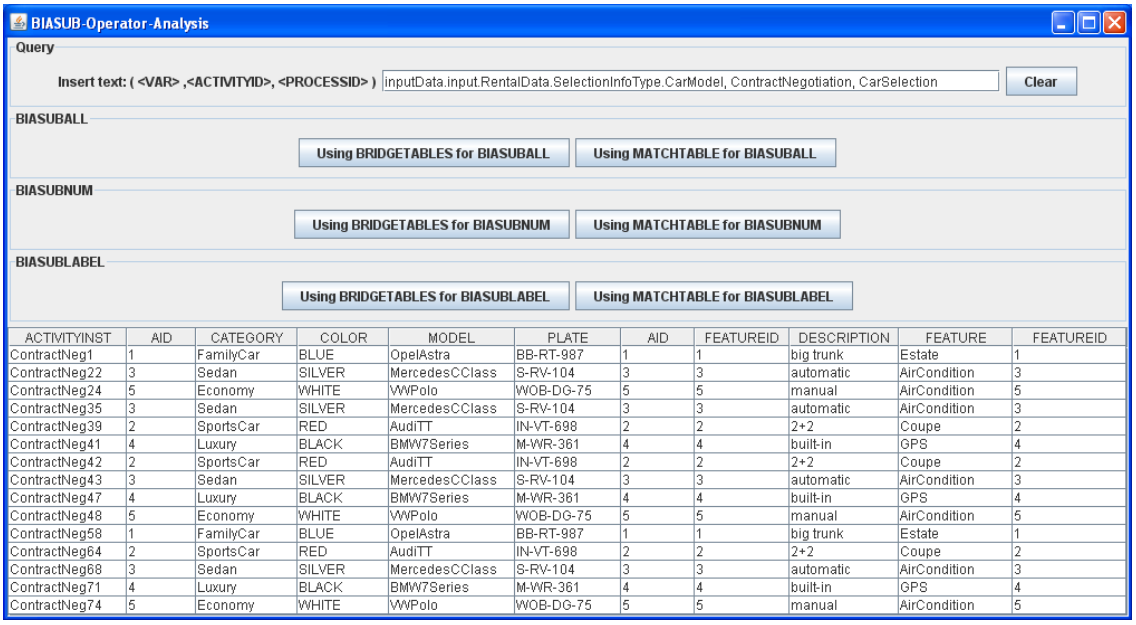
Die Analyse-Engine steuert die Ausführung der BIA-Operatoren. Sie führt die Operatoren gemäß der definierten Strategie aus Kapitel 5.2 aus, um alle Prozessattribute und ggf. ihre zugehörigen operativen Elemente aus dem Warehouse zurückzuliefern, auf welche die Bedingungen in den Operatoren zutreffen. Dazu wird der ausgesuchte Operator durch die dafür definierten SQL-Statements über die Datenbankschnittstelle auf dem Warehouse ausgeführt. Als Datenbankschnittstelle wird hier JDBC verwendet. Die Daten aus den Datenquellen werden über interne Wrapper des föderativen BIA-Warehouses selektiert. Prototypisch wurde es hier mit dem IBM Federation Server Version 9.7 umgesetzt [IBMa]. Das Ergebnis wird an die interne Struktur des Editors zur Darstellung über seine Benutzeroberfläche zurückgegeben. Wenn die Bedingungen der Operatoren auf keine Instanzwerte zutreffen, bleibt die Rückgabetabelle leer.

Kern der BIA-Analyse

Der Kern realisiert die Anwendungslogik des Analyseeditors, um die Operatoren auf der gewählten Warehouse-Architektur auszuführen und das Ergebnis auszugeben. Für die Realisierung steuert der Kern die Analyse-Engine. Um neue Erkenntnisse aus den OLAP-Ergebnissen zu erhalten, muss der Analyst bisher selbst Zusammenhänge in den Ergebnisdaten erkennen. Die Umsetzung von Analysetechniken, insbesondere Mining-Techniken, ist in zukünftigen Forschungsarbeiten [NRM10a] geplant und soll diese Erkennung von BIA-Analysewissen möglichst automatisieren. Wie die OLAP-Operatoren die Mining-Techniken dabei unterstützen können, ist in Kapitel 5.4 diskutiert.

5.3.2 Funktionalität des Analyseeditors

Die Umsetzung der Konzepte für die Analyseoperatoren auf beiden Warehouse-Architekturen ist in dem Analyseeditor realisiert. Abb. 5.9 zeigt das Hauptfenster des Editors. Das zentrale Element des Editors ist die Operatorenliste mit dem freien Feld für die Eingabeparameter, die sich bei jedem Operator etwas unterscheiden. In Abb. 5.9 wurde der Operator BIASUBALL angewendet auf das Variablenelement *InputData.input.RentalData.SelectionInfoType.CarModel* in der Aktivität *ContractNegotiation* im Prozess *CarSelection*. Es wurde auf der Warehouse-Architektur mit der Matchtabelle ausgeführt und ergibt die darunter dargestellte Ergebnistabelle. Hier sieht man in der ersten Spalte alle Aktivitätsinstanzen mit der angeforderten Variable und in den Spalten daneben alle operativen Elemente mit ihren Werten, die mit diesem Variablenelement zusammenhängen.



ACTIVITYINST	AID	CATEGORY	COLOR	MODEL	PLATE	AID	FEATUREID	DESCRIPTION	FEATURE	FEATUREID
ContractNeg1	1	FamilyCar	BLUE	OpelAstra	BB-RT-987	1	1	big trunk	Estate	1
ContractNeg22	3	Sedan	SILVER	MercedesCClass	S-RV-104	3	3	automatic	AirCondition	3
ContractNeg24	5	Economy	WHITE	VWPolo	WOB-DG-75	5	5	manual	AirCondition	5
ContractNeg35	3	Sedan	SILVER	MercedesCClass	S-RV-104	3	3	automatic	AirCondition	3
ContractNeg39	2	SportsCar	RED	AudiTT	IN-VT-698	2	2	2+2	Coupe	2
ContractNeg41	4	Luxury	BLACK	BMW7Series	M-WR-361	4	4	built-in	GPS	4
ContractNeg42	2	SportsCar	RED	AudiTT	IN-VT-698	2	2	2+2	Coupe	2
ContractNeg43	3	Sedan	SILVER	MercedesCClass	S-RV-104	3	3	automatic	AirCondition	3
ContractNeg47	4	Luxury	BLACK	BMW7Series	M-WR-361	4	4	built-in	GPS	4
ContractNeg48	5	Economy	WHITE	VWPolo	WOB-DG-75	5	5	manual	AirCondition	5
ContractNeg58	1	FamilyCar	BLUE	OpelAstra	BB-RT-987	1	1	big trunk	Estate	1
ContractNeg64	2	SportsCar	RED	AudiTT	IN-VT-698	2	2	2+2	Coupe	2
ContractNeg68	3	Sedan	SILVER	MercedesCClass	S-RV-104	3	3	automatic	AirCondition	3
ContractNeg71	4	Luxury	BLACK	BMW7Series	M-WR-361	4	4	built-in	GPS	4
ContractNeg74	5	Economy	WHITE	VWPolo	WOB-DG-75	5	5	manual	AirCondition	5

Abb. 5.9: Screenshot des Analyseeditors

5.3.3 Implementierungsaspekte des Analyseeditors

Der Analyseeditor ist unter Java Version 1.5 implementiert. Zur Erbringung seiner Funktionalität bezüglich der BIA-Operatoren aus Kapitel 5.2 bestimmt das Java-Programm in mehreren Schritten die Attribute und ihre Werte für die Ausgabetabelle der Operatoren, sobald sie durch einen Benutzer aufgerufen werden. Die Anzahl und auch die Namen der Attribute der Ausgabetabelle ändern sich, je nachdem für welches Element einer Prozessvariable oder für welchen Prozess der jeweilige Operator aufgerufen wurde. Aufgrund dieses dynamischen Verhaltens der Rückgabeattribute kann sowohl in einer IBM[1]- als auch in einer Oracle[2]-Datenbank keine benutzerdefinierte Tabellenfunktion (englisch: 'user-defined table function') verwendet werden, da diese eine genaue Angabe der Ausgabedatentypen zum Übersetzungszeitpunkt in der Datenbank verlangen.

Ansätze für eine dynamische Erstellung der Ausgabedatentypen in benutzerdefinierten Tabellenfunktionen existieren in [JM99]. Hier werden mit der Notation TABLENAME.+ Attribute aus der Eingabetabelle TABLENAME in die Ausgabetabelle propagiert. Jedoch können hier die Attribute aus dem Datenbankkatalog entnommen werden, da die Tabelle TABLENAME schon dort registriert ist.

Die BIA-Operatoren setzen die Ausgabetabelle allerdings erst während der Ausführung zusammen, da während der Ausführung erst bestimmt wird, welche operativen Tabellen und Spalten zu dem angegebenen Prozesselement zurückgegeben werden.

[1] http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.sql.ref.doc/doc/c0000873.html
[2] http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28370/create_function.htm

5.4 Anwendungsfälle für das ganzheitliche Mining

Als zusätzliche Option in der Analysephase sind Mining-Techniken für BIA möglich, um versteckte Optimierungsmuster in den Audit Trails und operativen Daten zu entdecken. [RM09] erläutert, wie die üblichen Mining-Verfahren Clustering, Klassifikation, Assoziationsregelerkennung und Regression, die im Detail z.B. in [Han05] beschrieben sind, auf kombinierte Prozess- und operative Subdimensionen in BIA angewendet werden könnten. In diesem Kapitel wird nun gezeigt, wie die BIA-Operatoren die Daten im Warehouse aufbereiten, um die Anwendung dieser Mining-Techniken für BIA zu vereinfachen. Auch diese Beispiele basieren auf dem Autovermietungsszenario aus Abb. 1.3. Dadurch sieht man auch, dass der Informationsgewinn durch BIA steigt, da nun viele weitere Analysen möglich sind. Ausführliche weitere Erläuterungen und Evaluationen in [NMR+11] belegen diesen Mehrwert der ganzheitlichen Geschäftsanalyse bei der Prozessoptimierung.

5.4.1 Clustering

Das Gruppieren von verwandten Objekten in eine bestimmte Klasse von Objekten, die einander ähnlich sind und gleichzeitig möglichst verschieden zu anderen Objektgruppen, nennt man Clustering [JMF99]. Besonders die Clustering-Methoden, die auf eine Vielzahl von Dimensionen angewendet werden können, sind für BIA interessant, da auch BIA eine große Anzahl von Subdimensionen bewältigen muss, die aus der großen Anzahl von Prozessvariablen der verschiedenen Prozessaktivitäten resultieren. Um das Clustering-Verfahren etwas zu beschleunigen, bietet sich in BIA die Verwendung der BIA-Operatoren an, um diejenigen Instanzen herauszufinden, die auf Optimierungsmöglichkeiten im Prozess hinweisen. Hier wird BIAFLUCT verwendet, um alle Aktivitäten aus dem Autovermietungsprozess zu erhalten, die großen Schwankungen in der Zeitdauer unterliegen. So wird man erkennen, dass dabei die Aktivität *SearchCar* sehr häufig auftaucht. Des Weiteren wird der Operator BIASUBNUM verwendet, da das Clustering nur mit numerischen Attributswerten arbeiten kann. Deshalb wird BIASUBNUM auf die Aktivität *SearchCar* und all ihre Variablen angewandt, um die operativen Attribute zu bestimmen und zu filtern. Dabei ist besonders die Variable *CustID* interessant und das Clustern der Instanzen in Bezug auf die subdimensionalen Attribute von *CustID*. Abb. 5.10 illustriert das Clustering innerhalb von zwei Dimensionen: der Ausführungszeit der Instanzen der Aktivität *SearchCar* und dem Attribut *CustomerRanking* (durch den Typ der *CreditCard* bestimmt) der Subdimension *Customer*. Daraus erhält man im Beispiel drei Cluster: *VIPService*, *FastService* und *DelayedVIPService*.

BIA-Clustering ist als ein erster Analyseschritt zur Erkennung der Aktivitäten sinnvoll, die problematisch sind und für eine spätere Optimierung weiter gehend analysiert werden sollten. Hier sollte deshalb das Cluster *DelayedVIPService* genauer untersucht werden, um die Gründe der Verspätung zu erkennen und den Prozess entsprechend zu restrukturieren.

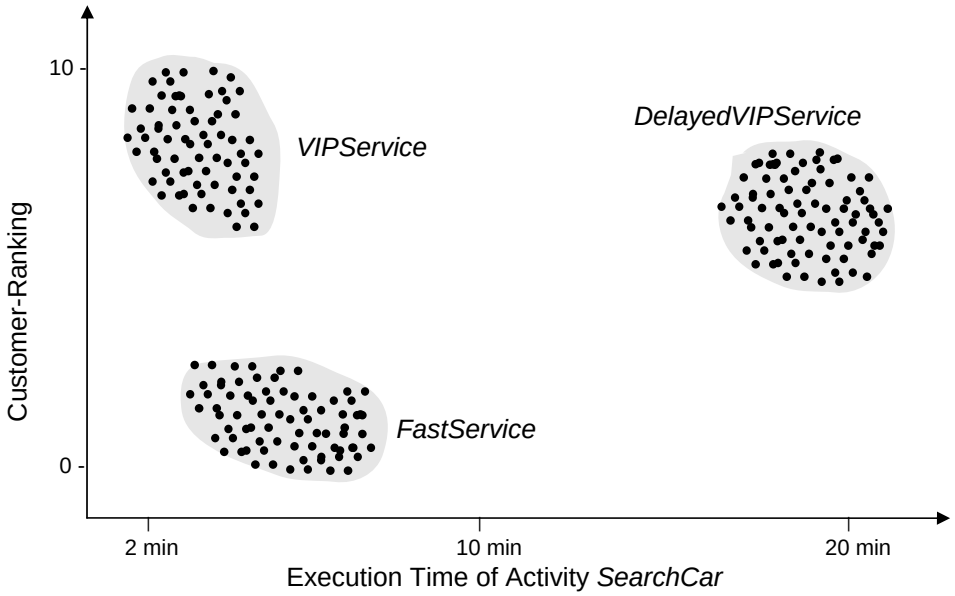


Abb. 5.10: Beispiel für BIA-Clustering

5.4.2 Klassifikation

Die Klassifikation ist eine Analysemethode, die anhand bestimmter Merkmale ein Modell erstellt, um die in Zusammenhang stehenden Informationen in Kategorien einzuteilen [MSTC94]. Für BIA ist es sinnvoll, Geschäftsprozesse in Kategorien einzuteilen, um die Prozesse in den für das Unternehmen negativen Kategorien zu optimieren. In dem folgenden Beispiel sollen die ausgeführten Aktivitäten bezüglich ihres Ausführungszustandes klassifiziert werden. Folglich werden die Daten des Autovermietungsprozesses zunächst mit dem Operator BIAERROR aufbereitet. Dadurch erkennt man, dass *ContractNegotiation* eine fehleranfällige Aktivität ist, welche häufig erfolglos abgebrochen wird. Deshalb wird die Klassifikation aus Performanzgründen auf diese Aktivität beschränkt. Die Instanzen werden mit dem Operator BIASUBLABEL weiter für die Klassifikation vorbereitet, da dadurch alle operativen Attribute und davon nur die nominalen Attribute bestimmt werden. Der Operator wird hier auf der Variablen *AcquiredBy* ausgeführt und bestimmt ihre subdimensionalen nominalen Attribute. Der Klassifikationsalgorithmus arbeitet nun auf den Ausführungsdaten von *ContractNegotiation* und den erhaltenen subdimensionalen Attributen des ausführenden Angestellten. Abb. 5.11 zeigt den Entscheidungsbaum und die Klassen, die nach den Prozesszuständen benannt sind (hier nur die zwei Klassen: korrekt (*complete*) oder fehlerhaft (*faulty*) beendet). Die Partitionierung der Tupel hängt von den Prozessdaten und den operativen Daten ab, die sich auf den Angestellten beziehen, der den Human Task ausführt. Das Attribut *Training* hat den höchsten Informationsgewinn und wird zu dem ersten Splitattribut. Wenn der Angestellte in Kommunikationstechniken (*communication*) fortgebildet ist, wird die Aktivität sofort akzeptiert und der Prozess kann weiterlaufen. Für das Training *Sales* ist eine weitere Auftrennung der Instanzen bezüglich des Attributes *Position* (angestellt als Verkaufsagent (*rental car supplier*) oder als *Mechaniker*) erforderlich. Auf der Seite der fortgebildeten Angestellten

im Bereich Werbung (*advertising*) wird eine weitere Aufteilung über die *Priorität der Ausführung* der Aktivität, die durch den Prozessdesigner spezifiziert wurde, für deren Klassifikation benötigt.

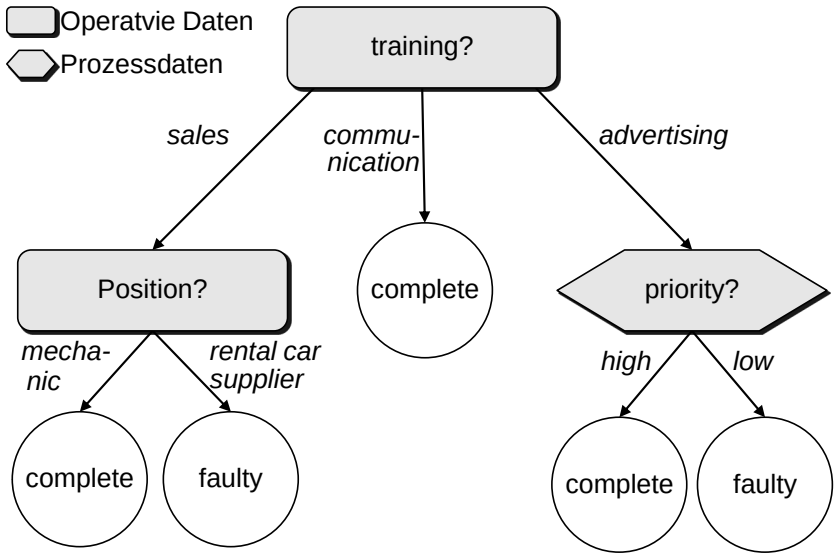


Abb. 5.11: Beispiel für BIA-Klassifikation

Die Klassifikationstechnik kann z.B. benutzt werden, um ein Prozessmodell zu analysieren und dann so zu restrukturieren, dass fehlerhaft abgebrochene Pfade vermieden werden. Nach der Klassifikation kann in diesem Beispiel für eine Prozessoptimierung die Gruppe der verantwortlichen Angestellten für die Aktivität *ContractNegotiation* geändert werden: Nur Verkäufer, die in Kommunikationstechniken fortgebildet sind, oder Mechaniker mit einem Verkaufstraining dürfen den Task annehmen anstatt aller Angestellten aus der Abteilung A, B und C, wie es in Abb. 1.3 dargestellt wurde. Für ein anderes Klassifikationsbeispiel könnte man den Operator BIAEXCEPT verwenden, um bestimmte Aktivitätsausführungen bezüglich ihrer Ausnahmebehandlungen zu klassifizieren. So könnte man herausfinden, ob in den Prozessen ein Fehler auftrat, der abgefangen werden musste und evtl. Optimierungspotenzial birgt.

5.4.3 Regression

Regression ist verwandt mit der Klassifikation. Während die Klassifikation allerdings auf diskreten Ergebnissen arbeitet, sagt der Regressionsalgorithmus kontinuierliche Werte voraus, z.B. durch statistische Methoden [UG99]. Ein Anwendungsgebiet ist eine optimale flexible Zuweisung der Angestellten zu den Human Tasks im Autovermietungsprozess. Hier sollte ihre durchschnittliche Dauer analysiert werden, um Engpässe oder lange Verspätungen zu reduzieren. Die Operatoren BIAHT und BIASUBNUM dienen in diesem Beispiel zur Aufbereitung der Daten. Zuerst findet BIAHT heraus, dass u.a. die Aktivität *CarHandOver* häufig eine lange Ausführungszeit besitzt. Wir nehmen an, dass

das Element *CarModel* tiefer gehend begutachtet werden sollte. Die numerischen Subdimensionen für das Element werden durch BIASUBNUM bestimmt. Über Standard-Regressionsalgorithmen auf der Ergebnistabelle wird die Ausführungsdauer der Aktivitätsinstanzen analysiert, die die Übergabe der Autos an den Kunden modellieren, zusammen mit ihren gematchten operativen Daten. Die Regression könnte dabei wie in Abb. 5.12 zeigen, dass die Dauer von der Anzahl der Extras abhängt, die in dem Modell des Leihautos verfügbar sind.

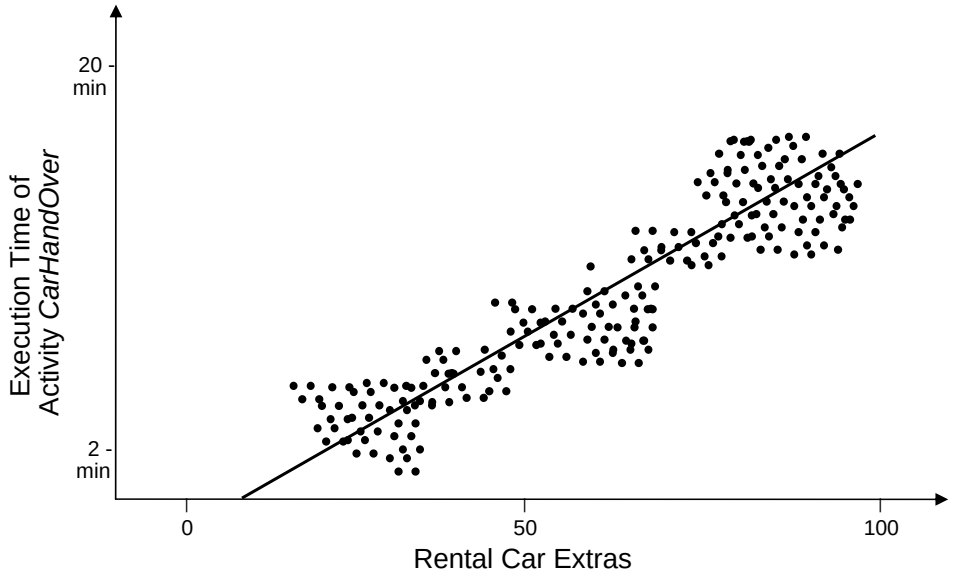


Abb. 5.12: Beispiel für BIA-Regression

5.4.4 Assoziationsregelerkennung

Assoziationsregelerkennung in BIA basiert auf einer Mischung von multidimensionalem Assoziationsregelmining und Prozessmining. Multidimensionale Assoziationsregelerkennung ist eine beliebte Methode, um interessante Beziehungen zwischen verschiedenartigen Attributen in einem großen Warehouse zu erkennen [KHC97]. Prozessmining dagegen betrachtet Prozessstrukturen und versucht herauszufinden, welche Aktivitäten mit welchen Variablen zusammen und in welcher Reihenfolge im Prozess auftreten [vdAvDH$^+$03]. Attribute die oft zusammen auftreten, werden Frequent Itemsets genannt. Sie sollen durch das Data Mining gefunden werden, um sie für die Erstellung von Assoziationsregeln zu verwenden. Die Assoziationsregel (A=a, B=b)→ (C=c) besagt: Wenn die Attribute A und B zusammen auftreten, belegt mit den Werten a und b, ist es wahrscheinlich, dass auch C mit dem Wert c auftaucht. A, B und C können sowohl für Prozessattribute als auch für operative Attribute stehen. Die Wahrscheinlichkeit der Regeln wird durch den Support (Anteil der Instanzen mit A,B,C und ihren Werten an allen ausgeführten Instanzen) und die Konfidenz (Anteil der Instanzen mit C=c an allen ausgeführten Instanzen,

die auch A=a und B=b enthalten) angegeben [Han05]. Assoziationsregeln, die Prozessvariablen, ihre Werte und ihre Effekte auf spätere Aktivitäten berücksichtigen, sind in BIA notwendig, um eine effiziente Ressourcenplanung zu erreichen, aber auch um Prozessaktivitäten zu parallelisieren oder die Reihenfolge der Aktivitäten zu ändern. Operative Daten können dabei helfen, diese Effekte zu kategorisieren.

rule	Left side	right side	support
①	{Activity = SearchCar, VariableElem = CarModel, CarModel.Value = CorvetteC6ZR1}	→ {Activity = SpecialBriefing}	0,001
②	{Activity = SearchCar, VariableElem = CarModel, CarModel.Value = RenaultScenic}	→ {Activity = InteriorCleaning}	0,013
③	{Activity = SearchCar, VariableElem = CarModel, CarModel.Category = sports}	→ {Activity = SpecialBriefing}	0,89
④	{Activity = SearchCar, VariableElem = CarModel, CarModel.Category = family}	→ {Activity = InteriorCleaning}	0,77

Abb. 5.13: Beispiel für BIA-Assoziationsregelerkennung

In dem Beispielszenario könnte man in einem ersten Schritt die Aktivität *SearchCar* analysieren und darauf folgende Aktivitäten im Prozess. Die Mining-Methode könnte Regeln wie z.B. in Abb. 5.13 (1) und (2) erkennen, die jedoch nicht wertvoll sind, da ihr Supportwert zu niedrig ist. Wenn wir jedoch den Operator BIASUBALL verwenden, können die operativen Daten bezüglich des Prozesselementes *CarModel* im Mining berücksichtigt werden. So können wir nun die gefundenen Aktivitätsausführungen im Hinblick auf das Attribut *Category* (z.B. Sportauto oder Familienauto) des Elements *CarModel* zusammenfassen und erhalten somit den notwendigen Support, um neue Assoziationsregeln zu etablieren, wie sie in Abb. 5.13 (3) und (4) gezeigt werden. Diese Regeln können verwendet werden, um den Prozess neu zu implementieren, um so eine schnelle Prozessausführung zu garantieren.

5.5 Zusammenfassung

In diesem Kapitel wurde auf die Warehousing- und die Analysephase des BIA-Optimierungszyklus eingegangen. In der Warehousingphase werden die Daten der gematchten Prozess- und operativen Schemamodelle integriert. Dazu wurde ein neues konzeptionelles Modell für einen Datenwürfel entwickelt. Dieser spezielle Würfel ermöglicht es, die Prozessdaten und operativen Daten zusammen zu betrachten, aber trotzdem dabei

den Fokus auf den Geschäftsprozessen zu behalten, da diese mit Hilfe der Analyseergebnisse optimiert werden sollen. Deshalb stehen die Prozessinstanzen in der Faktentabelle. Die Dimension mit den Geschäftsobjekten wird durch operative Subdimensionen ergänzt, die weitere Informationen für die Analyse liefern.

Für die logische Umsetzung dieses konzeptionellen Modells schlägt das BIA-Framework ein Föderationssystem vor, das die Daten in den Quellsystemen belässt und nur das Mapping zu den Subdimensionen in die föderative Datenbank aufnimmt. Es werden zwei Möglichkeiten der Realisierung dieses Mappings in zwei unterschiedlichen Warehouse-Architekturen vorgeschlagen:

- *mit einer Matchtabelle*: sie wird aus der Matchphase übernommen und lediglich mit Kurznamen für die Quellschemas versehen. Die Integration findet auf Schemaebene statt.
- *mit mehreren Bridgetabellen*: für jeden Match aus der Matchphase wird eine eigene Bridgetabelle erstellt, mit den Instanzdaten befüllt und mit Kurznamen versehen. Die Integration findet auf Instanzebene statt.

Beide Architekturen wurden im Hinblick unterschiedlicher Aspekte untersucht. Die Architektur mit der Matchtabelle liegt klar bei dem benötigten Platzbedarf und Zeitbedarf für ETL im Vorteil, da nur ein Tupel pro Match gespeichert wird. Stattdessen muss bei der anderen Architektur für jeden Match eine Bridgetabelle gespeichert werden, die vor dem ETL erstellt werden muss. Diese notwendige Schemaanpassung ist ein weiterer Minuspunkt. Dafür ist aber diese Architektur wiederum weitaus besser für adhoc-Analyseanfragen oder für die Verwendung von existierenden Analyseanwendungen geeignet, weil das Schema einer Standard-Warehouse-Architektur mit Bridgetabellen entspricht, im Gegensatz zu dem Warehouse mit der Matchtabelle, bei der erst die matchenden Instanzdaten herausgefunden und Cleansingschritte durchgeführt werden müssen. Falls man davon ausgehen kann, dass alle Daten in Audit Trail und operativer Datenbank bereinigt vorliegen und auf das Cleansing verzichtet werden kann, arbeiten Anfragen mit den BIA-Operatoren dagegen auf beiden Architekturen gleich schnell und halten sich auch im Hinblick auf die Benutzbarkeit die Waage. Insgesamt kann festgehalten werden, dass es letztlich von den individuellen Anforderungen des Benutzers abhängt, welche Warehouse-Architektur besser geeignet ist.

Für die Analysephase wurden für das BIA-Framework beispielhaft verschiedene OLAP-Operatoren entwickelt und in diesem Kapitel vorgestellt. Diese Operatoren ermöglichen eine gezielte Formulierung der SQL-Anfragen, um Hinweise für Prozessoptimierungen in den Warehousedaten ausfindig zu machen. Die acht Operatoren sind SQL-Tabellenausdrücke (engl. 'Table Expressions') und wurden im BIA-Framework als Anwendungsprogramm prototypisch umgesetzt:

- *für die Analyse der Subdimensionen*:
  - BIASUBALL: gibt alle operativen Attribute der operativen Tabellen und ihre Werte aus, die das gematchte Attribut zu einem gegebenen Prozesselement oder Fremdschlüssel der gematchten operativen Tabelle enthält

  - BIASUBNUM: wie BIASUBALL, gibt nur numerische operative Attribute aus
  - BIASUBLABEL: wie BIASUBALL, gibt aber nur alle nominalen Attribute aus
- *für die Analyse der Prozessdauer*:
  - BIAHT: gibt Geschäftsobjekte und die Ausführungsdauer aller Human-Task-Aktivitäten eines Prozesses aus
  - BIAPL: gibt Geschäftsobjekte und die Ausführungsdauer aller Invoke-Aktivitäten eines Prozesses aus
  - BIAFLUCT: gibt Geschäftsobjekte und alle Aktivitätsinstanzen aus, von denen ein gegebener Prozessanteil um einen geforderten Wert in seiner Ausführungszeit schwankt
- *für die Analyse von fehlerhaften Prozessen*:
  - BIAERROR: gibt Geschäftsobjekte und Aktivitätsinstanzen aus, von denen ein gegebener Prozessanteil in einem fehlerhaften Zustand endet
  - BIAEXCEPT: gibt Geschäftsobjekte und Aktivitätsinstanzen aus, bei denen bei einem gegebenen Anteil Fehlerbehandlungen stattfanden

Die Anwendbarkeit der Operatoren wurde beispielhaft an der Aufbereitung der Warehousedaten für vier wichtige Mining-Verfahren (Clustering, Klassifikation, Regression und Assoziationsregelerkennung) demonstriert. Über diese Mining-Beispiele wird auch der Informationsgewinn durch BIA deutlich gegenüber reinem Prozessmining ohne operative Daten.

KAPITEL 6

Schlussbetrachtungen

In diesem Kapitel wird die Arbeit zusammengefasst und ein Fazit gezogen. Es wird ein Ausblick auf zukünftige Erweiterungsmöglichkeiten der vorgestellten Lösungen in der Match-, Warehouse- und Analysephase gegeben. Außerdem folgt ein Ausblick in die Optimierungsphase von BIA, die die Ergebnisse aus den drei vorherigen Phasen verwendet.

6.1 Ergebnisse der Arbeit

Bevor in diesem Kapitel eine Bewertung zu den behandelten Aspekten von BIA erfolgt, werden zunächst die Ergebnisse der Arbeit zusammengefasst.

6.1.1 Zusammenfassung

Diese wissenschaftliche Abhandlung befasst sich mit einem Ansatz für eine ganzheitliche Geschäftsanalyse - in dieser Arbeit Business Impact Analysis, kurz BIA, genannt. BIA hat zum Ziel, Geschäftsprozesse für das Unternehmen gewinnbringend zu optimieren. Das bedeutet, es werden dabei die Faktoren beachtet, die einen betriebswirtschaftlichen Nutzen für das Unternehmen bringen, wie z.B. Kosten, Zeit und Ergebnis der Ausführung des Prozesses. Für eine ganzheitliche Analyse der Faktoren sind jedoch allein die Prozessausführungsdaten nicht ausreichend. Alle operativen Daten aus anderen Transaktionen im Unternehmen, die dasselbe Objekt wie das Geschäftsobjekt im Prozess beschreiben, beinhalten meist zusätzliche Information zu den Geschäftsobjekten. Diese Informationen stellen einen wichtigen Aspekt bei der Prozessoptimierung dar und sollen bei BIA berücksichtigt werden.

Bei der Optimierung in BIA werden unnötige Aktivitäten eliminiert oder neue Aktivitäten eingefügt bzw. der Kontroll- und Datenfluss komplett geändert. Bevor eine solche ganzheitliche Prozessoptimierung durchgeführt werden kann, ist die Realisierung dreier Bereiche notwendig: das Matchen der Prozessdaten mit den operativen Daten, die Speicherung der Daten für BIA in einer optimalen Warehouse-Architektur und die Durchführung der Analysen auf dieser Architektur, um das Optimierungspotenzial im Prozess zu erkennen.

Zunächst müssen die Beziehungen zwischen den Geschäftsobjekten und den operativen Daten gefunden werden. Das BIA-Framework unterstützt drei Möglichkeiten, diesen Matchprozess umzusetzen: das manuelle, semi-automatische und vollautomatische Matchen. Aufgrund der riesigen zu matchenden Datenmengen sind die (semi-)automatischen Verfahren zu bevorzugen. Da reine Schema-Matching-Verfahren oder Prozess-Matching-Verfahren dafür nicht ausreichend sind, sind in der Arbeit neue Matchtechniken entwickelt worden. Die Matchtechniken, die auf die Prozesselemente und die operativen Schemaelemente angewendet werden, sind Regeln, die auf existierenden, erweiterten und neuen Matchtechniken beruhen, die insbesondere die Namen und semantische Annotationen der zu matchenden Elemente berücksichtigen. Daneben werden aber auch prozessspezifische Eigenschaften, insbesondere die Verwendung der Variablen im Prozess, für das Matchen mit den operativen Daten herangezogen. So lassen sich mehr Matches zwischen den Elementen finden und eine höhere Genauigkeitsrate erzielen als allein durch Standard-Matching-Verfahren.

Zu den Matchtechniken gehört unter anderem das Erkennen von semantischen Beziehungen, die durch die Ontologie der Begriffe definiert werden, die für die Annotation verwendet wurden. Die semantischen Beziehungen betrachten unter anderem Synonyme und Subkonzepte zwischen zwei Annotationskonzepten und Axiome, bei denen ein Annotationskonzept von einem anderen als Inferenz definiert wurde. Darüber hinaus kann auch die partielle Annotation des operativen Schemas oder des Prozesses für das Matchen genutzt werden, indem die Namen der Annotationskonzepte mit den Namen der Matchpartner verglichen werden. Des Weiteren können auch für die nicht annotierten Elemente auf der Basis ihrer Namen Matches gefunden werden, oder die Elemente werden aufgrund ihrer Zugehörigkeit zu einer bestimmten Variable vom Matchprozess ausgeschlossen. Ein weiterer Schwerpunkt der Matchregeln liegt auf der Beachtung des Datenflusses oder definierten CorrelationSets im Prozess, wodurch weitere neue Matches entdeckt werden können.

Die Ergebnismenge aller gefundenen Matches wird durch weitere Matchtechniken verfeinert, die den Kontext der Matches betrachten. Dazu zählen wiederum existierende Techniken für den Vergleich der internen Struktur der matchenden Elemente oder ihrer Datentypen. Aber auch hier bieten neue Regeln eine wertvolle Ergänzung für ein genaueres Matchergebnis. Sie berücksichtigen bei dem Kontextvergleich der matchenden Elemente zum einen auch den Kontrollfluss des Prozesses, in den die Prozessvariablen eingebettet sind, und matchen die Namen der Kontrollflusselemente auf die operativen Elternelemente des matchenden Schemaelements. Zum anderen betrachten sie die semantischen Annotationen der Prozessaktivitäten in der Prozessbeschreibung und setzen sie

in Beziehung mit der (falls vorhandenen) semantischen Annotation des in der Ontologie am nächsten liegenden operativen Elternelements des gematchten Schemaelements.

Die Kontrollstrategie für den Ablauf der Matchtechniken berücksichtigt die Abhängigkeitsbeziehungen zwischen den einzelnen Matchregeln, um zuerst jene Regeln anzuwenden, deren Ergebnis bei den anderen Regeln auch betrachtet werden sollte. So können mehr Matchergebnisse gefunden werden, als wenn die Regeln in einer beliebigen Reihenfolge angewendet werden. Die Erstellung von Konzeptgruppen stellt sicher, dass semi-automatische Regeln in Abhängigkeit der Konzeptdefinitionen in der Ontologie auf Basis bereits gefundener Ergebnisse zusätzliche Matchergebnisse finden können. In einer Konzeptgruppe befinden sich alle Konzepte mit ihren annotierten Elementen, die in der Ontologie in einer äquivalenten semantischen Beziehung zueinander stehen oder die durch eine Vereinigung mit anderen für Annotationen verwendeten Konzepten definiert sind. Verschachtelte Konzeptgruppen beschreiben komplexere Beziehungen zwischen den beteiligten Konzepten und ihren Elementen. Gemäß der Kontrollstrategie können die annotierten Elemente aus gleichen oder verschachtelten Konzeptgruppen miteinander kombiniert werden. Ihr Ähnlichkeitswert berechnet sich aus der Kombination der einzeln durch die Matchtechniken bestimmten Ähnlichkeitswerte aus jeder am Match beteiligten Konzeptgruppe.

Die semantisch-basierten Matchregeln arbeiten auf Annotationen, die gemäß der SAWSDL-Strategie aus Kapitel 3 den Prozesselementen zugefügt wurden. Diese Strategie wurde für die manuelle Kombination so erweitert, dass mit ihr die Prozessbeschreibungssprache dBDD (deep Business Data Description) realisiert werden konnte. dBDD erweitert die Prozesselemente so, dass zu ihnen die gematchten operativen Elemente gespeichert werden. Das ist vor allem für Prozesse praktisch, deren Struktur noch mehrmals umgeändert wird und von denen deshalb mehrere Versionen in die Laufzeitumgebung gebracht werden. Mit dBDD muss der Match nicht jedes Mal mitgeändert werden, wenn das betreffende Prozesselement selbst nicht geändert wurde. Bei Verwendung der (semi-)automatischen Matchregeln speichert der Matcheditor die Ergebnisse dagegen in die Matchtabelle, was aufgrund der Referenz auf die Prozessversion für jede neue Version erneut ausgeführt werden muss.

Um eine ganzheitliche Analyse mit Hilfe von erweiterten OLAP- und Data-Mining-Techniken durchführen zu können, sind existierende Warehouse-Architekturen nur ungenügend für die integrierte Betrachtung der Prozessdaten und operativen Daten einsetzbar. Deshalb wurden im Rahmen dieser Arbeit zwei Warehouse-Architekturen entwickelt, die sowohl operative als auch Prozessdimensionen beinhalten. Die Architekturen unterscheiden sich in der Realisierung des Mappings zwischen den Prozessdaten und ihren zugehörigen operativen Daten (Matchtabelle mit Mappings auf Schemaebene versus Bridgetabellen mit Mappings auf Instanzebene). Außerdem entstand für die Arbeit eine Auswahl von Operatoren, die es erlauben, effizient OLAP-Anfragen über alle relevanten Attribute zu definieren, ohne dass dem Analysten Details über die Matches zwischen den operativen Daten und den Prozessdaten bekannt sein müssen. Neben den Operatoren bezüglich der Analyse der operativen Matches sind auch die Operatoren des Frameworks

über reine Prozessdaten hilfreich, um häufig gestellte Analysen zu formulieren, z.B. über die Dauer und Fehleranfälligkeit mancher Aktivitäten.

Die prototypischen Implementierungen der Match-, Warehousing- und Analysephase im BIA-Framework beweisen die praktische Einsatzfähigkeit der hier vorgestellten Ansätze. Die auf dem Prototypen der Matchphase basierenden Messungen belegen zudem, dass durch die neuen Matchregeln viel mehr und vor allem deutlich mehr korrekte Matches gefunden werden gegenüber der Version der Standardverfahren für Schema-Matching, die für die Evaluierung verwendet wurde. In der Warehousing- und Analysephase zeigt die Evaluierung des Prototypen einerseits, dass der Einsatz beider Warehouse-Architekturen von den jeweiligen Vorlieben des Benutzers abhängen und andererseits, dass bei Analyseanfragen auf beiden Architekturen die Verwendung von BIA-Operatoren hilfreich ist, wenn der Analyst die Matches zu den operativen Daten nicht kennt. Eine genauere Bewertung der entwickelten Ansätze der Arbeit folgt im Anschluss.

6.1.2 Bewertung von BIA

Die Bewertung von BIA lässt sich in drei Abschnitte unterteilen. Der erste Punkt der Bewertung ist die Evaluierung der Methoden, die die drei Phasen Matching, Warehousing und Analyse im Framework umsetzen. Außerdem sollen die betriebswirtschaftlichen Nutzenpotenziale des Frameworks diskutiert werden. Das Kapitel schließt mit einem allgemeinen Fazit zu BIA.

Evaluierung der Phasen im BIA-Framework

Im Folgenden werden die Methoden des Frameworks für die drei realisierten Phasen im BIA-Framework (vgl. Kapitel 1.2) bewertet, also für die Matchphase (Annotation und Matching) (1), für die Warehousephase (2) und die Analysephase (3):

(1) Matchphase: In den vorherigen Kapiteln wurden die drei Matchmethoden, die manuelle, semi-automatische und automatische Kombination, ausführlich erläutert. Sie werden in unterschiedlichen Situationen verwendet. Je nach Fachwissen des Designers ist die manuelle oder semi-automatische oder auch vollautomatische Methode schneller und effektiver. Das manuelle Verknüpfen der Prozessdaten mit operativen Daten funktioniert sehr effizient, wenn der Prozessdesigner sich auch bei den operativen Datenquellen auskennt. Dies ist nützlich bei einem Match zwischen Entitäten, deren Daten ähnlich strukturiert sind, z.B. um ihre Schlüsselwerte miteinander zu verbinden.

Sobald die Prozesselemente mit mehreren operativen Elementen matchbar sind oder dem Designer die zu matchenden operativen Datenquellen unbekannt sind, ist das semi-automatische oder vollautomatische Matching effektiver. Bei dem semi-automatischen Matching müssen jedoch auch die operativen Datenquellen von dem operativen DWH-Designer semantisch annotiert vorliegen. Außerdem ist das Vorhandensein eines Reasoners und einer Ontologie erforderlich, in deren Domäne sich der Designer für eine genaue Annotation der Daten gut auskennen muss, falls er die Annotation selbst vornimmt. Das

Matching der Daten ist auf diese Domäne beschränkt, im Gegensatz zur manuellen Kombination, bei dem die Daten aus verschiedenen Domänen vom Designer verknüpft werden können. Durch die Bereitstellung der Ontologie entsteht zwar ein zusätzlicher Aufwand; dieser fällt vor dem Hintergrund aktueller Entwicklungen jedoch nicht sehr stark ins Gewicht (siehe auch [RNM08]). Der Trend geht nämlich in die Richtung semantischer Web Services [CS06] und Semantik wird als zunehmend wichtiger Aspekt in der Prozessmodellierung angesehen. Daher bekommen entsprechende semantische Technologien wie SAWSDL gerade großen Auftrieb [HLD+05].

Das automatische Matching funktioniert nur für Elemente, die entweder einen sprechenden Namen des Elements oder des Annotationskonzepts besitzen (Regeln R6, R7), oder wenn vorherige gefundene Matches gemäß der Prozessstruktur auf übrige Prozesselemente kopiert werden können (Regeln R9, R10). Diese Regeln bringen einige zusätzliche Matchinformationen, da viele Elemente aus Zeit- oder Kostengründen nicht annotiert sind. Besonders R6, R7 und R9 können in den meisten Prozessen häufig genutzt werden. R10 kommt weniger oft zum Einsatz, da nicht jedes Element in einer Property definiert ist. Falls doch, können R10a-c bei Elementen, bei denen die Regeln R1-R9 kein Ergebnis liefern, oftmals noch über den Namen der Property oder schon gefundene Matches Informationen extrahieren und ggf. noch weitere Kombinationen finden.

Die Anwendbarkeit der Regeln wurde durch die Messungen der Trefferquote und der Genauigkeitswerte in Kapitel 4.9 gezeigt. Allerdings ist die Evaluierung nur bedingt aussagekräftig. Natürlich sind die Prozesse für die Messungen so konstruiert, dass alle Regeln zum Einsatz kommen können. Die Prozesselemente müssten auch durch andere Matching-Tools gematcht und das Ergebnis evaluiert werden, um eine Bestätigung der neuen Regeln zu erhalten. Dies ist jedoch nicht möglich, da die Prozessdaten dazu erst in großem Umfang aufbereitet werden müssten, damit sie als Eingabe der Tools akzeptiert würden. Vergleichen kann man das Ergebnis in Tabelle 4.38 Spalte (3) dennoch gut mit den Ergebnissen in Spalte (1), wo nur die Regel R6 und strukturelle Regeln ausgeführt wurden, was ähnlich zu den üblichen Schema-Matching-Tools ist. So kann man hierbei durch die Steigerung der Trefferquote und der Genauigkeit trotzdem die Effektivität des Frameworks erkennen.

(2) Warehouse-Architektur: Als Datenspeicher für die Analyse wurden gleich zwei Warehouse-Architekturen für BIA entwickelt. Die Architekturen unterscheiden sich in ihrer Mappingtabelle. Beide basieren auf einer föderativen Warehouse-Architektur, die die Daten in ihren Quellsystemen belässt. Das ist sinnvoll, da die riesigen Datenmengen aus Audit Trail und operativen Datenbanken, die nicht direkt an dem Match beteiligt sind, nicht im ETL-Prozess berücksichtigt werden müssen. Diese Datenmengen müssen nicht berücksichtigt werden, da im Audit Trail nur die Variablenelemente mit den operativen Daten gematcht werden können. Diese Daten müssen in einem ETL-Prozess für das Erstellen des Mappings sowieso gesondert betrachtet werden.

Das Mapping mit einer Matchtabelle scheint für viele Anwendungsgebiete besser zu sein: es erlaubt ein effizientes ETL-Verfahren und benötigt weniger Platz, um die Matches zu speichern. Die Bridgetabellen erlauben dagegen eine effizientere Anfrageverarbeitung,

wenn der Analyst die Matches zwischen den operativen Schema- und Prozessmodellen kennt. Dann kann er nämlich die korrekte Bridgetabelle mit den gespeicherten Instanzdaten herausziehen und muss nicht zusätzlich noch die Workflowtabelle und die Faktentabelle joinen, um die Ausführungsdaten zu erhalten, wie das bei der Matchtabelle der Fall ist. Das Kriterium der Benutzbarkeit und der Ausführungszeit bei Analyseanfragen, bei denen die Matches zu den operativen Daten dem Analysten unbekannt sind, ist in beiden Architekturen ausgewogen.

Für die Bewertung der beiden Architekturen gibt es also keine klare Entscheidung. Die Architektur mit der Matchtabelle ist jedoch im Vorteil, wenn viele Prozessmodelle, viele Variablen, dem Analysten unbekannte zugehörige operative Matches und viele Instanzdaten in den Datenquellen verfügbar sind. Das ist bei Prozessen, die mit BIA analysiert werden, meist der Fall.

(3) Analysemethoden: Wenn die Daten in einer Warehouse-Architektur mit einer Matchtabelle bereitliegen, kann die Formulierung der Analyseanfrage viel Zeit in Anspruch nehmen, da die Matches in der Matchtabelle erst aufgelöst werden müssen, bevor eine Anfrage gestartet werden kann. Dafür sind die BIA-Operatoren eine große Hilfe. Die existierenden OLAP-Operatoren allein können die operativen Subdimensionen im Warehouse nicht effizient bearbeiten, da sie nicht dafür ausgelegt sind. Die BIA-Operatoren machen es dem Analysten möglich, auf einfache Weise seine Anfragen an das BIA-Warehouse zu stellen. Ohne diese Operatoren wäre ein großer Mehraufwand nötig, da SQL-Experten komplexe Analyseanfragen formulieren müssten. Nun können die Anfragen von allen Benutzern formuliert werden, die ein Grundwissen in SQL besitzen. Dies wäre ansonsten aufgrund der vielfältigen und hohen Anzahl von Dimensionen und Attributen nicht in annehmbarer Zeit zu bewältigen.

Ob die BIA-Operatoren für BIA ausreichend sind, hängt davon ab, ob neben der Ausführungszeit, der Fehleranfälligkeit und den operativen Informationen der Prozesselemente noch andere Informationen für die Analyse als wichtig erachtet werden. Die Operatoren sind jedoch sicherlich ein guter Anhaltspunkt, da sie die offensichtlichen und dringlichsten Anfragen für BIA bewältigen können.

Diese Arbeit hat außerdem demonstriert, dass Data-Mining-Verfahren auch von diesen Operatoren profitieren, um die Daten für die verschiedenen Verfahren auf einfache Weise entsprechend aufbereiten zu können.

Betriebswirtschaftliche Nutzenpotenziale

Abb. 6.1 fasst die mit der Annotations-, Match- und Analysemethode und dem Ausführen einer ganzheitlichen Prozessoptimierung verfolgten Nutzenpotenziale zusammen. Die ganzheitliche Prozessoptimierung ermöglicht es einem Unternehmen, die Effizienz und Effektivität seiner Prozesse zu steigern. Indem die Annotations-, Match- und Analysemethoden und der Editor für die Umsetzung der Methoden Kosten-, Zeit- und Qualitätseffekte positiv beeinflussen, wird der Einsatz der Prozessoptimierung im Rahmen für BIA erst praktikabel ('Enabler').

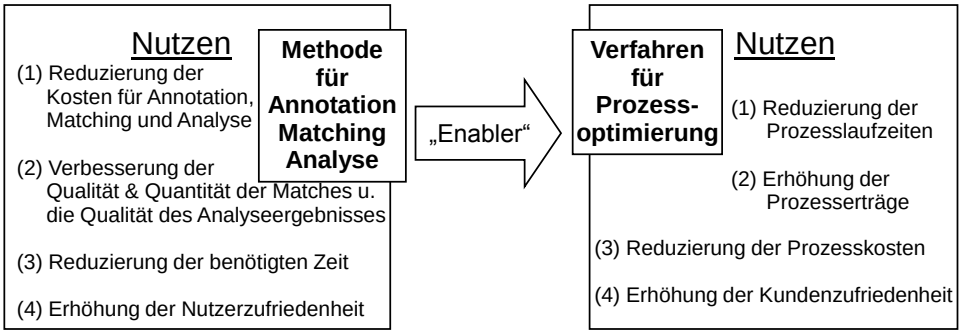


Abb. 6.1: Nutzenpotenziale des BIA-Frameworks

Im Folgenden werden die Nutzenpotenziale der Methoden und des Editors im Detail und analog zu Abb. 6.1 vorgestellt, die sich aus seiner Verwendung durch das BIA-Framework gegenüber einem manuellen Matching zur Analysezeit oder ohne BIA-Warehouse ergeben:

(1) Reduzierung der Kosten für Annotation, Matching und Analyse: Der Aufwand und damit die Kosten des Personals für die Erstellung der Verknüpfung verringern sich durch die gut bedienbare graphische Benutzeroberfläche des Annotations- und Matcheditors. Zudem muss der Match für die Variablen nur einmal pro Workflowmodell erfolgen und nicht für jede Analyse aufs Neue. Durch die Verwendung von SAWSDL im Editor ergeben sich noch weitere Kostenreduzierungen, da bereits erstellte Annotationen im Bereich der semantischen Web Services im Editor angezeigt und für die Analyse wieder verwendet werden können. Außerdem müssen bereits früher mit dem Editor erstellte semantische Annotationen nicht an neue operative Datenquellen angepasst werden, woraus sich weitere Einsparungen ergeben. Kosteneinsparungen ergeben sich durch die automatischen Matchregeln, bei denen die Genauigkeitsrate so gut ist, dass ggf. nur Personal für wenige Matchkorrekturen notwendig ist. Die Kosten zur Analysezeit bei der Formulierung der Anfragen verringern sich, da auf die Matchtabelle oder Bridgetabellen zurückgegriffen werden kann. Ebenso tragen die optimale Architektur des BIA-Warehouses mit den operativen Subdimensionen und die BIA-Operatoren zu einer Kostenreduzierung bei, da sie auf die für BIA interessanten Aspekte ausgelegt sind.

(2) Verbesserung der Qualität und Quantität des Matches und die Qualität des Analyseergebnisses: Die Fehlerquote der Kombinationen zwischen Prozessdaten und operativen Daten sinkt bei (semi-)automatischem Matching. Es können zudem Beziehungen gefunden werden, die einem Menschen nicht sofort aufgefallen wären und so kann auch die Quantität der Matches gesteigert werden. Besonders durch das vollautomatische Matchen werden neue Matches gefunden und die Genauigkeitsrate erhöht (siehe Kapitel 4.9). Eine standardisierte Ausgabe des Matches durch den Editor ermöglicht eine effektive Weiterverarbeitung im Warehouse durch den Analysten. Das BIA-Warehouse wiederum erhöht mit seiner passenden Architektur und der Verwendung der BIA-Operatoren nochmals die Qualität der Analysen, da Attribute zusammen betrachtet werden, für die sonst kein Zusammenhang erkannt worden wäre. Weiterhin erhöht die plattformübergreifende Nutzung verschiedener Workflowmanagementsysteme, Ontologiesprachen und beliebiger operativer relationaler Datenbanken die Quantität der Ergebnisse.

(3) Reduzierung der benötigten Zeit: Alle Faktoren, die bei den vorherigen Punkten eine Rolle spielen, können auch für die Zeitersparnis angeführt werden. Die graphische Benutzeroberfläche des Editors verkürzt die Zeit, die für eine Annotation und manuelles Matchen benötigt wird. Die vollautomatisch gefundenen Matches verkürzen die Zeit nochmals, die durch den Benutzer aufgewendet werden muss. Die Analysten wiederum können effizienter und effektiver ihre Analysen erstellen, da sie bereits qualitativ wertvolle Matches vorliegen haben und nicht selbst danach suchen müssen. Dadurch ergibt sich insgesamt eine schnellere Analyse der Prozesse. Auch die Anwendung der Operatoren spart Zeit bei der Formulierung der Anfragen.

(4) Erhöhung der Nutzerzufriedenheit: Eine gute Handhabung des Editors bezüglich verschiedener Richtlinien aus dem ISO-Standard [Iso06, RNM08] (z.B. Selbstbeschreibung und Individualisierbarkeit) und ein schnelles und zuverlässiges Match- und Analyseergebnis erhöhen die Bereitschaft des Nutzers diesen Editor zu verwenden.

Die Nutzenfaktoren Zeit- und Kostenersparnis machen es sinnvoll, den Editor für das Matchen und zum Teil auch für die Analyse zu verwenden. Die Erhöhung der Nutzerzufriedenheit führt dazu, dass der Editor auch wirklich benutzt wird und letztendlich die Annotationen und Matches überhaupt erst erstellt werden, was ohne den Editor selten der Fall wäre. Ansonsten bliebe die Erstellung des Matches die Aufgabe der Analysten, was aufgrund des hohen Zeitaufwandes in der Regel nicht durchgeführt wird. Auch die Analysen werden aufgrund der fehlenden optimalen Datenspeicher und OLAP-Operatoren nicht durchgeführt und neues Optimierungspotenzial aus den operativen Informationen nicht erkannt, selbst wenn ein Match gefunden wurde. Der Editor macht also ein ganzheitliches Optimierungsverfahren erst möglich. Das Nutzenpotenzial, das sich aus dem Optimierungsverfahren und der daraus resultierenden Prozessoptimierung ergibt, ist nachfolgend aufgelistet (siehe auch [BW05]):

(1) Reduzierung der Prozesslaufzeiten: Durch die Verknüpfung der Prozessdaten mit operativen Daten und ihre gemeinsame Analyse können Faktoren erkannt werden, die die Laufzeit eines Prozesses negativ beeinflussen. Mit der Erstellung neuer Prozessregeln, die diese Eigenschaften umgehen oder alternativ bearbeiten, kann die Laufzeit reduziert werden.

(2) Erhöhung der Prozesserträge: Prozesse, die abgebrochen wurden oder nur wenig Gewinn bringen, werden in Verbindung mit den operativen Datenquellen ganzheitlich optimiert.

(3) Reduzierung der Prozesskosten: Durch die Reduzierung der Prozesslaufzeit werden oft auch die Prozesskosten reduziert. So können z.B. die Kosten der im Prozess eingesetzten Ressourcen wie Maschinen und Mitarbeiter optimiert werden.

(4) Erhöhung der Kundenzufriedenheit: Die ganzheitliche Analyse erlaubt es, Kundenbedürfnisse und -wünsche schneller zu erkennen und in die Geschäftsprozesse zu integrieren. Dies erhöht die Kundenzufriedenheit und kann Kunden langfristig an das Unternehmen binden.

Allgemeines Fazit von BIA

Man kann erkennen, dass das hier vorgestellte Framework einen viel versprechenden Ansatz für die Umsetzung von BIA bildet. Alle Anforderungen aus Kapitel 1.2.2 wurden erfüllt und wie gefordert in einem Prototypen umgesetzt. Die Bewertung der Phasen und auch die betriebswirtschaftlichen Nutzenpotenziale lassen erkennen, dass das BIA-Framework für eine ganzheitliche Optimierung von Geschäftsprozessen gewinnbringend einsetzbar ist. Für die Umsetzung werden existierende Technologien effizient ausgenutzt, um durch neuartige BIA-Matchverfahren und Analysen zusätzliches Optimierungspotenzial zu finden. Natürlich sind ausführlichere Kosten-Nutzen-Analysen mit realen Geschäftsdaten und realen Benutzertests notwendig, um ein endgültiges Urteil zu fällen, wie profitabel der hohe Annotations- und Matchaufwand für eine automatisierte Prozessoptimierung wirklich ist.

Das BIA-Framework kann auf verschiedene Arten ergänzt und weiterentwickelt werden. Einige Ansätze werden im nächsten Kapitel näher erläutert.

6.2 Ausblick

Im Folgenden werden Erweiterungsmöglichkeiten beschrieben, die in den verschiedenen Phasen bisher noch nicht umgesetzt wurden, aber viel versprechende Ergebnisse für BIA erzielen könnten. Für diese Problemstellungen müssen in der Zukunft geeignete Lösungsansätze erarbeitet werden.

6.2.1 Annotation und Matching

Der in dieser Arbeit vorgestellte Optimierungszyklus beruht auf der Workflowbeschreibungssprache BPEL und relationalen operativen Datenbanken. Für eine Weiterentwicklung des Frameworks muss herausgefunden werden, welche Anpassungen an den Ein- und Ausgabefunktionen des Editors, den Matchregeln und der Architektur des Warehouses nötig sind, damit auch andere Workflowsprachen und operative Daten unterstützt werden können.

Wie in Kapitel 4.6 beschrieben, können die Prozesselemente über die Spracherweiterung dBDD manuell gematcht und im Prozess gespeichert werden. Damit die Spracherweiterung von einem WFMS erkannt wird, wäre es überlegenswert, dBDD als BPEL-Erweiterung anzubieten. Diese Erweiterung benötigt eine Workflow-Engine oder einen Parser, der dieses neue Attribut erkennt und bei oder nach dem Deployment den Match zu dem entsprechenden operativen Schemaelement in den Audit Trail oder separat abspeichert.

Auch eine Erweiterung der Matchregeln ist ein wichtiger Aspekt, sodass z.B. auch die Erkennung von einer explizit modellierten Korrelation zwischen operativen Daten und Prozesselementen im Geschäftsprozess über BPEL/SQL möglich ist. Dieser Aspekt wurde

in Kapitel 2.1.3 bereits angedacht und müsste durch Matchregeln realisiert werden. Dabei ist zu klären, wie die operativen Elemente, die in komplexen SQL-Anfragen verschachtelt sind, dabei auf die Prozesselemente abgebildet werden können.

Neben den vorgestellten Matchregeln R6, R11 und R14 aus bekannten Schema-Matching-Verfahren könnten auch weitere Regeln daraus interessant sein. So könnten auch noch weitere existierende Regeln aus Schema-Matching-Verfahren wertvolle Ergebnisse liefern, wie z.B. die *editDistance*-Regel [HD80] oder die *ngram*-Regel [DR02]. *EditDistance* berechnet die Ähnlichkeit über die Levenshtein-Distanz zwischen zwei Wörtern, d.h. wie viele Editierungen (Löschen, Ändern, Einfügen) notwendig sind, um das eine Wort in das andere Wort zu überführen. Die *nGram*-Regel vergleicht dagegen einzelne Ausschnitte zwischen zwei Wörtern auf Gleichheit, was bei den ähnlichen Aktivitätsnamen sowohl hinderlich als auch förderlich für das Matching sein kann. Der Einfluss solcher oder anderer existierender Regeln auf die Qualität der Matches bleibt zu diskutieren, um sie ggf. anschließend einzubinden.

Bei den semi-automatischen Matchregeln wäre es interessant herauszufinden, ob weitere modellierte semantische Beziehungen in der Ontologie eine Rolle bei der Qualität der Matchergebnisse spielen. So könnte es hilfreich sein, gefundene Matches, deren Annotationen jedoch als Komplement modelliert wurden, aus der Ergebnismenge auszuschließen.

Des Weiteren ist es sinnvoll, die Verwendung von instanzbasierten Matchverfahren wie z.B. den DUMAS-Matcher [BN05] zu betrachten. Dadurch kann ggf. eine Steigerung der Trefferquote und der Genauigkeitsrate erreicht werden. Beispielsweise können sich neue Matches ergeben, wenn die nicht annotierten Elemente unterschiedliche Namen haben und auch durch R9 und R10 die existierenden Matches nicht gefunden werden, aber die Elemente aus den Prozessinstanzen und Datenbanktupeln gleiche Werte haben. Für manche Matches, die durch R3 oder R4 gefunden wurden, ist es sogar unerlässlich, das Ergebnis für eine bessere Genauigkeit mit den einzelnen Instanzdaten zu überprüfen.

In einem weiterführenden Matchingszenario wäre es sinnvoll, einen einzelnen Matchprozess auf mehrere Geschäftsprozesse in einem Unternehmen auszuweiten. Falls ein Prozess schon mit operativen Schemaelementen gematcht wurde, können sich daraus wertvolle Hinweise für Matches des aktuell zu matchenden Prozesses ergeben. Viele Prozesse sind in einer Domäne ähnlich aufgebaut und verwenden gleiche Namen für semantisch äquivalente Aktivitäten oder Variablenelemente. Inwieweit die existierenden Matches mit ihren Ähnlichkeitswerten auf diese Prozesse übertragbar sind, oder ob dies nur für Prozesse des gleichen Prozessdesigners realisierbar ist, muss vorher herausgefunden werden.

Des Weiteren könnte eine genauere Ähnlichkeitswertberechnung der Matches durch das Einfließen der Ergebnisse aller Matchregeln R1-R10 erreicht werden, wie das im BIA-Framework bisher nur für die semi-automatischen Regeln mit Hilfe der Konzeptgruppen realisiert wurde.

Auch ist zu prüfen, ob das automatische Matching effizienter abläuft, wenn es statt auf der Prozessbeschreibung in der Datei auf den Prozessmodellinformationen im Audit Trail ausgeführt wird. Dadurch könnten längere Parsingvorgänge eingespart werden. Allerdings

wären dann aufgrund der unterschiedlichen Audit Trails je nach Ausführungsumgebung im Editor für die Matchregeln weitere Anpassungen nötig.

6.2.2 Warehousing und Analyse

In dieser Arbeit wurden zwei Warehouse-Architekturen für BIA skizziert. Die Alternative mit den Bridgetabellen kann dabei sehr platzeinnehmend sein und ggf. je nach Belegung der Parameter in der Platzabschätzung aus Kapitel 5.1.3, also der Anzahl der Prozesselemente, Aktivitäten, Matches usw., zu Platzproblemen führen. Eine Herausforderung besteht darin, statistische Auswertungen von früheren Analyseergebnissen zu evaluieren und in der Warehousephase für die Erstellung des Mappings im Warehouse zu verwenden. So könnte herausgefunden werden, welche operativen Attribute für die BIA-Analyse überhaupt relevant sind und der Match im Warehouse überhaupt gespeichert werden muss. Mit Hilfe der Statistiken kann so die Anzahl der Bridgetabellen drastisch reduziert werden. Genau die gleichen Statistiken wären für eine Reduzierung und Vorauswahl der Attribute für die BIA-Analyse sinnvoll, wie bereits in Kapitel 1.2.1 erwähnt. Eine Realisierung dieser Konzepte wäre sicherlich eine Verbesserung und würde die meisten Mining-Analysen in der Analysephase sogar erst ausführbar machen. Bei zu vielen Attributen werden die Miningschritte oft so komplex, dass sie nicht mehr in annehmbarer Zeit für den Analysten zu einem Ergebnis kommen.

Ein wichtiger Punkt ist zudem die praktische Umsetzung der ETL- und Cleansingschritte. Dazu müssen geeignete Methoden für das BIA-Warehouse begutachtet und realisiert werden.

Ein Schwerpunkt dieser Arbeit war das Analyseverfahren, für das BIA-Operatoren entwickelt wurden. Zunächst muss geklärt werden, ob noch weitere Analysen für das jeweilige Unternehmen interessant sind, in dem das BIA-Framework eingesetzt werden sollte. Eine detailreiche Quelle für weitere Analyseaspekte sind die Key Performance Indikatoren (KPI), die entweder schon im Unternehmen vorliegen oder die noch definiert werden müssen. Vorschläge für Strategien gibt es u.a. in [Pet06].

Falls sich die bereits umgesetzten Aspekte für BIA-Analysen mit Hilfe der Operatoren etablieren, ist es überlegenswert, ob die Operatoren nicht als OLAP-Erweiterung als eine *BuiltIn-Table*-Funktion in ein Datenbankprodukt mitaufgenommen werden könnten. Diese Funktionen könnten so realisiert werden, indem die Implementierung einer Datenbank so erweitert wird, dass ein BIA-Operator vor der Übersetzung abgefangen wird, durch ein Programm ausgeführt und als Ergebnistabelle BIA materialisiert wird. Diese Ergebnistabelle kann dann in die ursprüngliche Anfrage eingesetzt werden, bevor die Anfrage von der Datenbank übersetzt und zur Ausführung gebracht wird. Für diese Ergänzung ist jedoch Zugriff auf eine Datenbankimplementierung notwendig, um die Änderungen für die BuiltIn-Table-Funktion einzufügen.

Zukünftige Arbeiten sollten sich ausführlicher mit der Anpassung existierender Mining-Algorithmen für die BIA-Analysen beschäftigen. Ein Problem, das es zu lösen gilt, ist der bereits angesprochene explodierende Suchraum von Attributen. Neben den gerade

erwähnten Statistiken sollte noch nach weiteren Möglichkeiten gesucht werden, das BIA-Mining effizient ausführen zu können. Als eine Lösung könnte man sich vorstellen, dass man das BIA-Mining schrittweise durchführt. Zunächst werden dabei für die Analyse interessante Ausführungspfade im Prozess manuell bestimmt, bevor dieser Pfad mit den Variablen analysiert wird. Erst zum Schluss wird das Mining-Ergebnis mit Hilfe der operativen Informationen spezialisiert, falls eines gefunden wurde und Matches existieren.

6.2.3 Optimierungsklassen und ihre Verwendung in BIA

Um den BIA-Optimierungszyklus vollständig abzuschließen, muss in zukünftigen Arbeiten die Optimierungsphase realisiert werden. Dabei müssen die Erkenntnisse, die aus den BIA-Analysen gewonnen wurden, in Optimierungsregeln umgewandelt werden. Mit Hilfe dieser Regeln können die Prozesse dann verändert werden, sodass ein neuer optimal ablaufender Prozess entsteht. Auf reiner Prozessebene gibt es bereits bewährte Vorgehensweisen zum Prozessredesign [DvdAtH05]. Die Optimierungsregeln lassen sich in fünf Klassen einteilen:

- Aktivitätsklasse: Optimierung einzelner Aktivitäten z.B. durch Aktivitätsstreichung oder Automatisierung
- Kontrollflussklasse: Optimierung des Kontrollflusses z.B. durch Änderung der Reihenfolge der Aktivitäten oder Verzweigung des Kontrollflusses
- Ressourcenklasse: Optimierung der Ressourcenzuordnung z.B. durch Beförderung einzelner Angestellter, sodass neue Rollen entstehen, oder durch eine neue Aufteilung der Zuständigkeiten für bestimmte Aktivitäten
- Partnerklasse: Optimierung der Interaktion mit Partnern z.B. durch Integration von Partnerprozessen in den Kontrollfluss des Prozesses oder durch Outsourcing von Prozessabschnitten an Partner
- Prozessklasse: Optimierung des Prozesses aus ganzheitlicher Sicht z.B. durch spezielle Ausnahmebehandlung bei Fehlern im Prozess

Diese bewährten Methoden können mit Informationen aus BIA angereichert werden. Beispiele zu den Klassen wurden bereits indirekt in Kapitel 5.2 und 5.4 beschrieben und dabei auch vorgestellt, welche Rolle die operativen Daten bei der Optimierung spielen. Die Einteilung der Klassen angereichert mit den operativen Daten muss für zukünftige Arbeiten jedoch genauer diskutiert werden und ob noch zusätzliche Klassen für die Berücksichtigung der operativen Daten in BIA notwendig sind. Außerdem ist zu klären, wie die Optimierungsphase ablaufen und implementiert werden soll.

Als weiterer Punkt muss geklärt werden, wie die Analyseerkenntnisse diesen Optimierungsklassen möglichst automatisiert zugewiesen werden können. Zudem muss eine Strategie entwickelt werden mit der genauen Definition der Optimierungsregeln, wie die ausgewählte Regel auf den Prozess zur Optimierung angewendet wird (z.B. durch XSLT-Templates) und welche manuellen Korrekturen dennoch auszuführen sind. Erste Schritte

in diese Richtung sind in [NRM11, NRM10b] dargestellt. In [NRM11] bekommt der Analyst über einen Musterkatalog konkrete Vorschläge, wie er den Prozess optimieren könnte. Anders funktioniert die Optimierung bei [NRM10b], da hier die zu optimierenden Prozesse auf einen existierenden, bereits optimierten Prozess gematcht werden und dessen optimierte Struktur so auf die aktuellen Prozesse übertragen wird.

Das bisherige BIA-Framework arbeitet auf bereits abgelaufenen Prozessen und ihren operativen gematchten Daten und versucht die Prozesse für zukünftige Ausführungen zu optimieren. Dies war für die Anforderungen dieses Projekts auch völlig ausreichend. Für die Zukunft sollte jedoch über eine Erweiterung des BIA-Frameworks nachgedacht werden, wie der Zeitpunkt der Analyse und der Prozessoptimierung flexibilisiert werden kann. Damit die Prozessoptimierung auch vor oder während der Ausführung der Prozessinstanzen stattfinden kann, müssen ähnlich wie bei Monitoringsystemen für Prozesse, auch die operativen Daten, die in Zusammenhang mit dem Prozess stehen, überwacht werden. So kann in kritischen Situationen angemessen darauf reagiert werden.

ANHANG A

Quellcode der BPEL- und WSDL-Dateien des Beispielszenarios

Der Quellcode für einen Geschäftsprozess in diesem Autovermietungsszenario ist in einer BPEL-Datei mit dem Namen *CarSelection.bpel* gespeichert. Die Geschäftslogik des Prozesses wurde in Kapitel 1.3 beschrieben und ist hier in Abb. A.1, Abb. A.2 und Abb. A.3 in Auszügen dargestellt. Es werden dabei nur die Prozesskomponenten dargestellt, die für die Erläuterung der Matchregeln in Kapitel 4 relevant sind.

Im Prozess werden zunächst die Partnerlinks definiert. An erster Stelle steht hier die Definition der Schnittstelle des Prozesses selbst. Danach werden die Partnerlinks zu dem Verwaltungsservice *RentalSystem* der Autovermietung und dem Dienst *TaskRoutingService* definiert, der die Zuteilung des Personals zu den Human Tasks und ihren Ablauf koordiniert. Anschließend folgt die Deklaration der Prozessvariablen und die Definition des Kontrollflusses. Die strukturierten Aktivitäten, die Scopes *ReceiveCustomerData*, *ContractNegotiation* und *CarHandOver* sowie die Case- und Sequence-Aktivitäten, beinhalten einfache Assign- und Invoke-Aktivitäten und implementieren die Prozesslogik aus Abb. 1.3.

Die Schnittstelle des Prozesses mit der Deklaration seiner Variablentypen, Nachrichtentypen, Operationen und <*portType*>-Elementen ist in einer WSDL-Datei dargestellt. Abb. A.4 und Abb. A.5 zeigen Auszüge aus der Beschreibung. Die Elemente sind teilweise mit Konzepten aus einer Ontologie, die durch die URI *http://BIA-Project/Org/2007/Info* identifiziert wird, annotiert. Ebenso ist die Operation *initiate* mit Hilfe des Konzepts *DataSelection* aus dieser Ontologie semantisch annotiert, da die erste Aktivität des Prozesses *receiveCustomerData* die Datenauswahl des Kunden entgegennimmt. Das Element *rentalStartDate* ist direkt mit einem Verweis auf die Spalte *CalendarDate* in der Tabelle *CalendarData* im Schema *CanvassSchema* aus Anhang B unter Angabe der notwendigen

Informationen für den Aufbau einer JDBC-Verbindung zu der Marketingdatenbank annotiert. Außerdem ist eine Property *customerAddress* als Alias sowohl für das Element *contact* der Nachricht *input* als auch für das Element *Info* der Nachricht *request* definiert.

WSDL-Dateien, die die Schnittstellen zu den aufgerufenen Web Services *RentalSystem* und *TaskManagerService* im Geschäftsprozess *CarSelection* beschreiben, sind in Abb. A.6 und A.7 nur kurz aufgelistet, soweit sie für das Verständnis des Matchszenarios wichtig sind.

```
<process name="CarSelection" …>

//Partnerlink-Definitionen
<partnerLinks>
   <partnerLink name="CustomerProcess" partnerLinkType="client:CarSelection"
         myRole="CarSelectionProcessProvider" partnerRole="CarSelectionProcessRequester"/>
   <partnerLink name="RentalSystem" partnerLinkType="ns1:CarSearch“
                              myRole="CarSearchRequester" partnerRole="CarSearchProvider"/>
   <partnerLink name="TaskManagerService" partnerLinkType= "taskmngr:TaskManager“
                                                              partnerRole="TaskManager“ …/>
   …
</partnerLinks>

//Variablen
<variables>
   <variable name="inputData" messageType="client:input"/>
   <variable name="outputData" messageType="client:output"/>
   <variable name="InvokeCarSearchVar" messageType="ns1:searchInput"/>
   <variable name="CarAvailability" messageType="ns1:CarSearchResponseMessage"/>
   <variable name=„ServiceInfo" messageType="client:request"/>
   <variable name="CallCustomerVar" element="task:task"/>
   <variable name="NewInputData" messageType="client:ChangeDate"/>
   <variable name="CarHandOverVar" element="task:task"/>
   …
</variables>

// Prozessdefinition
<sequence name="main">
//Aktivität ReceiveCustomerData
<scope name="CustomerInfoAcquisition">
  <sequence>
   <receive name="receiveCustomerData" partnerLink="CustomerProcess" portType=
            "client:selectionPort" operation="initiate„ variable="inputData" createInstance="yes"/>
   <assign name=„Assign_1“>
      <copy>
         <from variable="inputData" part="RentalData"/>
         <to variable="InvokeCarSearchVar" part="RentalData"/>
      </copy>
   </assign>
  </sequence>
   …
</scope>
…
//Aktivität SearchCar
<invoke name="SearchCar" partnerLink="RentalSystem" portType="ns1:CarSearch“
                                          operation="initiate" inputVariable="InvokeCarSearchVar"/>
<receive name="ReceiveCarSearch" partnerLink="RentalSystem" portType="ns1:
      CarSearchCallback“ operation="onResult"  variable="CarAvailability" createInstance="no"/>
```

Abb. A.1: Auszug aus dem BPEL-File (1)

```
<switch name=„CarAvailable“>
  <case condition="bpws:getVariableData('CarAvailability','payload','/ns1:
                                          CarSearchProcessResponse/ns1:result') = true">
    <sequence>
      //Aktivität SetServiceData
      <assign name="SetServiceData">
        <copy>
          <from variable="inputData" part="RentalData"
                                      query="/client:SelectionInfoType/client:rentalStartDate"/>
          <to variable=„ServiceInfo" part="InfoData"
                                          query="/client:ServiceRequest/client:StartDate"/>
        </copy>
        ...
      </assign>
    </sequence>
  </case>
  <otherwise>
    <sequence>
      //HumanTask ContractNegotiation
      <scope name="ContractNegotiation"  ...>
        ...
        <assign name="setUserDefinedAttributes">
          <copy>
            <from expression="CallCustomer"/>
            <to variable="CallCustomerVar" query="/task:task/task:title"/>
          </copy>
          <copy>
            <from expression="string('DepartmentA')"/>
            <to variable="CallCustomerVar" query="/task:task/task:assigneeGroups[1]"/>
          </copy>
          <copy>
            <from expression=bpws:getVariableData(‚inputData‘,‘RentalData‘)/>
            <to variable="CallCustomerVar" query="/task:task/task:payload"/>
          </copy>
          ...
        <invoke name="initiateTask" partnerLink="TaskManagerService"
                                                  portType="taskmngr:TaskManager"    ... />
        <switch> //Task.outcome accepted?
          <case condition="bpws:getVariableData('CallCustomerVar', '/task:task/task:state') =
                  'COMPLETED' and bpws:getVariableData('CallCustomerVar','/task:task/task:
                                                                      conclusion') = 'ACCEPT'">
            <assign><copy>
                  <from variable=„CallCustomerVar" query="/task:task/task:payload "/>
                  <to variable=„NewInputData" part=„payload“ />
            </copy> ...</assign>
            //Aktvität SetServiceData
            <assign name="SetServiceData">
              <copy>
                <from variable="NewInputData" part="payload"
                              query="/client:CarSelection/client:input/client:CarModel"/>
                <to variable=„ServiceInfo" part="InfoData“
                                      query="/client:ServiceRequest/client:requestedCar"/>
              </copy>
```

Abb. A.2: Auszug aus dem BPEL-File (2)

```
                </assign>
                …
              <otherwise>
                <terminate name="Terminate_CarSelectionProcess"/> …
              </otherwise>
        </scope>
        …
   </otherwise>
</switch>
<scope name="PickupService">
…
   //HumanTask CarHandOver
   <scope name="CarHandOver“ …>
      <assign name="setUserDefinedAttributes">
        <copy>
          <from expression=„CarHandOver"/>
          <to variable="CarHandOverVar" query="/task:task/task:title"/>
        </copy>
        <copy>
           <from expression=bpws:getVariableData(‚serviceInfo‘,‘InfoData‘)/>
           <to variable=„CarHandOverVar" query="/task:task/task:payload"/>
        </copy>
        <copy>
          <from expression="string('DepartmentD')"/>
          <to variable="CarHandOverVar" query="/task:task/task:assigneeGroups[1]"/>
        </copy> …
      </assign>
      <invoke name="initiateTask" partnerLink="TaskManagerService" …"/>
        <switch>
          <case condition="bpws:getVariableData('CarHandOverVar', '/task:task/task:state') =
              'COMPLETED' and bpws:getVariableData('CarHandOverVar',
                                                   '/task:task/task:conclusion') = 'ACCEPT'">
            <assign name="AssignPayload">
              <copy>
                <from expression="true"/>
                <to variable="outputData" part="payload“
                                        query="/client:response/client:CarHandOverFinished"/>
              </copy>
              …
            </assign>
          </case>
          <otherwise>
            <terminate name="Terminate_CarHandOver"/>
            …
        </switch> …
</scope>
…
<invoke name="callbackClient" partnerLink="CustomerProcess" portType="client:
              CarSelectionProcessCallback" operation="onResult" inputVariable="outputData"/>
</sequence> //end main
</process>
```

Abb. A.3: Auszug aus dem BPEL-File (3)

```
<definitions name="CarSelection“ …>
<types>
  <schema … xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="CustComplexType">
      <complexType>
        <sequence>
          <element name="custID" type="int“ sawsdl:modelReference=
                                        "http://BIA-Project.org/2007/Info#identifier" />
          <element name="name" type="string“ sawsdl:modelReference=
                                          "http://BIA-Project.org/2007/Info#name" />
          <element name="contact" type="string“ sawsdl:modelReference=
                                        "http://BIA-Project.org/2007/Info#address" />
        …
    </element>
    <element name="response">
      <complexType>
        <sequence>
          <element name="CarHandOverFinished" type="boolean"/>
        …
    </element>
    <element name="SelectionInfoType">
      <complexType>
        <sequence>
          <element name="rentalStartDate" type=„date“ bia:source=„jdbc:oracle:thin:
            login1/password1@localhost:1521:XE#CanvassSchema.CalendarData.
            CalendarDate  bia:sourceType=„relational“/>
          <element name="rentalEndDate" type=„date“ sawsdl:modelReference=
                                    "http://BIA-Project.org/2007/Event#CalendarDate" />
          <element name=„CarModel" type=„string"/>
        …
    </element>
    <element name="ServiceRequest">
      <complextype>
        <element name=„Info" type="string"/>
        <element name=„startDate" type=„date"/>
        <element name=„requestedCar“ type=„string“/>
        …
      </complextype>
    </element>

    <bpws:property name="customerAddress" type="string"/>
    <bpws:propertyAlias propertyName="client:customerAddress“
                                    messageType="client:input"  part="CustomerData"
                                      query="/client:CustComplexType/client:contact"/>
    <bpws:propertyAlias propertyName="client:customerAddress“
                                        messageType="client:request"  part="InfoData"
                                          query="/client:ServiceRequest/client:Info"/>
  </schema>
</types>
```

Abb. A.4: Auszug aus dem WSDL-File (1)

```
<message name="input">
   <part name="CustomerData" element="client:CustComplexType"/>
   <part name="RentalData" element="client:SelectionInfoType"/>
</message>
<message name="output">
   <part name=„outputData" element="client:response"/>
</message>
<message name="request">
   <part name="InfoData" element="client:ServiceRequest"/>
</message>
<message name=„changeDate“>
   <part name=„payload“ element=„client:SelectionInfoType“/>
</message>
 …

<portType name="selectionPort">
   <operation name="initiate“ sawsdl:modelReference=
                                        "http://BIA-Project.org/2007/Operations#DataSelection">
      <input message="client:input"/>
   </operation>
</portType>
<portType name="CarSelectionProcessCallback">
   <operation name="onResult">
      <input message="client:output"/>
   </operation>
</portType>

<plnk:partnerLinkType name="CarSelection">
    <plnk:role name="CarSelectionProcessProvider">
       <plnk:portType name="client:selectionPort"/>
   </plnk:role>
   <plnk:role name="CarSelectionProcessRequester">
      <plnk:portType name="client:CarSelectionProcessCallback"/>
   </plnk:role>
</plnk:partnerLinkType>

</definitions>
```

Abb. A.5: Auszug aus dem WSDL-File (2)

```
<definitions name=„RentalSystem“ …>
<types>
   <schema … xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="result“ type="boolean“ />

      <element name="SelectionInfoType">
         <complexType>
            <sequence>
               <element name="rentalStartDate" type=„date“ />
               <element name="rentalEndDate" type=„date“ />
               <element name=„CarModel" type=„string"/>
         …
      </element>

<message name=„searchInput">
   <part name="RentalData" element="client:SelectionInfoType"/>
</message>
<message name=„CarSearchResponseMessage">
   <part name="payload" element="client:result"/>
</message>
…

<portType name=„CarSearch“ …>
   …
</definitions>
```

Abb. A.6: Auszug aus dem WSDL-File für RentalSystem

```
<definitions name=„TaskManager“ …>
<types>
   <schema … xmlns="http://www.w3.org/2001/XMLSchema">
      <element name=„task“>
          <complexType>
            <sequence>
              <element name=„title“ type=„string“ />
              <complexType name=„EmployeeTask“>
                    <element name=„assigneeGroups“ type=„string“
                              sawsdl:modelReference=„http://BIA-
                              Project.org/2007/Info#SpecialRentalAgent“/>
                    <element name=„acquiredBy“ type=„int“
                                    sawsdl:modelReference=
                                     "http://BIA-Project.org/2007/Info#ID" />
              </complexType>
              <element name=„expirationDuration“ type=„duration“/>
              <element name=„payload“ type=„anyType“ />
              <element name=„state“ type=„string“ />
          …
      </element>

<message name=„taskMessage">
   <part name=„payload" element=„task"/>
</message>
…

<portType name=„TaskManager“>
  <operation name=„initiateTask“>
     <input name=„TaskManagerInput“ message=„taskMessage“ />
   …
</definitions>
```

Abb. A.7: Auszug aus dem WSDL-File für TaskManagerService

ANHANG B

Modell des operativen Schemas des Beispielszenarios

Dieses Kapitel zeigt die operativen Schemas, die im Unternehmen in anderen Anwendungen verwendet werden. Abb. B.1 beschreibt das *CanvassSchema* der Marketingabteilung, die für das Anwerben neuer Kunden zuständig ist und die Daten zur Werbekampagne sowie den neuen Kunden speichert. Abb. B.2 speichert im *VehicleSchema* Informationen zu der Fahrzeugflotte des Unternehmens und Abb. B.3 beinhaltet Auszüge des *HumanResourceSchemas* der Personalabteilung. Es werden in allen drei Schemas nur die Tabellen und Spalten angezeigt, die für das Matching in Kapitel 4 und der Analyse in Kapitel 5 eine Rolle spielen.

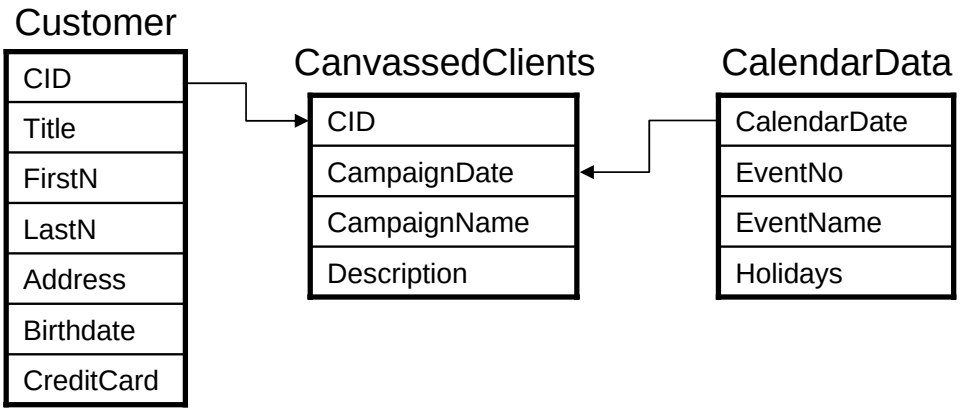


Abb. B.1: Auszug aus dem CanvassSchema der Marketingabteilung

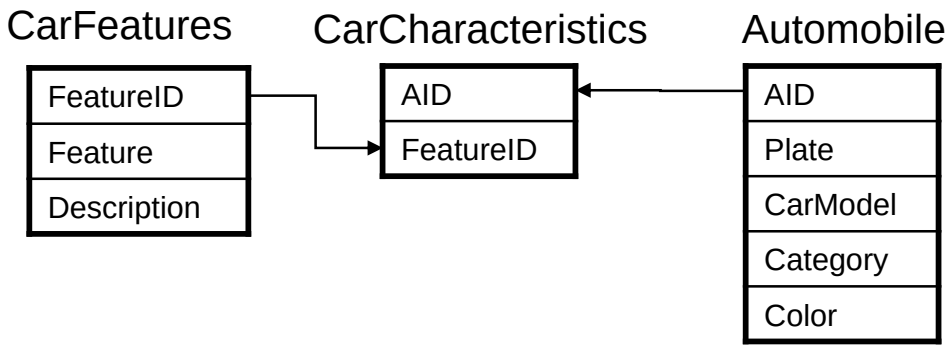


Abb. B.2: Auszug aus dem VehicleSchema der KFZ-Abteilung

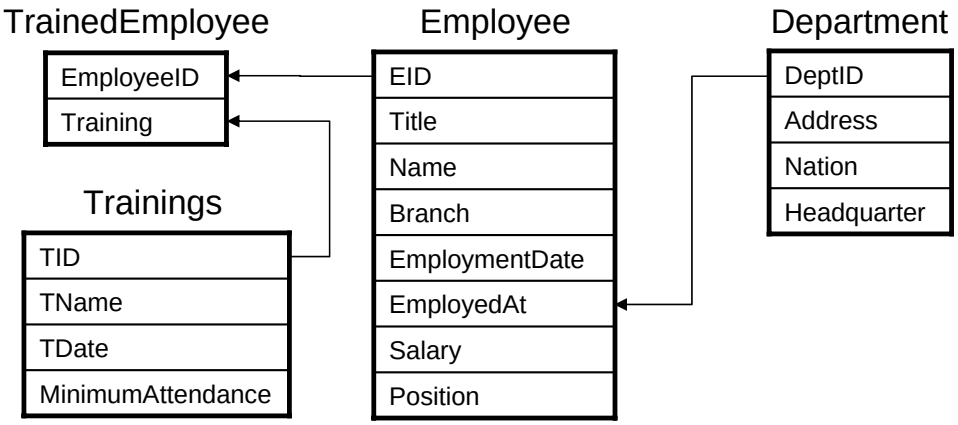


Abb. B.3: Auszug aus dem HumanResourceSchema der Personalabteilung

Um diese Schemas in den Matchregeln so verwenden zu können, wie sie in den Beispielen in Kapitel 4 jeweils angegeben sind, müssen sie teilweise semantisch annotiert sein. Diese Annotation wird hier jedoch nur kurz textuell erläutert und nicht graphisch dargestellt. Im *CanvassSchema* in Abb. B.1 ist deshalb die Tabelle *Customer* durch das Konzept *Customer*, die Spalte *Customer.CID* durch das Konzept *ID*, die Spalte *Customer.FirstN* durch das Konzept *firstname*, die Spalte *Customer.LastN* durch das Konzept *surname* und die Spalte *Customer.Address* durch das Konzept *address* annotiert. Im Schema *HumanResourceSchema* sind hingegen nur die Spalten *Employee.EmployedAt* durch das Konzept *Department* und *Employee.EID* durch das Konzept *ID* annotiert. Im Schema *VehicleSchema* ist die Tabelle *Automobile* mit dem Konzept *Car* annotiert.

ANHANG C

Auszug aus den gespeicherten Matchergebnissen

Im Folgenden werden die Matchergebnisse aus COM zwischen den Variablen des Autovermietungsprozesses aus Anhang A und den operativen Schemas aus Anhang B dargestellt. Für die Erstellung der Matches wurden alle Pairing- und Filterregeln verwendet, die in Kapitel 4 beschrieben wurden. In Klammer steht die Pairingregel, die bei dem jeweiligen Match ausschlaggebend war, d.h. mit deren Hilfe der größte Ähnlichkeitswert berechnet werden konnte. Die Filterregeln wurden immer alle angewendet. Mit der Formel aus Definition 9 wird die Ähnlichkeit der jeweiligen Kombination berechnet mit $w_{weight} = 0.6$. Alle Variablen in COM stammen aus dem Prozess *CarSelection*. Aus Platzgründen wird dies hier aber nur im Match M_1 angezeigt und bei den anderen Matches weggelassen.

$COM = \{$
$M_1 = (CarSelection.inputData.input.CustomerData.CustComplexType.custID,$
$CanvassSchema.Customer.CID,\ 0.82)$ `(R2)`
$M_2 = (inputData.input.CustomerData.CustComplexType.name,$
$CanvassSchema.Customer.LastN, 0.82)$ `(R5)`
$M_3 = (inputData.input.CustomerData.CustComplexType.name,$
$CanvassSchema.Customer.FirstN, 0.82)$ `(R5)`
$M_4 = (inputData.input.CustomerData.CustComplexType.contact,$
$CanvassSchema.Customer.Address, 0.82)$ `(R1)`
$M_5 = (inputData.input.RentalData.SelectionInfoType.rentalStartDate,$
$CanvassSchema.CalendarData.CalendarDate, 1)$ `(manuell)`
$M_6 = (inputData.input.RentalData.SelectionInfoType.rentalEndDate,$
$CanvassSchema.CalendarData.CalendarDate, 0.74)$ `(R7)`
$M_7 = (inputData.input.RentalData.SelectionInfoType.CarModel,$

$VehicleSchema.Automobile.CarModel, 0.83)$ (R6)
$M_8 = (InvokeCarSearchVar.searchInput.RentalData.SelectionInfoType.$
$rentalStartDate, CanvassSchema.CalendarData.CalendarDate, 1)$ (R9)
$M_9 = (InvokeCarSearchVar.searchInput.RentalData.SelectionInfoType.$
$rentalEndDate, CanvassSchema.CalendarData.CalendarDate, 0.74)$ (R9)
$M_{10} = (InvokeCarSearchVar.searchInput.RentalData.SelectionInfoType.$
$CarModel, VehicleSchema.Automobile.CarModel, 0.83)$ (R9)
$M_{11} = (CallCustomerVar.Task.EmployeeTask.AssigneeGroup,$
$HumanResourceSchema.Employee.employedAt, 0.7\overline{3})$ (R4+R3)
$M_{12} = (CarHandOverVar.Task.EmployeeTask.AssigneeGroup,$
$HumanResourceSchema.Employee.employedAt, 0.7\overline{3})$ (R4+R3)
$M_{13} = (ServiceInfo.request.InfoData.ServiceRequest.Info,$
$CanvassSchema.Customer.Address, 0.82)$ (R10)
$M_{14} = (ServiceInfo.request.InfoData.ServiceRequest.startDate,$
$CanvassSchema.CalendarData.CalenderDate, 1)$ (R9)
$M_{15} = (ServiceInfo.request.InfoData.ServiceRequest.requestedCar,$
$VehicleSchema.Automobile.CarModel, 0.83)$ (R9)
$M_{16} = (NewInputData.ChangeDate.payload.SelectionInfoType.$
$rentalStartDate, CanvassSchema.CalendarData.CalendarDate, 1)$ (manuell)
$M_{17} = (NewInputData.ChangeDate.payload.SelectionInfoType.$
$rentalEndDate, CanvassSchema.CalendarData.CalendarDate, 0.74)$ (R7)
$M_{18} = (NewInputData.ChangeDate.payload.SelectionInfotype.$
$CarModel, VehicleSchema.Automobile.CarModel, 0.83)$ (R6)
$M_{19} = (CallCustomerVar.Task.EmployeeTask.acquiredBy,$
$HumanResourceSchema.Employee.EID, 0.7\overline{3})$ (R1)
$M_{20} = (CarHandOverVar.Task.EmployeeTask.acquiredBy,$
$HumanResourceSchema.Employee.EID, 0.7\overline{3})$ (R1)
}

Die Berechnung der Ähnlichkeitswerte basiert u.a. auf der Regel R12, der der Prozesskontext zugrunde liegt. Dafür wurde die BPEL-Prozessstruktur aus Anhang A in einen Prozessbaum wie in Abb. C.1 überführt. Dieser Prozessbaum listet in den jeweiligen Hierarchieebenen die Namen der Prozesskomponenten auf, je nachdem wo sie im Prozess vorkommen. Die Variablen sind in Abb. C.1 nur vereinfacht dargestellt und nicht bis hin zu den Elementen angegeben. Unbenannte Prozesselemente und die Aktivität *main* werden dabei übersprungen. So ist für Regel R12 genau ersichtlich, welche Komponenten über den am Match beteiligten Variablenelementen in der Hierarchie stehen, deren Namen in R12 mit den Namen der operativen Schemaelemente und ihrer Elternelemente verglichen werden sollen.

M_{18} hat zwar den gleichen Ähnlichkeitswert wie M_7. Der Match wurde trotzdem erneut berechnet, da sich der Prozesskontext bei den zwei Variablen unterscheidet. Für Regelerweiterungen des BIA-Frameworks wäre es möglich, die Einführung einer neuen Regel zu diskutieren, die bei Variablen mit gleichen WSDL-Beschreibungen die gefundenen Matches mit den höchsten Ähnlichkeitswerten auf die anderen Matches überträgt.

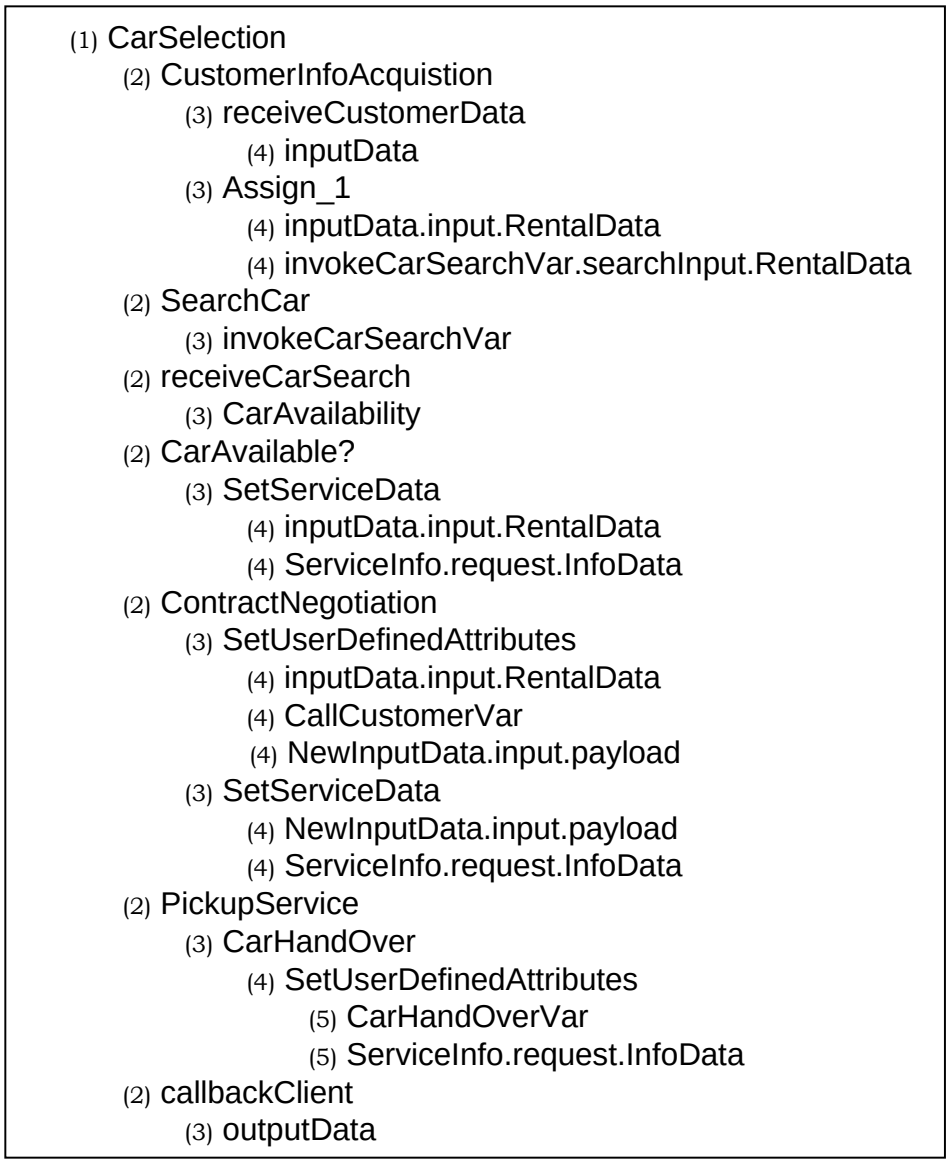


Abb. C.1: Schematischer Prozessbaum des Beispielprozesses

Der Matchkandidat $M_{21} = (CarSelection.inputData.input.CustomerData.CustComplexType.custID, HumanResSchema.Employee.EID, 0.7)$ wird korrekterweise nicht in COM aufgenommen, da der berechnete Ähnlichkeitswert zwar genau auf, aber nicht über dem Schwellwert 0.7 liegt.

Auch Regel R8 verhindert, dass falsche Matches zu den Elementen der Variablen *CallCustomerVar* und *CarHandOverVar* aus dem Human Task gefunden werden. So wird der falsche Match $M_{22} = CarSelection.CallCustomerVar.Task.title, CanvassSchema.Customer.Title, 0.725)$ eliminiert, obwohl sein Ähnlichkeitswert über dem Schwellwert liegt. Der Titel im Human Task

beschreibt semantisch allerdings etwas ganz anderes, er bezeichnet die Überschrift des Human Tasks.

Für die Variablenelemente *CarSelection.CarAvailability.payload.CarSearchResponseMessage.result* und *CarSelection.outputData.output.outputPart.response.CarHandOverFinished* wird richtigerweise ebenso kein Match gefunden. Es können von den Pairingregeln nur R6 und R7 für diese Elemente ausgeführt werden und die berechneten Ähnlichkeitswerte liegen zu allen operativen Elementen schon allein für das Pairing unter dem Schwellwert ($\frac{0.7-0.4*1}{0.6} = 0.5$), selbst wenn die Filterregeln den Wert 1 ergeben würden.

ANHANG D

Formale Definition der BIA-Operatoren

BIA-Operatoren definieren Methoden, mit denen man Informationen aus dem BIA-Warehouse extrahieren kann, die für die Prozessoptimierung eine wichtige Rolle spielen könnten. Die formale Definition beschreibt die Ergebnistupel der Operatoren. Wenn nicht anders spezifiziert, beziehen sich die Prädikate und Attribute dabei auf Tabellen und Spalten aus dem BIA-Warehouse.

Jeder BIASUB-Operator bezieht sich je nach Anwendung auf unterschiedliche Geschäftsobjektelemente und andere operative Tabellen und Attribute. Da deshalb die auszugebenden Spaltennamen und evtl. referenzierenden operativen Tabellennamen dynamisch bestimmt werden müssen, werden für die Realisierung der Operatoren die Metainformationen über das Schema des BIA-Warehouses hinzugezogen. Die Ergebnistupel werden im Folgenden mit Hilfe von spezifischen Metadateninformationen aus Tabellen berechnet, die hier *biareferences* (Informationen zu allen referenzierten Tabellen im BIA-Warehouse) und *biacolumns* (Informationen zu allen Tabellenspalten im BIA-Warehouse) sowie *biatables* (Informationen zu allen Tabellen im BIA-Warehouse) genannt werden.

Aus *biareferences* werden die Namen der Tabelle, ihrem Primärschlüssel, einem Fremdschlüssel und der durch den Fremdschlüssel referenzierten Tabelle als Attribute *tabname*, *pk_colnames*, *fk_colnames* und *reftabname* verwendet. Aus *biacolumns* werden die Namen der Spalten und ihrer Tabelle als Attribute *colname* und *tabname* verwendet. *biatables* selektiert nur die Tabellennamen als Attribut *tabname*. Je nach Hersteller des Warehousesystems müssen die Namen für die Metainformationen dementsprechend ersetzt werden.

D.1 Definition der BIASUB-Operatoren für Bridgetabellen

D.1.1 BIASUBALL-Operator

Die Anfrage des Operators wird in vier Teilanfragen zerlegt. Das Ergebnistupel wird in **BIASUBALL** mit Angabe der Aktivitätsinstanzen und operativen Spalten gespeichert. **MetaOP** berechnet aus den Metadaten alle Bridgetabellen die gemäß dem Muster in **BIASUBALL** auf die angegebenen Variablenelemente **var** und **processID** passen. Dabei wird die Angabe in **var** in ihre verschiedenen benötigten Bestandteile Variablenname (VAR(**var**)), Pfad (PATH(**var**)) und Element (ELEM(**var**)) aufgeteilt. Außerdem bestimmt **MetaOP** die referenzierten operativen Tabellen und den betreffenden Fremdschlüssel. **MetaOP** wird in die Hauptanfrage **BIASUBALL** eingesetzt. Da sich die Ergebnismenge von **MetaOP** jedoch auf die Metadaten bezieht, muss der jeweilige Wert aus **MetaOP** umgewandelt werden. Dies wird hier mit *value()* dargestellt. Der Wert wird entweder in eine Menge (wie in $\exists bt \in value(t.tabname)$) oder ein qualifizierendes Element daraus (wie in $bt.value(t.fk)$) umgewandelt.

Weitere operative Tabellen und Spalten werden im Ergebnistupel **OP** bereitgestellt. Es verwendet **MetaFK**, um alle Tabellen zu finden, die ausgehend von der Ergebnismenge **MetaOP** noch referenziert werden. Dies geschieht in den Metadaten des Warehouses über Aufnahme eines Fremdschlüssels ($reftabname$) in die aktuelle Tabelle oder der Aufnahme des Primärschlüssels ($tabname$) der aktuellen Tabelle in eine andere Tabelle als Fremdschlüssel. Die Erstellung von **MetaFK** ist iterativ, bis alle Referenzen in den Metadaten betrachtet wurden. **OP** speichert auch alle Spaltennamen der referenzierten operativen Tabellen. Duplikate sollten dabei eliminiert werden.

Der Join der Bridgetabellen aus **MetaOP** mit den Ergebnistupeln aus **OP** in **BIASUBALL** findet wie zuvor mit Hilfe von *value()* durch eine Umwandlung der Werte in die Tabellen und Attribute des Warehouse-Schemas statt. Auch die operativen Ergebnisattribute in **BIASUBALL** müssen mit *value()* in ihre qualifizierenden Elemente umgewandelt werden. Da alle Spaltenwerte zeilenweise bei jedem gefundenen Tupel ausgegeben werden sollen, muss zuvor eine Pivotisierung [BG04] der operativen Spalten durchgeführt werden, d.h. für jeden unterschiedlichen Wert von *op.col* bzw. *t.col* wird eine eigene Spalte erzeugt.

BIASUBALL (**var**, **processid**, **activityid**) =
$\{ [a.ActivityInst,\ value(PIVOT(t.col \cup op.col))] \mid$
$a \in ActivityExecution\ \wedge\ \exists w \in Workflow(w.ActivityInst = a.ActivityInst \wedge$
$w.processid = \mathbf{processid}\ \wedge\ w.activityid = \mathbf{activityid}) \wedge$
$\exists bo \in BusinessObject(bo.BOID = a.BOID\ \wedge\ bo.varname = VAR(\mathbf{var})$
$\exists boe \in BusinessObjectElement(boe.BOID = bo.BOID \wedge$
$boe.varElemName = ELEM(\mathbf{var})\ \wedge\ boe.path = PATH(\mathbf{var})) \wedge$
$\exists t \in MetaOP(\exists bt \in value(t.tabname)(boe.BOIDelem = bt.BOIDelem \wedge$
$\exists o \in value(t.ref)(o.value(t.pk) = bt.value(t.fk)) \wedge$
$\forall op \in OP(\exists o2 \in value(op.tabname) \wedge$
$\exists o3 \in value(op.ref)(o2.value(op.pk) = o3.value(op.fk)) \wedge$

$\exists op2 \in OP(op2.tabname = t.ref)))))\}$

mit **MetaOP** =
$\{ [r.tabname,\ r.reftabname\ as\ ref,\ c.colname\ as\ col,\ r.fk\_colnames\ as\ fk,$
$r.pk\_colnames\ as\ pk] \mid$
$r \in biareferences \wedge$
$r.tabname\ like\ 'BIABT\_$**processid**$\_VAR($**var**$)\_ELEM($**var**$)\%' \wedge$
$r.reftabname\ != \ 'BusinessObjectElement' \wedge$
$\exists c \in biacolumns(c.tabname = r.reftabname)\}$

und mit **OP** =
$\{ [m1.tabname, m2.tabname\ as\ ref, m1.fk, m2.pk, c.colname\ as\ col] \mid$
$m1, m2 \in MetaFK\ \wedge\ m1.ref = m2.tabname \wedge$
$\exists c \in biacolumns(c.colname = m1.col\ \vee c.colname = m2.col)\}$

und **MetaFK** =
$\{ [r.tabname, r.reftabname\ as\ ref,\ c.colname\ as\ col,\ r.fk\_colnames\ as\ fk,$
$r.pk\_colnames\ as\ pk] \mid$
$r \in biareferences \wedge$
$\exists o \in MetaFK_{old}($
$(o.ref = r.tabname\ \wedge \exists c \in biacolumns(c.tabname = r.reftabname)) \vee$
$(o.ref = r.reftabname \wedge \exists c \in biacolumns(c.tabname = r.tabname)) \vee$
$(\exists op \in MetaOP(op.ref = r.tabname \wedge$
$\exists c \in biacolumns(c.tabname = r.reftabname))))\}$

D.1.2 BIASUBNUM-Operator

Der Operator ist genauso aufgebaut wie der zuvor erläuterte **BIASUBALL**-Operator. Zusätzlich wird jedoch in **OP** und **MetaOP** jeweils bei *biacolumns* eine weitere Zeile eingefügt, um die Spalten nach ihren numerischen Datentypen zu filtern:
$...\wedge c.typename \in \{bigint, dec, decimal, double, float, int, integer, numeric, real, smallint\}$

D.1.3 BIASUBLABEL-Operator

Auch dieser Operator ist genauso aufgebaut wie der **BIASUBALL**-Operator. Zusätzlich wird jedoch in **OP** und **MetaOP** jeweils bei *biacolumns* eine weitere Zeile eingefügt, um die Spalten nach ihren nominalen Datentypen zu filtern:
$... \wedge\ c.typename \in \{char, character, varchar, long\ varchar\}$

D.2 Definition der BIASUB-Operatoren für die Matchtabelle

Die Anfrage des Operators auf der Matchtabelle wird hier wie bei der Bridgetabelle in vier Teilanfragen zerlegt. Auch hier wird wiederum *value()* und *PIVOT* benutzt, um die

Umwandlung der Metadaten in qualifizierende Elemente und ihre zeilenweise Darstellung wiederzugeben.

MetaOP2 bestimmt aus den Metadaten alle operativen Spalten für die gespeicherte gematchte Verbindung aus der Matchtabelle und wird in die Hauptanfrage **BIASUBALL** eingesetzt. Da in der Matchtabelle Kurznamen verwendet werden, um auf die richtigen Schemas und Tabellen zu verweisen, wird daraus die richtige operative Tabelle im Ergebnistupel abgeleitet ($\exists o \in value(TABLE(m.operational))$).

Weitere operative Tabellen und Spalten werden im Ergebnistupel **OP2** bereitgestellt. Es verwendet MetaFK2, um alle Tabellen zu finden, die ausgehend von der Ergebnismenge **MetaOP2** noch referenziert werden. Diese Ergebnisse gleichen der Definition **OP** und **MetaFK** in der Bridgetabelle oben.

BIASUBALL (**var**, **processid**, **activityid**) =
$\{ [a.ActivityInst,\ value(PIVOT(t.col \cup op.col))] \mid$
$a \in ActivityExecution \wedge\ \exists w \in Workflow(w.ActivityInst = a.ActivityInst \wedge$
$w.processid = \mathbf{processid}\ \wedge\ w.activityid = \mathbf{activityid}) \wedge$
$\exists bo \in BusinessObject(bo.BOID = a.BOID\ \wedge\ bo.varname = VAR(\mathbf{var}) \wedge$
$\exists boe \in BusinessObjectElement(boe.BOID = bo.BOID \wedge$
$boe.varElemName = ELEM(\mathbf{var})\ \wedge\ boe.path = PATH(\mathbf{var}) \wedge$
$\exists m \in MatchTable(m.process = w.processid \wedge$
$m.varElem = boe.varElemName\ \wedge\ m.var = bo.varName \wedge$
$m.path = boe.path) \wedge$
$\exists o \in value(TABLE(m.operational))(o.(m.operational) = boe.VarValue))) \wedge$
$\exists t \in MetaOP2($
$\forall op \in OP2(\exists\ o2 \in value(op.tabname) \wedge$
$\exists o3 \in value(op.ref)(op.value(op.pk) = op.value(op.fk)) \wedge$
$\exists op2 \in OP2(op2.tabname = t.tabname)))\}$

mit **MetaOP2** =
$\{ [r.tabname,\ c.colname\ as\ col] \mid$
$r \in biatables \wedge$
$r.tabname = TABLE(m.operational) \wedge$
$\exists c \in biacolumns(c.tabname = r.tabname)\}$

und mit **OP2** =
$\{ [m1.tabname,\ m2.tabname\ as\ ref,\ m1.fk,\ m2.pk,\ c.colname\ as\ col] \mid$
$m1, m2 \in MetaFK2\ \wedge\ m1.ref = m2.tabname \wedge$
$\exists c \in biacolumns(c.colname = m1.col\ \vee c.colname = m2.col)\}$

und **MetaFK2** =
$\{ [r.tabname,\ r.reftabname\ as\ ref,\ c.colname\ as\ col,\ r.fk\_colnames\ as\ fk,$
$r.pk\_colnames\ as\ pk] \mid$
$r \in biareferences\ \wedge\ \exists o \in MetaFK2_{old}($
$(o.ref = r.tabname\ \wedge \exists c \in biacolumns(c.tabname = r.reftabname)) \vee$

$(o.ref = r.reftabname \ \wedge \ \exists c \in biacolumns(c.tabname = r.tabname)) \vee$
$(\exists op \in MetaOP2(op.tabname = r.tabname \wedge$
$\exists c \in biacolumns(c.tabname = r.reftabname))))\}$

Auch die Operatoren **BIASUBNUM** und **BIASUBLABEL** haben die gleiche Ergänzung wie in der Variante mit den Bridgetabellen. In **OP2** und **MetaOP2** wird jeweils bei *biacolumns* eine weitere Zeile eingefügt, um die Spalten nach ihren numerischen bzw. nominalen Datentypen zu filtern.

D.3 Definition der Operatoren für die Analyse der Ausführungszeit

D.3.1 BIAHT-Operator

BIAHT (**processid**) =
$\{ [a.ActivityInst, \ (s2.time - s1.time) \ as \ duration, \ a.resourceID,$
$PIVOT(boe.VarElemName, \ boe.VarValue)] \mid$
$a \in ActivityExecution \wedge \exists bo \in BusinessObject(bo.BOID = a.BOID \wedge$
$\exists boe \in BusinessObjectElement(boe.BOID = bo.BOID)) \wedge$
$\exists w \in Workflow(w.ActivityInst = a.ActivityInst \wedge w.processid = \mathbf{processid} \wedge$
$w.type \ = \ 'invoke' \wedge w.partnerlink \ = \ 'TaskManager' \wedge$
$\exists s1, s2 \in Time(a.startTimeID = s1.timeID \wedge$
$\exists a2 \in ActivityExecution(a2.ActivityInst = w.successorActInst \wedge$
$a2.startTimeID = s2.timeID)))\}$

Das Attribut $boe.VarElemName$ ändert sich dynamisch bei jeder Anfrage. Es gibt als Spaltennamen der Ergebnistupel die Namen der Geschäftsobjektelemente der jeweiligen HumanTask-Aktivität ($w.type \ = \ 'invoke' \wedge w.partnerlink \ = \ 'TaskManager'$) an und als ihre Attributwerte die Werte von $VarValue$. Deshalb wird für die Spalten $boe.VarElemName$ und $boe.VarValue$ die Pivot-Spalten [BG04] berechnet. Für jeden Wert von $boe.VarElemName$ (ohne Duplikate) wird eine eigene Spalte erzeugt.

D.3.2 BIAPL-Operator

Die Ergebnismenge dieses Operators wird durch eine fast gleiche Formel wie der **BIAHT**-Operator beschrieben. Allerdings fehlt die Einschränkung $w.partnerlink = 'TaskManager'$, da der Operator die Instanzen aller Invoke-Aktivitäten ohne Ausnahme ausgeben soll.

D.3.3 BIAFLUCT-Operator

Diese Ergebnistupel werden der Übersicht wegen hier in drei Teilergebnisse aufgeteilt. Die Menge **Fluct2** beinhaltet dabei die für **BIAFLUCT** wichtigen Merkmale. Es wird ge-

prüft, ob die Ausführungsdauer der Aktivitätsinstanzen länger oder kürzer als die durchschnittliche Dauer ($AVG(f3.duration)$) dieser Aktivität ist. Außerdem wird geprüft, für wie viele Tupel der gleichen Aktivität $COUNT(f2)$ diese Abweichung in der Dauer prozentual an der gesamten Anzahl der Tupel in der Ergebnismenge $COUNT(f3)$ zutrifft. Wenn dies mehr als die gewünschten Prozent in den angegebenen **percent** sind, ist der Tupel Teil von **Fluct2**.

BIAFLUCT (**processid**, **duration**,**percent**) =
$\{ [f.ActivityInst,\ f.duration,\ a.resourceID,\ PIVOT(boe.VarElemName, boe.VarValue)] \mid$
$a \in ActivityExecution \wedge \exists bo \in BusinessObject(bo.BOID = a.BOID \wedge$
$\exists boe \in BusinessObjectElement(boe.BOID = bo.BOID)) \wedge$
$\exists f \in Fluct2(f.activityInst = a.activityInst)\}$

mit **Fluct1** =
$\{ [a.ActivityInst,\ (s2.time - s1.time)\ as\ duration,\ w.activityID\ ] \mid$
$a \in ActivityExecution \wedge$
$\exists w \in Workflow(w.ActivityInst = a.ActivityInst \wedge w.processid = \mathbf{processid} \wedge$
$\exists s1, s2 \in Time(a.startTimeID = s1.timeID \wedge$
$\exists a2 \in ActivityExecution(a2.ActivityInst = w.successorActInst \wedge$
$a2.startTimeID = s2.timeID)))\}$

und mit **Fluct2** =
$\{ [f.ActivityInst,\ f.duration,\ f.activityID\ ] \mid$
$f \in Fluct1 \wedge$
$\forall f2, f3 \in E1(f.activityID = f2.activityID = f3.activityID \wedge$
$(f2.duration\ >\ (\mathbf{duration} + AVG(f3.duration)) \vee$
$f2.duration\ <\ (AVG(f3.duration) - \mathbf{duration})) \wedge$
$\mathbf{percent}\ <\ \frac{100 * COUNT(f2)}{COUNT(f3)}\}$

Fluct2 wird in **BIAFLUCT** verwendet und dazu noch die *PIVOT*-Spalten von den zugehörigen Geschäftsobjektelementen wie beim **BIAHT**-Operator berechnet. Neben *PIVOT* dienen auch *AVG* und *COUNT* zur Vereinfachung der formalen Definition von **BIAFLUCT**.

D.4 Definition der Operatoren für die Analyse von Fehlern

D.4.1 BIAERROR-Operator

Auch hier werden die Ergebnismenge in drei Teilergebnisse aufgeteilt. Wie im **BIAFLUCT**-Operator mit **Fluct2** wird in **E2** mit *COUNT* der Anteil aller Instanzen eines Aktivitätsmodells berechnet, die nicht terminieren ($state\ != \ 'completed'$) und zwar öfter als in dem Parameter **percent** angegeben.

In **E1** wird für die Berechnung die Ergebnismenge in **E2** bereits auf die Tupel beschränkt, von denen mindestens eines eine nicht terminierende Instanz des gewünschten Prozesses enthält.

BIAERROR (**processid**, **percent**) =
$\{ [e.ActivityInst,\ e.state,\ a.resourceID,\ PIVOT(boe.VarElemName, boe.VarValue)] \mid$
$a \in ActivityExecution \wedge \exists bo \in BusinessObject(bo.BOID = a.BOID \wedge$
$\exists boe \in BusinessObjectElement(boe.BOID = bo.BOID)) \wedge$
$\exists e \in E2(e.activityInst = a.activityInst)\}$

mit **E1** =
$\{ [a.ActivityInst,\ a.state,\ w.activityID\ ] \mid$
$a \in ActivityExecution \wedge$
$\exists w \in Workflow(w.ActivityInst = a.ActivityInst\ \wedge w.processid = \textbf{processid} \wedge$
$\exists a2 \in ActivityExecution\ \wedge a2.state\ != \ 'completed' \wedge$
$\exists w2 \in Workflow(w2.ActivityInst = a2.ActivityInst \wedge$
$w2.activityID = w.activityID\ \wedge w2.processid = \textbf{processid}))\}$

und mit **E2** =
$\{ [e1.ActivityInst,\ e1.state,\ e1.activityID\ ] \mid$
$e1 \in E1 \wedge$
$\forall e2, e3 \in E1(e1.activityID = e2.activityID = e3.activityID \wedge$
$e2.state != \ 'completed' \wedge\ \textbf{percent}\ < \ \frac{100 * COUNT(e2)}{COUNT(e3)})\}$

D.4.2 BIAEXCEPT-Operator

Der **BIAEXCEPT**-Operator funktioniert analog zu **BIAERROR**. Allerdings interessieren hier (siehe **Ex1** und **Ex2**) nicht der Zustand des Prozesses wie zuvor, sondern der Typ der ausgeführten Aktivität, d.h. ob ein Fehler im Prozess auftrat, der behandelt wurde.

BIAEXCEPT (**processid**, **percent**) =
$\{ [e.ActivityInst,\ a.resourceID,\ PIVOT(boe.VarElemName, boe.VarValue)] \mid$
$a \in ActivityExecution \wedge \exists bo \in BusinessObject(bo.BOID = a.BOID \wedge$
$\exists boe \in BusinessObjectElement(boe.BOID = bo.BOID)) \wedge$
$\exists e \in Ex2(e.activityInst = a.activityInst)\}$

mit **Ex1** =
$\{ [a.ActivityInst,\ w.type,\ w.activityID\ ] \mid$
$a \in ActivityExecution \wedge$
$\exists w \in Workflow(w.ActivityInst = a.ActivityInst\ \wedge w.processid = \textbf{processid} \wedge$
$\exists w2 \in Workflow(w2.activityID = w.activityID\ \wedge w2.processid = \textbf{processid}$
$\wedge\ w2.FaultHandlerActivated = true))\}$

und mit **Ex2** =
$\{ [e1.ActivityInst,\ e1.type,\ e1.activityID\ ] \mid$
$e1 \in Ex1 \wedge$
$\forall e2, e3 \in Ex1(e1.activityID = e2.activityID = e3.activityID \wedge$
$e2.FaultHandlerActivated = true\ \wedge\ \mathbf{percent}\ <\ \frac{100 * COUNT(e2)}{COUNT(e3)})\}$

Auch hier bestimmt **Ex1** zunächst alle Tupel, für die mindestens eine Instanz existiert, in der ein Fault-Handler aktiviert wurde. Mit *COUNT* wird daraus in **Ex2** die Anzahl berechnet, bei denen ein gewisser Anteil die geforderte Grenze überschreitet.

Abbildungsverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[ABK+04] S. Arroyo, C. Bussler, J. Kopecky, R. Lara, A. Polleres, and M. Zaremba. Web service capabilities and constraints in wsmo. In *W3C Workshop on Constraints and Capabilities for Web Services*, USA, 2004.

[AGL98] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology*. Springer-Verlag, 1998.

[AL91] Hans-Jürgen Appelrath and Jochen Ludewig. *Skriptum Informatik: eine konventionelle Einführung*. Teubner, 1991.

[BG04] Andreas Bauer and Holger Günzel. *Data Warehouse Systeme*. dpunkt.verlag, 2004.

[BH08] Philip A. Bernstein and Laura M. Haas. Information integration in the enterprise. *Communications of the ACM*, 51(9), 2008.

[BKI06] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme. Grundlagen – Algorithmen – Anwendungen*. Vieweg, 3 edition, 2006.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5), 2001.

[BLS02] Robert M. Bruckner, Beate List, and Josef Schiefer. Striving towards near real-time data integration for data warehouses. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, 2002.

[BN05] Alexander Bilke and Felix Naumann. Schema matching using duplicates. *Proceedings of the 21st International Conference on Data Engineering*, 2005.

[BS08] Stefan Berger and Michael Schrefl. From federated databases to a federated data warehouse system. In *Proceedings of the 41st Hawaii International Conference on System Sciences*, 2008.

[BvHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. W3C Recommendation: Web Ontology Language. Available: http://www.w3.org/TR/owl-ref/, 2004.

[BW05] Eva Best and Martin Weth. *Geschäftsprozesse optimieren.* Gabler, 2 edition, 2005.

[CCDS07] Fabio Casati, Malú Castellanos, Umeshwar Dayal, and Norman Salazar. A generic solution for warehousing business process data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, 2007.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. 26(1), 1997.

[CGBB08] Juan Carlos Corrales, Daniela Grigori, Mokrane Bouzeghoub, and Javier Ernesto Burbano. Bematch: a platform for matchmaking service behavior models. In *Proceedings of the 11th International Conference on Extending database technology*, 2008.

[Cha95] James Champy. *Reengineering Management.* HarperCollins, 1995.

[CS06] Jorge Cardoso and Amit P. Sheth. *Semantic Web Services, Processes and Applications (Semantic Web and Beyond: Computing for Human Experience).* Springer-Verlag New York, Inc., 2006.

[dBBD+05] Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta König-Ries, Jacek Kopecky, Rubin Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. W3C Submission: Web Service Modeling Ontology. Available: http://www.w3.org/Submission/WSMO/, 2005.

[dBFK+05] Jos de Bruijn, Dieter Fensel, Uwe Keller, Michael Kifer, Holger Lausen, Reto Krummenacher, Axel Polleres, and Livia Predoiu. W3C Submission: Web Service Modeling Language. Available: http://www.w3.org/Submission/WSML/, 2005.

[Den04] Michael Denny. Ontology tools survey, revisited. *XML.com*, 2004.

[DGBD09] Marlon Dumas, Luciano García-Bañuelos, and Remco M. Dijkman. Similarity search of business process models. In *IEEE Data Engineering Bulletin*, volume 32, 2009.

[DHM+04] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004.

[DMDH04] Anhai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies in Information Systems*. Springer, 2004.

[DMR02] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, 2002.

[DR02] Hong-Hai Do and Erhard Rahm. Coma - a system for flexible combination of schema matching approaches. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[DSHB98] Barbara Dinter, Carsten Sapia, Gabriele Höfling, and Markus Blaschka. The olap market: state of the art and research issues. In *Proceedings of the 1st ACM International Workshop on Data warehousing and OLAP*, 1998.

[DSK+09] Marin Dimitrov, Alex Simov, Mihail Konstantinov, Luchesar Cekov, Barry Norton, and Carlos Pedrinaci. Wsmo studio: Semantic web service environment for wsmo. http://www.wsmostudio.org/, 2009.

[DSMK07] Marin Dimitrov, Alex Simov, Vassil Momtchev, and Mihail Konstantinov. Wsmo studio - a semantic web services modelling environment for wsmo (system description). In *Proceedings of the 4th European Semantic Web Conference*, 2007.

[DvdAtH05] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

[EKO07] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. Measuring similarity between semantic business process models. In *Proceedings of the 4th Asia-Pacific Conference on Conceptual Modelling*, 2007.

[FL07] Joel Farrell and Holger Lausen. W3C Recommendation: Semantic Annotations for WSDL and XML Schema. Available: http://www.w3.org/TR/sawsdl/, 2007.

[Hal03] Alon Y. Halevy. Data integration: A status report. In *Proceedings of the 10th Conference on Datenbanksysteme für Business, Technologie und Web*, 2003.

[Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[HD80] Patrick Hall and Geoff Dowling. Approximate string matching. *ACM Computing Surveys*, 12, 1980.

[HLD+05] Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In *Proceedings of the International Conference on e-Business Engineering*, 2005.

[HPO08] Hewlett-Packard and OpenSourceCommunity. Jena - a semantic web framework for java. Available: http://jena.sourceforge.net/, 2008.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. W3C Submission: A semantic web rule language combining OWL and RuleML. Available: http://www.w3.org/Submission/SWRL/, 2004.

[IBMa] IBM. Infosphere Federation Server. Available: http://www-01.ibm.com/software/data/infosphere/federation-server/.

[IBMb] IBM. Semantic tools for web services. Available: http://www.alphaworks.ibm.com/tech/wssem.

[iib] II4BPEL: Information integration for bpel on ibm websphere process server. Available: http://www.alphaworks.ibm.com/tech/ii4bpel.

[Iso03] International organization for standardization: Iso / iec norm 9075-2 – information technology – database languages – SQL – part 2: Foundation, 2003.

[Iso06] International organization for standardization: Iso norm 9241-110 - ergonomics of human-system interaction - part 110: Dialogue principles. Available: http://www.iso.org/iso/home.htm, 2006.

[JM99] Michael Jaedicke and Bernhard Mitschang. User-defined table operators: Enhancing extensibility for ordbms. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 494–505, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3), 1999.

[JMN03] Vanja Josifovski, Sabine Massmann, and Felix Naumann. Super-fast xml wrapper generation in db2: A demonstration. In *Proceedings of the 19th International Conference on Data Engineering*, 2003.

[KFNM04] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. The protégé owl plugin: An open development environment for semantic web applications. Springer, 2004.

[KHC97] Micheline Kamber, Jiawei Han, and Jenny Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1997.

[KKP+05] Matthias Kloppmann, Frank Koenig, Dieter Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic. Ws-bpel extension for people - bpel4people. In *Joint white paper, IBM and SAP*, 2005.

[Kün06] Thomas Künneth. Using the wizard api. Available: http://today.java.net/article/2006/02/24/using-wizard-api, 2006.

[KRMF06] Jacek Kopecký, Dumitru Roman, Matthew Moran, and Dieter Fensel. Semantic web services grounding. In *Proceedings of the International Conference on Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, 2006.

[KRT+98] Ralph Kimball, Laura Reeves, Warren Thornthwaite, Margy Ross, and Warren Thornwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses with CD Rom.* John Wiley & Sons, Inc., New York, NY, USA, 1998.

[KVLL+08] Dimka Karastoyanova, Tammo Van Lessen, Frank Leymann, Joerg Nitzsche, and Daniel Wutke. Ws-bpel extension for semantic web services (bpel4sws), version 1.0. Technical report, Institut für Architektur von Anwendungssystemen, Universität Stuttgart, 2008.

[LN07] Ulf Leser and Felix Naumann. *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* dpunkt, 2007.

[LR00] Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques.* Ed. Prentice Hall, Inc., Upper Saddle River, New Jersey, second edition, 2000.

[MBH+04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. W3C Recommendation: Owl-s: Semantic markup for web services. Available: http://www.w3.org/Submission/OWL-S/, 2004.

[MBR01] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. Technical report, Microsoft Research, 2001.

[McC02] D. W. McCoy. Business activity monitoring: Calm before the storm. Technical Report LE-15-9727, Gartner, 2002.

[MSTC94] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell. *Machine learning, neural and statistical classification.* Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[Nie08] Florian Niedermann. Development of a method for the integrated handling of process variables and operational data sources. Master's thesis, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2008.

[NMR+11] Florian Niedermann, Bernhard Maier, Sylvia Radeschütz, Holger Schwarz, and Bernhard Mitschang. Automated process decision making based on integrated source data. In *Proceedings of the 14th International Conference on Business Information Systems*, 2011.

[NR04] Frank Neumann and Anke Robeller. Websphere application server enterprise process choreographer - enhanced audit log data analysis and query. Technical report, IBM Corporation, 2004.

[NRM10a] Florian Niedermann, Sylvia Radeschütz, and Bernhard Mitschang. Deep business optimization: A platform for automated process optimization. In *Proceedings of the 3rd International Conference on Business Process and Services Computing*, 2010. (best paper award).

[NRM10b] Florian Niedermann, Sylvia Radeschütz, and Bernhard Mitschang. Design-time process optimization through optimization patterns and process model matching. In *Proceedings of the 12th IEEE International Conference on Commerce and Enterprise Computing*, 2010.

[NRM11] Florian Niedermann, Sylvia Radeschütz, and Bernhard Mitschang. Business process optimization using formalized optimization patterns. In *Proceedings of the 14th International Conference on Business Information Systems*, 2011.

[OAS07] OASIS. Web Services Business Process Execution Language Version 2.0. Available: http://docs.oasis-open.org/wsbpel, 2007.

[OMG08] OMG. Bpmn: Business process model and notation. Available: http://www.omg.org/spec/BPMN/, 2008.

[Pap08] Michael P. Papazoglou. *Web Services: Principles and Technology.* Pearson, Prentice Hall, 2008.

[PCTM03] John Poole, Dan Chang, Douglas Tolbert, and David Mellor. *Common Warehouse Metamodel Developer's Guide.* Wiley, January 2003.

[Pet04] Eric Peterson. *Web Analytics Demystified: A Marketer's Guide to Understanding How Your Web Site Affects Your Business.* Celilo Group Media, 2004.

[Pet06] Eric Peterson. *The Big Book of Key Performance Indicators*. Available: http://www.webanalyticsdemystified.com, 2006.

[Pet07] Dusan Petkovic. Die unterstützung von sql/olap im sql-standard und in relationalen datenbanksystemen. *Datenbank-Spektrum*, 7(22), 2007.

[Rah94] Erhard Rahm. *Mehrrechner-Datenbanksysteme - Grundlagen der verteilten und parallelen Datenbankverarbeitung*. Addison-Wesley, 1994.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10:2001, 2001.

[RGvdA+07] Vladimir Rubin, Christian W. Günther, Wil M. P. van der Aalst, Ekkart Kindler, Boudewijn F. van Dongen, and Wilhelm Schäfer. Process mining framework for software processes. In *Proceedings of the International Conference on Software Process*, 2007.

[RM08] Sylvia Radeschütz and Bernhard Mitschang. An annotation approach for the matching of process variables and operational business data models. In *Proceedings of the 21st International Conference on Computer Applications in Industry and Engineering*, 2008.

[RM09] Sylvia Radeschütz and Bernhard Mitschang. Extended analysis techniques for a comprehensive business process optimization. In *Proceedings of the International Conference on Knowledge Management and Information Sharing*, 2009.

[RML08] Sylvia Radeschütz, Bernhard Mitschang, and Frank Leymann. Matching of process data and operational data for a deep business analysis. In *Proceedings of the 4th Interoperability for Enterprise Software and Applications*, 2008.

[RNB10] Sylvia Radeschütz, Florian Niedermann, and Wolfgang Bischoff. Biaeditor - matching process and operational data for a business impact analysis. In *Proceedings of the 13th International Conference on Extending Database Technology*, 2010.

[RNM08] Sylvia Radeschütz, Florian Niedermann, and Bernhard Mitschang. Ein annotationsansatz zur unterstützung einer ganzheitlichen geschäftsanalyse. In *Proceedings of the 4th Conference on Data Warehousing*, 2008.

[RVSM11] Sylvia Radeschütz, Marko Vrhovnik, Holger Schwarz, and Bernhard Mitschang. Exploiting the symbiotic aspects of process and operational data for optimizing business processes. In *Proceedings of the 12th IEEE International Conference on Information Reuse and Integration*, 2011.

[SCDS02] Mehmet Sayal, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. Business process cockpit. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[SdBF08] Nathalie Steinmetz, Jos de Bruijn, and Andreas Frankl. Wsml/owl mapping. *WSML Working Draft*, (D37 v0.2), 2008.

[SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22, 1990.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: a practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2), 2007.

[SS04] Steffen Staab and Rudi Studer. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[UG99] İlhan Uysal and H. Altay Güvenir. An overview of regression techniques for knowledge discovery. *Knowledge Engineering Review*, 14(4), 1999.

[vdAvDH+03] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowledge Engineering*, 47(2), 2003.

[VGS+05] Kunal Verma, Karthik Gomadam, Amit P. Sheth, John A. Miller, and Zixin Wu. The meteor-s approach for configuring and executing dynamic web processes. Technical report, University of Georgia, Athens, Greece, 2005.

[VSRM08] Marko Vrhovnik, Holger Schwarz, Sylvia Radeschütz, and Bernhard Mitschang. An overview of sql support in workflow products. In *Proceedings of the 24th International Conference on Data Engineering*, 2008.

[WCL+05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[Wor98] Workflow Management Coalition. Workflow management coalition: Audit data specification, document number wfmc-tc-1015. Available: www.wfmc.org, September 1998.

[zM04] Michael zur Muehlen. *Workflow-based Process Controlling. Foundation, Design, and Application of workflow-driven Process Information Systems.* Logos, 2004.

[zMS09] Michael zur Muehlen and Robert Shapiro. Business process analytics. In *Handbook on Business Process Management*, volume 2. Springer Verlag, Berlin, 2009.