Ralph Bobrik

Konfigurierbare Visualisierung komplexer Prozessmodelle

Als Buch erschienen im Dr. Hut-Verlag, München, in der Reihe Informatik
http://www.dr.hut-verlag.de

ISBN: 978-3-89963-778-6
http://www.dr.hut-verlag.de/978-3-89963-778-6.html

Ralph Bobrik

Konfigurierbare Visualisierung komplexer Prozessmodelle

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat. der Fakultät für Ingenieurwissenschaften und Informatik der Universität Ulm, angefertigt am Institut für Datenbanken und Informationssysteme in Kooperation mit dem Forschungszentrum der Daimler AG in Ulm

Amtierender Dekan:	Prof. Dr. Helmuth Partsch
Gutachter:	Prof. Dr. Manfred Reichert
Gutachter:	Prof. Dr.-Ing. Michael Weber
Tag der Promotion:	5. Juni 2008

Informatik ist die Philosophie des 21. Jahrhunderts

N.N.

Kurzfassung

Die in heutigen Unternehmen durch Informationssysteme unterstützten Geschäftsprozesse werden zunehmend komplexer. Häufig existieren keine zentralen Steuereinheiten, sondern die Ausführung eines Prozesses ist auf viele heterogene Systeme verteilt. Ohne entsprechende Werkzeugunterstützung ist es daher schwer, einen Überblick über den aktuellen Ausführungsstatus solcher fragmentierter Prozesse zu bewahren. Eine Visualisierungskomponente, welche die Prozesse (inkl. relevanter Applikationsdaten) durchgängig darstellt, ist hier essenziell. Allerdings muss eine solche Komponente in der Lage sein, die Informationsbedürfnisse der verschiedenen Benutzergruppen adäquat zu befriedigen. Typischerweise gibt es hier unterschiedliche Anforderungen an eine Prozessvisualisierung im Hinblick auf Detaillierungsgrad, angezeigte Daten und graphische Informationsaufbereitung. Heutige Werkzeuge stellen Prozesse meist in exakt derselben Form dar, wie sie vom Prozessmodellierer ursprünglich gezeichnet worden sind. Eine flexible Anpassung der Darstellung an die Bedürfnisse des Betrachters ist nicht oder nur in sehr engen Grenzen möglich.

Diese Arbeit stellt mit Proviado ein Rahmenwerk für die konfigurierbare Visualisierung komplexer Prozesse vor. Proviado ermöglicht sowohl eine strukturelle als auch eine graphische Anpassung der Prozessvisualisierung. Mit Hilfe eines mächtigen *View-Mechanismus* können Prozessmodelle strukturell an die Bedürfnisse ihrer Betrachter angepasst werden, indem Prozesselemente reduziert oder zu abstrakten Elementen aggregiert werden. Es werden View-Bildungsoperationen bereitgestellt, die in mehreren Schichten organisiert sind. Mittels Konfigurationsparametern, die die Eigenschaften der resultierenden Prozessmodelle beeinflussen, kann die View-Bildung flexibel konfiguriert und an die Bedürfnisse des jeweiligen Anwendungsfalls angepasst werden. Weitere Möglichkeiten zur graphischen Konfiguration einer Prozessvisualisierung bietet ein fortschrittlicher *Template-Mechanismus*. Zum einen können die für die Visualisierung zu verwendenden Symbole einfach definiert werden. Zum anderen erlaubt dieser Mechanismus eine flexible Zuordnung der Symbole einer Prozessnotation zu Prozesselementen. Diese Zuordnung kann entweder statisch (z.B. abhängig vom Prozesselementtyp) oder dynamisch, d.h. abhängig von Laufzeitdaten (z.B. Ausführungszustand), erfolgen. Diese beiden Basismechanismen werden ergänzt um Konzepte, die für die Realisierung einer umfassenden Visualisierungskomponente unverzichtbar sind. Dazu zählen unter anderem die Anbindung prozessunterstützender Systeme (d.h. die Integration von Modell- und Laufzeitdaten), sowie Konzepte für das automatische Layout dynamisch berechneter Prozessgraphen.

Insgesamt können mit Proviado Prozessvisualisierungen strukturell und graphisch an die Bedürfnisse des jeweiligen Betrachters angepasst werden. Die entsprechenden Darstellungen bieten allen in die Prozesse involvierten Personen eine wesentlich bessere Unterstützung bei der täglichen Arbeit als derzeit verfügbare Systeme.

Vorwort

Diese Arbeit entstand im Rahmen eines Forschungsprojekts der Daimler AG, in dem fortschrittliche Themen einer Prozessmanagementinfrastruktur bearbeitet werden. Die Prozessvisualisierung war eine funktionale Anforderung, die in den Jahren 2001-2003 wiederholt von IT-Projekten der unterschiedlichen Geschäftsbereiche nachgefragt wurde. Da die Lösungsansätze existierender Werkzeuge nicht ausreichend und der Aufwand einer Konzeptentwicklung mit anschließender Implementierung für ein Einzelprojekt zu groß war, wurde das Thema im Jahr 2004 als Dissertationsthema ausgeschrieben. Heute liegt mit dieser Arbeit ein umfassendes Konzept für die konfigurierbare Visualisierung komplexer Prozessmodelle vor. Es liegt nun an den Herstellern von Prozessmanagementwerkzeugen, entsprechende Lösungen in ihre Produkte zu integrieren, um so dem ungebrochenen Bedarf an einer Visualisierungskomponente für Prozesse nachzukommen.

An dieser Stelle möchte ich einigen Personen danken, ohne deren Unterstützung diese Arbeit so nicht möglich gewesen wäre.

Mein Dank gilt Prof. Dr. Manfred Reichert für die Betreuung dieser Arbeit, für das akribische Korrekturlesen jeglicher Schriftstücke und für die vielen Diskussionen in Ulm, Enschede und diversen anderen Orten dieser Welt: *Het was een geweldige tijd!* Gleichermaßen bedanke ich mich auch bei Dr. Thomas Bauer für die Betreuung dieser Arbeit bei der Daimler AG und die zahllosen, unvergesslichen Momente. Mit seiner einzigartigen Kombination aus Praxiswissen, Zielorientierung und strukturiertem Denken, verbunden mit dem nötigen Pragmatismus, gab er in unzähligen Diskussionen wichtige Anregungen. Ich danke Herrn Prof. Dr. Peter Dadam für sein Interesse an meinem Thema und die „Beherbergung“ im Institut für Datenbanken und Informationssysteme (DBIS) der Universität Ulm. Herrn Prof. Dr.-Ing. Michael Weber danke ich für die Übernahme des Koreferats. Für die Schaffung der organisatorischen Rahmenbedingungen bei Daimler Research & Development und die Impulse (z.B. hinsichtlich Planung) gilt mein Dank Reiner Siebert. Mein herzlicher Dank gebührt Rüdiger Pryss für die Vermittlung des Kontaktes und die wahre Freundschaft. Besonders bedanke ich mich auch bei Dominic Müller für die vielen heißen Diskussionen und lustigen Stunden im Daimler-Büro, auf Reisen und jenseits der Arbeit. Allen Kollegen des Instituts DBIS der Universität Ulm und des Teams Process-Management der Daimler AG danke ich für das angenehme Arbeitsumfeld und den Spaß bei und neben der Arbeit. Besonders bedanke ich mich bei den Kollegen, die mich zum Schluss beim Korrekturlesen dieser Arbeit unterstützt haben. Ebenso bedanke ich mich bei allen Studenten für ihr Engagement, die im Rahmen von Diplomarbeiten und Praktika zu dieser Arbeit beigetragen haben.

Ulm, im März 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Grundlagen und Anforderungen

1 Einleitung

Unternehmen müssen heute eine Vielzahl von Geschäftsprozessen unterstützen, in die verschiedene Partner, Abteilungen und Bearbeiter involviert sein können. In diesem Kontext gibt es ein zunehmendes Interesse an Prozessmanagement-Technologie sowie aufkommenden Standards für die Orchestrierung und Choreographie von Prozessen bzw. der durch sie verknüpften Services (z.B. WS-BPEL [OAS07] bzw. WS-CDL [KBR+07]) [Hav05]. Diese Technologien und Standards ermöglichen die Definition und Ausführung der operativen Prozesse eines Unternehmens. In Verbindung mit Web-Service Technologie können darüber hinaus die Vorteile der Prozessautomatisierung auf unternehmensübergreifende Prozesse übertragen werden [Wic06].

Durch die Fragmentierung von Prozessimplementierungen auf verschiedene operative Systeme sowie die unternehmensübergreifende Vernetzung der Prozesse nimmt gleichermaßen die Komplexität zu. Insbesondere der Gesamtzusammenhang und der Überblick über den Gesamtprozess gehen dabei oftmals verloren. Will sich ein Prozessbeteiligter in einer solchen Umgebung, in der die Prozessdaten auf verschiedene, heterogene Informationssysteme verteilt sind, zum Beispiel einen Überblick über den aktuellen Ausführungszustand des Gesamtprozesses verschaffen, muss er heute auf diverse Informationsquellen zugreifen. Jedes System bietet eigene Log-Dateien, Berichte oder Visualisierungen an, die oftmals heterogene Struktur und Inhalte aufweisen und die der Bearbeiter selbständig zu einem Gesamtbild zusammensetzen muss.

Eine integrierte, graphische Visualisierung des Gesamtprozesses wäre hier sehr hilfreich. Mit einer derartigen Visualisierungskomponente könnten sich die Prozessbeteiligten über den aktuellen Ausführungszustand sowie über das Zusammenspiel der verschiedenen dargestellten, am Prozess beteiligten Personen informieren. So kann die Projektleitung auf Basis der gewonnen Informationen Projekte neuplanen bzw. deren Planung adaptieren. Der einzelne Prozessbearbeiter wiederum kann durch die Kenntnis der Prozessstruktur die eigenen Aufgaben besser in den Gesamtprozess einordnen. Gleichzeitig erhöht sich die Wahrnehmung der eigenen Verantwortung für die Prozessergebnisse. Aus der Visualisierung des Gesamtprozesses sind zudem die anderen

Prozessbeteiligten ersichtlich, so dass die Kommunikation mit Bearbeitern nachfolgender Aktivitäten erleichtert bzw. erst ermöglicht wird. Der aus der Prozessdarstellung zu entnehmende Bearbeitungszustand des Prozesses ermöglicht dem Beteiligten eine bessere Planung der eigenen, zukünftigen Aufgaben. Er kann frühzeitig auf Verzögerungen reagieren und besser einschätzen, welche Auswirkungen Verzögerungen der eigenen Aufgaben auf den Gesamtprozess haben (z.B. ob sie auf dem kritischen Pfad liegen) [Mue04, Cas05, SJHK06, Tan08].

Heutige Workflow-Management-Systeme, die zur Modellierung und Ausführungsunterstützung von Prozessen verwendet werden [JBS97, LR00, DAH05, Wic06], bieten meist eine Komponente für das Monitoring der Prozesse an. Ein gravierender Nachteil dieser ist jedoch die statische Art der Visualisierung: Die Prozesse werden in den Werkzeugen exakt so dargestellt, wie sie zuvor vom Prozessentwickler gezeichnet wurden. Insbesondere ist eine Anpassung an die Bedürfnisse des jeweiligen Betrachters nicht oder in nur sehr eingeschränktem Maße möglich. Abbildung 1.1 zeigt die Darstellung eines Prozesses in der Visualisierungskomponente von Lotus Workflow, einem bezüglich der Funktionalität der Prozessvisualisierung relativ mächtigen Workflow-Management-System. Dieses weist aber, wie die meisten anderen Werkzeuge, eine Reihe gravierender Schwächen auf [Bau04b].

- Wie erwähnt wird der Prozess von den Monitoring-Komponenten exakt so dargestellt, wie er bei der Modellierung definiert wurde. Diese Darstellung zeigt meist eine sehr technische Sicht, d.h. er enthält auch Aktivitäten, die z.B. nur für die Transformation von Daten benötigt werden und automatisch in den Systemen ausgeführt werden (z.B. Assign-Aktivitäten in WS-BPEL). Für den Endanwender ist eine solche Darstellung wegen der großen Anzahl für ihn irrelevanter Prozessobjekte dagegen sehr unübersichtlich.
- In der Prozessvisualisierung können nur Daten, die vom Workflow-Management-System verwaltet werden (Workflow-relevante Daten [WfM99]), dargestellt werden. Die für den Prozessbeteiligten wichtigen Applikationsdaten (z.B. Statusinformationen über den bearbeiteten Antrag) lassen sich integriert nicht anzeigen.

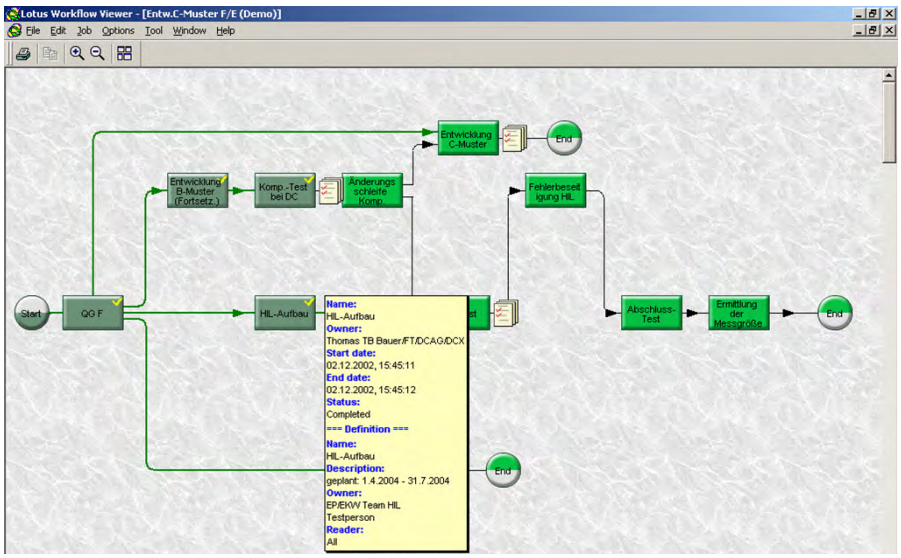


Abbildung 1.1: Darstellung eines laufenden Prozesses in Lotus Workflow

Für den in Abbildung 1.1 gezeigten Prozess wurde eine optimierte Visualisierung implementiert. Durch die Verwendung einfacher graphischer Mittel kann eine wesentlich anschaulichere Visualisierung erreicht werden (vgl. Abbildung 1.2). Neben den Daten, die schon in der ursprünglichen Darstellung enthalten waren, enthält die neue Visualisierung auch Applikationsdaten (z.B. Baureihe in der Überschrift). Bei der vorliegenden Implementierung handelt es sich jedoch um eine unflexible Lösung, die kaum Konfigurationsmöglichkeiten bietet, in diesem konkreten Fall aber ausreichend war [Bau04b]. Für eine generische, flexibel konfigurierbare Lösung ist zusätzlicher Aufwand erforderlich.

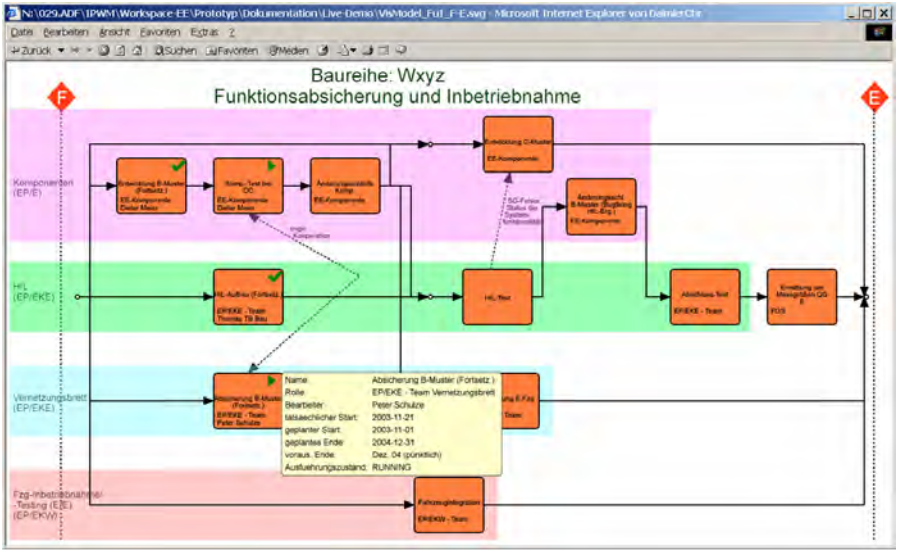


Abbildung 1.2: Optimierte Darstellung des Prozesses aus Abbildung 1.1

An einem komplexen Prozess, wie beispielsweise dem Entwicklungsprozess eines Fahrzeuges in der Automobilindustrie [Tie03, Ehr06, GHK06, LRZ06], sind viele verschiedene Nutzergruppen beteiligt: Fahrzeugkonstrukteure, Produktionsplaner, Einkaufsagenten, Zulieferer, Prozessverantwortliche, IT-Entwickler, IT-Verantwortliche, Manager, etc. Jede dieser Nutzergruppen betrachtet den Prozess aus einer anderen Perspektive und hat dementsprechend unterschiedliche Fragestellungen bzw. Motivation. Will man diese optimal befriedigen, genügt es nicht, eine einzige Form der Visualisierung anzubieten. Vielmehr benötigt man adaptierbare Darstellungen, die jeweils unterschiedliche Aspekte des Prozesses darstellen bzw. hervorheben können. Während für den Bearbeiter einer Aktivität alle Details (inkl. Dokumenten und Systemen) relevant sind, benötigt der Manager in der Regel eine abstrahierte Sicht auf den Prozess, aus der lediglich die groben Zusammenhänge ersichtlich sein sollten.

In Unternehmen werden häufig große Prozessmodelle zu Dokumentationszwecken angefertigt [Sch96a, SN00, Bro03, Sei06, DB07]. Abhängig von der Modellierungsmethodik ergeben sich mitunter riesige, die Größe von DIN A0 weit überschreitende Prozessmodelle („Wandtapeten“). Für die unterschiedlichen Prozessbeteiligten ist es sehr schwer, „ihren“ Anteil in der Gesamtvisualisierung des Prozesses zu identifizieren. Üblicherweise wird durch farbliche Hinterlegungen oder Anordnung der Aktivitäten in Bahnen versucht, die Orientierung der Betrachter zu unterstützen. Dringend erforderlich sind hier Mechanismen, um aus dem komplexen Gesamtprozessmodell übersichtliche, rollenspezifische Teilmodelle abzuleiten.

1.1 Grundlagen

Bevor wir die grundlegende Problemstellung und den Beitrag dieser Arbeit erörtern, behandeln wir hier zunächst einige grundlegende Begriffe aus dem Bereich des Prozessmanagements. Sie sind für das weitere Verständnis dieser Arbeit unabdingbar.

Ein *Geschäftsprozessmodell* besteht aus einer Menge von *Aktivitäten*, die ausgeführt werden sollen, um ein bestimmtes Geschäftsziel (z.B. Bearbeitung eines Antrages) zu erreichen. Das Modell beschreibt die Anordnungsbeziehung der Aktivitäten explizit durch Kontrollflusskanten (vgl. Abbildung 1.3). Ein mögliches Ziel einer Prozessmodellierung kann die Dokumentation

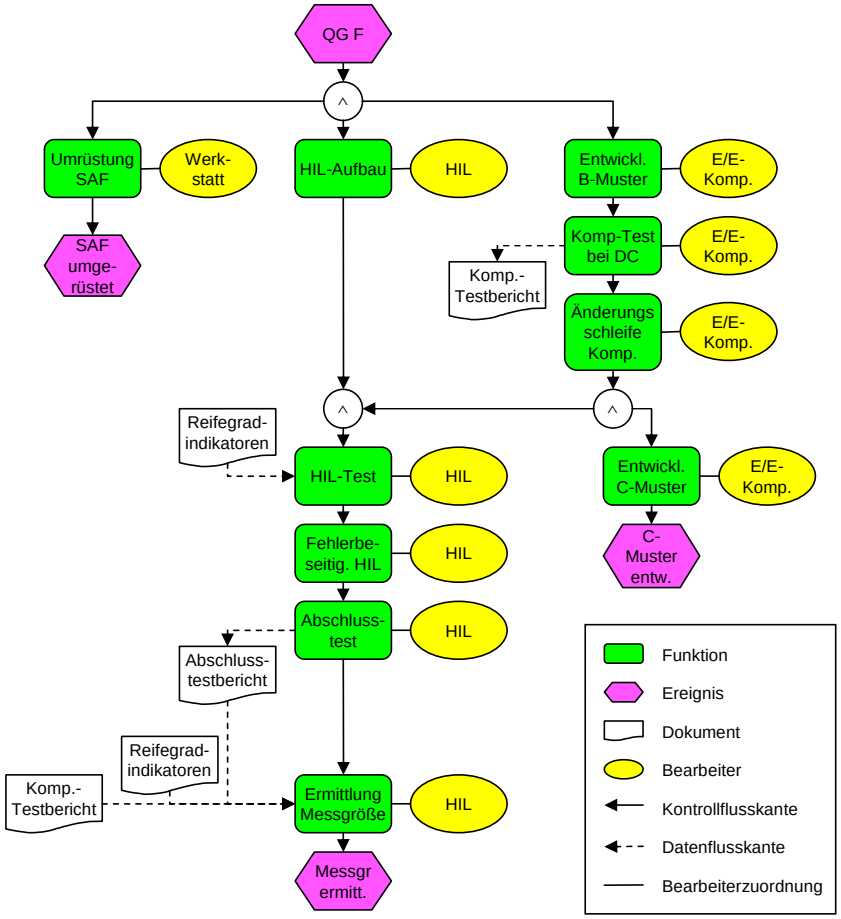


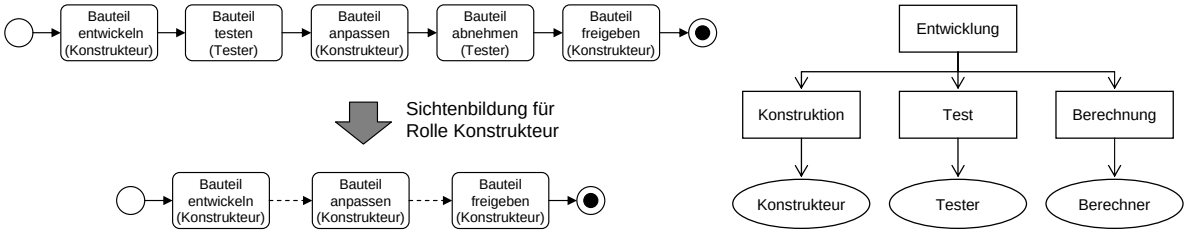
Abbildung 1.3: Beispiel für ein Geschäftsprozessmodell

der Abläufe sein. Soll die Ausführung der Geschäftsprozesse IT-technisch unterstützt werden, kommen häufig *Workflow-Management-Systeme* zum Einsatz. Diese ermöglichen die Trennung der Prozesslogik vom Anwendungscode. Die Prozesslogik wird einmalig im *Prozessschema* modelliert. Auf Basis des Prozessschemas wird für jeden konkret auszuführenden Geschäftsfall eine *Prozessinstanz* erzeugt, die dann vom Workflow-Management-System über ihre komplette Lebensdauer gesteuert und überwacht wird. Wir verwenden in dieser Arbeit den Begriff des *Prozessmodells* für Prozessschemata und Prozessinstanzen.

Ein Prozessmodell basiert auf einem *Prozessmetamodell*, welches die für die Modellierung zur Verfügung stehenden Elemente definiert. Neben dem zuvor angesprochenen Kontrollfluss, bestehend aus Aktivitäten und Kontrollflusskanten, bilden Prozessmodelle häufig weitere *Prozessaspekte* ab. Der *Datenfluss* beispielsweise spezifiziert mittels Datenelementen und Datenflusskanten, welche Daten von den Prozessaktivitäten gelesen bzw. geschrieben werden. Die *Bearbeiterzuordnungen* geben für jede Aktivität an, welche organisatorischen Einheiten bzw. Personen einer Aktivität zugeordnet sind und diese ausführen (vgl. Abbildung 1.3).

Ein Prozessmetamodell definiert zusammen mit einer Modellierungskonvention eine *Prozesssprache*. Sie gibt an, welche auf dem Metamodell basierenden Prozessmodelle gültige Prozesse beschreiben. Für die graphische Darstellung von Prozessen benötigen wir eine *Prozessnotation*. Sie besteht aus einer Menge von Symbolen, von denen üblicherweise ein bestimmtes Symbol einem Metamodellelement zugeordnet ist, d.h. eine Prozessnotation spezifiziert je ein Symbol für eine Aktivität, ein Datenelement, eine Kontroll- bzw. Datenflusskante etc. Diese klare Trennung zwischen Sprache und zugehöriger Notation kann bei den BPMI-Standards (Business Process Management Initiative) BPML (Business Process Modeling Language [Ark01]) und BPMN (Business Process Modeling Notation [Bus06]) beobachtet werden[1].

Häufig sind Prozessmodelle, die alle Aspekte abbilden, sehr groß und für den Benutzer unübersichtlich. Ein vereinfachtes Prozessmodell, aus dem Elemente entfernt wurden, bezeichnen wir als *Prozesssicht* (vgl. Abbildung 1.4a). *Prozessperspektiven* fokussieren dagegen auf einen bestimmten Prozessaspekt. Sie zeigen beispielsweise die Organisationsstruktur der am Prozess beteiligten Organisationseinheiten (vgl. Abbildung 1.4b).



(a) Prozessmodell mit zugehöriger Prozesssicht

(b) Organisatorische Perspektive

Abbildung 1.4: Beispiele für die Darstellung von Prozessaspekten

Die Begriffe, die sich um die Darstellung von (Teilen von) Prozessen drehen, werden in der Literatur teilweise auch in anderer Bedeutung verwendet (Perspektiven - Aspekte - Sichten). In dieser Arbeit verwenden wir diese Begriffe jedoch einheitlich mit der zuvor beschriebenen Semantik.

1.2 Problemstellung und Fragestellungen

Für prozessorientierte Applikationen besteht der konkrete Bedarf, Prozesse zu visualisieren. Dabei soll die Visualisierung an die Bedürfnisse der Betrachter anpassbar sind, etwa hinsichtlich

[1] Nach dem Zusammenschluss der Standardisierungsorganisationen BPMI und OMG wurde BPML durch BPDM (Business Process Definition Metamodel [OMG03]) abgelöst.

Detaillierungsgrad, verwendeter Symbole sowie Menge der angezeigten Daten. Allgemein ergeben sich folgende Forschungsfragestellungen:

Integration der heterogenen Prozesssysteme In verteilten Prozesslandschaften müssen vor der Visualisierung die Prozessmodelldaten aus den verschiedenen Quellsystemen integriert und homogenisiert werden. Für die Visualisierung von Prozessinstanzen müssen darüber hinaus sowohl aktuelle Statusdaten als auch Applikationsdaten betrachtet werden. Mögliche Quellsysteme umfassen neben Workflow-Management-Systemen auch konventionell implementierte, prozessorientierte Anwendungssysteme, Altapplikationen und Datenbanken.

Anpassung des Detaillierungsgrades der Prozessmodelle Prozessmodelle sind oftmals sehr detailliert und umfassen viele Schritte, d.h. sie enthalten Prozesselemente, die je nach Szenario für den Betrachter irrelevant sein können bzw. unnötig zur Komplexität der Prozessmodelle und somit zur Verunsicherung des Betrachters beitragen (z.B. zugeordnete Systeme oder Dokumente). Andererseits erfordern die einzelnen Benutzergruppen unterschiedlich detaillierte Prozessdarstellungen. Wir benötigen daher ein Konzept, um Details der Prozessmodelle bei Bedarf zu verbergen. Dadurch kann zum einen die Komplexität reduziert und zum anderen die Visualisierung besser an die Bedürfnisse der Betrachter angepasst werden. Dies erfordert natürlich auch Modifikationen an der Prozessstruktur.

Konfiguration der Darstellung Heutige Prozesswerkzeuge sind in Bezug auf die graphischen Gestaltungsfreiräume einer Prozessdarstellung sehr stark begrenzt. Häufig besteht jedoch der Wunsch, die verwendete Notation individuell an die Erfordernisse der Anwendung bzw. Benutzer anzupassen. Dazu muss die Prozessnotation frei konfigurierbar sein, sowohl was die Gestaltung der Symbole (Form und dargestellte Daten) als auch die flexible Zuordnung der Symbole zu Prozesselementen betrifft (abhängig von beliebigen Prozessdaten, Ausführungszuständen und Applikationsdaten).

Bereitstellung alternativer Darstellungsformen Konventionell werden Prozesse in Form eines Graphen dargestellt. Für bestimmte Anwendungen sind aber andere Formen der Darstellung besser geeignet, abhängig davon welcher Aspekt für den Betrachter besonders wichtig ist. Daher muss untersucht werden, welche alternativen Darstellungsformen realisierbar sind und sich für eine zielgruppengerechte Visualisierung anbieten.

Berechnung eines Prozessgraph-Layouts Prozesse werden in heutigen Systemen in der Regel so dargestellt, wie sie zuvor vom Prozessentwickler gezeichnet wurden. Mit der Umsetzung der oben beschriebenen Anpassungsmöglichkeiten ändern sich die Darstellungen dynamisch. Die zuvor gültigen Positionsdaten der Prozesselemente sind damit nicht mehr ideal. Ist die Prozessvisualisierung zudem dynamisch, d.h. der Detaillierungsgrad und das Aussehen hängen von Laufzeitdaten (z.B. Ausführungszustand des Prozesses) ab, wird eine Positionierung von Hand zu aufwendig. Wir benötigen daher adäquate Algorithmen zur Berechnung eines geeigneten Prozess-Layouts.

Eine systematische Aufbereitung der mit den jeweiligen Teilaspekten einhergehenden Anforderungen und technischen Problemstellungen erfolgt im Verlauf der Arbeit.

1.3 Beitrag der Arbeit

Proviado (Process Visualization in the Automotive Domain) ist Teil eines großen Forschungsprojektes, in dem Konzepte für eine umfassende Prozessmanagementinfrastruktur erstellt werden. In Proviado wird ein Rahmenwerk für die Visualisierung systemübergreifender Prozesse entwickelt. Dabei geht es vor allem um die Konzeption generischer, grundlegender Technologien sowie deren prototypische Umsetzung und weniger um eine Visualisierungs- oder Monitoring-Komponente für ein spezielles Prozessmodellierungs- oder Ausführungssystem. Diese Arbeit adressiert die zuvor aufgelisteten Fragestellungen. Einige davon sind für eine umfassende Gesamtlösung sehr wohl wichtig, aber eher praktischer Natur. Dazu zählen alle system- bzw. anwendungsspezifischen Aspekte (z.B. wie kann die Datenanbindung für System X realisiert werden), die nicht generisch gelöst werden können. In dieser Arbeit fokussieren wir auf diejenigen Fragestellungen, die auf einer abstrakten Ebene, unabhängig von speziellen Anwendungen, gelöst werden können. Der Hauptbeitrag dieser Arbeit liegt in den folgenden beiden Aspekten:

1. Proviado stellt Mechanismen zur Verfügung, mit denen sich sowohl die Visualisierung von Prozessschemata als auch von Prozessinstanzen an die Bedürfnisse der Betrachter anpassen lassen. Dazu führen wir ein mächtiges Sichtenkonzept für Prozesse ein, mittels dem sich die Komplexität der Modelle signifikant reduzieren lässt. Zum einen können beliebige Prozesselemente aus dem Modell entfernt und dadurch das Modell übersichtlicher gestaltet werden. Zum anderen erlaubt der Mechanismus, mehrere Prozesselemente zu einem neuen zusammenzufassen und so eine abstraktere Sicht auf den Prozess zu generieren. Die dabei zum Einsatz kommenden Operationen lassen sich mittels Parametern frei konfigurieren, so dass das resultierende Prozessmodell optimal an die Anforderungen des Betrachters angepasst werden kann.
2. Wir präsentieren einen mächtigen Mechanismus, um die graphische Darstellung eines Prozesses anzupassen. Die verwendete Notation kann frei definiert und flexibel angewendet werden. Die Konfiguration der Prozessvisualisierung erfolgt über ein Visualisierungsmodell. Dieses subsumiert alle für die Generierung der Darstellung erforderlichen Parameter.

Darüber hinaus werden weitere wichtige Aspekte, die für eine Gesamtlösung zur Visualisierung großer Prozesse erforderlich sind, betrachtet. Dazu zählt die Datenintegrationsproblematik, die vor allem bei verteilten Prozessapplikationen mit heterogenen Quellsystemen auftritt. Am Rande betrachten wir ferner die Herausforderungen, die in der automatischen Berechnung eines geeigneten Prozess-Layouts (d.h. der Positionierungsinformation für die Prozesselemente auf der Zeichenebene) liegen.

Proviado hat nicht zum Ziel, eine Empfehlung für eine ergonomisch optimale Visualisierung (d.h. Prozessnotation oder Darstellungsform) zu geben. In der Literatur existieren viele Arbeiten, die sich mit einer nutzergerechten Gestaltung von Bedienoberflächen befassen [Shn94, CR03]. [LK01] geht speziell auf Aspekte, die bei der Gestaltung der Benutzeroberfläche einer Prozessvisualisierung zu berücksichtigen sind, ein. In Proviado soll vielmehr ein Rahmenwerk konzipiert werden, das die erforderlichen Technologien zur Verfügung stellt, damit die Darstellung von Prozessen flexibel an die Anforderungen der jeweiligen Anwendung angepasst werden kann.

1.4 Gliederung und Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

Teil I erläutert die Herausforderungen an eine Prozessvisualisierung im Detail. Dazu motivieren wir in Kapitel 1 die Problematik. Kapitel 2 schildert die Problemstellung anhand konkreter Fallstudien aus der Praxis. Anschließend werden daraus Anforderungen abgeleitet. Das grobe Lösungskonzept unseres Ansatzes erörtern wir in Kapitel 3.

Im zweiten Teil der Arbeit stellen wir die Kernkonzepte von Proviado detailliert vor. Dabei diskutieren wir eng verwandte Ansätze zu den einzelnen Teilaspekten bereits in den jeweiligen Kapiteln. Kapitel 4 führt die Grundlagen der Proviado-Prozesssichten ein. Mit dem vorgestellten Mechanismus lassen sich Prozesse strukturell adaptieren und durch Entfernen bzw. Zusammenfassen von Prozesselementen vereinfachen. Dabei decken wir alle Prozessaspekte ab (Kontrollfluss, Datenfluss, Bearbeiterzuordnungen). Aufbauend auf diesen Grundlagen präsentieren wir in Kapitel 5 mächtige Operationen für die Bildung von Sichten. Sie erlauben es, Sichten auf Prozessen einfach und parametrisierbar zu definieren. Durch diese Konfigurationsparameter der Sichtenbildung geben wir dem Benutzer Werkzeuge an die Hand, die Prozessmodelle optimal an seine Vorstellungen zu adaptieren. In Kapitel 6 stellen wir mit dem Proviado-Template-Mechanismus ein Konzept zur graphischen Adaption von Prozessvisualisierungen vor. Die Prozessdarstellung kann mit diesem Konzept durch die flexible Definition und Verwendung der Prozessnotation frei gestaltet werden. Weitere wichtige Aspekte wie die Integration von Prozessmodell-, Laufzeit- und Applikationsdaten sowie die Berechnung eines möglichst optimalen Prozessgraph-Layouts thematisieren wir in Kapitel 7.

Der dritte Teil der Arbeit beschreibt das Zusammenspiel der verschiedenen Konzepte (Kapitel 8). Hier wird unter anderem der Gesamtablauf einer Prozessvisualisierung beschrieben und anhand eines Beispiels illustriert.

Teil IV schließt die Arbeit ab. Kapitel 9 diskutiert verwandte Ansätze zur Prozessvisualisierung aus Wissenschaft und Praxis. Kapitel 10 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick.

2

Fallstudien und Anforderungen

2.1 Motivation

Das Projekt Proviado (Process Visualization in the Automotive Domain) ist Teil eines größeren Forschungsprojekts zur Entwicklung einer Infrastruktur für partner- bzw. organisationsübergreifendes Prozessmanagement. Abbildung 2.1 zeigt die hierbei insgesamt behandelten Themen. Soweit möglich wird bei der Realisierung der einzelnen Komponenten auf existierende Technologien zurückgegriffen, d.h. nur wenn keine brauchbaren Werkzeuge existieren, werden eigene Konzepte entwickelt. Dies gilt insbesondere auch für die in Proviado entwickelte Prozessvisualisierungskomponente. Daneben finden weitere konzeptionelle Arbeiten im Bereich des Arbeitslistenmanagements und der prozessorientierten Applikationsintegration statt [Bau05, Buc07, UB08].

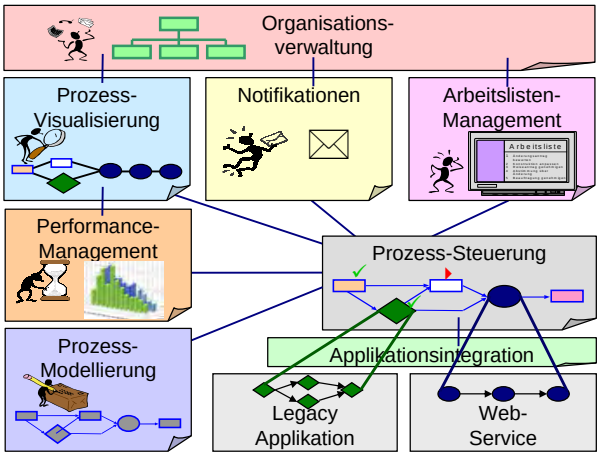


Abbildung 2.1: Komponenten der Prozessmanagementinfrastruktur

Im Bereich der Prozessmodellierung werden fortschrittliche Konzepte zur Modellierung von Prozessvarianten entwickelt [HBR07, HBR08a, HBR08c, HBR08b]. Eine Einbindung eines Werkzeugs für das Process-Performance-Management wird in [BB07a] diskutiert. Im Projekt Proviado wird ein umfassendes Konzept für die Visualisierung system- und partnerübergreifender Prozesse (Schemata wie Instanzen) entwickelt. Dies bezieht sowohl die Visualisierung von Prozessschemata als auch von Prozessinstanzen mit ein. Letztere müssen bezüglich der Darstellung angepasst werden, wenn sich der Status der visualisierten Instanz ändert.

In diesem Kapitel beschreiben wir zunächst Praktische Beispiele und Szenarien anhand von Fallstudien. Die daraus abgeleiteten Anforderungen für die Visualisierung von Prozessen werden im Anschluss daran behandelt.

2.2 Fallstudien und -beispiele

Im Folgenden stellen wir drei Fallstudien aus der Praxis[1] vor, die zu Beginn des Projekts Proviado durchgeführt wurden. Ziel war es, Anforderungen an die Prozessvisualisierung seitens der Anwender aufzunehmen und besser zu verstehen. Dabei abstrahieren wir von einer Darstellung aller Prozess- bzw. Projektdetails, da dies den Umfang der vorliegenden Arbeit sprengen würde. Nichtsdestotrotz geben die Beispiele die vorgefundene Realität authentisch wieder.

2.2.1 Fallstudie I: Änderungsmanagement in der Produktentwicklung

Das Änderungsmanagement dient der Koordination von Produktänderungsvorhaben (Change Request, CR) von der Fahrzeugentwicklung über die Produktionsplanung bis hin zu Änderungen, die sich auf die laufende Produktion auswirken. Durch die hohe Komplexität des zu entwickelnden Produktes sowie die zahlreichen Abhängigkeiten seiner Teilkomponenten müssen alle Änderungen (z.B. an Konstruktion oder Design) ab einem bestimmten Zeitpunkt im Entwicklungsprozess einen strukturierten Änderungsprozess durchlaufen. Dadurch wird sichergestellt, Dass die Änderung später reibungslos erfolgen kann. Beispielsweise sollte bei einer Erhöhung der elektrischen Leistungsaufnahme eines Gerätes der Entwicklungsverantwortliche der Lichtmaschine informiert werden, sodass diese gegebenenfalls angepasst werden kann und es später im Betrieb zu keiner Überlastung kommt. Ebenso müssen wirtschaftliche und produktionstechnische Aspekte vor der Umsetzung einer Änderung geprüft werden.

Eine vereinfachte Darstellung eines Änderungsmanagement-Prozesses zeigt Abbildung 2.2. Sie vermittelt neben dem Kontrollflussaspekt einen Eindruck zu prozessrelevanten Daten sowie den Akteuren, die an der Bewertung eines Änderungsvorhabens beteiligt sind. In Phase I wird das Änderungsvorhaben angelegt und die von der Änderung betroffenen Teile werden identifiziert. In Phase II findet die technische Bewertung des Änderungsvorhabens statt. Die automatischen Schritte 4 und 5 ermitteln dazu die Teilenummern aus dem Produktdatenmanagement-System (PDM-System) und transformieren diese in das benötigte Format. Anschließend werden die Änderungen durch verschiedene Entwicklungsexperten bezüglich ihrer technischen Realisierbarkeit bewertet (Schritte 6–8). Der CR-Manager aggregiert diese Expertenmeinungen zu einer Gesamtstellungnahme (Schritt 9), welche dann durch eine automatische Aktivität archiviert

[1] Die Darstellungen der Beispiele erfolgt verfremdet, um den Vertraulichkeitsanforderungen zu entsprechen.

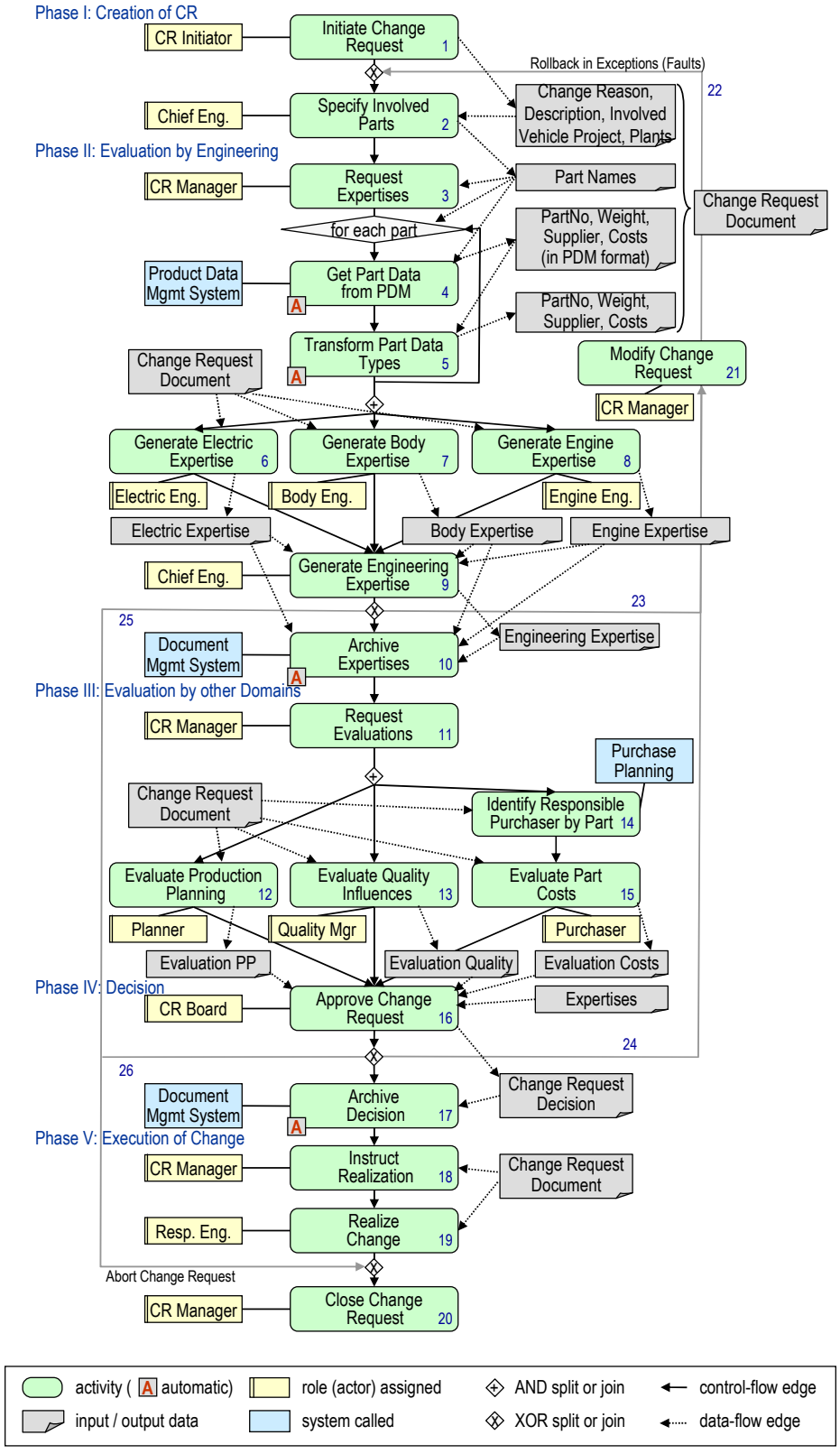


Abbildung 2.2: Vereinfachtes Prozessmodell für das Änderungsmanagement

wird (Schritt 10). Phase III beurteilt in einer zweiten Begutachtungsrunde die wirtschaftliche, organisatorische und zeitliche Machbarkeit des Änderungsvorhabens. Dies erfolgt durch den Produktionsplanungsbereich, die Qualitätsabteilung und den Einkauf, der dazu Angebote von Zulieferern anfordert (Schritt 15). Das CR-Gremium entscheidet auf Grundlage der diversen Stellungnahmen der Fachbereiche in Schritt 16 über die Realisierung. Schließlich erfolgt in Phase V die konstruktive und produktive Umsetzung des Änderungsvorhabens. An diversen Stellen innerhalb des Prozesses ist eine Modifikation des Änderungsvorhabens (Schritt 21) mit einem Rückwärtssprünge zu Schritt 2 möglich.

[Tan05, TF05, TW07, Tan08] untersuchen im Detail, welche Unterstützung dem Benutzer des Änderungsmanagement-Systems durch prozess-orientierte Hilfesysteme gegeben werden kann. Eine Studie belegt auch die positiven Auswirkungen einer Prozessvisualisierung auf die Arbeitsleistung der Bearbeiter.

Eine wichtige Anforderung an die Prozessvisualisierung im Änderungsmanagement ist die Darstellung des aktuellen Prozessstatus für die Prozessbeteiligten. Dabei sollen aus fachlicher Sicht irrelevante, technische Aktivitäten (z.B. Schritte 4, 5, 10, 17) ausgeblendet und die bereits abgeschlossenen Aktivitäten zusammengefasst werden. Besonders wertvoll für den Betrachter sind Zusatzinformationen über das aktuelle Änderungsvorhaben, wie z.B. die Liste der von der Änderung betroffenen Teile. Eine Verlinkung der prozessrelevanten Dokumente aus der Prozessdarstellung heraus, erleichtert den Zugriff auf diese.

Während Prozessbeteiligte meist eine detaillierte Sicht auf den Gesamtprozess wünschen, soll die Visualisierung für das Management keine Detailaktivitäten mehr enthalten, sondern die Aktivitäten zu Prozessphasen zusammengefasst darstellen. Eine derartige Darstellung, ergänzt um aktuelle Prozesskennzahlen, ermöglicht einen schnellen Überblick über den aktuellen Status des Änderungsvorhabens.

[Ada04] dokumentiert die Anforderungen an eine Prozessvisualisierung im Änderungsmanagement basierend auf Interviews mit Prozessbeteiligten.

2.2.2 Fallstudie II: Werkstattprozesse

Werkstattprozesse beschreiben Abläufe mit Kundenbeteiligung in den Niederlassungen und Werkstätten. Beispiele sind die Abläufe zur Inspektionsuntersuchung des Kraftfahrzeugs oder zur Fehlersuche und -behebung aufgrund einer Symptombeschreibung des Kunden. Folgende grobe Phasen werden durchlaufen:

1. **Erstkontakt** Der Kunde meldet sich per Telefon, E-Mail oder vor Ort, um einen Termin zu vereinbaren.
2. **Auftragsannahme** Das Fahrzeug wird übergeben. In einem Kundengespräch werden die durchzuführenden Arbeiten besprochen und evtl. eine erste Diagnose aufgrund der Problembeschreibung des Kunden gestellt.
3. **Werkstattbefund** In der Werkstatt werden der Fehler analysiert und die gegebenenfalls benötigten Ersatzteile identifiziert.

4. **Auftragserweiterung** Sollte die Diagnose ein bislang unerkanntes Problem ergeben bzw. übersteigt der notwendige Aufwand die Vorgaben des Kunden, wird bei diesem nachgefragt und der Auftrag entsprechend erweitert.
5. **Teilebereitstellung** Die für eine Reparatur benötigten Ersatzteile werden beschafft.
6. **Reparatur** Die Ersatzteile werden eingebaut und auf Funktion geprüft. Bei Bedarf findet eine Probefahrt statt.
7. **Reparaturdokumentation** Die durchgeführten Arbeiten werden dokumentiert.
8. **Auftragsabschluss** Mit der Übergabe des Fahrzeugs an den Kunden endet der Prozess.

Abbildung 2.3 zeigt einen Prototyp für die graphisch ansprechende Visualisierung des Prozesses für den Werkstattleiter. In der Darstellung sind sowohl die groben Prozessschritte erkennbar als auch die aktuelle Anzahl der Fahrzeuge, die sich in der jeweiligen Phase befinden. Die Visualisierung zeigt auch die wichtigsten Kennzahlen für die Qualitätsbewertung.

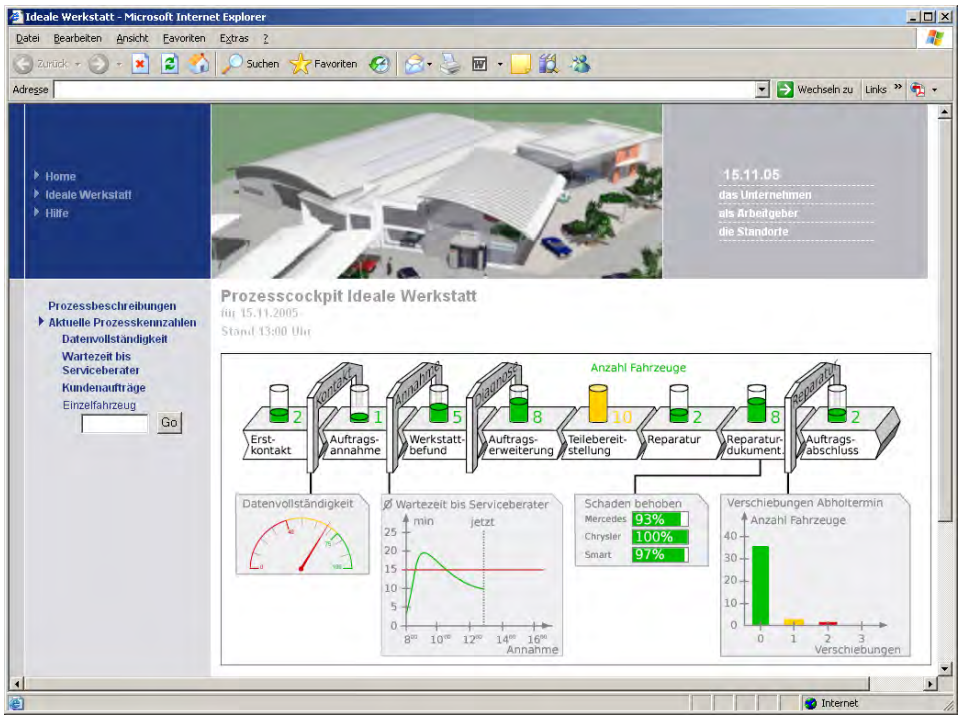


Abbildung 2.3: Visualisierung des Werkstattprozesses

Eine kompakte Visualisierung der aktuell in der Werkstatt befindlichen Fahrzeuge und deren Fortschritt im Prozess gibt Abbildung 2.4. Für eine derartige Darstellung wird der Prozess in wenige Phasen unterteilt. Jedes Fahrzeug bzw. jede Prozessinstanz kann aufgrund der aktuellen Ausführungsinformationen einer Phase zugeordnet werden. In Abbildung 2.4 verwenden wir Farben zur Kodierung der Phasen.

Neben den Werkstattleitern, die aus den gezeigten Darstellungen den größten Nutzen ziehen können, existieren weitere Personen, für die eine Prozessdarstellung hilfreich ist. In [Mac04]

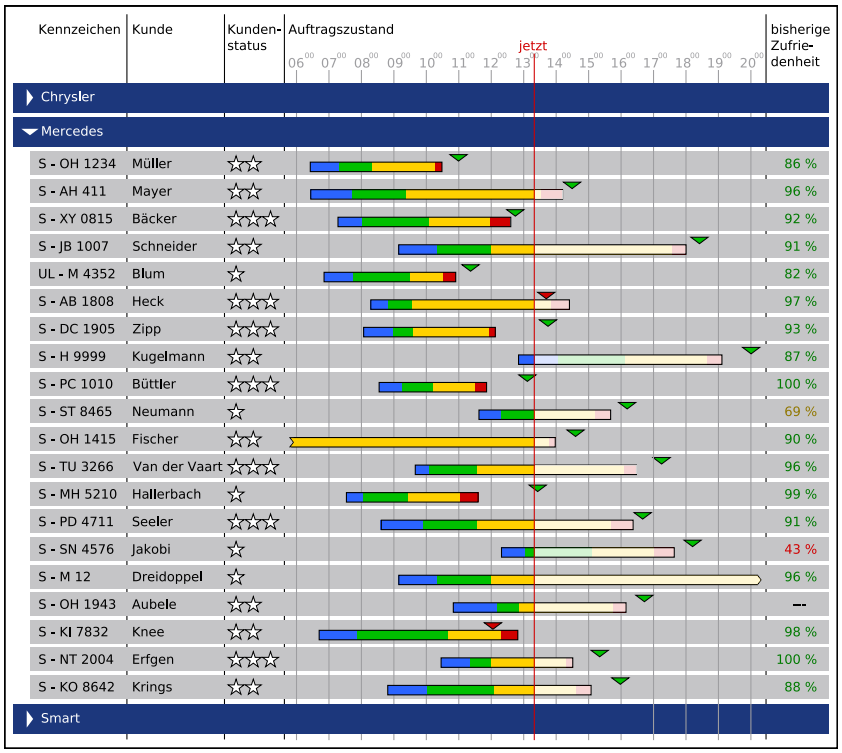


Abbildung 2.4: Überblick über den aktuellen Zustand der Instanzen

werden entsprechende Benutzergruppen identifiziert und die jeweiligen Anforderungen an eine Visualisierung der Werkstattprozesse untersucht. Im Folgenden geben wir einen Überblick über die wichtigsten prozessbeteiligten Gruppen und deren Anforderungen.

Management Das Management, welches durch die Niederlassungsleiter verkörpert wird, benötigt eine einfache, übersichtliche Darstellung der Werkstattprozesse. Idealerweise lassen sich Prozesskennzahlen in die Prozessdarstellung integrieren, die Ausführungsdaten aus dem Prozess aggregieren. So lässt sich die Qualität der Prozesse beispielsweise anhand der Anzahl verschobener Abholtermine bewerten.

Prozessverantwortliche Der Prozessverantwortliche – hier der Werkstattleiter – benötigt ein detailliertes Bild aller Prozesse. Aus den Darstellungen müssen neben dem Kontrollfluss auch alle weiteren Prozessaspekte wie Datenfluss, relevante Dokumente (z.B. Reparaturanweisungen oder Ersatzteilbestellformulare), IT-Systeme und Bearbeiter erkennbar sein. Neben den reinen Prozessausführungsdaten sollten zudem Kontaktdaten der Bearbeiter oder im Prozess verwendete Dokumente aus Visualisierung zugänglich sein. Zusätzlich sollen sich die Darstellungen um Prozesskennzahlen erweitern lassen.

Prozessbeteiligte Die Prozessbeteiligten, im vorliegenden Fall die Mitarbeiter der Werkstatt, des Empfangs und der Verwaltung, müssen aus den Prozessdarstellungen in erster Linie „ihre“ Aktivitäten entnehmen können. Dabei sollten relevante Fristen, Dokumente oder Systeme visualisiert werden. Für ein besseres Verständnis und eine bessere Einordnung der eigenen Tätigkeiten sollten auch die „Schnittstellen“ zu anderen Prozessbeteiligten bzw. die Vorgänger- bzw. Folgeaktivitäten der „Schnittstellen“ dargestellt werden.

Schulungen Für die Verwendung der Prozessvisualisierungen im Rahmen von Schulungen müssen neben einer Überblicksdarstellung der Werkstattprozesse detaillierte Ablaufdarstellungen unterstützt werden. Technische Details der implementierten Prozesse, wie Datentransformationen oder andere automatisch ablaufende Schritte, sollen jedoch ausgeblendet werden.

IT-Systementwickler Die dokumentierten Werkstattprozesse werden häufig als Grundlage für die Implementierung von IT-Systemen verwendet. Dementsprechend müssen die Darstellungen für diese Anwendergruppe sämtliche Details (inkl. Datenfluss, Dokumenten und Systemen) enthalten, um die Abläufe im Werkstatt-System abzubilden.

Zur Überwachung von Prozessinstanzen sowie zur Berechnung und Darstellung von Prozesskennzahlen (Key Performance Indicator, KPI) existieren, neben allgemeinen Business-Intelligence-Lösungen [IBM07, SAP07] speziell auf Prozesse zugeschnittene Lösungen. Sie werden unter dem Schlagwort Process-Performance-Management subsumiert. Dabei werden die Ausführungsdaten gleichartiger Prozessinstanzen zu Kennzahlen aggregiert und zum Beispiel in Form von Säulen- oder Balkendiagrammen visualisiert. Eine reine Kennzahlendarstellung (ohne Prozessmodell) ist vor allem dann interessant, wenn sehr viele Instanzen durchlaufen werden. Für langlaufende, komplexe Prozesse mit wenigen Instanzen ist dagegen vor allem eine kombinierte Darstellung von Prozessmodell und aktuellen Statusinformationen relevant.

2.2.3 Fallstudie III: Produktionsplanung

Der Produktionsplanungsprozess (PP-Prozess) ist Teil des gesamten Fahrzeugentwicklungsprozesses. Er wird für jedes Fahrzeugprojekt durchlaufen. In dem betrachteten Abschnitt beschreibt der PP-Prozess die notwendigen Tätigkeiten für die Planung der Produktionsmittel (z.B. Press- und Gießwerkzeuge). Ferner dokumentiert er den Fluss der ca. 50 relevanten Dokumente. Als unterstützende Dokumentation existiert ein Prozesshandbuch, in dem alle Aktivitäten detailliert beschrieben sind. Die Prozessdokumentation in dieser Form dient vorrangig als Diskussionsgrundlage, Schulungsunterlage oder Aufgabenbeschreibung für Mitarbeiter. Eine Ausführungsunterstützung steht momentan nicht im Fokus.

Zum Zeitpunkt der Fallstudie ist der Prozess mit Hilfe der OMEGA-Methode [Jan05] dokumentiert und steht den Anwendern in Form einer ca. 1,5×5m großen „Wandtapete“ zur Verfügung. Im Rahmen einer Harmonisierungsinitiative sollen zukünftig alle Planungsprozesse einheitlich dokumentiert und über das Intranet verfügbar gemacht werden. Einen Eindruck der verwendeten Prozessbeschreibungssprache vermittelt Abbildung 2.5. Mit der verwendeten OMEGA-Methode lassen sich die komplexen Abhängigkeiten zwischen Aktivitäten und Dokumentenflüssen relativ kompakt darstellen. Die im Prozess verwendeten IT-Systeme können ebenfalls dokumentiert werden. Im PP-Prozess sind die Aktivitäten horizontal in „Swimlanes“ angeordnet, von denen jede für einen Zuständigkeitsbereich einer Abteilung steht. Zusätzlich wird die ausführende Abteilung durch die Hintergrundfarbe der Aktivitäten kodiert. Horizontal sind die Aktivitäten entlang der Prozessphasen des globalen Produktentstehungsprozesses angeordnet. Die für die Planer wichtigste Information, die sie dem PP-Prozess entnehmen können, betreffen die Schnittstellen zwischen den Abteilungen. Typische Fragen, die sich aus dieser Darstellung beantworten lassen, sind: Welche Abteilung liefert Material für meine anstehende Aktivität oder an welche Abteilung muss Dokument X weitergeleitet werden.

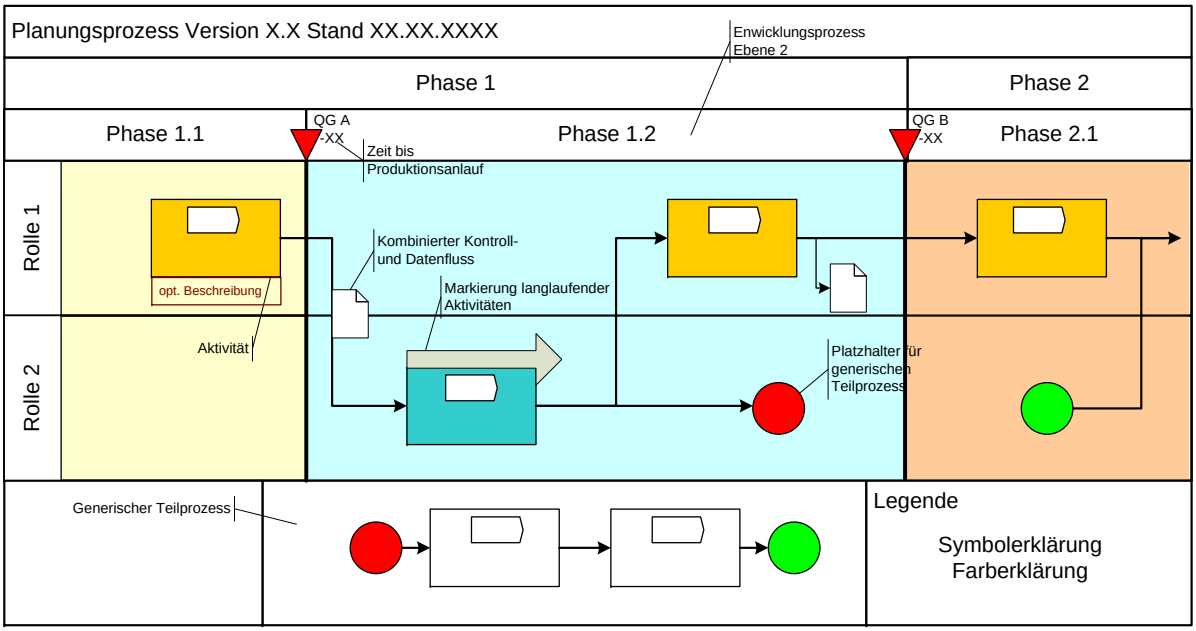
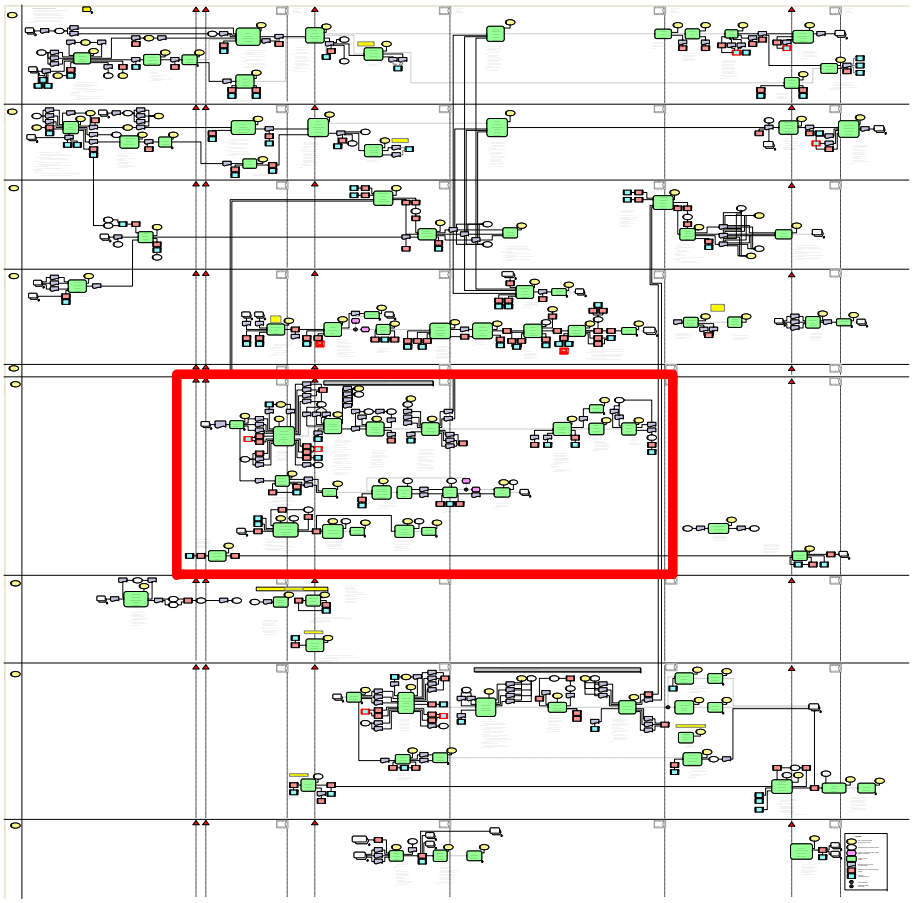


Abbildung 2.5: Schematische Darstellung der Modellierungsmethodik des Planungsprozesses

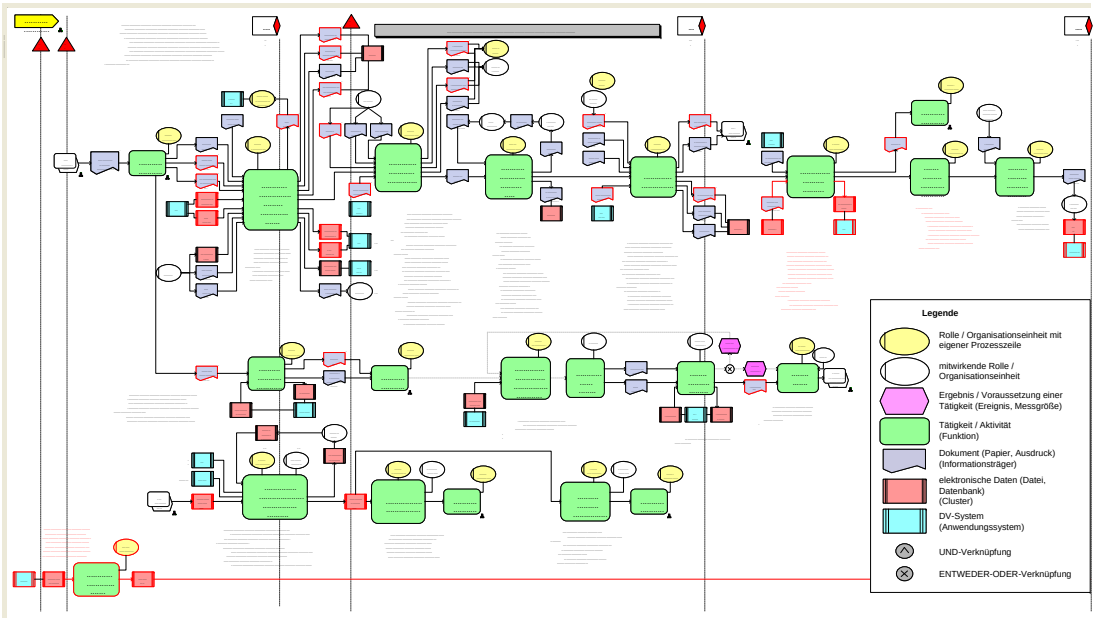
[Gär04] ermittelt mittels einer Reihe von Experteninterviews Verbesserungsvorschläge für die zukünftige Darstellung des PP-Prozesses. Im Bezug auf die Prozessvisualisierung bestehen folgende Zielvorstellungen an eine Visualisierung:

- Visualisierung der Schnittstellen zwischen Abteilungen.
- Darstellung des Dokumentenflusses zwischen Abteilungen.
- Verwendung einer intuitiven Prozessnotation (z.B. ein Rechnersymbol für IT-Systeme).
- Verlinkung der Tätigkeitsbeschreibung aus den dargestellten Aktivitäten.
- Verlinkung der Dokumente aus der Prozessvisualisierung, um einen einfachen und schnellen Zugriff auf die Standarddokumente des PP-Prozesses zu erhalten.
- Ableitung rollenspezifischer Darstellungen, indem aus dem Gesamtprozess Submodelle extrahiert werden, die nur die Aktivitäten einer Rolle bzw. Abteilung enthalten.

Inzwischen wurde der PP-Prozess von der OMEGA-Methode auf ARIS umgestellt. Dadurch können die Prozessmodelle mittels eines Export-Werkzeugs (ARIS Business Publisher) im Intranet zur Verfügung gestellt werden. Dieser Verbesserung der Online-Verfügbarkeit stehen einige Nachteile aus Sicht der Visualisierung gegenüber. Während die OMEGA-Methode eine sehr kompakte Repräsentation der Geschäftslogik erlaubt, sind für die Abbildung desselben Vorgangs in ARIS wesentlich mehr Objekte erforderlich (vgl. Abbildung 2.6). In ARIS kann beispielsweise der Bearbeiter nicht durch eine Farbe kodiert werden, sondern hängt als Zusatzobjekt mittels einer Kante an der Aktivität. Um die Modelle dennoch übersichtlich zu halten, ist ein Mechanismus erforderlich, mit dem sich nicht benötigte Detailsymbole ausblenden und deren Information in anderer Form oder an anderer Stelle zugänglich machen lassen.



(a) Ausschnitt des Gesamtprozesses



(b) Detailprozess einer Abteilung (Teilausschnitt aus Abbildung 2.6a)

Abbildung 2.6: Beispiel eines Prozessmodells aus der Produktionsplanung modelliert in ARIS

2.2.4 Ergebnisse der Fallstudien

Neben den hier vorgestellten Fallstudien wurden weitere Anwendungsfälle aus dem Automobilbereich analysiert. Charakteristisch für alle ist, dass es sich um langlaufende Prozesse (Durchlaufzeiten im Bereich von mehreren Tagen bis Monaten) mit einem hohen Grad an Benutzerinteraktion handelt. Somit konzentrieren sich viele Anforderungen auf die Darstellung eines Prozesses (bzw. einer Prozessinstanz). Im Gegensatz dazu existieren in anderen Bereichen Prozesse, die extrem kurzlebig und hochgradig automatisiert sind, dafür aber in großer Zahl ausgeführt werden. Beispiele sind Finanztransaktionen oder Call-Center-Prozesse. Diese können mit den Proviado-Konzepten ebenfalls visualisiert werden, jedoch können dort zusätzliche Anforderungen auftreten, die sich in unserem Kontext nicht stellen.

Bei den betrachteten Fällen einer Instanzvisualisierung handelt es sich allesamt um systemübergreifende Prozesse. Insbesondere existiert in der Praxis keine übergeordnete Ausführungseinheit, welche die Koordination der Subsysteme übernimmt, d.h. der Gesamtprozess ist fragmentiert über verschiedene (autonome) Systeme verteilt. Für die Visualisierung bedeutet dies, dass kein zentrales System existiert, das die Ausführungsdaten für den Gesamtprozess zur Verfügung stellt. Vielmehr müssen die Daten aus den diversen Quellsystemen abgefragt bzw. importiert werden.

Zwei wichtige Aspekte der Prozessvisualisierung sind die jeweilige Zielgruppe der Visualisierung und die gewünschte Darstellungsform.

2.2.4.1 Zielgruppen der Visualisierung

Wie anhand der Fallstudien deutlich gemacht wurde, sollte eine Prozessvisualisierung auf die Bedürfnisse des jeweiligen Betrachters zugeschnitten sein. So unterscheiden sich die Visualisierungsanforderungen eines Managers fundamental von denen eines IT-Mitarbeiters, der zum Beispiel auf Grundlage des Prozessmodells ein System implementieren soll. Verschiedene Arbeiten haben sich inzwischen der Untersuchung der Visualisierungsanforderungen der verschiedenen Zielgruppen gewidmet [BLM04, Gär04, Mac04, Mol06]. Im Speziellen untersucht [Mac04] die Zielgruppen der Fallstudie II (Werkstattprozesse), während [Gär04] sich der Fallstudie III (Produktionsplanung) widmet.

Im Folgenden geben wir einen Überblick über die verschiedenen Rollen, deren Aufgabe im Prozesskontext und deren Anforderungen an einer Prozessvisualisierung. Prinzipiell unterscheiden wir zwischen Produzenten und Konsumenten einer Visualisierung. Zu den Produzenten zählen neben den Prozessmodellierern die Visualisierungsdesigner. Zu den Konsumenten einer Visualisierung gehören neben den unmittelbar Prozessbeteiligten auch Manager und IT-Systementwickler, die die Visualisierung für die Erfüllung ihrer Aufgaben verwenden.

Prozessmodellierer (auch *Prozess-Designer*) Er erstellt die Prozessmodelle mit einem Modellierungswerkzeug. Die Visualisierung der Prozesse ist für den Modellierer meist von untergeordneter Bedeutung, da er alle benötigten Informationen aus dem Modellierungswerkzeug bezieht.

Visualisierungsdesigner Der Visualisierungsdesigner gestaltet die Prozessdarstellung. Er definiert, welche Prozesse, auf welche Art und Weise und in welchem Detaillierungsgrad visualisiert werden sollen. Heutige Visualisierungswerkzeuge stellen die Prozesse meist in exakt derselben Form dar, wie sie zuvor durch den Modellierer gemalt wurden. Eine Unterscheidung der beiden Rollen „Prozessmodellierer“ und „Visualisierungsdesigner“ ist in diesen Werkzeugen nicht vorgesehen.

Prozessbeteiligter Unter einem Prozessbeteiligten verstehen wir eine Person, die mindestens eine Aktivität im Prozess bearbeitet. Für diesen Personenkreis ist eine Darstellung wichtig, aus der sie die eigenen Aktivitäten im Kontext direkt benachbarter Aktivitäten einordnen können. Weiter entfernte Aktivitäten sind dabei weniger relevant. In jedem Fall aber sollte die Darstellung eine Einordnung der eigenen Aktivitäten in den Kontext des Gesamtprozesses erlauben (z.B. *Was geschieht mit den Ergebnissen meiner Arbeit im weiteren Verlauf des Prozesses?*). Gleichzeitig benötigt der Prozessbeteiligte einen Überblick über den Status des Prozesses (z.B. über anstehende Aufgaben).

Prozessverantwortlicher Der Prozessverantwortliche hat die Aufsicht über den Prozess. Kommt es zu Problemen in dessen Ablauf (z.B. Verzögerungen in der Bearbeitung der Aktivitäten), muss er eingreifen (z.B. Warnmeldung an Mitarbeiter). Dazu benötigt er stets aktuelle Informationen über den Zustand der Prozessinstanzen. Um die Gesamtleistung aller Prozessinstanzen zu steuern, sind aggregierte Darstellungen aller Instanzen mit den entsprechenden Prozesskennzahlen (z.B. Laufzeit und Kosten) wünschenswert. Im Gegensatz zum Manager benötigt er aber auch eine Darstellung, die die nötigen Details für eine Prozessoptimierung enthält.

Manager Der Manager steht in der Unternehmenshierarchie über dem Prozessverantwortlichen. Naturgemäß interessieren ihn Prozessdetails weniger, vielmehr wünscht er aggregierte Darstellungen mit aussagekräftigen Kennzahlen (bzw. im Extremfall lediglich Kennzahlen) sind gefragt.

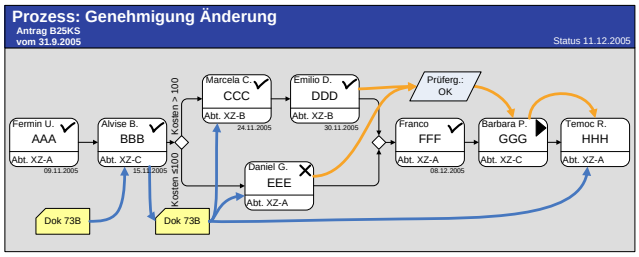
IT-Systementwickler Auf Grundlage eines Prozessmodells implementiert der IT-Systementwickler das entsprechende prozessorientierte Informationssystem. Dementsprechend benötigt er sämtliche Details einer Prozessdarstellung (inklusive Datenelementen, Datenflüssen, Bearbeiterzuordnungen und beteiligten IT-Systemen).

Die Konsumenten einer Prozessvisualisierung fassen wir im Folgenden unter dem Begriff *Prozessbetrachter* zusammen.

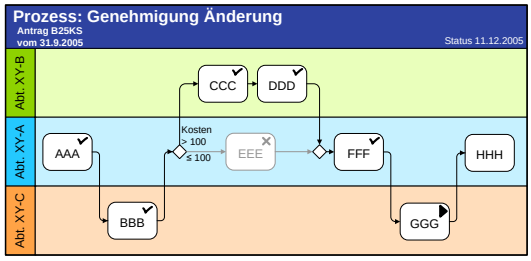
2.2.4.2 Darstellungsformen

Prozesse und zugehörige Informationen werden oftmals in Form eines Prozessgraphen dargestellt. Je nach Detaillierungsgrad umfasst dieser neben dem Kontrollfluss weitere Aspekte wie Datenfluss, Bearbeiterzuordnungen, IT-Systeme, zeitliche Beschränkungen (z.B. Aktivitätendauern, Fristen oder zeitliche Abstände von Aktivitäten) und Laufzeitdaten (z.B. Ausführungszustand, Bearbeitungszeiten, tatsächlicher Bearbeiter, Applikationsdaten). Unsere Fallstudien haben auch gezeigt, dass neben der Prozessgraphdarstellung (vgl. Abbildung 2.7a) weitere Darstellungsformen für Prozessinformationen gewünscht werden. In [Mol06] haben wir mögliche

Darstellungsformen untersucht, die aus den in den Prozessmodellen hinterlegten Informationen generiert werden können. Abbildung 2.7 zeigt einige Beispiele hierfür.



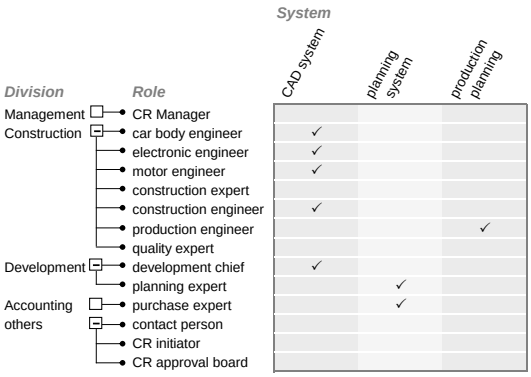
(a) Prozessgraph



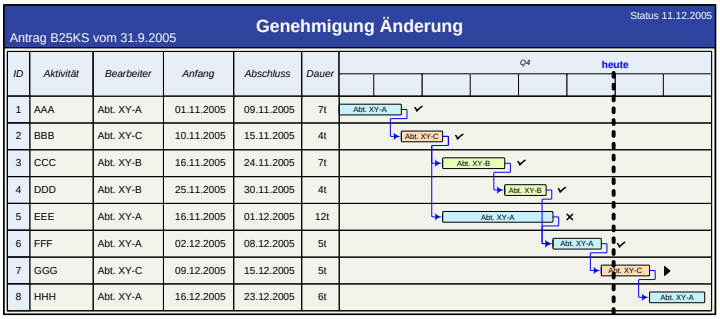
(b) Swimlane

Activity	Role	Input documents	Output documents
initiate change request	CR initiator		change request
determine CR manager	contact person		
instruct expertise	CR manager		
parallel block			
generate expertise	car body engineer	change request	expertise->car body
generate expertise	electronic engineer	change request	expertise->electronic
generate expertise	motor engineer	change request	expertise->motor
generate expertise	development chief	change request expertise	expertise
request evaluation	CR manager		
parallel block			
provide evaluation	planning expert	change request expertise	
provide evaluation	purchase expert	change request expertise	
provide evaluation	quality expert	change request expertise	
request comments	CR manager		
parallel block			
provide comments	planning expert	evaluation change request	
provide comments	planning expert	evaluation change request	
provide comments	construction expert	evaluation change request	
provide comments	quality expert	evaluation change request	
approve CR	CR approval board	evaluation change request	
start realization	CR manager	change request	
realize CR (construction)	construction engineer		
start realization	CR manager		
realize CR (production)	production engineer		
conclude CR	CR manager		

(c) Tabelle



(d) Matrix-Darstellung



(e) Gantt-Diagramm

Abbildung 2.7: Verschiedene Darstellungsformen für Prozessinformationen

Prozessgraph (vgl. Abbildung 2.7a) Der Prozessgraph fokussiert bei der Darstellung auf den Kontrollflusses. Je nach Bedarf kann die Darstellung um weitere Details wie Datenfluss und Bearbeiterzuordnungen angereichert werden. Die Komplexität der Visualisierung kann dadurch jedoch schnell steigen, so dass unbedarfte Betrachter oft Probleme mit allzu detaillierten Prozessdarstellungen haben.

Swimlane (vgl. Abbildung 2.7b) Besonderheit der Swimlane-Darstellung ist die Partitionierung des Prozessgraphen in mehrere Bahnen. Jeder Bahn kann zum Beispiel eine Organisationseinheit des Unternehmens zugeordnet sein, so dass aus der Darstellung

erkennbar wird, welche Aktivitäten von welchen Einheiten ausgeführt werden. Im Allgemeinen können für die Partitionierung beliebige Prozessinformationen herangezogen werden. Auch Hierarchisierungen der Bahnen sind möglich.

Gantt-Diagramm (vgl. Abbildung 2.7e) Sind zu den Aktivitäten des Prozesses entsprechende Start- und Endzeiten oder Ausführungsdauern hinterlegt, kann mit Hilfe dieser Daten ein Gantt-Diagramm generiert werden. Diese Darstellung ist im Projektmanagement sehr verbreitet und Managern daher geläufig. Besonders die zeitlichen Zusammenhänge innerhalb eines Prozesses können dieser Darstellungsform besonders gut entnommen werden.

Prozessterminplan Ähnlich dem Gantt-Diagramm rückt auch der Prozessterminplan die temporalen Aspekte des Prozesses in den Vordergrund. Im Gegensatz zum Gantt-Diagramm verwendet der Prozessterminplan aber nicht für jede Aktivität eine eigene Zeile. Dadurch ergeben sich vor allem bei großen Prozessen kompaktere Darstellungen.

Tabelle (vgl. Abbildung 2.7c) Für eine übersichtliche Darstellung eines Prozesses eignet sich auch eine Tabelle. In der einfachsten Version werden dabei nur die Aktivitäten aufgelistet. Bei Bedarf können zusätzliche Tabellenspalten Informationen zu Daten oder Bearbeitern der Aktivitäten enthalten. Während sequentielle Prozesse mittels dieser Darstellungsform sehr gut und einfach abgebildet werden können, gestaltet sich die Darstellung von nichtsequentiellen Prozessen (z.B. parallele oder zyklische Prozesse) schwierig.

Matrix-Darstellung (vgl. Abbildung 2.7d) Die Matrix-Darstellung abstrahiert von der Ablauflogik des Prozesses und versucht, Zusammenhänge zwischen Prozessobjekten zu vermitteln. Beispielsweise lässt sich in einer Matrix übersichtlich darstellen, welche Bearbeiter auf welche Dokumente innerhalb des Prozesses zugreifen. Liegen zu den Bearbeitern auch Informationen zu deren Position innerhalb der Organisationshierarchie (z.B. aus dem Organisationsmodell des Unternehmens) vor, kann man die Daten in der Matrix hierarchisiert darstellen und entsprechend „auf- bzw. zuklappen“. Diese Darstellungstechniken sind aus Data-Mining-Anwendungen bekannt, wo sie der Exploration von Datenwürfeln dienen.

Interaktionsdiagramm Aus dem Swimlane-Diagramm können wir entnehmen, welche Abteilungen an welchen Stellen der Prozesse interagieren. Das Interaktionsdiagramm abstrahiert nun von den einzelnen Aktivitäten und illustriert stattdessen nur die Schnittstellen (z.B. Austausch von Dokumenten) zwischen zwei Partnern. Diese Darstellung erleichtert bei der Optimierung von Prozessen das Erkennen von Abhängigkeiten.

Neben diesen Beispielen werden in der Literatur weitere Darstellungsformen vorgeschlagen bzw. verwendet. In [BM04, BM05, BMR+05, Bur05] etwa werden Prozessmodelle in Form von U-Bahnnetzplänen dargestellt.

Die Realisierung der unterschiedlichen Darstellungsformen erfordert einen unterschiedlichen Aufwand. Einige Formen (z.B. Tabellen oder Matrix-Darstellung) sind graphisch sehr einfach realisierbar. Für andere, wie Gantt-Diagramme, existieren eine Reihe mächtiger Werkzeuge (vgl. Kapitel 9). Die bezüglich der Generierung anspruchsvollsten Darstellungsformen sind der Prozessgraph und die damit stark verwandte Swimlane-Darstellung. Sie zeichnen sich durch eine hohe Komplexität aus, die aus der komplizierten graphischen Struktur herrührt. Sowohl Gestalt und Inhalt der Prozesssymbole wie auch deren Position kann beliebig verändert werden. Im Gegensatz dazu kann bei Tabellen nur die Darstellung der Tabellenzeile variiert werden, wobei die

Symbolik sehr beschränkt ist. Außerdem werden die Tabellenzeilen immer sequentiell angeordnet, eine freie Positionierung oder eine Darstellung zeitlicher Abhängigkeiten durch Kanten ist nicht möglich. In Kapitel 9 werden wir zudem sehen, dass die auf dem Markt verfügbaren Werkzeuge für die Prozessvisualisierung in den hier untersuchten Szenarien keine zufrieden stellenden Lösungen anbieten. Wir fokussieren daher in dieser Arbeit auf Prozessdarstellungen mittels Prozessgraphen, wobei die vorgestellten Konzepte teilweise auch auf andere Darstellungsformen anwendbar bzw. übertragbar sind.

2.3 Anforderungsanalyse

Aus den vorgestellten sowie weiteren Fallstudien zur Prozessvisualisierung haben wir charakteristische Anforderungen abgeleitet [Bob04a, BRB05]. Im Folgenden werden sowohl funktionale als auch nicht-funktionale Anforderungen an die Prozessvisualisierung beschrieben (vgl. Tabelle 2.1).

Anforderung	Beschreibung
Funktionale Anforderungen - Allgemein	
Anforderung 2-1	Visualisierung von Prozessschemata und Prozessinstanzen
Anforderung 2-2	Ansprechende Darstellung durch fortschrittliche Layout-Verfahren
Anforderung 2-3	Visualisierung von systemübergreifenden Prozessen
Anforderung 2-4	Einbindung von Applikationsdaten in die Darstellung
Anforderung 2-5	Anbindung von verschiedenen Systemtypen
Funktionale Anforderungen - Anpassung an die Bedürfnisse des Betrachters	
Anforderung 2-6	Vereinfachung des Prozessmodells durch Prozess-Views
Anforderung 2-7	Flexible Definition von Prozess-Views
Anforderung 2-8	Flexible Definition der Prozessnotation
Anforderung 2-9	Einfache Konfigurierbarkeit einer Visualisierung
Anforderung 2-10	Wartbarkeit
Anforderung 2-11	Darstellung der Prozessinformation in verschiedenen Formen
Nicht-Funktionale Anforderungen	
Anforderung 2-12	Prozessdarstellung über Inter-/Intranet zugreifbar
Anforderung 2-13	Zugriffsbeschränkung
Anforderung 2-14	Benutzerfreundliche Gestaltung der Anwendung

Tabelle 2.1: Übersicht über die Anforderungen an eine Prozessvisualisierung

2.3.1 Funktionale Anforderungen

Anforderung 2-1 (Visualisierung von Prozessschemata und Prozessinstanzen): Eine Visualisierungskomponente muss in der Lage sein, sowohl Prozessschemata als auch Prozessinstanzen (inkl. Statusdaten) angemessen darzustellen.

Anforderung 2-2 (Ansprechende Darstellung durch fortschrittliche Layout-Verfahren): Die Prozessvisualisierung muss ansprechende Darstellungen erzeugen. Beispielsweise soll bei der Zeichnung der Prozesse die zur Verfügung stehende Fläche sinnvoll ausgenutzt werden

und Überlappungen von Symbolen bzw. Kantenkreuzungen – soweit möglich und sinnvoll – vermieden werden.

Anforderung 2-3 (Visualisierung von systemübergreifenden Prozessen): Komplexe Prozesse werden meist verteilt durch mehrere Systeme ausgeführt. Gerade bei solch fragmentierten Prozessen ist eine durchgängige Visualisierung der systemübergreifenden Prozesse wünschenswert, selbst wenn keine übergeordnete Workflow-Steuerung existiert.

Anforderung 2-4 (Einbindung von Applikationsdaten in die Darstellung): Für die Betrachter einer Prozessvisualisierung ist es wichtig, dass alle relevanten Informationen aus den verschiedenen Laufzeitsystemen in der Prozessvisualisierung verfügbar sind. Zum einen sind dies Laufzeitdaten, wie beispielsweise Nummer und Name des aktuellen Änderungsvorhabens. Zum anderen müssen auch relevante Dokumente (z.B. CAD-Modell und Stellungnahmen) in die Visualisierung integriert werden, so dass auf diese bei Bedarf direkt zugegriffen werden kann.

Anforderung 2-5 (Anbindung von verschiedenen Systemtypen): Nicht immer werden die Prozesse von Workflow-Management-Systemen mit wohldefinierten Schnittstellen für den Zugriff auf Laufzeitdaten ausgeführt. Häufig handelt es sich bei den ausführenden Systemen um Legacy-Applikationen (z.B. konventionell implementierte Applikationen, ohne Prozesssteuerung). Unabhängig vom Systemtyp muss die Visualisierung Zugriff auf die Laufzeitdaten haben, um den Prozessstatus und weitergehende relevante Informationen darzustellen.

Anpassbarkeit an Benutzer

Zusätzlich zu diesen eher allgemeinen Anforderungen existiert eine Reihe spezifischer Anforderungen, welche die Personalisierung, d.h. die Anpassbarkeit der Prozessdarstellungen an die Bedürfnisse des jeweiligen Betrachters, betreffen.

Anforderung 2-6 (Vereinfachung des Prozessmodells durch Prozess-Views): Prozessmodelle enthalten oftmals zu viele Details für den jeweiligen Betrachter. Weiter sollen für manche Darstellungen nur Teile des Prozesses angezeigt werden. Der Rest des Prozesses soll entweder komplett aus dem Modell entfernt oder nur in abstrakter Form dargestellt werden. Daher benötigen wir einen Mechanismus, der es erlaubt, eine Sicht auf das Prozessmodell zu definieren, d.h. mit sich dieses Modell durch Zusammenfassen oder Entfernen von nicht benötigten Prozessobjekten vereinfachen lässt.

Anforderung 2-7 (Flexible Definition von Prozess-Views): Derartige Sichten auf Prozessmodelle müssen einfach und flexibel definiert werden können. Beispielsweise sollte die Menge der zu entfernenden Prozesselemente anhand von Attributwerten festlegbar sein. So lässt sich schnell eine Sicht definieren, die zum Beispiel nur die Aktivitäten des aktuellen Betrachters enthält oder die alle bereits abgeschlossenen Aktivitäten zusammenfasst.

Anforderung 2-8 (Flexible Definition der Prozessnotation): Zur optimalen Anpassung der Prozessdarstellung an die jeweiligen Erfordernisse der Anwendung bzw. des Betrachters, muss die Prozessnotation frei definierbar sein. Dabei sollten sowohl die Form und Farbe der verwendeten Symbole konfigurierbar sein, als auch die darin dargestellten Daten. Die Verwendung der Symbole kann statisch festgelegt sein oder dynamisch in Abhängigkeit von Laufzeitdaten bestimmt werden.

Anforderung 2-9 (Einfache Konfigurierbarkeit einer Visualisierung): Die Konfiguration einer Prozessvisualisierung sollte möglichst einfach erfolgen können. Alle für eine Visualisierung benötigten Konfigurationsparameter sollten daher am besten zentral festgelegt werden können. Insbesondere wäre eine Werkzeugunterstützung für den Visualisierungsdesigner wünschenswert.

Anforderung 2-10 (Wartbarkeit): Der Aufwand für die Definition und Pflege einer Prozessvisualisierung muss in Relation zu dem erwarteten Nutzen liegen. Vor allem die Pflege kann bei sich ändernden Prozessmodellen, Visualisierungsvorstellungen oder Quellsystemen schnell sehr viel Aufwand generieren. Deshalb muss von vornherein die Wartbarkeit der Gesamtlösung im Auge behalten werden, beispielsweise durch die Wiederverwendung von Notationsdefinitionen.

Anforderung 2-11 (Darstellung der Prozessinformation in verschiedenen Formen): Neben der Darstellung der Prozesse in Form von Prozessgraphen sollen auch weitere Darstellungsformen unterstützt werden (vgl. Abschnitt 2.2.4.2).

2.3.2 Nicht-funktionale Anforderungen

Neben funktionalen Anforderungen, welche die direkt erlebbaren Funktionen der Visualisierungskomponente beschreiben, existieren weitere Anforderungen, die eher die allgemeine Beschaffenheit der Anwendung widerspiegeln [ISO01]. Insgesamt sind diese Anforderungen jedoch ebenso wichtig für die Akzeptanz der Visualisierungskomponente.

Anforderung 2-12 (Prozessdarstellung über Inter-/Intranet zugreifbar): Die Prozessvisualisierungen müssen für den Endanwender einfach zugänglich sein. Dies wird am besten durch eine Integration in das bestehende Intranet erreicht. Für die zu entwickelnde Prozessvisualisierungskomponente bedeutet dies, dass die Prozessdarstellungen in einem geeigneten Format erstellt werden müssen. Gleichermaßen muss die Visualisierungskomponente eine Integration in das Unternehmensportal erlauben. Ein weiterer Vorteil dieser Lösung ist der minimale Installationsaufwand auf dem Klientenrechner, wenngleich dieses Argument durch die immer besseren Software-Verteilungsmechanismen zunehmend an Gewicht verliert.

Anforderung 2-13 (Zugriffsbeschränkung): Prozessmodelle beinhalten sensible Informationen, die meist nur einem begrenzten Nutzerkreis zur Verfügung gestellt werden sollen. Es muss daher durch entsprechende Maßnamen sichergestellt werden, dass nur authentifizierte und autorisierte Benutzer Zugriff auf die Prozessdarstellungen erhalten. Für den Fall, dass ein Benutzer beispielsweise eine Aktivität nicht sehen darf, sollten Mechanismen vorgesehen werden, welche die entsprechenden sensiblen Daten aus dem Prozessmodell ausblenden, anstatt den Zugriff auf das Modell vollständig zu verweigern.

Anforderung 2-14 (Benutzerfreundliche Gestaltung der Anwendung): Für den Betrachter einer Prozessvisualisierung ist die benutzerfreundliche Ausgestaltung der Visualisierungsapplikation sehr wichtig. Sie ist neben den funktionalen Aspekten mit entscheidend für die Akzeptanz der Gesamtlösung und damit auch für die Akzeptanz prozessorientierter Systeme an sich.

2.4 Zusammenfassung

In diesem Kapitel haben wir ausgewählte Fallstudien vorgestellt, welche die Situation in der Praxis illustrieren. In den untersuchten Szenarien sollen komplexe, langlaufende Prozessmodelle mit einem hohen Anteil an Benutzerinteraktion visualisiert werden. Dabei existieren mehrere Benutzergruppen, die jeweils unterschiedliche Anforderungen an die Inhalte einer Prozessvisualisierung haben (z.B. Prozessbeteiligte, Manager und IT-Systementwickler). Ausgehend von diesen Szenarien und weiteren hier nicht erörterten Fallstudien [Bob04a] werden Anforderungen an eine Prozessvisualisierungskomponente vorgestellt. Neben allgemeinen funktionalen Anforderungen existieren weitere Anforderungen bezüglich der Konfigurierbarkeit der Darstellung von Prozessen. Insgesamt haben wir all diese Anforderungen in Proviado aufgegriffen. In dieser Arbeit fokussieren wir auf die innovativen Aspekte einer Prozessvisualisierung.

Proviado ist originär im Automobilbereich angesiedelt. Daher wird die Prozessvisualisierung hier anhand von Beispielen der hochkomplexen Prozesse aus verschiedenen Bereichen der Automobilindustrie (Fahrzeugentwicklung, Produktionsplanung, Verkauf und Service) diskutiert. Ähnliche Fallbeispiele findet man jedoch auch in anderen Domänen (z.B. Prozesse aus dem klinischen Umfeld [Kon96, Mey96, Sch96b, LR07]). Die in Proviado entwickelten Konzepte sind generisch und lassen sich auf beliebige Bereiche übertragen. Auch in Anwendungen mit weniger komplexen oder kleineren Prozessen können die entwickelten Visualisierungskonzepte entsprechenden Nutzen stiften.

3

Rahmenwerk für die Prozessvisualisierung

3.1 Motivation

In Kapitel 2 wurden verschiedene Fallbeispiele vorgestellt und Anforderungen daraus abgeleitet. Mit Proviado soll eine Visualisierungskomponente bereitgestellt werden, die in der Lage ist, sehr große, komplexe Prozessmodelle in einer an die Bedürfnisse des Benutzers angepassten Art und Weise darzustellen. Prinzipiell existieren mehrere Möglichkeiten, wie eine Prozessdarstellung konfiguriert bzw. angepasst werden kann. Diese sind in Abbildung 3.1 zusammengefasst und sollen hier kurz erklärt werden.

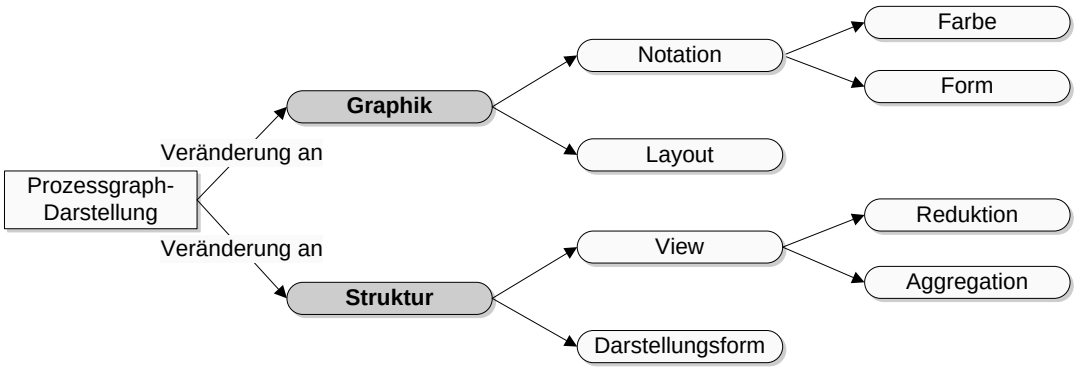


Abbildung 3.1: Freiheitsgrade der Gestaltung einer Prozessvisualisierung

Graphische Anpassung

Generell kann eine Prozessvisualisierung durch rein graphische Methoden verändert werden, ohne inhaltliche Änderungen vorzunehmen.

Zum einen kann die *Notation*, d.h. die zur Darstellung verwendeten Symbole, variiert werden. Durch Form- und Farbveränderungen der einzelnen Symbole erzielt man zudem eine unterschiedliche Wahrnehmung bei den Betrachtern. Die Gestaltung der Notation hat deshalb erheblichen Einfluss auf die Akzeptanz einer Visualisierung. Wird ein Prozess in einer dem Betrachter vertrauten Art und Weise dargestellt, erhöht dies die Akzeptanz und verringert den Einarbeitungsaufwand.

Zum anderen kann das graphische Erscheinungsbild durch Anpassung des *Layouts* beeinflusst werden. Die Position der verschiedenen Prozessobjekte ist zum Beispiel ein sehr wichtiger Aspekt einer Prozessvisualisierung. Das Bild, das sich beim Betrachten eines Prozesses beim Betrachter einprägt, bezeichnet man auch als *Mental Map* [PHG06]. Es spielt eine große Rolle für die Wiedererkennung eines Prozesses.

Strukturelle Anpassung

Neben graphischen Anpassungen kann die Darstellung von Prozessen auch strukturell verändert werden.

Um die Komplexität der Prozessmodelle zu reduzieren, kann zum Beispiel die Menge der dargestellten Information reduziert werden. Dies legt die Unterstützung von *Prozess-Views* nahe. Durch sie werden Teile des Prozessgraphen entfernt (Reduktion) oder zusammengefasst (Aggregation), ohne die Integrität des Prozessmodells zu verletzen.

Die im Prozessmodell enthalten Informationen werden gewöhnlich als Prozessgraph dargestellt. Alternative *Darstellungsformen* verwenden dieselben Informationen, bereiten sie jedoch auf eine andere Art und Weise auf. So können Prozessinformationen beispielsweise auch in Form von Tabellen oder Projektplänen dargestellt werden.

In Proviado werden Prozessvisualisierungen durch Ausnutzung all dieser Freiheitsgrade auf die Bedürfnisse des Betrachters zugeschnitten.

3.2 Arten der Prozessvisualisierung

Bei der Visualisierung von Prozessmodellen kann man verschiedene Arten unterscheiden. Neben der Darstellung von Schemata müssen oftmals auch Prozessinstanzen visualisiert werden. Diese enthalten zusätzlich zu den Schemadaten auch Laufzeitdaten (z.B. Informationen zum Status der verschiedenen Aktivitäten oder Werte von Prozessdatenelementen). Weiter ist zwischen der Darstellung eines einzelnen Schemas (bzw. einer Instanz) und der Darstellung mehrerer Schemata (Instanzen) zu unterscheiden. Eine Kategorisierung der möglichen Arten einer Prozessvisualisierung zeigt Abbildung 3.2.

Schemadarstellung Ein wichtiger Anwendungsfall einer Prozessvisualisierung ist die Anzeige von Prozessvorlagen bzw. -schemata. Hier geht es in erster Linie um die Darstellung der vom Prozessmodellierer entworfenen Abläufe, wie sie zum Beispiel zu Dokumentationszwecken erfasst werden. Neben dem Kontrollfluss enthält eine solche Darstellung zum Beispiel Informationen zu anderen Prozessaspekten (z.B. Datenfluss, Bearbeiterzuordnungen oder Systemzuordnungen).

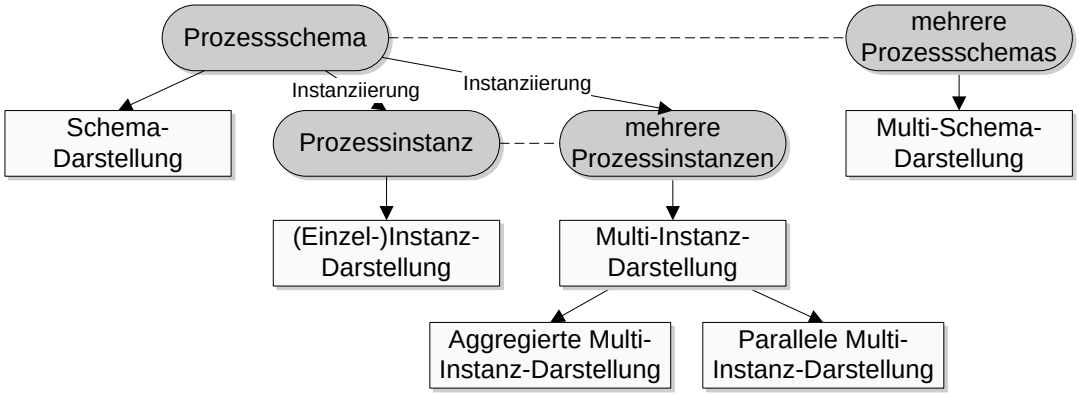


Abbildung 3.2: Kategorisierung der existierenden Visualisierungen für Prozesse

(Einzel-)Instanzdarstellung Bei der (Einzel-)Instanzdarstellung handelt es sich um die Darstellung des konkreten Ablaufs eines Prozesses. Zu den Daten des Schemas kommen weitere zu visualisierende Informationen hinzu, wie Ausführungszeiten, Bearbeiter und aktueller Zustand des Prozesses (bzw. seiner einzelnen Aktivitäten).

Multi-Instanzdarstellung Die Multi-Instanzdarstellung stellt mehrere Instanzen zur gleichen Zeit in einem Diagramm dar. Es existieren zwei Alternativen für diese Art der Darstellung:

Bei der *aggregierten Multi-Instanzdarstellung* werden Daten wie Bearbeiter und Zustand für mehrere Prozessinstanzen zusammengefasst und aggregiert dargestellt. Eine derartige Darstellung findet sich zum Beispiel beim Process-Performance-Monitoring, wo schnell ein Überblick über den Status einer großen Kollektion von Instanzen nötig ist.

Alternativ stellt die *parallele Multi-Instanzdarstellung* mehrere Instanzen gleichzeitig dar. Meist wird hier nicht das komplette Schema angezeigt, sondern nur eine abstrahierte Sicht auf das Originalschema, in dem beispielsweise nur noch die wesentlichen Prozessphasen unterschieden werden. Durch die direkte Gegenüberstellung mehrerer Instanzen kann der Betrachter eine begrenzte Anzahl von Instanzen direkt vergleichen.

Multi-Schemadarstellung Die Multi-Schemadarstellung visualisiert mehrere unterschiedliche Prozessschemata. Eine spezielle Variante ist die Delta-Darstellung. Hierbei werden zwei unterschiedliche Versionen desselben Prozesses dargestellt und die Unterschiede hervorgehoben.

Der Fokus dieser Arbeit liegt auf der (Einzel-)Schema- und Instanzdarstellung.

3.3 Grober Ablauf der Visualisierung

Aufgabe der Informationsvisualisierung ist das Sichtbarmachen von Zusammenhängen, die versteckt innerhalb von Systemen liegen [Tuf90, War04]. Bei Prozessen sollen die Zusammenhänge, speziell die Reichenfolge der Aktivitäten sowie deren Relation zu Prozessdaten, Bearbeitern und Systemen graphisch dargestellt werden. Der Ablauf von den Rohdaten im System zum

fertigen Bild richtet sich nach dem allgemeinen Visualisierungsprozess, wie er aus der Literatur bekannt ist [SM00, Grö02]. Abbildung 3.3 zeigt die vier Phasen, die sich in Datenbereitstellung und Darstellungsgenerierung unterteilen lassen.

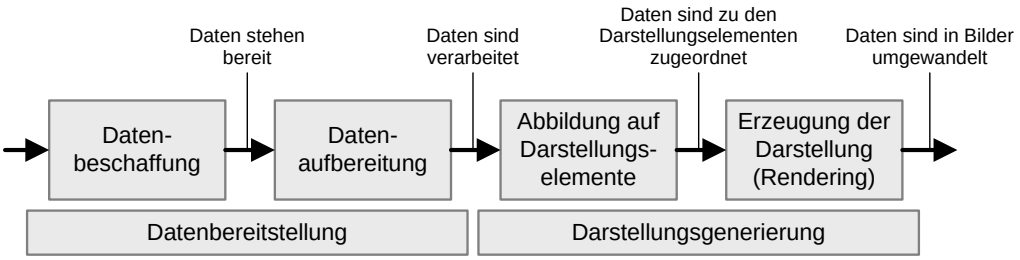


Abbildung 3.3: Visualisierungsprozess

1. **Datenbeschaffung** Die darzustellenden Rohdaten werden bereitgestellt, so dass sie für die Visualisierung zur Verfügung stehen.
2. **Datenaufbereitung** Die Rohdaten werden derart aufbereitet, dass sie sich für die gewählte Art der Visualisierung eignen. Dazu können die Daten beispielsweise gruppiert, aggregiert oder gefiltert werden, oder es werden neue Datenwerte auf Grundlage der existierenden berechnet.
3. **Abbildung auf Darstellungselemente** In dieser Phase werden die Daten graphischen Objekten oder Eigenschaften zugeordnet. Dadurch wird festgelegt, wie ein bestimmter Datenwert in der Darstellung repräsentiert werden soll. Bei Prozessen kann dies beispielsweise die Zuordnung eines Datenobjekttyps (z.B. Aktivität) zu einem Symbol (z.B. Aktivitätensymbol) sein, oder die Zuordnung eines Aktivitätenstatus zu einer Hintergrundfarbe.
4. **Erzeugung der Darstellung (Rendering)** In der Rendering-Phase wird aus den Daten und deren zugeordneten Darstellungselementen ein Bild erzeugt. D.h. aus den in einem internen Repräsentationsformat vorliegenden Informationen über die darzustellenden Objekte wird eine Graphikdatei erzeugt, die auf dem Bildschirm des Betrachters angezeigt werden kann.

Die Übertragung dieses allgemeinen Ablaufs für die Visualisierung von Daten auf die Domäne der Prozessvisualisierung ergibt die in Abbildung 3.4 dargestellte Grobarchitektur einer Visualisierungskomponente. Die allgemeinen Phasen werden im Folgenden bezüglich ihrer Aufgabe bei der Prozessvisualisierung erörtert.

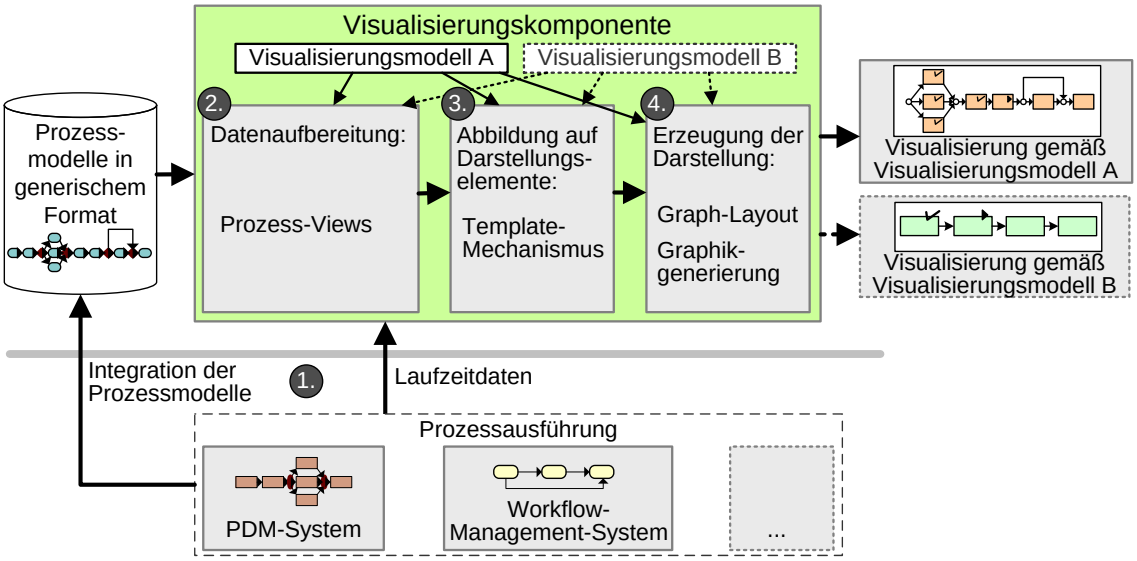


Abbildung 3.4: Überblick über die Komponenten der Visualisierung

1. **Datenbeschaffung** In dieser Phase werden die Prozessdaten zur Verfügung gestellt. Dazu zählen insbesondere die Prozessmodelldaten. Diese liegen oftmals fragmentiert (d.h. über mehrere Informationssysteme verteilt), in diversen Formaten auf den Quellsystemen vor. Sie müssen nun in ein einheitliches Format überführt werden, das dann als Grundlage für die Visualisierung dient. In diesem Zusammenhang werden die Prozessfragmente der verschiedenen Quellsysteme noch zu einem Gesamtmodell integriert. Bei der Visualisierung von Prozessinstanzen ist eine Anbindung an die Laufzeitdaten der Quellsysteme zu realisieren, so dass aktuelle Daten zum Prozessstatus sowie andere Applikationsdaten in die Visualisierung integriert werden können. Als Ergebnis dieser Phase liegen die Prozessdaten als einheitliches Gesamtmodel im System vor. Auf die zugehörigen Laufzeitdaten der Prozesse kann über eine definierte Schnittstelle (z.B. Schnittstelle 5 der WfMC [Hol95]) zugegriffen werden.

2. **Aufbereitung der Daten** Das integrierte Gesamtmodell ist für die Visualisierung im Allgemeinen noch zu komplex und enthält gegebenenfalls für den aktuellen Betrachter irrelevante Details. Daher ist es erforderlich, die Prozessmodelle aufzubereiten bzw. zu vereinfachen. Der später vorgestellte Proviado-View-Mechanismus versetzt uns z.B. in die Lage, derartige Strukturänderungen am Prozessmodell vorzunehmen. Konkret können gewisse Modellanteile aus dem Prozess entfernt (Reduktion) bzw. zusammengefasst (d.h. abstrahiert) werden (Aggregation). Durch parametrisierbare View-Operationen mit komplexer Verarbeitungslogik erlaubt es der in dieser Arbeit vorgestellte Proviado-Ansatz, die Modelle individuell auf die Bedürfnisse der Betrachter zu zu schneiden. Auf Grundlage solcher vereinfachender Prozess-Views wird anschließend die Visualisierung generiert.

3. **Abbildung auf Darstellungselemente** Für jedes Prozessobjekt (z.B. Aktivität) wird in dieser Phase festgelegt, welches Symbol der Prozessnotation für dessen Darstellung verwendet werden soll. Dadurch wird das spätere Aussehen der Visualisierung definiert. In Proviado ermöglicht ein mächtiger Template-Mechanismus durch geschickte Kombination verschiedener Technologien eine hochgradig flexible Zuordnung zwischen Prozessobjekten und Symbolen. Sogar eine dynamische Auswahl der Symbole, abhängig von Laufzeitdaten, wird unterstützt.

4. **Erzeugung der Darstellung** Bei der Visualisierung von Prozessen umfasst diese Phase drei Abschnitte.

   (a) Bevor die fertige Prozessgraphik erzeugt werden kann, muss zunächst das Layout berechnet werden. D.h. für jedes darzustellende Element der Visualisierung muss festgelegt werden, an welcher Stelle der Zeichnungsebene es positioniert werden soll. Die Positionsdaten können entweder manuell erzeugt oder, wie im Falle von Proviado ebenfalls möglich, durch einen Graph-Layout-Algorithmus berechnet werden.

   (b) Aus dem Prozessmodell, den zugeordneten Symbolen und den berechneten Positionsdaten wird nun die Prozessvisualisierung als Graphik erzeugt.

   (c) Die erzeugte Graphikdatei wird an den Visualisierungs-Client übertragen. Dieser kann, abhängig vom jeweiligen Einsatzszenario, unterschiedlich realisiert sein (Thin-Client im Inter-/Intranet oder Rich-Client).

3.4 Zusammenfassung

Zur Anpassung von Prozessvisualisierungen an die Bedürfnisse der Betrachter können diese entweder graphisch oder strukturell verändert werden (vgl. Abbildung 3.1). In Proviado entwickeln wir eine Visualisierungskomponente, die in der Lage ist, die individuellen Anforderungen eines Betrachters an die Darstellung von Prozessinformationen zu erfüllen. Die Visualisierung wird entlang des vorgestellten allgemeinen Visualisierungsprozesses erzeugt. Im Fokus dieser Arbeit steht die strukturelle Anpassung der Prozessmodelle mittels eines mächtigen und flexibel konfigurierbaren Prozess-View-Mechanismus (siehe Kapitel 4 und 5). Hinzu kommt die graphische Anpassung der Darstellung an die Bedürfnisse des Benutzers durch die flexible Definition der Prozessnotation mittels eines fortschrittlichen Template-Mechanismus (siehe Kapitel 6). In Kapitel 8 werden diese fundamentalen Konzepte (Prozess-Views und Template-Mechanismus) durch Arbeiten zu Prozessmodellintegration, zum Prozessgraph-Layout und zu Darstellungsformen für Prozessmodelle flankiert. Sie runden das entwickelte Gesamtkonzept ab. Abbildung 3.5 illustriert die Zusammenhänge zwischen dem allgemeinen Visualisierungsprozess und den weiteren Kapiteln dieser Arbeit.

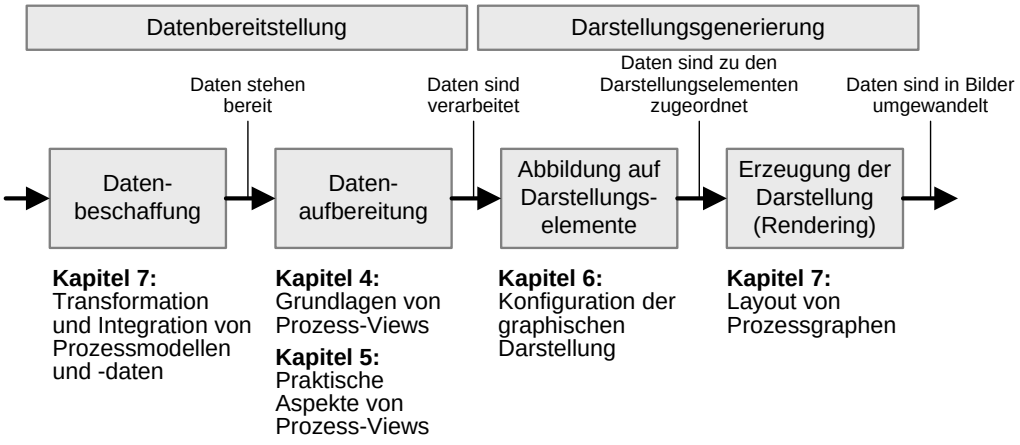


Abbildung 3.5: Zusammenhang zwischen den Kapiteln und dem Visualisierungsprozess

Teil II

Visualisierungskonzepte

4

Prozess-Views: Grundlagen und Elementaroperationen

Geschäftsprozessmodelle weisen oftmals eine beachtliche Größe auf. So umfassen Entwicklungsprozesse in der Automobilindustrie hunderte bis tausende von Aktivitäten. Wird eine auf eine bestimmte Benutzergruppe angepasste Prozessvisualisierung gewünscht, sind entsprechende Modelle aber meist zu umfangreich für die Darstellung. Typischerweise sind die Aktivitäten eines bestimmten Prozessbeteiligten über große Teile des Gesamtprozesses verteilt. Um ihm in einem solchen Szenario einen Überblick zu seinen Aktivitäten im Gesamtprozess zu verschaffen, müssen irrelevante Objekte (z.B. Aktivitäten anderer Beteiligter) aus der Prozessdarstellung entfernt bzw. ausgeblendet werden können.

Wir stellen in diesem und im folgenden Kapitel einen mächtigen Ansatz zur Generierung von Prozess-Views für die Prozessvisualisierung vor. Damit wird es möglich, die Komplexität von Prozessmodellen bei Bedarf zu reduzieren, um zum Beispiel eine optimierte Darstellung für eine bestimmte Benutzergruppe zu realisieren [BRB07, BB07b].

Dieses Kapitel gliedert sich wie folgt: Abschnitt 4.1 motiviert Prozess-Views im Kontext der Visualisierung und gibt einige konkrete Beispiele für die praktische Anwendung von Views. Davon ausgehend leiten wir konkrete Anforderungen für die Unterstützung von Prozess-Views bei der Prozessvisualisierung ab. Das diesem View-Ansatz zugrunde gelegte Prozessmodell und formale Grundlagen werden in Abschnitt 4.2 eingeführt. Abschnitt 4.3 stellt Elementaroperationen zur View-Generierung auf Kontrollflussgraphen vor und diskutiert die Eigenschaften der generierten Views. Abschnitt 4.4 erweitert diesen Ansatz um View-Operationen, die auf Prozessattributen anwendbar sind. In Abschnitt 4.5 expandieren wir das Konzept weiter und beziehen Aspekte wie Datenfluss ein. Abschnitt 4.6 gibt einem Überblick über die eingeführten Operationen und deren Eigenschaften, bevor in Abschnitt 4.7 verwandte Arbeiten diskutiert werden.

Abschließend fasst Abschnitt 4.8 die Erkenntnisse dieses Kapitels zusammen. Aufbauend auf den theoretischen Grundlagen, die in diesem Kapitel gelegt werden, betrachten wir im nachfolgenden Kapitel 5 praktische Aspekte der View-Bildung und definieren dazu höherwertige View-Bildungsoperationen.

4.1 Einleitung

4.1.1 Motivation

Eine von uns durchgeführte Analyse existierender Werkzeuge zur Darstellung von Prozessmodellen (meist in Form von Komponenten von Modellierungswerkzeugen und Workflow-Management-Systemen) hat ergeben, dass bestehende Werkzeuge den Benutzer bei der Bildung personalisierter Sichten nicht oder nicht angemessen unterstützen. D.h. um eine bestimmte Prozesssicht zu erhalten, muss diese bei existierenden Werkzeugen von Hand erzeugt (d.h. nachmodelliert) werden. Eine benutzerspezifische Visualisierung erfordert jedoch flexible Mechanismen, mit denen sich die Komplexität der Prozessmodelle durch Reduktions- bzw. Abstraktionstechniken adaptieren lassen. Dazu gehört zum einen die Anforderung, nicht benötigte Objekte aus der Darstellung komplett entfernen zu können (Reduktion). Zum anderen muss es möglich sein, eine Menge von Objekten (z.B. Aktivitäten) in abstrahierter Form darstellen zu können (Aggregation). Abbildung 4.1 zeigt ein einfaches Beispiel, das die genannten Aspekte illustriert.

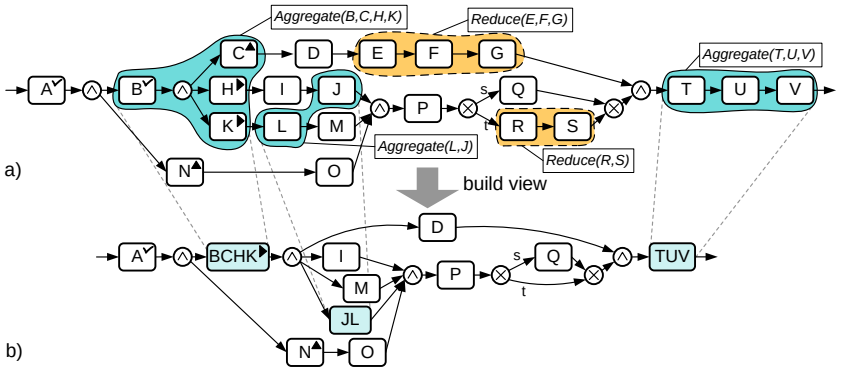


Abbildung 4.1: Beispiel einer Prozess-View

Dieses und das nachfolgende Kapitel stellen einen mächtigen Ansatz zur Generierung solcher Prozess-Views vor. Damit wird es möglich, die Komplexität von Prozessmodellen, wie eingangs gefordert, zu reduzieren. Da in Proviado die Visualisierung von Prozessen im Fokus steht, bestehen hinsichtlich der Bildung von Views weniger stringente Anforderungen bezüglich der Konsistenz der resultierenden Prozessmodelle als zum Beispiel bei Szenarien, in denen auf Grundlage von Prozess-Views unternehmensübergreifende Prozesse koordiniert werden (vgl. hierzu Abschnitt 4.7). Existierende Ansätze zur View-Bildung sind eher theoretischer Natur und operieren meist auf einem stark eingeschränkten Modell.

4.1.2 Beispiele für Views bei der Visualisierung von Prozessen

Abbildung 4.1 zeigt Beispiele für zwei Formen von View-Operationen: Aggregation und Reduktion. Wie motiviert, dient Reduktion dazu, Prozesselemente aus dem Prozessmodell zu entfernen, während die Aggregation mehrere Prozesselemente zu einem neuen, abstrakten Element zusammenfasst.

Nachfolgend skizzieren wir einige typische Beispielszenarien für die Verwendung von Views im Rahmen der Prozessvisualisierung.

Beispiel 4-1: Personalisierung der Darstellung mittels Reduktion
In komplexen Prozessmodellen ist es für Prozessbeteiligte schwierig zu erkennen, an welchen Stellen im Prozess sie involviert sind. Daher müssen alle Aktivitäten, die ohne Beteiligung des aktuellen Betrachters ablaufen, aus der jeweiligen Prozessdarstellung entfernt werden können.

Beispiel 4-2: Gewährleistung der Vertraulichkeit von Darstellungen durch Entfernen sicherheitskritischer Informationen
Realisierungsdetails eines unternehmensübergreifenden Geschäftsprozesses sollen vor Zugriffen der Konkurrenz oder Zulieferer geschützt werden. Durch Entfernen der betreffenden Aktivitäten aus der Prozessdarstellung kann die Vertraulichkeit von Prozessdaten gewährleistet werden.

Beispiel 4-3: Entfernen technischer Details aus der Prozessdarstellung
Ausführbare Workflow-Modelle umfassen oftmals eine große Anzahl „technischer“ Aktivitäten, mit denen keine Anwendungsfunktionen verknüpft sind, sondern lediglich Schritte zur Übertragung oder Konvertierung von Daten. Für Prozessbeteiligte sind diese Aktivitäten oftmals irrelevant bzw. unverständlich und sollten daher bei Bedarf aus der Darstellung entfernt bzw. ausgeblendet werden können.

Beispiel 4-4: Zusammenfassung von Informationen abgeschlossener Aktivitäten durch Aggregation
Eine zentrale Anwendung einer Prozessvisualisierung besteht darin, sich aktuell anstehende Aktivitäten einer Prozessinstanz zusammen mit zukünftigen Aktivitäten anzeigen zu lassen. Die zu diesem Zeitpunkt bereits abgeschlossenen Aktivitäten sind in diesem Szenario irrelevant und sollen deshalb nicht oder nur in aggregierter Form dargestellt werden können.

Beispiel 4-5: Ausblenden bzw. Entfernen abgewählter bzw. nicht mehr ausführbarer Ablaufpfade
Beim Monitoring beendeter bzw. sich in Ausführung befindlicher Prozessinstanzen sind die abgewählten Pfade von XOR-Verzweigungen (z.B. Abwahl durch Deadpath Elimination) irrelevant und sollen deshalb nicht dargestellt werden. Unter Umständen können auf Basis der vorhandenen Prozessdaten auch für in der Zukunft liegende Verzweigungspfade entsprechende Vorhersagen getroffen werden, d.h. auch hier sollte eine Abwahl bzw. ein Ausblenden dieser Zweige möglich sein, wenn diese nicht mehr relevant sind (Look-Ahead).

Beispiel 4-6: Aggregation aller Schritte, die durch bestimmte Organisationseinheiten ausgeführt werden
In der Automobilindustrie ist die Kooperation von Automobilhersteller (OEM) und Zulieferunternehmen fundamental. Ein stark abstrahierter, aber noch realitätsnaher Prozess aus diesem

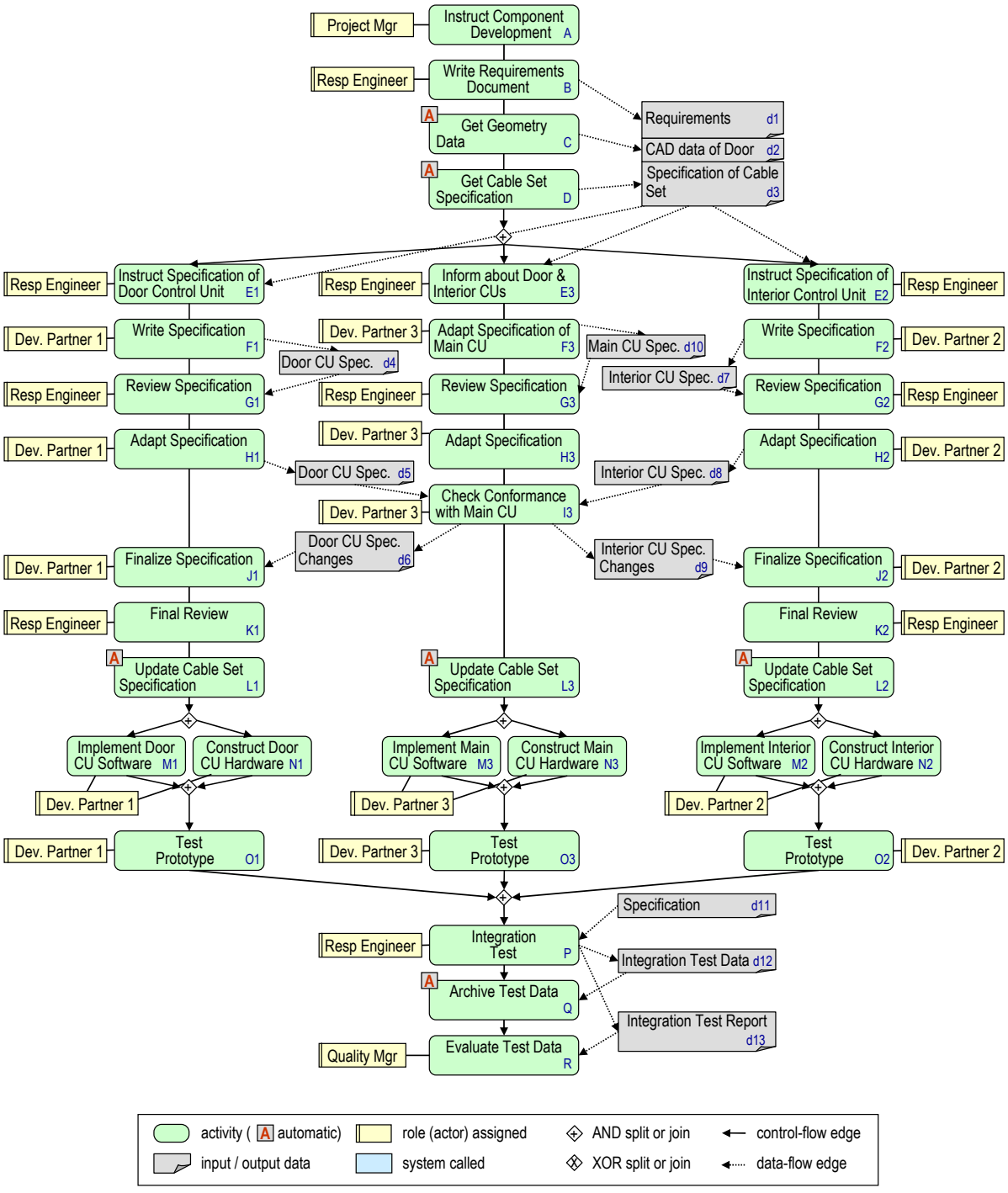


Abbildung 4.2: Prozess für die Entwicklung der Steuergeräte einer Tür

Umfeld ist in Abbildung 4.2 dargestellt. Konkret zeigt diese Abbildung einen Prozess zur Entwicklung der Elektrik/Elektronik-Komponenten (E/E-Komponenten) für eine Fahrzeugtür: nach Festlegung der Anforderungen für die zu entwickelnden Komponenten, werden diese unabhängig voneinander von unterschiedlichen Partnern entwickelt. Dies umfasst die Spezifikation der E/E-Komponente durch den Partner, Überprüfungen durch den OEM und die Implementierung der Komponenten in Hard- und Software. Die einzeln gestesteten Prototypen werden schließlich in einem Integrationstest auf ihr Zusammenspiel hin geprüft.

Der OEM möchte zum Beispiel einen Überblick über den Zustand der Aktivitäten dieses Prozesses gewinnen, die in seinem Unternehmen ausgeführt werden. Deshalb sollen jeweils alle Aktivitäten eines Zulieferers aggregiert werden, wobei die exakte Darstellung der Interaktionen zwischen OEM und Zulieferer sekundär ist. Es genügt die Information in welchem Abschnitt des Prozesses der Zulieferer tätig ist und ob die Bearbeitung dieses Abschnitts abgeschlossen ist oder noch läuft. Abbildung 4.3 zeigt das entsprechend dieser Kriterien aufbereitete Prozessmodell. Hier wurden auch, wie in Beispiel 4-3 beschrieben, die technischen Aktivitäten aus der Prozessdarstellung entfernt. Im aufbereiteten Prozessmodell sind weiterhin die drei parallelen Zweige des Entwicklungsprozesses erkennbar. Mit Hilfe der in Kapitel 6 vorgestellten Mechanismen können ferner Datenelemente ausgeblendet und Bearbeiterzuordnungen in die Aktivitätensymbole integriert werden. Dies verbessert die Übersichtlichkeit des Modells weiter. Die parallelen Zweige mit den aggregierten Aktivitäten der Entwicklungspartner geben zusätzlich an, in welchem Abschnitt der Partner tätig ist. Gleichzeitig bleibt die Reihenfolge der Aktivitäten des OEM erhalten.

4.1.3 Anforderungen

Im Folgenden beschreiben wir die wichtigsten Anforderungen an einen View-Mechanismus, wie sie unsere Anforderungsanalyse aufgeworfen hat. Die Beispiele aus Abschnitt 4.1.2 reflektieren ebenfalls die Ergebnisse der Anforderungsanalyse und dienen der Illustration der hier vorgestellten Anforderungen. Tabelle 4.1 fasst die Anforderungen an Prozess-Views zusammen.

Anforderung	Beschreibung
Anforderungen an die Mächtigkeit des View-Ansatzes	
Anforderung 4-1	Reduktion von Aktivitäten
Anforderung 4-2	Aggregation von Aktivitäten
Anforderung 4-3	Berücksichtigung aller Prozessaspekte
Anforderung 4-4	Kombinierbarkeit der Operationen
Anforderung 4-5	Bildung von Views abhängig von Laufzeitdaten
Anforderung 4-6	Parametrisierbarkeit der Views
Anforderung 4-7	Erweiterbarkeit der Operationsmenge
Anforderungen bezüglich der praktischen Anwendbarkeit	
Anforderung 4-8	Views für die Visualisierung
Anforderung 4-9	Bereitstellung komplexer Operationen
Anforderung 4-10	Definition von Views auf abstrakter Ebene
Anforderung 4-11	Bereitstellung einer View-Definitionssprache

Tabelle 4.1: Übersicht über die Anforderungen an Prozess-Views

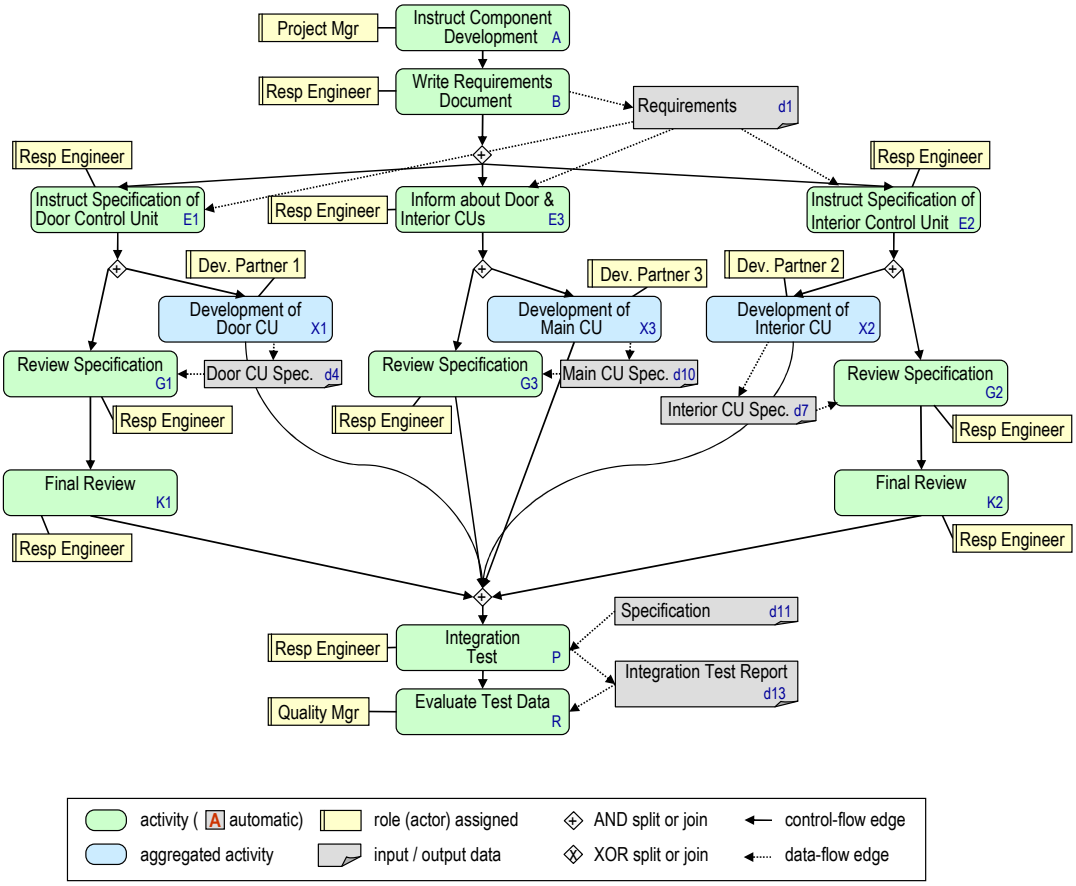


Abbildung 4.3: View auf den Entwicklungsprozess aus Abbildung 4.2

Eine erste Gruppe von Anforderungen betrifft die Mächtigkeit des View-Ansatzes:

Anforderung 4-1 (Reduktion von Aktivitäten): Zwecks Vereinfachung der Darstellung komplexer Prozessmodelle muss eine View-Komponente angemessene Operationen zum Entfernen nicht relevanter Aktivitäten anbieten (*Reduktion*). Zu beachten ist, dass durch die Reduktion der Zusammenhang des Kontrollflusses nicht zerstört werden darf und letztlich wieder ein gültiges Prozessmodell resultiert.

Anforderung 4-2 (Aggregation von Aktivitäten): Neben dem Entfernen von Aktivitäten muss das Zusammenfassen einer Menge von Aktivitäten zu einer neuen, abstrakten Aktivität unterstützt werden. Dies muss sowohl für zusammenhängende Kontrollflussabschnitte, als auch für nicht zusammenhängende Bereiche möglich sein (vgl. hierzu das in Abschnitt 4.1.2 beschriebene Beispiel 4-6 einer Aggregation aller Aktivitäten eines bestimmten Zulieferers). Bei der Aggregation müssen ferner die Attribute der zu aggregierenden Aktivitätenmenge auf die abstrakte Aktivität übertragen werden.

Anforderung 4-3 (Berücksichtigung aller Prozessaspekte): Ein Prozessmodell umfasst neben dem Kontrollfluss weitere Aspekte (Datenfluss, Bearbeiterzuordnungen, IT-Systeme, etc.). Ein View-Mechanismus muss in der Lage sein, diese weiteren Aspekte zu verarbeiten. So müssen beispielsweise View-Operationen auf Datenelementen definiert werden können, die diese sowohl

in Verbindung mit Kontrollflussoperationen als auch unabhängig als eigenständige Operationen verarbeiten (Reduktion und Aggregation).

Anforderung 4-4 (Kombinierbarkeit der Operationen): View-Operationen sollen frei kombinierbar sein, um auch komplexe Transformationen des Basisprozessmodells realisieren zu können.

Anforderung 4-5 (Bildung von Views abhängig von Laufzeitdaten): Im einfachen Fall wird eine View durch Aufzählung der zu reduzierenden bzw. aggregierenden Objekte definiert. Darüber hinaus sollen Views abhängig von Prozessdaten definiert werden können. Das heißt, die Menge der zu reduzierenden bzw. aggregierenden Prozesselemente soll durch Prädikat-ähnliche Ausdrücke beschrieben werden können. Dadurch wird es dann zusätzlich möglich, die Aggregationsmenge dynamisch, abhängig vom aktuellen Ausführungszustand der Aktivitäten, zu definieren.

Anforderung 4-6 (Parametrisierbarkeit der Views): Nicht in allen Fällen einer View-Bildung ist eine Aggregation, wie in Beispiel 4-6 gezeigt, erwünscht bzw. erlaubt. Dies muss bei der View-Bildung berücksichtigt werden. Daher sollen die View-Operationen parametrisierbar sein und in Fällen, in denen gewisse Konsistenz-Anforderungen nicht erfüllbar sind, die View-Bildung mit einer Fehlermeldung abbrechen.

Anforderung 4-7 (Erweiterbarkeit der Operationsmenge): Es muss die Möglichkeit bestehen, neue View-Bildungsoperationen zu definieren, die mit den standardmäßig verfügbaren View-Operationen interagieren bzw. auf diesen aufsetzen. Hierdurch lässt sich sicherstellen, dass zukünftige Anforderungen abgedeckt werden können. Hierzu zählt zum Beispiel die Neudefinition einer komplexen Operation mit dem Ziel, eine häufig benötigte View-Bildung nur einmalig definieren zu müssen und anschließend bei Bedarf einfach anwenden zu können.

Ein View-Ansatz, der die Anforderungen 4-1 bis 4-7 erfüllt, verbessert die Möglichkeiten der Personalisierung von Prozessdarstellungen signifikant. Für eine praktische Anwendbarkeit sind noch weitere Aspekte zu beachten, die in den folgenden Anforderungen zusammengefasst sind.

Anforderung 4-8 (Views für die Visualisierung): Ein wichtiges Ziel der View-Bildung in Proviado ist die Aufbereitung von komplexen Prozessmodellen, um sie in einer verständlichen, übersichtlichen Art und Weise dem Benutzer darzustellen. In der Praxis spielen Übersichtlichkeit bzw. Verständlichkeit der Prozessmodelle bei der Visualisierung oftmals eine größere Rolle als eine stets korrekte Prozessstruktur (bzw. Konsistenz zwischen Basismodell und visualisierter View). Aus diesem Grund sollte eine View-Komponente auch Operationen bereitstellen, die bei Bedarf die Aggregation nicht zusammenhängender Bereiche ermöglicht. Wie wir später zeigen werden, kann dies gegebenenfalls zum Verlust von Informationen bzw. zu Unschärfe führen. Daher sollte für ein aus einer View-Bildung hervorgehendes Prozessmodell immer feststehen, welchem Grad an Korrektheit es genügt. Dadurch wird zudem eine breite Anwendbarkeit des View-Mechanismus ermöglicht.

Anforderung 4-9 (Bereitstellung komplexer Operationen): Eine zentrale Anforderung für die praktische Nutzung von Views ist, die auf den verschiedenen Prozessaspekten (z.B. Kontroll-, Datenfluss) aufsetzenden Basisoperationen zu komplexen Operationen, die alle Aspekte

in die View-Bildung einbeziehen, zusammenfassen zu können. Bei Bedarf sind Parameter vorzusehen, über die das exakte Verhalten der komplexen Operationen gesteuert werden kann. Beispielsweise muss festgelegt werden können, ob bei der Reduktion von Aktivitäten adjazente Datenelemente gelöscht werden sollen oder nicht.

Anforderung 4-10 (Definition von Views auf abstrakter Ebene): In vielen Anwendungsdomänen werden bestimmte Szenarien der View-Bildung repetitiv auftreten (vgl. hierzu die Beispiele in Abschnitt 4.1.2). Die entsprechenden Views sollen daher auf einer möglichst abstrakten Ebene definiert werden können.

Anforderung 4-11 (Bereitstellung einer View-Definitionssprache): Die Definition einer View soll möglichst einfach und flexibel für den Benutzer gestaltet werden können. Zusätzlich zu einer Programmbibliothek sollte daher eine View-Definitionssprache bereitgestellt werden.

Nachfolgend führen wir die Grundlagen des Proviado-View-Mechanismus ein. Dabei werden in diesem Kapitel zunächst die Anforderungen 4-1 bis 4-7 behandelt. Die Anforderungen 4-8 bis 4-11 hinsichtlich praktischer Anwendbarkeit werden anschließend in Kapitel 5 aufgegriffen.

4.2 Grundlagen

Dieser Abschnitt vermittelt einen Überblick über die grundlegenden Konzepte des Proviado-View-Mechanismus. Wir führen im Folgenden wichtige Begriffe und formale Grundlagen ein, die für die nachfolgenden Betrachtungen zu Prozess-Views benötigt werden.

In prozessorientierten Informationssystemen wird jeder Geschäftsprozess durch ein *Prozessschema* beschrieben (vgl. Abbildung 4.2). Ein solches Prozessschema repräsentiert einen abstrakten Geschäftsprozess und bezieht Aspekte wie Kontrollfluss, Datenfluss und Bearbeiterzuordnungen mit ein [JB96]. Ausgehend von einem solchen Prozessschema werden zur Laufzeit *Prozessinstanzen* erzeugt, von denen jede einen konkreten Geschäftsfall repräsentiert. Wie in Abbildung 4.4 dargestellt, kann eine Prozess-View sowohl auf einem Prozessschema (Anwendungsfall 1) als auch auf einer Prozessinstanz (Anwendungsfall 2) gebildet werden.

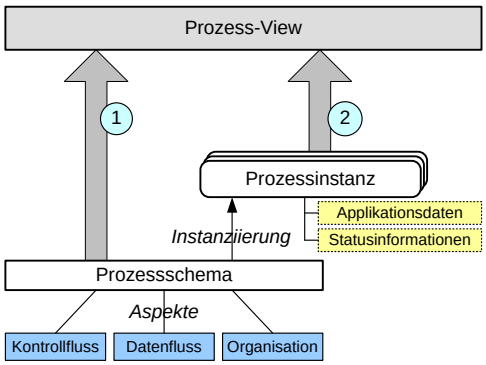


Abbildung 4.4: Überblick über grundlegende Konzepte

Dieses Kapitel stellt elementare Operationen vor, die für die Bildung von Views notwendig sind. Diese Operationen beziehen sowohl die verschiedenen Prozessaspekte als auch Instanzdaten mit ein. Jede dieser Operationen adressiert einen bestimmten Prozessaspekt.

Aufbauend auf diesen elementaren Operationen werden in Kapitel 5 in einem mehrschichtigen Ansatz komplexe View-Bildungsoperationen definiert, die mehrere elementare Operationen kombinieren, um höheren praktischen Anforderungen zu genügen, und auf semantisch höherer Ebene angesiedelt sind. Die Definition einer View gestaltet sich dadurch für Benutzer (im Speziellen für den Visualisierungsdesigner) wesentlich einfacher als bei der Verwendung der elementaren Operationen.

In Bezug auf Prozessinstanzen unterscheiden wir vier Arten von Views, je nachdem wie sich die Menge der zu reduzierenden bzw. aggregierenden Aktivitäten ergibt (z.B. welchen Einfluss Laufzeitdaten auf die View-Bildung haben).

1. **Explizite Views:** Die Menge der zu reduzierenden bzw. aggregierenden Aktivitäten wird explizit spezifiziert.
2. **Statische Views:** Die View-Bildung hängt von statischen Daten ab, die sich zur Laufzeit nicht ändern. Beispielsweise sollen aus dem Prozess alle Aktivitäten reduziert werden, die durch ein statisches Attribut als „technisch“ gekennzeichnet sind (vgl. Beispiel 4-3).
3. **Dynamische Views:** Die View-Bildung ist von Laufzeitdaten abhängig. Beispielsweise sollen die Aktivitäten eines bestimmten Bearbeiters aggregiert werden, wobei sich die konkrete Bearbeiterzuordnung zur Laufzeit ergibt.
4. **Hoch dynamische Views:** Ist die View-Bildung von Aktivitätenzuständen abhängig, so sprechen wir von hoch dynamischen Views, da sich diese Daten mit jeder Statusänderung im Prozess verändern und somit unter Umständen die View bei jeder Statusänderung neu berechnet werden muss. Beispiel 4-4 auf Seite 39 skizziert diese Art von View.

4.2.1 Prozessschema

Wir führen nun die Grundlagen des Prozessmodells ein, auf dem der Proviado-View-Mechanismus basiert. Der Begriff Prozessmodell wird hier als Überbegriff für Prozessschemata und Prozessinstanzen verwendet. Ein Prozessschema besteht aus (atomaren) Aktivitäten und Kontrollflussabhängigkeiten zwischen ihnen (vgl. Abbildung 4.5). Für die Modellierung der Kontrollflussstruktur verwenden wir Kontrollflusskanten und Strukturknoten (z.B. $ANDSplit, XORSplit$).

Definition 4.1 (Prozessschema) Ein Prozessschema ist ein Tupel $P = (N, E, EC, NT, ET)$ mit

- N ist eine Menge von Knoten.
- $E \subseteq N \times N$ ist eine Präzedenzrelation (Notation: $e = (n_{src}, n_{dest}) \in E$). Sie beschreibt die Kontrollflussabhängigkeiten im Prozess und legt die Reihenfolge der Knoten fest.
- $EC : E \rightarrow Conds \cup \{\text{TRUE}\}$ ordnet Kontrollflusskanten optional eine Transitionsbedingung zu (z.B. bei ausgehenden Kanten eines $(X)ORSplit$).

- $NT : N \rightarrow NodeTypes$ ordnet jedem Knoten $n \in N$ einen Knotentyp $NT(n)$ zu. Die Menge an Knotentypen in N ist gegeben durch $NodeTypes = \{Activity, ANDSplit, ANDJoin, ORSplit, ORJoin, XORSplit, XORJoin, LoopSplit, LoopJoin\}$. Dadurch kann N in disjunkte Mengen aus Aktivitäten A ($NT(n) = Activity$ für $n \in N$) und Strukturknoten S ($NT(n) \neq Activity$ für $n \in N$) zerlegt werden, d.h. $N = A \dot{\cup} S$.
- $ET : E \rightarrow \{ControlFlowEdge, LoopEdge\}$ weist jeder Kante $e \in E$ einen Kantentyp $ET(e)$ zu, d.h. $ET(e)$ drückt aus, ob e eine gewöhnliche Kontrollflusskante oder eine Rücksprungkante einer Schleife ist.

$\mathcal{P}$ bezeichnet die Menge aller Prozessschemata.

Diese Definition eines Prozessschemas beinhaltet zunächst nur den Kontrollflussaspekt. Der Proviado-View-Mechanismus deckt jedoch auch weitere Prozessaspekte (z.B. Datenelemente und Datenfluss) ab. Diese werden in Abschnitt 4.5 behandelt und dort formal eingeführt.

Prozessschemata (vgl. Definition 4.1) müssen einige strukturelle Bedingungen erfüllen:

1. Das Prozessschema ist ein zusammenhängender Graph.
2. Es gibt genau einen Start- und einen End-Knoten, d.h. es existiert genau ein Knoten $s \in N$ ($e \in N$), der keinen Vorgängerknoten (Nachfolgerknoten) besitzt.
3. Jeder Knoten ist vom Startknoten aus erreichbar und von jedem Knoten ist der End-Knoten erreichbar.
4. Die Zweige einer Verzweigung müssen mit einem „passenden“ Join wieder zusammengeführt werden. Beispielsweise darf ein von einem $ANDSplit$ ausgehender Zweig nicht in einen $XORJoin$ münden, oder umgekehrt ein $XORSplit$ nicht in einen $ANDJoin$.
5. Mit Strukturknoten (AND, OR, XOR) gebildete Verzweigungen können beliebig geschachtelt sein, solange Bedingung 4 erfüllt ist.
6. P ist kein Multigraph: $\forall x, y \in N : \nexists e_1, e_2 \in E : e_1 = (x, y) \wedge e_2 = (x, y) \wedge e_1 \neq e_2$.
7. Eine Schleife darf nicht in einem Zweig einer Verzweigung beginnen bzw. enden (Schleifen umspannen abgeschlossene Kontrollflussbereiche mit nur je einer eingehenden und ausgehenden Kante).

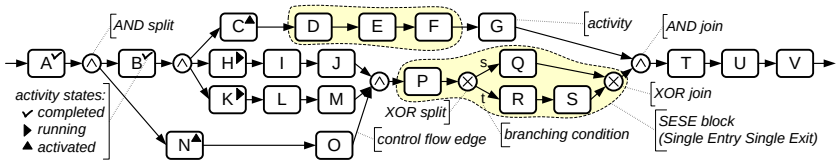


Abbildung 4.5: Beispiel für ein Prozessschema

Die Menge der hier definierten Prozessschemata umfasst alle Modelle, deren Verzweigungen beliebig geschachtelt und überlappend sein dürfen, solange deren Semantik noch korrekt ist. Abbildung 4.5 zeigt ein Beispiel für einen Prozessschema, das Definition 4.1 entspricht und die Bedingungen einhält. Eine Untermenge davon sind die *blockstrukturierten Prozessmodelle*. Sie zeichnen sich dadurch aus, dass zu jedem Split-Knoten ein eindeutiger Join-Knoten existiert [RD98]. Dadurch ergibt sich eine Schachtelung der Blöcke. Vorteil dieser Modellklasse ist ihre einfache Struktur, was sich effiziente Algorithmen zunutze machen. Manche praxisrelevante Prozessmodelle lassen sich mit dieser Art von Modellen jedoch nicht abbilden. View-Mechanismen auf blockstrukturierten Prozessgraphen sind wesentlich einfacher zu realisieren und wurden im Kontext der vorliegenden Arbeit begleitend untersucht [Klo04].

Ziel des Proviado-View-Mechanismus ist es, Prozessmodelle vor ihrer Visualisierung zu vereinfachen. Unter anderem sollen, wie in Beispiel 4-6 auf Seite 39 erwähnt, Aktivitäten auf einem parallelen Pfad zusammengefasst (aggregiert) werden können. In Abbildung 4.6 sollen beispielsweise die Aktivitäten K und M aggregiert und als neuer abstrakter Knoten (hier mit KM bezeichnet) dargestellt werden. Dazu müssen wir zunächst die so genannten Aufsetzpunkte bestimmen, von denen der parallele Pfad ausgehen kann. Um die Korrektheit des resultierenden Modells (im Sinne der vorangehend beschriebenen Modelleigenschaften) zu gewährleisten, wählen wir dazu den größten gemeinsamen Vorgänger bzw. Nachfolger der aggregierten Aktivitätenmenge. Während diese in blockstrukturierten Graphen immer einfach und eindeutig zu bestimmen sind, ist dies bei allgemeinen Prozessgraphen nicht trivial. Anhand von Abbildung 4.6 werden wir nachfolgend einige Definition von Vorgängern und Nachfolgern einführen, die dann später bei der Definition der View-Operationen relevant werden (siehe Abschnitt 4.3.2.1).

Betrachten wir zunächst den Fall, dass im Prozessschema aus Abbildung 4.6 die Aktivitäten K und M aggregiert werden sollen. Als größter gemeinsamer Vorgänger und damit als Startknoten für den parallelen Pfad kommt zunächst $s6$ in Frage (vgl. Variante (a)). Problematisch daran ist jedoch, dass $s6$ nicht immer ausgeführt wird und zwar genau dann nicht, wenn zuvor bei $s5$ der andere Pfad (d.h. der obere Pfad mit Aktivität I) gewählt worden ist. Dies führt uns zur Definition von $s6$ als *unsicheren Vorgänger*.

Definition 4.2 (Unsicherer Vorgänger/Nachfolger) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema und $S \subseteq N$ eine Menge von Knoten in N. Der größte gemeinsame Knoten, der Vorgänger von allen Knoten aus S ist, wird als *größter gemeinsamer unsicherer Vorgänger (gguV)* bezeichnet. Analog ist der *kleinste gemeinsame unsichere Nachfolger (kguN)* der erste Knoten, der Nachfolger aller Knoten aus S ist.

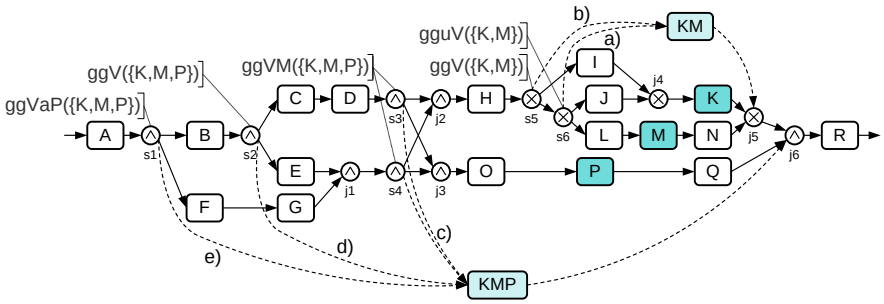


Abbildung 4.6: Arten von Vorgängern in komplexen Prozessmodellen

Bevor wir den sicher ausgeführten, größten gemeinsamen Vorgänger definieren, zeigt das folgende Beispiel, dass der gguV in bestimmten Fällen nicht eindeutig ist. Im Prozessschema aus Abbildung 4.6 soll zusätzlich zu den Aktivitäten K und M auch P aggregiert werden. Hier existieren nun mehrere Knoten, die zugleich größte gemeinsame Vorgänger von $\{K, M, P\}$ sind, nämlich die beiden Verzweigungsknoten $s3$ und $s4$. Wir bezeichnen diese Menge als *größte gemeinsame Vorgängermenge* (ggVM). Von jedem dieser Knoten könnten wir die neue Kante zu der abstrakten Aktivität ziehen, ohne die Korrektheit des Modells zu verletzen (Variante (c)).

Um Probleme, die aus dieser Mehrdeutigkeit und der nicht garantierten Ausführung resultieren, zu vermeiden, definieren wir für eine Menge S den *größten gemeinsamen Vorgänger*. Dieser wird immer vor allen anderen Knoten aus S ausgeführt. Ein solcher Knoten ist eindeutig, existiert immer und lässt sich durch rekursives Bestimmen von $ggVM(S)$ berechnen. Das Verfahren terminiert spätestens am eindeutigen Startknoten des Prozesses. In unserem Beispiel wäre Knoten $s2$ der ggV von $\{K, M, P\}$ (Variante (d) in Abbildung 4.6).

Definition 4.3 (Größter gemeinsamer Vorgänger/Nachfolger) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema und $S \subseteq N$. Der eindeutige, größte Knoten, der Vorgänger von allen Knoten aus S ist und der bei allen möglichen Ausführungen[1] des Prozesses P sicher ausgeführt wird, wird als *größter gemeinsamer Vorgänger (ggV)* bezeichnet. Analog ist der *kleinste gemeinsame Nachfolger (kgN)* der eindeutige, kleinste Knoten der Nachfolger aller Knoten aus S ist und der sicher ausgeführt wird.

Um sicherzustellen, dass der größten gemeinsamen Vorgängers ggV einer Aktivitätenmenge S sicher ausgeführt wird, dürfen bei dessen Berechnung alle Knoten auf XOR- oder OR-Zweigen nicht berücksichtigt werden, die zwischen den Knoten aus S und $gguV(S)$ zusammengeführt werden. In unserem Beispiel aus Abbildung 4.6 dürfen für $S = \{K, M\}$ die Knoten I, J und $s6$ wegen $j4$ nicht berücksichtigt werden. Es ergibt sich somit $ggV(\{K, M\}) = s5$.

Die dritte Art von Vorgängern ergibt sich, wenn man auch bei AND-Verzweigungen alle Zweige mit einbezieht. Der *größte gemeinsame Vorgänger aller Pfade* (ggVaP) ist in unserem Beispiel Knoten $s1$ (Variante (e) in Abbildung 4.6).

In Tabelle 4.2 führen wir wichtige Begriffe aus der Graphentheorie [Die06] ein, die wir im weiteren Verlauf dieser Arbeit verwenden.

Adjazent und inzident Zwei Knoten eines Graphen heißen *adjazent*, wenn eine Kante zwischen ihnen existiert. Eine Kante ist inzident zu einem Knoten, wenn der Knoten entweder Start- oder Endpunkt der Kante ist. Entsprechend verwenden wir Formulierungen wie „adjazente Datenelemente einer Aktivität“ und meinen damit alle Datenelemente, die durch eine Datenkante mit der Aktivität verbunden sind.
Direkte Vorgänger/Nachfolger (*pred*/*succ*) Für einen Knoten a im Kontrollfluss bezeichnet $pred(a)$ alle direkten Vorgänger, d.h. die Menge aller Knoten, die mit a durch eine eingehende (Kontrollfluss-)Kante verbunden sind. Analog bezeichnet $succ(a)$ alle direkten Nachfolger.

[1] „alle möglichen Ausführungen von P“ bezieht sich hierbei auf Ausführungen, in denen auch mindestens eine Aktivität aus S ausgeführt wird

Vorgänger/Nachfolger ($pred^*/succ^*$) Für einen Knoten a im Kontrollfluss sind $pred^*(a)$ die direkten und indirekten Vorgänger, d.h. die Menge aller Knoten, die entweder $pred(a)$ oder in $pred^*(pred(a))$ liegen. Analog bezeichnet $succ^*(a)$ alle direkten und indirekten Nachfolger.
(Kontrollfluss-)Weg ($(CF)path$) Ein Weg in einem Prozess ist eine Folge von Kontrollflussknoten $(x_0, \ldots, x_n)$, die jeweils durch eine (Kontrollfluss-)Kante verbunden sind, d.h. $\forall i \in \{1, \ldots, n\} : (x_{i-1}, x_i) \in E$
Zusammenhang – Zusammenhangskomponenten Eine Menge von Knoten ist zusammenhängend, wenn in dem durch sie induzierten ungerichteten Graphen ein Weg zwischen je zwei Knoten existiert. Die einzelnen zusammenhängenden Bereiche werden Zusammenhangskomponenten genannt.
Erweiterte Zusammenhangskomponente (S^*) Für eine Menge von Aktivitäten S in P bezeichne S^* die minimale, um adjazente Strukturknoten erweiterte Menge S, so dass die Zusammenhangskomponenten maximal sind. Ist S^* zusammenhängend, bezeichnen wir S als erweitert zusammenhängend.

Tabelle 4.2: Übersicht über wichtige Begriffe

Abbildung 4.7 stellt den Unterschied zwischen Zusammenhangskomponenten und erweiterten Zusammenhangskomponenten an einem Beispiel dar. Bei ersteren sind die Strukturknoten nicht enthalten.

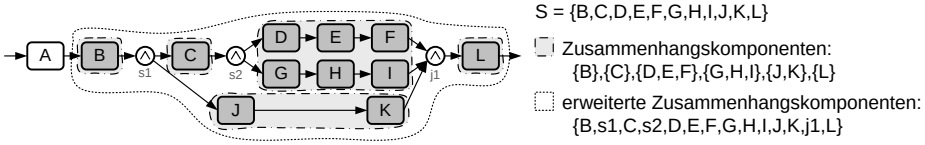


Abbildung 4.7: Beispiele Zusammenhang und erweiterten Zusammenhang

Eine wichtige Struktur in einem Prozessgraphen ist ein SESE (vgl. Abbildung 4.8). Die Abkürzung steht für *Single Entry Single Exit* und stammt aus dem Bereich der Compiler Theorie. Ein SESE ist ein Subgraph, der exakt einen Ein- und Ausgang besitzt, d.h. es existieren keine weiteren Kanten, die inzident zu einem Knoten des SESE sind. [JPP93] zeigt, wie sich ein SESE in linearer Zeit bestimmen lässt. In [VVL07] wird eine SESE-Dekomposition verwendet, um den Prozessgraphen effizient auf strukturelle Korrektheit zu überprüfen. Andere Arbeiten bezeichnen eine derartige Graphstruktur auch als *Hammock Graph* [ZD04].

Definition 4.4 (SESE) Seien $P = (N, E, EC, NT, ET)$ ein Prozessschema und $X \subseteq N$ eine Menge von Knoten. Der durch X induzierte Graph $P[X]$ wird als *SESE* (Single Entry Single Exit) bezeichnet, wenn $P[X]$ zusammenhängend ist und durch genau eine eingehende und eine ausgehende Kante mit dem Rest von P verbunden wird.

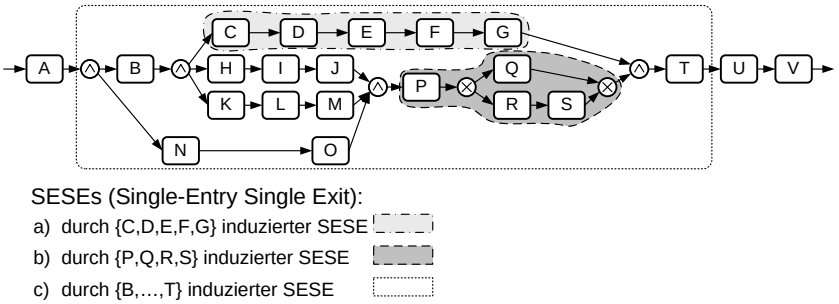


Abbildung 4.8: Beispiel für SESE

Eine weitere Struktur, die wir für die Definition unserer View-Operationen benötigen, ist der Verzweigungsbaum. Ein Beispiel hierfür zeigt Abbildung 4.9.

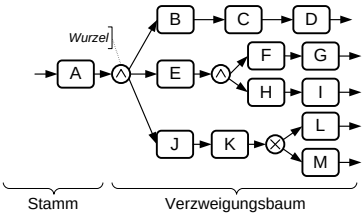


Abbildung 4.9: Beispiel für einen Verzweigungsbaum

Definition 4.5 (Verzweigungsbaum, Vereinigungsbaum) Seien $P = (N, E, EC, NT, ET)$ ein Prozessschema und $X \subseteq N$ eine Menge von Aktivitäten. X^* induziert einen *Verzweigungsbaum*, wenn gilt:

- X^* ist zusammenhängend
- $a := ggV(X, P)$ ist ein Splitknoten
- a liegt in X^*
- Die von a ausgehenden Zweige werden in X nicht mehr zusammengeführt

Analog für *Vereinigungsbaum*.

Ein Verzweigungsbaum (Vereinigungsbaum) mit Stamm ist ein Verzweigungsbaum dem eine Sequenz oder ein SESE vorangeht (nachfolgt).

4.2.2 Prozessinstanz

Ausgehend von einem Prozessschema können zur Laufzeit neue Prozessinstanzen erstellt und ausgeführt werden. Im Beispiel der Prozessinstanz aus Abbildung 4.10 sind etwa die Aktivitäten A und B abgeschlossen, die Aktivitäten C und N aktiviert (d.h. werden in den Arbeitslisten der Bearbeiter angeboten) und die Aktivitäten H und K laufend. Die restlichen Aktivitäten wurden noch nicht gestartet.

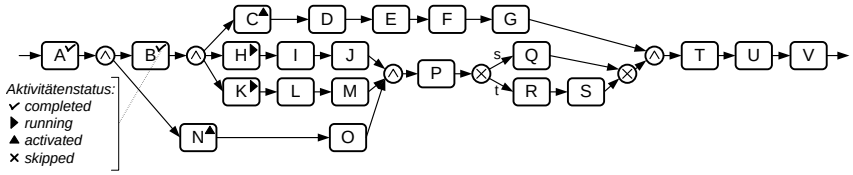


Abbildung 4.10: Beispiel für eine Prozessinstanz

Definition 4.6 (Prozessinstanz) Eine Prozessinstanz I ist ein Tupel $(P, NS, \mathcal{H})$ mit

- P bezeichnet das Prozessschema auf dem I basiert
- $NS : N \rightarrow ExecutionStates$ beschreibt für jeden Knoten $n \in N$ seinen Ausführungszustand. Die Menge $ExecutionStates = \{NotActivated, Activated, Running, Skipped, Completed\}$ bezeichnet dabei die Menge der möglichen Ausführungszustände.
- $\mathcal{H} = \langle e_1, \ldots, e_n \rangle$ bezeichnet die Ausführungshistorie von I, wobei jeder Eintrag $e_k \in \mathcal{H}$ dem Start oder Ende einer Prozessaktivität entspricht ist.

Hinsichtlich der Ausführung einer Instanz und ihrer Zustände setzen wir die Einhaltung gewisser Konsistenzbedingungen voraus:

- Bei einer XOR-Verzweigung wird genau ein Zweig ausgeführt.
- Aktivitäten auf nicht ausgeführten Zweigen werden durch den Zustand *Skipped* markiert.
- Bei einer AND-Verzweigung (Parallelität) werden alle Zweige ausgeführt.
- Auf allen Pfaden vom Start- zum End-Knoten, auf denen sich keine Aktivitäten im Zustand *Skipped* befinden, existiert genau eine Aktivität n mit $NS(n) \in \{Activated, Running\}$. Eine Ausnahme bildet eine komplett abgeschlossene Instanz, bei der alle Aktivitäten dieser Pfade im Zustand *Completed* sind.
  - Alle Aktivitäten vor einer solchen Aktivität n sind im Zustand *Completed* oder *Skipped*
  - Es gibt einen Pfad p vom Startknoten zum Knoten n, der nur abgeschlossene Aktivitäten enthält ($\forall n' \in p$: $NS(n') = Completed$).
  - Alle auf n folgende Aktivitäten befinden sich im Zustand $NotActivated$.
- Bei Schleifen werden die Zustände der Aktivitäten des Schleifenrumpfes mit jeder Iteration neu initialisiert (d.h. auf $NotActivated$ gesetzt).

Die hier informell aufgeführten Konsistenzbedingungen sollen sicherstellen, dass sinnvolle Prozessinstanzen vorliegen, die frei von Verklemmungen (Deadlocks) und anderen inkonsistenten Zuständen sind. Da die Visualisierungskomponente letztlich Prozesse anzeigen soll, die auf anderen Systemen ablaufen, sind dies realistische Forderungen an die zugrunde liegenden Ausführungskomponenten.

Das verwendete, kanonische Zustandsmodell spiegelt die Realität in gängigen Prozesssystemen (IBM WebSphere Process Server, IBM WebSphereMQ Workflow, Tibco Staffware, IBM Lotus

Workflow) wider, wurde jedoch der Übersichtlichkeit halber auf eine minimale Menge von Zuständen reduziert. Die Details dazu sind in [Mih05] nachzulesen.

4.2.3 Prozess-View

Entsprechend Abbildung 4.1 entsteht eine Prozess-View aus einem Prozessschema durch Reduktion oder Aggregation von Aktivitäten. Eine Prozess-View wird im Folgenden mengentheoretisch definiert, die Festlegung der konkreten Semantik folgt später durch Angabe von View-Operationen zur Reduktion bzw. Aggregation. Wie bereits erwähnt, fokussieren wir zunächst auf den Kontrollflussaspekt und erweitern die Definition später um weitere Prozessaspekte wie zum Beispiel Datenfluss.

Definition 4.7 (Prozess View) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema mit Aktivitätenmenge $A \subseteq N$. Dann gilt: Eine *Prozess-View* auf P ist ein Prozessschema $V(P) = (N', E', EC', NT', ET')$ dessen Aktivitätenmenge $A' \subseteq N'$ durch Reduktion und/oder Aggregation von Aktivitäten aus A bzw. P abgeleitet wurde. Formal bedeutet dies:

- $A_U = A \cap A'$ bezeichnet die Menge der Aktivitäten, die sowohl in P als auch $V(P)$ enthalten sind
- $A_D = A \setminus A'$ bezeichnet die Menge der Aktivitäten, die in P aber nicht in $V(P)$ enthalten sind; dies sind entweder reduzierte oder aggregierte Aktivitäten:

$$A_D \equiv AggrNodes \cup RedNodes$$

- $A_N = A' \setminus A$ bezeichnet die Menge der Aktivitäten, die in $V(P)$ aber nicht in P enthalten sind. Jede Aktivität $a \in A_N$ entspricht einer *abstrakten* Aktivität, die eine Menge von Aktivitäten aus A aggregiert:
  1. $\exists\ AggrNodes_i\ (i = 1 \ldots n)$ mit $AggrNodes = \dot{\bigcup}_{i=1\ldots n} AggrNodes_i$
  2. Es existiert eine bijektive Funktion $aggr$ mit:

$$aggr : \{AggrNodes_1, \ldots, AggrNodes_n\} \rightarrow A_N$$

Für ein gegebenes Prozessschema P mit zugehöriger View $V(P)$ führen wir die Funktion $VNode\colon A \rightarrow A' \cup \{undefined\}$ ein, basierend auf der Notation von Definition 4.7. $VNode(n)$ ordnet jeder Prozessaktivität $n \in A_U \cup AggrNodes$ eine zugehörige View-Aktivität zu:

$$VNode(c) = \begin{cases} c & c \in A_U \\ aggr(AggrNodes_i) & \exists\ i \in \{1, \ldots, n\}\colon c \in AggrNodes_i \\ undefined & c \notin A_U \cup AggrNodes \end{cases}$$

Für eine Aktivität $c' \in A'$ bezeichnet ferner $VNode^{-1}(c')$ die zugehörige Aktivität im Basisprozessschema oder die Menge der Aktivitäten, die zu c' aggregiert wurden.

In Proviado wird eine View durch Komposition einer Menge von View-Operationen gebildet:

$$V(P) = Op_n \circ \ldots \circ Op_1(P)$$

Die Semantik der generierten View wird durch die zur Anwendung gekommenen Operationen Op_i bestimmt. Wie erwähnt, verwendet Proviado einen mehrschichtigen operationalen Ansatz (für Details siehe Kapitel 5). Elementaroperationen beschreiben, wie eine gegebene Menge von Aktivitäten in einer bestimmten Struktur (z.B. Aktivitätensequenz, Verzweigung) verarbeitet werden muss (reduziert oder aggregiert). Darauf aufbauend werden in Kapitel 5 höherwertige Operationen definiert, welche die Struktur der Aktivitätenmenge analysieren und entscheiden, welche Elementaroperationen letztlich angewendet werden müssen.

4.3 Elementare Kontrollflussoperationen

In diesem Abschnitt stellen wir elementare Operationen für die Manipulation von Kontrollflussgraphen (d.h. für die Reduktion bzw. Aggregation von Aktivitäten) vor.

4.3.1 Reduktion

Reduktion dient dazu, Aktivitäten aus dem Prozessmodell zu entfernen. Anwendung findet diese Operation zum Beispiel, wenn nicht benötigte Details bei der Prozessvisualisierung ausgeblendet werden sollen (vgl. Beispiele 4-1 und 4-3). Reduktion wird insbesondere auch dann verwendet, wenn bestimmte Prozessinformationen aufgrund von Sicherheitsbeschränkungen im gegebenen Kontext nicht angezeigt werden sollen (vgl. Beispiel 4-2).

4.3.1.1 Elementaroperationen für die Reduktion

Durch Reduktion sollen Aktivitäten aus dem Prozessschema entfernt werden. Ein einfaches Beispiel zeigt Abbildung 4.11. In der gegebenen Sequenz sollen die Aktivitäten B, C und D reduziert werden. Es resultiert eine Prozess-View, die nur noch die Aktivitäten A und E enthält. Anzumerken ist, dass der Zusammenhang des Prozesses erhalten bleibt.

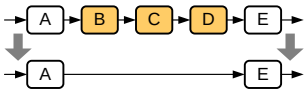


Abbildung 4.11: Reduktion von Aktivitäten in einer Sequenz

Proviado realisiert die Reduktion durch die mehrfache Anwendung der Elementaroperation `RedActivity`. Letztere löscht eine Aktivität aus dem gegebenen Prozessschema und ersetzt diese durch eine Kontrollflusskante (vgl. Abbildung 4.12).

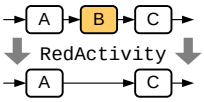


Abbildung 4.12: Reduktion einer Aktivität durch `RedActivity`

Die algorithmische Realisierung der Operation zeigt Algorithmus 4-1. Dieser Algorithmus fügt zuerst eine neue Kante zwischen dem Vorgänger der zu löschenden Aktivität und deren Nachfolger ein. Falls die Aktivität die erste innerhalb einer (X)OR-Verzweigung ist, wird die Kantenbedingung der eingehenden Kante auf die neue Kante übertragen. Schließlich wird die Aktivität mitsamt den inzidenten Kanten gelöscht.

Algorithmus 4-1 : `RedActivity`(P, x)

Input :
$P = (N, E, EC, NT, ET)$ Prozessschema
x: Aktivität, die reduziert werden sollen

```
1 // Einfügen der neuen Kante
2 e = InsertEdge(P, pred(x), succ(x))
3 // Kantenbedingung der alten Kante auf die neue übertragen
4 if EC((pred(x),x)) ≠ TRUE then
5     EC(e) = EC((pred(x),x))
6 // Löschen der Aktivität
7 DeleteNodes(P, {x})
```

Die Beschreibung des Algorithmus bedient sich diverser Hilfsfunktionen, die wir an dieser Stelle im Vorgriff auf weitere Algorithmenbeschreibungen mit Tabelle 4.3 einführen.

DeleteNodes(P, X) Löscht die Aktivitäten der Menge X aus der Menge der Knoten N in P. Adjazente Kanten werden aus der Kantenmenge von P entfernt.
AddNode(P, x) Fügt Aktivität x in die Knotenmenge N von P ein.
InsertNode(P, x, a, b) Fügt Aktivität x zwischen Knoten a und b in P ein.
InsertEdge(P, x_1, x_2) Fügt Kante zwischen x_1 und x_2 in die Menge der Kanten E von P ein.
createAggrNode(X) Erstellt einen neuen Knoten aus den Knoten der Menge X.

Tabelle 4.3: Hilfsfunktionen für Prozessgraphen

Die Reduktion komplexerer Strukturen erfolgt durch wiederholtes Anwenden von `RedActivity`. So wird die Aktivitätensequenz aus Abbildung 4.11 bei der View-Bildung wie in Abbildung 4.13 gezeigt verarbeitet. Konkret wird eine Aktivität nach der anderen aus der Sequenz entfernt, um die gewünschte Reduktion zu realisieren. Das genaue Vorgehen wird in Abschnitt 5.2.1 beschrieben, an dieser Stelle erfolgt aber zumindest eine exemplarische Betrachtung. Beispiele für komplexere Strukturen sind in Abbildung 4.14 dargestellt. Hier wird die Reduktion ebenfalls durch mehrfaches Anwenden der Operation `RedActivity` realisiert.

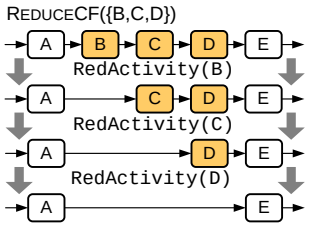


Abbildung 4.13: Ablauf der Reduktion unter Verwendung von `RedActivity`

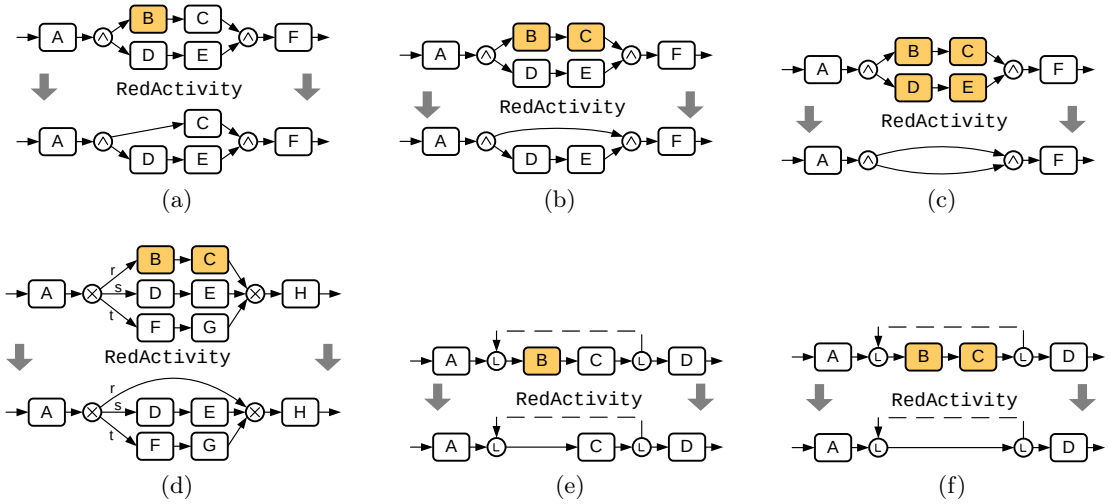


Abbildung 4.14: Reduktion von Aktivitäten in komplexen Prozessen

In einigen Fällen verbleiben nach der Reduktion leere Kanten oder andere überflüssige Kontrollflussstrukturen im Modell (vgl. Abbildungen 4.14b, 4.14c und 4.14f). Hierfür sieht Proviado Vereinfachungsoperationen vor, die im Anschluss an die Anwendung der Elementaroperationen das Prozessmodell analysieren und – falls nötig – zum Beispiel leere bzw. redundante Kanten löschen oder leere Schleifen entfernen. Abbildung 4.15 zeigt einige Beispiele für solche Vereinfachungsoperationen, darunter eine Vereinfachungsoperation für die Entfernung „leerer“ Kanten aus einer Parallelität (siehe Abbildung 4.15a). Falls bei der View-Bildung lediglich ein Zweig oder gar kein weiterer übrig bleibt, werden durch weitere Vereinfachungsoperationen auch der Verzweigungs- und Vereinigungsknoten (Split/Join) entfernt.

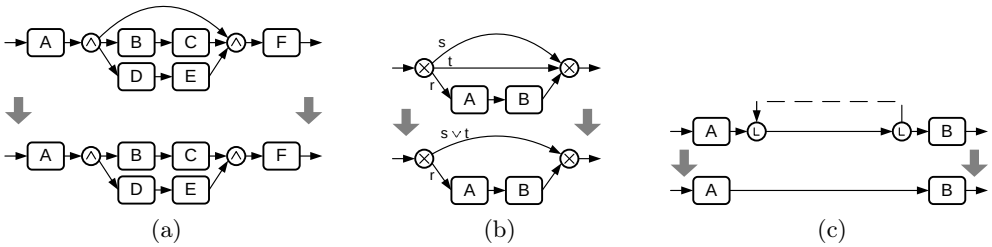


Abbildung 4.15: Beispiele für Vereinfachungsoperationen

Bei alternativen Verzweigungen (OR oder XOR) gilt es zu beachten, dass leere Kanten nicht

einfach gelöscht werden können, ohne die Durchgängigkeit des Kontrollflusses zu zerstören. Hier können leere Zweige nur zusammengefasst werden, wenn gleichzeitig die Kantenbedingungen angepasst werden. Sind die leeren Kanten $e_1, \ldots, e_n$ mit Kantenbedingungen $EC(e_1), \ldots, EC(e_n)$ assoziiert, ergibt sich die Bedingung der neuen Kante e durch eine Oder-Konkatenation der ursprünglichen Bedingungen, d.h. $EC(e) = EC(e_1) \vee \ldots \vee EC(e_n)$ (vgl. Abbildung 4.15b).

4.3.1.2 Eigenschaften der Reduktion

Im Folgenden charakterisieren wir View-Operationen basierend auf verschiedenen formalen Eigenschaften der generellen Prozess-Views. Die Reduktion von Aktivitäten ist zwangsläufig mit einem Informationsverlust verbunden. Gleichzeitig bleibt die Gesamtstruktur der im Prozessschema verbleibenden Aktivitäten im Vergleich zum Ausgangsschema aber erhalten. Letzteres bedeutet, dass wenn eine Aktivität a im Originalschema vor einer Aktivität b im Kontrollfluss steht, gilt dies auch für die korrespondierenden Aktivitäten in der View. Diese Eigenschaft kann man mit Hilfe des Begriffs der *Ordnungserhaltung* beschreiben. Dazu führen wir zunächst eine partielle Ordnungsrelation $\preceq$ ($\subseteq N \times N$) auf Prozessschemata ein. Diese erfasst absichtlich keine zyklischen Strukturen.

Definition 4.8 (Ordnung auf Kontrollfluss) Sei $P = (N, E, EC, NT, ET)$ ein Prozess. Sei ferner durch $\preceq \subseteq N \times N$ eine partielle Ordnungsrelation auf N definiert, mit $n_1 \preceq n_2 \Leftrightarrow \exists$ Pfad p in P von n_1 nach n_2, d.h.

$$\exists p = \{x_0 \ldots x_k\} \text{ mit } x_0 = n_1 \wedge x_k = n_2 \text{ und}$$
$$\forall i \in \{1, \ldots, k\} : x_i \in N \wedge (x_{i-1}, x_i) \in E \wedge ET((x_{i-1}, x_i)) = ControlFlowEdge$$

Ausgehend von dieser Definition können wir *ordnungserhaltende View-Operationen* definieren:

Definition 4.9 (Ordnungserhaltende View-Operationen) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema mit Aktivitätenmenge $A \subseteq N$. Sei ferner $V(P) = (N', E', EC', NT', ET')$ eine durch Anwendung von V entstandene View auf P mit Aktivitätenmenge $A' \subseteq N'$. V heißt *ordnungserhaltend* genau dann wenn

$$\forall n_1, n_2 \in A \text{ mit } n_1 \preceq n_2 :$$
$$\text{falls } \exists\ n_1', n_2' \in A' \text{ mit } n_1' = VNode(n_1) \wedge n_2' = VNode(n_2) \Rightarrow \neg(n_2' \preceq n_1')$$

Diese Eigenschaft spiegelt die Anforderung wider, dass die Ordnung zweier Aktivitäten in einem Prozessschema durch die Bildung einer View nicht vertauscht werden darf.

Wie man leicht sieht, ist die elementare Reduktionsoperation `RedActivity` ordnungserhaltend. Generell ist die Eigenschaft der Ordnungserhaltung eine Grundlage für die Integrität von Prozessschemata und zugehörigen Views. Daher sind alle View-Operationen, die in dieser Arbeit vorgestellt werden, zumindest ordnungserhaltend. Eine strengere Eigenschaft beschreibt Definition 4.10. Diese fordert nicht nur, dass zwei Aktivitäten nicht vertauscht werden dürfen, sondern auch dass die vor der View-Bildung gegebenenfalls bestehende Reihenfolgebeziehung durch die View-Bildung erhalten bleibt.

Definition 4.10 (Streng ordnungserhaltende View-Operationen) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema mit Aktivitätenmenge $A \subseteq N$. Sei ferner $V(P) = (N', E', EC', NT', ET')$ eine durch Anwendung von V entstandene View auf P mit Aktivitätenmenge $A' \subseteq N'$. V heißt *streng ordnungserhaltend* genau dann wenn

$$\forall n_1, n_2 \in A \text{ mit } n_1 \preceq n_2 : \\ \text{falls } \exists n_1', n_2' \in A' \text{ mit } n_1' = VNode(n_1) \wedge n_2' = VNode(n_2) \Rightarrow n_1' \preceq n_2'$$

Strenge Ordnungserhaltung ist in Verbindung mit Reduktionsoperationen gegeben. Wir werden später aber auch Aggregationsoperationen für die View-Bildung vorstellen, für die die Eigenschaft aus Definition 4.10 nicht notwendigerweise erfüllt ist.

4.3.2 Aggregation

Die zweite wichtige Operation zur Bildung von Views ist Aggregation. Wie in Anforderung 4-2 dargelegt, dient die Aggregation dazu, mehrere Aktivitäten in einer abstrakten Aktivität zusammenzufassen (für ein Beispiel siehe Abbildung 4.1). Wir stellen im Folgenden zunächst Elementaroperationen für die Aggregation von Aktivitäten vor. Wie wir sehen werden, sind diese weitaus komplexer als bei der Reduktion. Anschließend analysieren wir wieder die Eigenschaften der Elementaroperationen für die Aggregation im Detail.

4.3.2.1 Elementaroperationen für die Aggregation

Ziel der Aggregation ist das Zusammenfassen mehrerer Aktivitäten zu einer abstrakten Aktivität. Diese abstrakte Aktivität soll so in das bestehende Prozessschema integriert werden, dass möglichst wenige Veränderungen vorgenommen werden müssen. Der wichtigste Unterschied zur Reduktion besteht darin, dass die Aggregation die Informationen der aggregierten Knoten zumindest in abstrahierter Form erhält, während sie bei der Reduktion vollständig verloren gehen. Dies hat auch Auswirkungen auf das algorithmische Vorgehen: Bei der Reduktion werden komplexe Strukturen durch sequenzielles Reduzieren der betroffenen Einzelaktivitäten verarbeitet. Bei der Aggregation würde solch ein sequenzielles Vorgehen zu verfälschten Werten führen, da sich beispielsweise die Durchschnittsbildung über die Ausführungszeit von Aktivitäten nicht sequenzialisieren lässt, sondern atomar ausgeführt werden muss. Daher werden für die Aggregation spezielle Elementaroperationen bereitgestellt, die jeweils für eine bestimmte Kontrollflussstruktur entwickelt wurden und die diese dann atomar aggregiert.

Betrachten wir zunächst die Aggregation einfacher Kontrollflussstrukturen. Ein Beispiel zeigt Abbildung 4.16a. Die Elementaroperation `AggrSequence` aggregiert die sequenziell angeordneten Aktivitäten B, C und D zu einer neuen, abstrakten Aktivität BCD. Dazu entfernt diese Operation die betreffenden Aktivitäten aus dem Prozessschema und fügt stattdessen die abstrakte Aktivität BCD ein. Allgemein funktioniert dieser Ansatz für alle SESE-Strukturen, weshalb wir die Operation `AggrSequence` durch die Operation `AggrSESE` verallgemeinern können (vgl. Abbildung 4.16b).

(a) Sequenz (b) SESE

Abbildung 4.16: Aggregation von Aktivitäten

Die Operation `AggrSESE` lässt sich genau dann anwenden, wenn die zur Aggregation selektierte Aktivitätenmenge S bzw. der durch diese Knotenmenge definierte erweiterte Zusammenhangskomponente S^* einen SESE darstellen. Dies wird in der folgenden Vorbedingung festgehalten.

Vorbedingung an S für `AggrSESE`:

- S^* ist ein SESE

Algorithmus 4-2 beschreibt das Vorgehen bei der Aggregation eines SESE-Blocks. Hierbei werden zunächst der erste Knoten a und der letzte Knoten b des SESE bestimmt (als größter gemeinsamer Vorgänger bzw. kleinster gemeinsamer Nachfolger der Aktivitäten des SESE). Anschließend wird die aggregierte Aktivität erstellt und dem Prozessmodell hinzugefügt. Die aggregierte bzw. neu eingefügte Aktivität wird daraufhin mit zwei Kanten mit dem Vorgänger von a und dem Nachfolger von b verbunden. Sollte die eingehende Kante in a zuvor eine Kantenbedingung besessen haben, wird diese auf die neue Kante übertragen. Schließlich werden alle bisherigen Knoten zwischen a und b (und damit der SESE) aus dem Prozess entfernt.

Algorithmus 4-2 : `AggrSESE`(P, S)

Input :
$P = (N, E, EC, NT, ET)$ Prozessschema
S: Menge von Aktivitäten, die aggregiert werden sollen

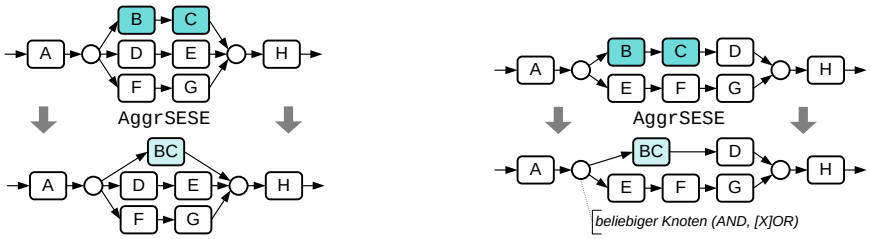
```
1  a = ggV(S, P)
2  b = kgN(S, P)
3  // Anlegen des aggregierten Knotens
4  n = createAggrNode(S)
5  AddNode(P, n)
6  // Einfügen der neuen Kanten
7  e1 = InsertEdge(P, pred(a), n)
8  e2 = InsertEdge(P, n, succ(b))
9  // Kantenbedingung der alten Kante auf die neue übertragen
10 EC(e1) = EC((pred(a),a))
11 // Löschen der Knoten des SESE
12 DeleteNodes(P, {x|x ∈ succ*(a) ∩ pred*(b)})
```

Der Algorithmus zeigt, dass das Vorgehen aus relativ einfachen Schritten besteht. Die Aggregation komplexerer Strukturen verläuft ähnlich vorhersehbar – unerwartete neue Schritte kommen nicht hinzu. Daher werden wir für die weiteren Fälle auf eine detaillierte Darstellung der Algorithmen

verzichten und die Funktionsweise der Operationen stattdessen an Beispielen bzw. graphischen Illustrationen verdeutlichen.

Aggregation von Aktivitäten in Verzweigungen

Für die Aggregation zusammenhängender Bereiche innerhalb eines Verzweigungsastes (AND, OR oder XOR) gibt es die Elementaroperation `AggrSESE` (vgl. Abbildung 4.17).

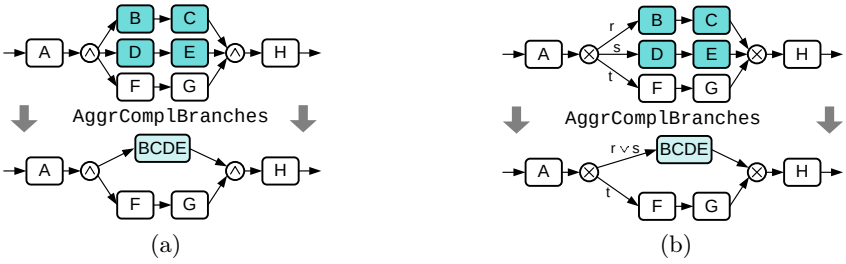


(a) (b)

Abbildung 4.17: Aggregation von Aktivitäten in einem Zweig einer Verzweigung

Sind die zu aggregierenden Knoten auf mehrere Zweige verteilt, kann man zwischen weiteren Fällen unterscheiden. Zunächst widmen wir uns dem Szenario, dass jeweils vollständige Zweige aggregiert werden sollen, wie in Abbildung 4.18 angedeutet. Dieser Fall wird in Proviado durch die Elementaroperation `AggrComplBranches` abgedeckt. Sie aggregiert die jeweiligen Aktivitäten der Zweige zu einer neuen abstrakten Aktivität. Bei einer (X)OR-Verzweigung muss anschließend zusätzlich die Kantenbedingung für den Zweig der abstrakten Aktivität berechnet werden. Dazu werden die Bedingungen der ursprünglichen Zweige $(e_1, \ldots, e_n)$ durch $\vee$-Konkatenation zu einer Bedingung für den neuen Zweig e' kombiniert, d.h.

$$EC(e') = \bigvee_{i \in \{1,\ldots,n\}} EC(e_i)$$



(a) (b)

Abbildung 4.18: Aggregation von Aktivitäten mittels `AggrComplBranches`

Die Elementaroperation `AggrComplBranches` kann genau dann angewendet werden, wenn die erweiterte Menge der selektierten Aktivitäten S^* zusammenhängend ist. Des Weiteren muss der größte gemeinsame Vorgänger a (kleinste gemeinsame Nachfolger b) von S ein Split(Join)-Knoten

sein, und es müssen zwischen a und b noch weitere Zweige existieren, die nicht in S liegen (ansonsten wäre S^* ein SESE und die Aggregation durch `AggrSESE` abgedeckt). Zusammengefasst ergibt sich:

Vorbedingung an S für `AggrComplBranches`:

- S^* ist zusammenhängend
- Für $a = ggV(S)$ Split-Knoten, $b = kgN(S)$ Join-Knoten und $Y = \{x | a \preceq x \preceq b\} \setminus S^*$ die Menge der Knoten, die zwischen a und b, aber nicht in S^* liegen, muss gelten:

  $\exists\ path(a, b)$ in S und $\exists\ path(a, b)$ in Y und $\nexists$ Kante zwischen S^* und Y

Eine weitere Elementaroperation behandelt den Fall, dass alle zu aggregierenden Aktivitäten am Beginn (Ende) einer Verzweigung liegen (vgl. Abbildung 4.19). Die Operation `AggrShiftOut` aggregiert diese Aktivitäten, indem sie vor (hinter) den Verzweigungsknoten gezogen werden. Bei (X)OR-Verzweigungen müssen, wie in Abbildung 4.19c angedeutet, die Verzweigungsbedingungen angepasst werden.

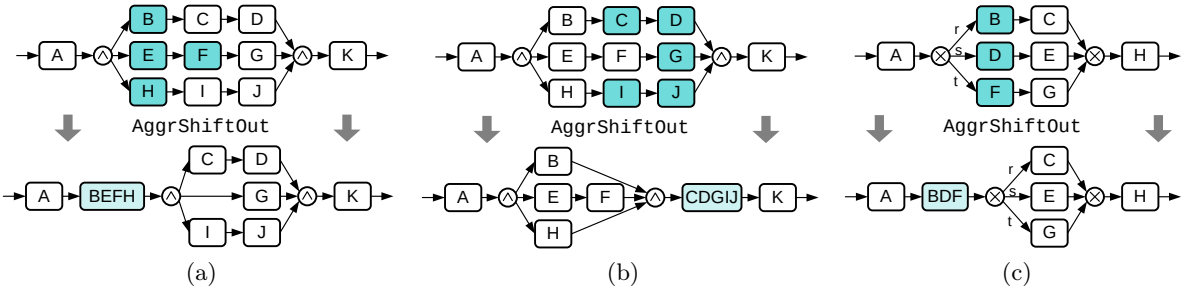


Abbildung 4.19: Aggregation von Aktivitäten mittels `AggrShiftOut`

Ein Sonderfall dieser Operation ist in Abbildung 4.20a dargestellt. Da hier die zu aggregierenden Aktivitäten nicht alle Zweige abdecken, wird zunächst in einem vorbereitenden Schritt ein neuer Verzweigungsknoten vor die zu aggregierenden Aktivitäten eingefügt und erst im Anschluss daran die zuvor beschriebene Operation `AggrShiftOut` ausgeführt.

Die Operation `AggrShiftOut` deckt einen weiteren Fall ab, bei dem weitere, vor dem Verzweigungsknoten liegende Aktivitäten selektiert sind (vgl. Abbildung 4.20b). Definition 4.5 bezeichnet diese Struktur als Verzweigungsbaum mit Stamm. Alle beschriebenen Fälle einer Aggregation von Aktivitäten am Beginn einer Verzweigung werden spiegelsymmetrisch auch am Ende einer Verzweigung durch `AggrShiftOut` unterstützt.

Die Vorbedingungen der Operation `AggrShiftOut` geben all diese Fälle wieder.

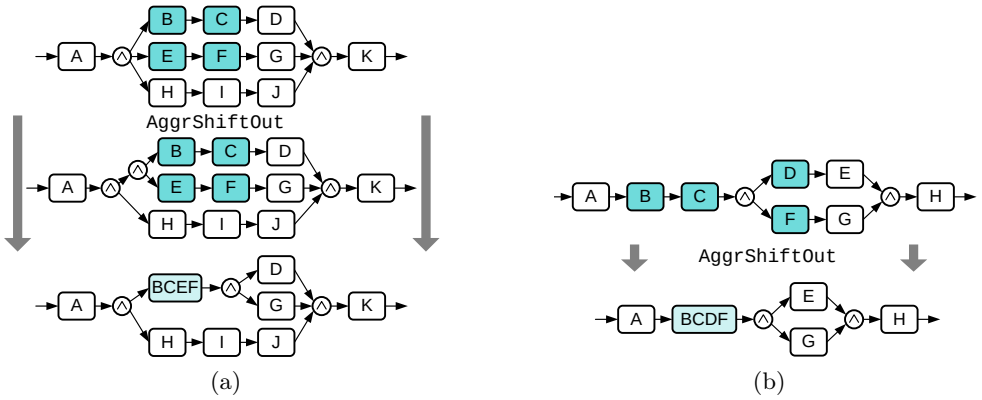


Abbildung 4.20: Sonderfälle einer Aggregation mittels `AggrShiftOut`

Vorbedingung an S für `AggrShiftOut`:

Fall 1: Beginn einer Verzweigung:

- **Fall 1a:** S^* ist ein Verzweigungsbaum ohne Stamm.
- **Fall 1b:** S^* ist ein Verzweigungsbaum mit Stamm.
- **Fall 1c:** S^* ist ein Verzweigungsbaum ohne Stamm und nicht alle Zweige der Wurzel a enthalten Knoten aus S.

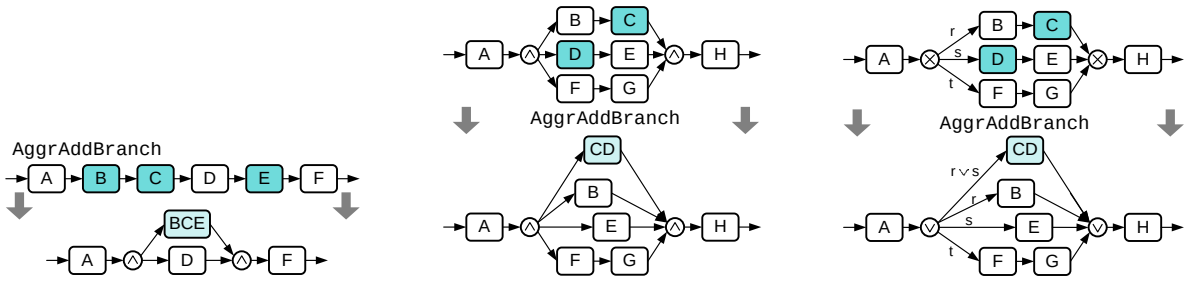
Fall 2: Ende einer Verzweigung: analog

Bei der View-Bildung mittels `AggrShiftOut` gilt es, einige Besonderheiten zu beachten. Zum Beispiel wird durch die Aggregation in Abbildung 4.19a eine Kontrollflussabhängigkeit zwischen der aus B resultierenden, abstrakten Aktivität $BEFH$ und G eingefügt, die im Originalprozess nicht vorhanden ist. Wir werden diese „Inkonsistenz“ in Abschnitt 4.3.2.2 anhand der Eigenschaft *abhängigkeitserzeugend* diskutieren und die Auswirkungen solcher View-Operationen auf das Prozessmodell erörtern (vgl. Definition 4.12).

Ist keine der bisherigen Aggregationsoperationen zur Bildung der gewünschten View anwendbar oder ist S^* nicht zusammenhängend, so verbleibt folgende Möglichkeit zur Aggregation der betreffenden Aktivitätenmenge: Es wird ein paralleler Zweig um den Bereich mit den zu aggregierenden Aktivitäten erzeugt (vgl. Abbildung 4.21a). Gerade für die Visualisierung bringt die Bereitstellung einer derartigen Operation einen erheblichen Nutzen. Will sich ein Prozessbeteiligter beispielsweise einen Überblick über seine eigenen Aktivitäten im Prozess verschaffen, sollen Aktivitäten anderer Prozessbeteiligter oftmals nur in aggregierter Form dargestellt werden. Die daraus ersichtliche Information, dass (a) ein weiterer Akteur am Prozess mitarbeitet und dass (b) dies im Bereich des parallelen Zweiges geschieht, ist für den beabsichtigten Zweck der Visualisierung ausreichend. Die Tatsache, dass diese Form der Aggregation mit einem Informationsverlust (hier dem Verlust von Kontrollflussabhängigkeiten) einhergeht, ist für den Benutzer nebensächlich. Er kann der Visualisierung der Prozess-View die für ihn relevanten Informationen (d.h. seine Aktivitäten und deren Reihenfolge) weiterhin vollständig entnehmen.

Solche Situationen sind in Proviado im Gegensatz zu den bekannten, restriktiven Ansätzen für die View-Bildung (siehe [LS03, EG07, EG08]) sehr gut abbildbar.

In Proviado realisiert die Elementaroperation `AggrAddBranch` diese Art der Aggregation. Ihre Anwendung ist immer möglich, unabhängig davon ob die Aktivitäten in einer Sequenz, einer Verzweigung (vgl. Abbildungen 4.21b und 4.21c) oder anderen komplexen Strukturen liegen (vgl. Abbildung 4.22). Anders ausgedrückt kann für jede beliebige Aktivitätenmenge eine Aggregation durchgeführt werden, sodass eine Prozessvisualisierung in allen gewünschten Konstellationen möglich wird.



(a) Sequenz (b) AND-Verzweigung (c) XOR-Verzweigung

Abbildung 4.21: Aggregation von Aktivitäten mittels `AggrAddBranch`

Grundsätzlich kann die Operation `AggrAddBranch` zu gewissen Inkonsistenzen führen, da Kontrollflussbeziehungen verloren gehen, in unserem Beispiel aus Abbildung 4.21a etwa die Kontrollflussbeziehung zwischen B und D. In Abschnitt 4.3.2.2 führen wir zur Charakterisierung und Diskussion dieses Sachverhalts die Eigenschaft *abhängigkeitslöschend* ein.

Abbildung 4.22 macht deutlich, dass es in Verbindung mit `AggrAddBranch` mehrere potentielle Aufsetzpunkte für den einzufügenden parallelen Zweig geben kann. Um die Struktur des Prozessschemas möglichst gut zu erhalten, wählen wir als Aufsetzpunkte prinzipiell Punkte in Prozessgraphen, die vor bzw. nach gemeinsamen Vorgängern bzw. Nachfolgern der zu aggregierenden Aktivitätenmenge liegen. Unterschiede ergeben sich daraus, je nachdem, wie nah sich die Aufsetzpunkte an der Aggregationsmenge befinden. Dies führt entsprechend zu einem Resultat mit mehr oder weniger gelöschten Kontrollflussabhängigkeiten. Je enger die Aufsetzpunkte gesetzt werden, desto weniger Rücksicht wird auf eine strukturelle Korrektheit des Prozesses genommen. In Abbildung 4.22 äußert sich dies dadurch, dass sich in Fall (a) stark verschränkte Verzweigungen ergeben, während Fall (b) derartige Verschränkungen vermeidet.

In Abschnitt 4.2.1 haben wir verschiedene Arten von gemeinsamen Vorgängern bzw. Nachfolgern definiert (vgl. Definition 4.3). Für die Wahl der Aufsetzpunkte kommen zwei dieser Vorgänger (bzw. Nachfolger) in Frage: die größten gemeinsamen Vorgänger ggV (bzw. Nachfolger kgN) und die größten gemeinsamen Vorgänger aller Pfade (ggVaP) (bzw. Nachfolger kgNaP).

Proviado ermöglicht beide Festlegungen. Zu diesem Zweck besitzt die Operation `AggrAddBranch` einen Parameter ***`Branching`***. Er entscheidet darüber, welche der zwei im Rahmen dieser Arbeit vorgestellten Strategien zur Auswahl der Aufsetzpunkte verfolgt werden soll: eng und weit (***`tight`*** und ***`wide`***). Die Aufsetzpunkte der Operation `AggrAddBranch` ergeben sich wie folgt:

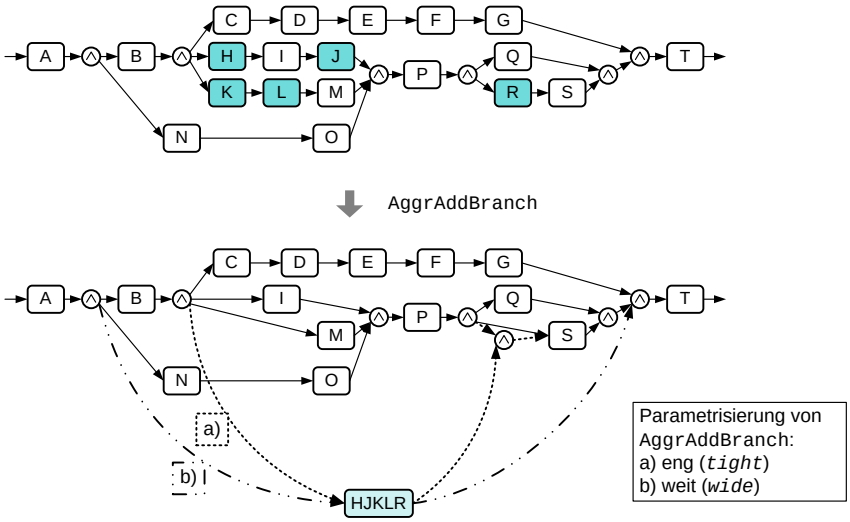


Abbildung 4.22: Parametrisierung von `AggrAddBranch`

eng (*tight*) Die neuen Verzweigungsknoten werden direkt nach dem ggV bzw. vor dem kgN eingefügt und der neu erzeugte, abstrakte Knoten dort als Zweig entsprechend angehängt. Falls der ggV bzw. kgV bereits Verzweigungsknoten sind, wird der abstrakte Knoten dort einfach als zusätzlicher Zweig angehängt (vgl. Abbildung 4.22 Fall a).

weit (*wide*) Die neuen Verzweigungsknoten werden direkt nach dem ggVaP bzw. vor dem kgNaP eingefügt und der neu erzeugte, abstrakte Knoten dort als Zweig entsprechend angehängt. Falls der ggVaP bzw. kgNaP bereits Verzweigungsknoten sind, wird der abstrakte Knoten dort einfach als zusätzlicher Zweig angehängt (vgl. Abbildung 4.22 Fall b).

Bei der Aggregation von Aktivitäten in (X)OR-Verzweigungen führen die beiden Strategien *eng* und *weit* zum selben Ergebnis. Tendenziell führt die Strategie *weit* zu einfacher strukturierten Modellen, da Verschachtelungen von Verzweigungen vermieden werden. Dagegen kann die Struktur blockstrukturierter Graphen bei Anwendung von `AggrAddBranch[eng]` verletzt werden.

Neben den vorgestellten Strategien könnte man prinzipiell auch die Knoten aus der größten gemeinsamen Vorgängermenge $ggVM(S)$ verwenden. Dann sind jedoch die Aufsetzpunkte nicht mehr eindeutig. Zudem ist die Vorgängermenge in den meisten Fällen einelementig, d.h. $ggVM(S) = ggV(S)$. Aus diesen Gründen haben wir auf einen zusätzlichen Parameterwert für eine weitere Strategie verzichtet.

Abbildung 4.22 illustriert die Auswirkungen der Parametrisierung für AND-Verzweigungen. In diesem Beispiel wird als Aufsetzpunkt der Variante *tight* der Knoten $R = kgN(S)$ gewählt. Somit wird der Join-Knoten, der den Zweig mit dem abstrakten Knoten $HJKLR$ mit dem Prozess zusammenführt, direkt nach R eingefügt. Läge R, wie in Abbildung 4.23 dargestellt, in einer XOR- oder OR-Verzweigung, würde bei derselben Aggregation ein Prozess mit unklarer Semantik resultieren. Hierbei würde nach Beendigung der abstrakten Aktivität $HJKLR$, unabhängig von der Verzweigungsbedingung des XOR-Split-Knotens, immer Aktivität T ausgeführt, was zu einem Problem bei der Synchronisation am XOR-Join-Knoten führt (Multiple Instanziierung).

Dies ist ein Beispiel für einen unsicheren gemeinsamen Nachfolger (kguN, vgl. Definition 4.2). Dies unterstreicht, weshalb wir für die Aufsetzpunkte nur sichere Vorgänger und Nachfolger verwenden (ggV bzw. kgN, vgl. Definition 4.3).

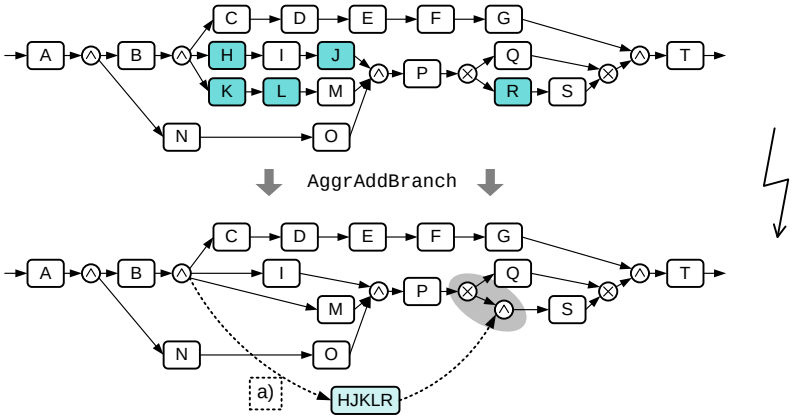


Abbildung 4.23: Problem bei enger Kantenführung von `AggrAddBranch` und XOR-Verzweigungen

Da die Realisierung der Operation `AggrAddBranch` einige Besonderheiten im Vergleich zu den vorhergehend vorgestellten Operationen aufweist, wird der zugehörige Algorithmus 4-3 hier anhand von Pseudo-Code skizziert.

Algorithmus 4-3 ermittelt zunächst mittels der Hilfsoperation `getInsertionPoints` die Aufsetzpunkte a und b. Wie dabei abhängig von der jeweiligen Strategie vorgegangen wird, wurde zuvor beschrieben. Handelt es sich bei diesen Knoten bereits um Split- bzw. Join-Knoten wird bei Bedarf deren Typ angepasst. Andernfalls werden neue AND-Knoten eingefügt. Schließlich wird der neu erzeugte abstrakte Knoten zwischen den zuvor ermittelten Aufsetzpunkten a und b eingefügt. Abschließend werden die Aktivitäten in S mittels der Reduktionsoperation `RedActivity` entfernt.

Algorithmus 4-3 : `AggrAddBranch`$(P, S, param)$

Input :
$P = (N, E, EC, NT, ET)$ Prozessschema
S: Menge von Aktivitäten, die aggregiert werden sollen
$param$: Parameter für die Bestimmung der Aufsetzpunkte

```
1  (a,b) = getInsertionPoints(P, S, param)
2  if a Split-Knoten then
3      // Anpassung des Split-Knotentyps
4      if NT(a) = XORSplit then
5          NT(a) = ORSplit
6  else
7      // Einfügen eines zusätzlichen AND-Splits
8      a' = newNode(ANDSplit)
9      InsertNode(P, a', pred(a), a)
10     a = a'
11 if b Join-Knoten then
12     // Anpassung des Split-Knotentyps
13     if b XORJoin then
14         NT(b) = ORJoin
15 else
16     // Einfügen eines zusätzlichen AND-Splits
17     b' = newNode(ANDSplit)
18     InsertNode(P, b', b, succ(b))
19     b = b'
20 n = createAggrNode(S)
21 InsertNode(P, n, a, b)
22 // Kantenbedingung setzen
23 EC((a,n)) = calcEdgeCondition(a,b,S,P)
24 // Löschen der Knoten in S
25 forall x ∈ S do
26     RedActivity(P, x)
```

Ein im vorliegenden Kontext zu beachtender Aspekt ist die Berechnung der Verzweigungsbedingungen bei `AggrAddBranch` in XOR- oder OR-Verzweigungen. In einigen Fällen kann die Kantenbedingung des neuen Zweiges mit dem abstrakten Knoten als $\vee$-Konkatenation der betroffenen Zweige angegeben werden (vgl. Seite 59). Hieraus resultieren aber meist komplexe Prädikate, die für eine Endbenutzer-Visualisierung zu komplex bzw. unverständlich erscheinen. In der Mehrzahl der Fälle ist eine sinnvolle Berechnung einer Kantenbedingung ohnehin nicht möglich. Abbildung 4.24 zeigt ein Beispiel, in dem im Vorgriff auf Abschnitt 4.5 ein Datenelement vorkommt. Das Datenelement $D1$ wird von Aktivität A geschrieben und danach abhängig von der Auswertung der Bedingung am ersten XOR-Split gegebenenfalls durch Aktivität F überschrieben. In der zweiten Verzweigung hängt die Ausführung eines Pfades vom Wert von $D1$ ab. Durch die Aggregation von C, D und H zur abstrakten Aktivität CDH auf einem parallelen Pfad wird die Bedingung, die das Datenelement $D1$ verwendet, vor die mögliche Veränderung durch F gezogen. Somit stimmt das Ergebnis der Auswertung der konkatenierten Bedingung unter Umständen nicht mehr mit der Realität des ursprünglichen Prozesses überein. Im Kontext unserer Anwendung stellt dies jedoch kein Problem dar, da wir ausschließlich visualisieren, was von darunter liegenden Systemen ausgeführt wird. Eine Interpretation der Prozessmodelle, wozu

eine formale Semantik benötigen würde, findet daher in Proviado nicht statt. Die Funktion `calcEdgeCondition` in Zeile 23 des Algorithmus 4-3 setzt die Kantenbedingung des Zweiges mit dem abstrakten Knoten daher in allen Fällen, in denen der Verzweigungsknoten sich nicht eindeutig dem Vereinigungsknoten zuordnen lässt, grundsätzlich auf TRUE.

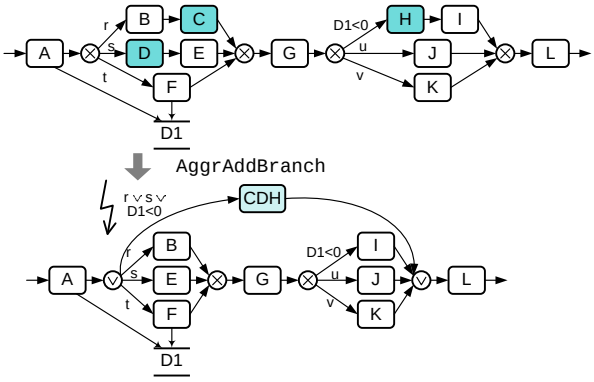


Abbildung 4.24: Beispiel für eine nicht mögliche Berechnung einer Kantenbedingung

Wie erwähnt, kann die Operation `AggrAddBranch` auf beliebige Mengen S angewendet werden. Eine Anwendung auf SESE-Bereiche ergibt keinen Sinn, da dies nach Anwendung entsprechender Vereinfachungsoperationen zum selben Ergebnis wie `AggrSESE` führt. Somit definieren wir als einzige Vorbedingung für `AggrAddBranch` der Vollständigkeit halber, dass S^* keinen SESE bilden darf.

Vorbedingung an S für `AggrAddBranch`:

- S^* ist kein SESE

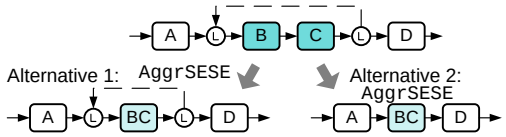
Aggregation von Schleifen

Enthält ein Prozess Schleifen und die zu aggregierenden Aktivitäten liegen entweder alle innerhalb oder alle außerhalb der Schleife, kann die Aggregation wie gewohnt mittels der bereits vorgestellten Operationen erfolgen.

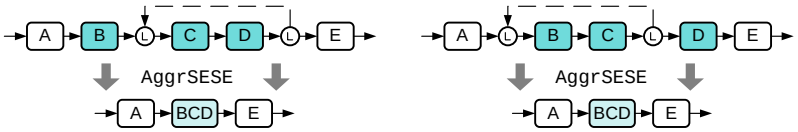
Sollen alle Aktivitäten eines Schleifenkörpers aggregiert werden, kann die bereits vorgestellte Operation `AggrSESE` angewendet werden. Hierzu existieren zwei Alternativen, die in Abbildung 4.25a dargestellt sind. Bei Alternative 1 bleibt die Schleife erhalten, Alternative 2 aggregiert die Schleifenanfangs- und Schleifenendknoten mit. Zur Unterscheidung dieser beiden Alternativen führen wir den Parameter *`LoopInSESE`* für die Operation `AggrSESE` ein mit den Werten: *`no`* (Alt. 1) und *`yes`* (Alt. 2).

Umfasst die Menge der zu aggregierenden Aktivitäten alle Aktivitäten einer Schleife und zusätzlich adjazente Aktivitäten unmittelbar vor oder nach dieser Schleife, so aggregiert `AggrSESE` wieder alles zu einem einzigen abstrakten Knoten (vgl. Abbildung 4.25b).

Interessante Fälle ergeben sich, wenn eine Teilmenge der zu aggregierenden Aktivitäten innerhalb und eine andere Teilmenge außerhalb einer Schleife liegen. Für zusammenhängende Aktivitätenmengen zeigt Abbildung 4.26 die existierenden Varianten einer Aggregation. Ein



(a) Aggregation aller Aktivitäten im Schleifenrumpf



(b) Aggregation von Aktivitäten innerhalb und außerhalb von Schleifen

Abbildung 4.25: Aggregation von Aktivitäten in Schleifen durch `AggrSESE`

solcher Anwendungsfall für die Aggregation kommt in der Praxis zum Beispiel beim Testen von Komponenten vor:

Beispiel 4-7: Aggregation in Verbindung mit Schleifen im Kontrollfluss
Wir nehmen an, dass eine E/E-Komponente in mehreren Iterationen auf einem Prüfstand getestet werden soll und jede Iteration wieder mit denselben vorbereitenden Schritten beginnt, in denen die Anlage neu justiert wird. Zudem muss direkt vor dem Start der Iterationen die Anlage initial auf das Prüfobjekt eingestellt werden. Für die Visualisierung sollen all diese Vorbereitungsaktivitäten in einem Schritt vor der Testschleife aggregiert dargestellt werden. Für die visualisierte View ist es primär unerheblich, ob (a) ursprünglich ein Schritt vor der Schleife lag und ob (b) sich die aggregierte Aktivität innerhalb oder außerhalb der Schleife befindet.

Proviado stellt für die Aggregation zusammenhängender Aktivitätenmengen in Verbindung mit Schleifen die Operation `AggrShiftLoop` zur Verfügung (vgl. Varianten 1a und b in Abbildung 4.26). Diese Operation bietet zwei Möglichkeiten für die Positionierung der bei der Aggregation erzeugten, abstrakten Aktivität: entweder innerhalb oder außerhalb der Schleife. Bei jeder dieser Varianten suggeriert die View, dass Aktivitäten entweder wiederholt werden, die eigentlich nur einfach ausgeführt werden (B in Variante 1a), oder dass Aktivitäten nur einfach ausgeführt werden, obwohl sie im Basisprozess wiederholt werden können (D in Variante 1b). Welche Variante der Operation `AggrShiftOut` zur Anwendung kommt, entscheidet der Parameter *`LoopShift`* mit den Werten *`in`* und *`out`*.

Alternativ dazu ist immer eine Aggregation mit Hilfe von `AggrAddBranch` möglich. Zu beachten ist, dass ein Einmünden des parallelen Zweiges in die Schleife nicht zulässig ist (vgl. Abbildung 4.27). Dies würde die postulierten Anforderungen an eine korrekte Struktur des Prozessschemas gemäß Abschnitt 4.2.1 verletzten und letztlich zu einer unklaren Semantik des Modells führen.

Vorraussetzung für die Anwendung von `AggrShiftLoop` ist, dass die zu aggregierenden Knoten zusammenhängend sind und innerhalb der Zusammenhangskomponente ein Schleifenknoten (Anfang oder Ende) liegt. Der Teil vor und nach dem Schleifenknoten kann dann beliebig gestaltet sein (z.B. einzelner Knoten, SESE, oder Vereinigungs- bzw. Verzweigungsbaum).

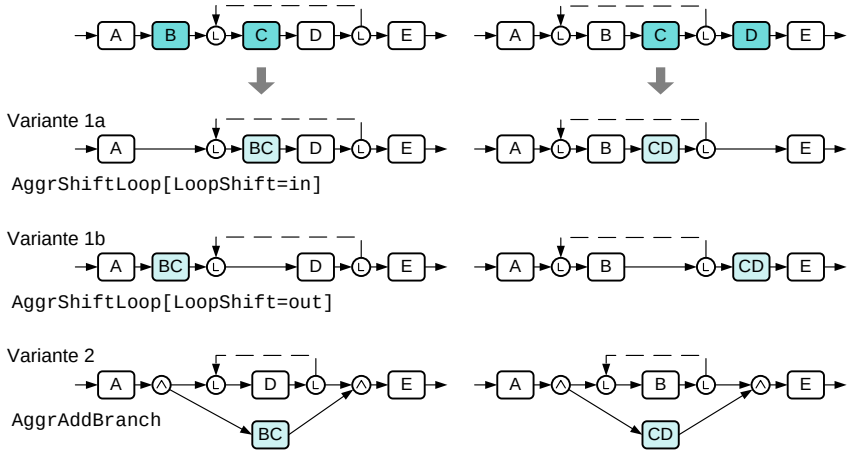


Abbildung 4.26: Varianten für die Aggregation von Aktivitäten in Schleifen

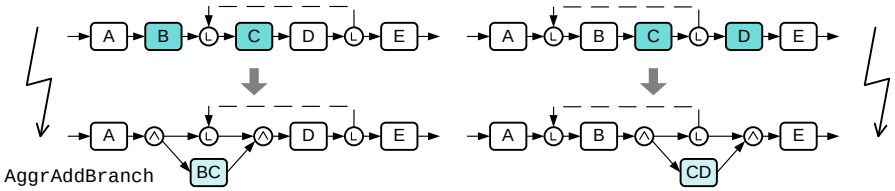


Abbildung 4.27: Strukturverletzung bei der Aggregation von Aktivitäten

Vorbedingung an S für `AggrShiftLoop`:

- S^* ist zusammenhängend
- S^* kann partitioniert werden in $S_1 \rightarrow l \rightarrow S_2$ mit $S_1 \cap S_2 = \emptyset$, l ist ein Schleifenknoten und es gilt $kgN(S_1) = l = ggV(S_2)$.
  - S_i kann eine Sequenz, ein SESE oder ein Vereinigungs- bzw. Verzweigungsbaum sein $(i = 1, 2)$

Wir haben in diesem Abschnitt eine Reihe elementarer Aggregationsoperationen vorgestellt, die sich mit Ausnahme von `AggrAddBranch` jeweils für die Aggregation spezieller Kontrollflussstrukturen eignen. Durch Kombination dieser Operationen können komplexe Aggregationen realisiert werden. Darauf aufbauend werden wir in Abschnitt 5.2.2 eine komplexe Operation AggregateCF vorstellen, die für eine gegebene Menge von Aktivitäten automatisch die richtige Elementaroperation auswählt. Durch die Existenz der Operation `AggrAddBranch` wird zudem sichergestellt, dass immer eine Aggregation möglich ist. D.h. unabhängig von der Struktur der zu aggregierenden Aktivitäten im Prozess existiert immer mindestens eine „passende“ Aggregationsoperation. Falls keine Elementaroperation aus der Menge `AggrSESE`, `AggrComplBranches`, `AggrShiftOut` und `AggrShiftLoop` anwendbar ist, kann die Aggregation in jedem Fall mittels `AggrAddBranch` durchgeführt werden. Neben der komplexen Operation AggregateCF werden wir in Kapitel 5 weitere, semantisch höherwertige Operationen kennen lernen, die die Aggregation einer gegebenen Aktivitätenmenge auf elementare Aggregationsoperationen abbilden.

Sollte in einem bestimmten Szenario das Ergebnis der Aggregation von der Vorstellung des Betrachters abweichen, so ist vorgesehen, die Menge der Elementaroperationen erweitern zu können. Einen Kandidaten für eine solche elementare Aggregationsoperation zeigt Abbildung 4.28.

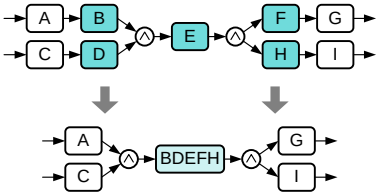


Abbildung 4.28: Mögliche Erweiterung der Operationsmenge

4.3.2.2 Eigenschaften der Aggregationsoperationen

In diesem Abschnitt analysieren wir die Elementaroperationen aus Abschnitt 4.3.2.1 im Detail. Dazu benötigen wir folgende Definition.

Definition 4.11 (Abhängigkeitsmenge) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema mit Aktivitätenmenge $A \subseteq N$. Sei weiter $\mathbb{D}_P \subseteq A \times A$ die Menge aller direkten und indirekten (über Strukturknoten hinweg aber ohne Berücksichtigung von Abhängigkeiten über Schleifenkanten) Kontrollflussabhängigkeiten zwischen Aktivitäten. Wir bezeichnen $\mathbb{D}_P$ als *Abhängigkeitsmenge*.

$$\begin{aligned}\mathbb{D}_P = \{(n_1, n_2) \in A \times A | & e = (n_1, n_2) \in E, ET(e) = ControlFlowEdge \quad \vee \\ & \exists \text{Pfad p von } n_1 \text{ nach } n_2 \text{ mit } p = (n_1, x_1, \ldots, x_k, n_2) : \\ & NT(x_i) \neq Activity \quad \wedge \quad ET((x_{i-1}, x_i)) = ControlFlowEdge\}\end{aligned}$$

Die Abhängigkeitsmenge eines Prozesses ist also genau die Menge der durch die Kontrollflusskanten gegebenen Abhängigkeiten zwischen Aktivitäten, erweitert um die Menge der indirekten Abhängigkeiten, in denen zwischen zwei Aktivitäten beliebig viele Strukturknoten und Kontrollflusskanten (aber keine Schleifenkanten) liegen. Ein Beispiel zeigt Abbildung 4.29. Die partielle Ordnungsrelation $\preceq$ (vgl. Definition 4.8) ist eine transitive Hülle bzgl. $\mathbb{D}_P$.

Wir interessieren uns für die Beziehung der Abhängigkeitsmengen eines Prozesses P und einer zugehörigen Prozess-View $V(P)$. Zu diesem Zweck sei $\mathbb{D}_P$ die Abhängigkeitsmenge von P und $\mathbb{D}_{V(P)}$ die Abhängigkeitsmenge der auf P definierten View $V(P)$. Des Weiteren benötigen wir eine Projektion der Abhängigkeiten von $V(P)$ auf P, bezeichnet mit $\mathbb{D}'_{V(P)}$. Diese Menge ergibt sich, indem man die Abhängigkeiten zwischen den durch die View nicht veränderten Aktivitäten und der abstrakten Aktivität durch die Abhängigkeiten im Originalprozess ersetzt. Als Beispiel betrachten wir Abbildung 4.29. Wie man leicht erkennen kann, enthält $\mathbb{D}'_{V(P)}$ zusätzliche Abhängigkeiten (hier $\{(D, C), (B, E)\}$). Wir bezeichnen diese Eigenschaft als *abhängigkeitserzeugend*.

Algorithmus 4-4 zeigt die Berechnung der projizierten Abhängigkeitsmenge. Dabei werden zunächst die Abhängigkeiten zwischen der abstrakten Aktivität und der Umgebung gelöscht und anschließend die projizierten Abhängigkeiten zwischen den aggregierten Aktivitäten und der Umgebung der abstrakten Aktivität wieder eingefügt. Der

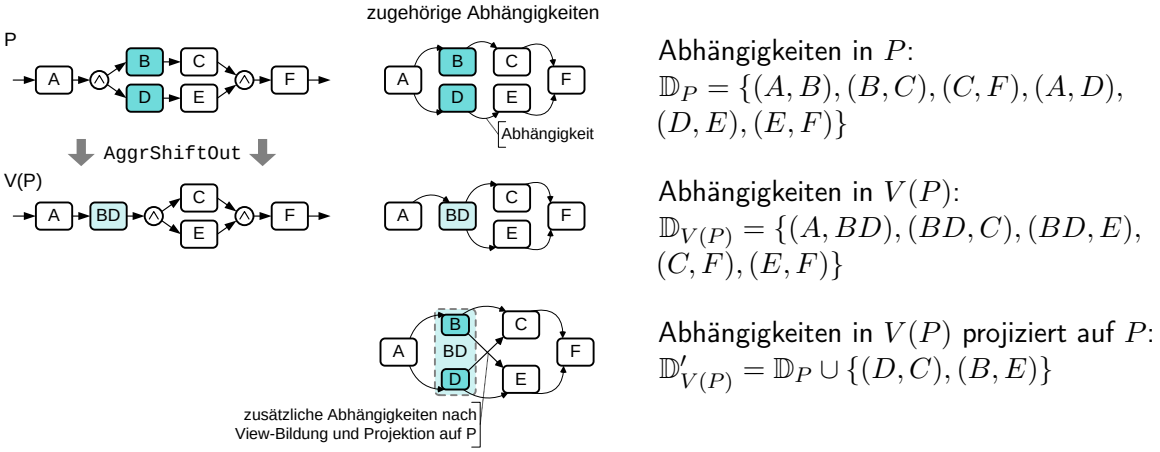


Abbildung 4.29: Beispiele für Abhängigkeitsmengen

Vollständigkeit halber werden auch die ursprünglich in P vorhandenen Abhängigkeiten zwischen den aggregierten Aktivitäten wieder eingefügt (vgl. auch Abbildung 4.29).

Algorithmus 4-4 : CalculateProjectedDependencies$(P, V(P))$

Input :
$P = (N, E, EC, NT, ET)$ Prozessschema und
$P' = V(P) = (N', E', EC', NT', ET')$ Prozess-View auf P mit
$AggrNodes$ Menge der aggregierten Aktivitäten von P
A' Menge der durch die View nicht veränderten Aktivitäten
A'_N Menge der neuen, aggregierten (abstrakten) Aktivitäten

Output :
$\mathbb{D}'_{V(P)}$ projizierte Abhängigkeitsmenge

1 $\mathbb{D}'_{V(P)} = \mathbb{D}_{V(P)}$
2 **forall** $n_1 \in A'_N$, $n_2 \in A'$ *mit* $(n_1, n_2) \in \mathbb{D}_{V(P)}$ **do**
3 entferne $(n_1, n_2) \in \mathbb{D}_{V(P)}$ aus $\mathbb{D}'_{V(P)}$
4 füge $\{(n, n_2) | n \in aggr^{-1}(n_1)\}$ in $\mathbb{D}'_{V(P)}$ ein
5 *// analog für* $n_2 \in A'_N$
6 **forall** $n_1 \in A'$, $n_2 \in A'_N$ *mit* $(n_1, n_2) \in \mathbb{D}_{V(P)}$ **do**
7 entferne $(n_1, n_2) \in \mathbb{D}_{V(P)}$ aus $\mathbb{D}'_{V(P)}$
8 füge $\{(n_1, n) | n \in aggr^{-1}(n_2)\}$ in $\mathbb{D}'_{V(P)}$ ein
9 *// füge die Abhängigkeiten zwischen* $AggrNodes$ *ein*
10 $\mathbb{D}'_{V(P)} = \mathbb{D}'_{V(P)} \cup \{d = (n_1, n_2) | d \in \mathbb{D}_P \wedge n_1 \in AggrNodes \wedge n_2 \in AggrNodes\}$

An dieser Stelle sei darauf hingewiesen, dass Betrachtungen zu erhaltenen bzw. gelöschten Kontrollflussabhängigkeiten zwischen Aktivitäten theoretischer Natur sind. Durch die Aggregation werden die betroffenen Aktivitäten aus dem Prozessmodell entfernt und sind daher in der View nicht mehr enthalten. Stattdessen wird ein neuer Knoten (an derselben Stelle oder parallel zu den ursprünglichen Aktivitäten) eingefügt, der in den schematischen Abbildungen dieses Kapitels aus Gründen der Lesbarkeit den Namen der ursprünglichen Aktivität enthält. In der Realität lässt sich unter Umständen logisch rekonstruieren, das ein Teil der abstrakten Aktivität aus der ursprünglichen Aktivität stammt (z.B. anhand von Parameterwerten).

Wir können nun exakt angeben, was mit Kontrollflussabhängigkeiten bei der Bildung einer View passiert. Dazu betrachten wir die transitiven Hüllen der Abhängigkeitsmengen. $\mathcal{H}$ sei der entsprechende Hüllenoperator.

Definition 4.12 (Abhängigkeitsbeziehungen) Seien $P = (N, E, EC, NT, ET)$ ein Prozessschema und $V(P)$ eine durch Anwendung der View-Operation V entstandene View von P. Seien $\mathbb{D}_P$ und $\mathbb{D}'_{V(P)}$ die Abhängigkeitsmengen wie von Definition 4.11 festlegt.

- V wird als **abhängigkeitslöschend** bezeichnet $\iff$ es gibt in $\mathcal{H}(\mathbb{D}_P)$ Abhängigkeitsbeziehungen, die in $\mathcal{H}(\mathbb{D}'_{V(P)})$ nicht mehr vorhanden sind.
- V wird als **abhängigkeitserzeugend** bezeichnet $\iff$ es gibt in $\mathcal{H}(\mathbb{D}'_{V(P)})$ Abhängigkeitsbeziehungen, die in $\mathcal{H}(\mathbb{D}_P)$ nicht enthalten sind.
- V wird als **abhängigkeitserhaltend** bezeichnet $\iff$ $V(P)$ ist weder abhängigkeitslöschend noch abhängigkeitserzeugend.

Es ist offensichtlich, dass eine Reduktionsoperation immer abhängigkeitslöschend ist. Bezogen auf die Aggregation von Aktivitäten existieren Elementaroperationen zu allen drei Arten. Anhand von Abbildung 4.16 ist zum Beispiel erkennbar, dass `AggrSESE` abhängigkeitserhaltend ist. Im Gegensatz dazu ist `AggrAddBranch` abhängigkeitslöschend. So gilt beispielsweise in Abbildung 4.21a $(C, D) \notin \mathbb{D}'_{V(P)}$. Demgegenüber zeigt Abbildung 4.29, dass `AggrShiftOut` wegen $(B, E) \in \mathcal{H}(\mathbb{D}'_{V(P)})$ abhängigkeitserzeugend ist. Diese Aussagen gelten generell für die entsprechenden Operationen.

Satz 4.1 verdeutlicht die Relation zwischen den Abhängigkeitsbeziehungen (vgl. Definition 4.12) und der Ordnungserhaltung (vgl. Definition 4.9).

Satz 4.1 (Abhängigkeiten und Ordnungserhaltung) Sei V eine aggregierende View-Operation. Dann:

(i) V ist abhängigkeitslöschend $\Rightarrow$ V ist nicht streng ordnungserhaltend

(ii) V ist abhängigkeitserhaltend $\Rightarrow$ V ist streng ordnungserhaltend

Der Beweis von Satz 4.1 folgt direkt aus der Definition der Eigenschaften und der Definition bzw. Berechnung der Abhängigkeitsmengen.

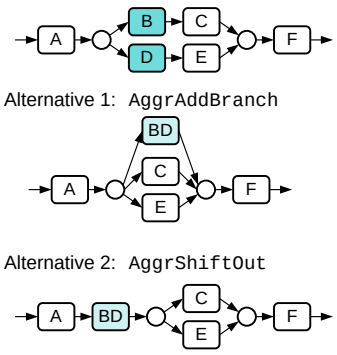
Beweis: Seien $P = (N, E, EC, NT, ET)$ ein Prozessschema und $V(P)$ eine durch Anwendung der Aggregationsoperation V entstandene View von P.

(i) Da V abhängigkeitslöschend ist, existiert $d = (a, b) \in \mathbb{D}_P$ mit $d \notin \mathbb{D}'_{V(P)}$ (o.b.d.A. $a \in AggrNodes$ und $b \notin AggrNodes$). Somit gilt $a \not\preceq b$. Seien $a' = VNode(a)$ und $b' = VNode(b)$, d.h. $a \in A'_N$ und $b \in A'_U$. Gemäß Algorithmus 4-4 gilt demnach auch $a' \not\preceq b'$ (da sonst in Zeile 8 die Abhängigkeit (a, b) in $\mathbb{D}'_{V(P)}$ eingefügt worden wäre).
Wäre V streng ordnungserhaltend, müsste aber auch $a' \preceq b'$ gelten.

(ii) Da V abhängigkeitserhaltend ist, gilt $\mathcal{H}(\mathbb{D}_P) = \mathcal{H}(\mathbb{D}'_{V(P)})$. Somit gilt für alle Abhängigkeiten $d = (a, b) \in \mathbb{D}_P$: $a \preceq b$. Aus Algorithmus 4-4 ergibt sich damit auch $a' \preceq b'$ mit $a' = VNode(a)$ und $b' = VNode(b)$. Die Behauptung folgt somit aus der Transitivität der Ordnung $\preceq$. □

In der Praxis sind die Eigenschaften einer View nicht erst in der Retrospektive (also welche Eigenschaften besitzt eine gerade erstellte View) von Interesse. Die Eigenschaften formulieren eher Anforderungen an die View-Bildung, die bei der View-Definition zu berücksichtigen sind. Auf diesen Aspekt werden wir später im Rahmen der Definition höherwertiger Operationen noch detailliert eingehen, wollen hier aber schon einen ersten Eindruck vermitteln.

Zu diesem Zweck betrachten wir das Beispiel aus Abbildung 4.30. Hier sollen die Aktivitäten B und D aggregiert werden. Mit den uns zur Verfügung stehenden Elementaroperationen existieren zwei Alternativen: (1) Aggregation durch `AggrAddBranch` und damit Anlegen eines weiteren Zweiges mit der abstrakten Aktivität, oder (2) Aggregation durch `AggrShiftOut`, wodurch die abstrakte Aktivität vor die Verzweigung gezogen wird. Betrachten wir kurz die Auswirkungen der jeweiligen Operationen. Ein Blick auf die Abhängigkeitsmengen zeigt, dass durch `AggrAddBranch` Abhängigkeiten gelöscht und durch `AggrShiftOut` Abhängigkeiten erzeugt werden. Insbesondere können wir durch Vorgabe bestimmter erwünschter Eigenschaften zwischen verschiedenen möglichen Elementaroperationen auswählen.



Abhängigkeitsmenge von P:
$\mathbb{D}_P = \{(A,B),(B,C),(C,F),(A,D),(D,E),(E,F)\}$

Abhängigkeiten die durch `AggrAddBranch` entfernt wurden:
$\{(B,C),(D,E)\}$

d.h. $\mathbb{D}'_{V(P)} = \mathbb{D}_P \setminus \{(B,C),(D,E)\}$

Abhängigkeiten die durch `AggrShiftOut` erzeugt wurden:
$\{(D,C),(B,E)\}$
d.h. $\mathbb{D}'_{V(P)} = \mathbb{D}_P \cup \{(D,C),(B,E)\}$

Abbildung 4.30: Abhängigkeitsmengen im Vergleich für `AggrShiftOut` und `AggrAddBranch`

Bei der Aggregation von Schleifen unter Anwendung der Operation `AggrShiftLoop` werden je nach Parametrisierung der Operation entweder Aktivitäten aus einer Schleife herausgezogen oder umgekehrt in sie hineingezogen. Zur genauen Charakterisierung dieses Verhaltens führen wir folgende Definition ein.

Definition 4.13 (Iterationen) Sei $P = (N, E, EC, NT, ET)$ ein Prozessschema mit einer Schleife und sei $V(P)$ eine zugehörige View.

- $V(P)$ wird als **Iterations-einbeziehend** bezeichnet $\iff$ es existiert eine Aktivität, die in P außerhalb der Schleife liegt, aber in $V(P)$ innerhalb (als Teil einer abstrakten Aktivität).
- $V(P)$ wird als **Iterations-ausschließend** bezeichnet $\iff$ es existiert eine Aktivität, die in P innerhalb der Schleife liegt, aber in $V(P)$ außerhalb (als Teil einer abstrakten Aktivität).

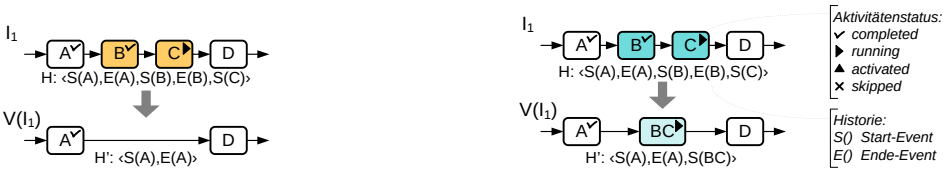
In besonderen Fällen kann `AggrShiftOut` die Eigenschaft „Iterations-ausschließend“ besitzen. Dies ist genau dann gegeben, wenn ein Zweig eine Schleife enthält, diese aber nicht vollständig aggregiert werden soll.

4.3.3 Bildung von Views auf Prozessinstanzen

Bislang haben wir nur Views auf Prozessschemata betrachtet. Im folgenden Abschnitt behandeln wir zusätzlich Views auf Prozessinstanzen (vgl. Definition 4.6).

4.3.3.1 Instanz-Views

Bei der Bildung von Instanz-Views muss der Ausführungszustand der erzeugten View, d.h. die Zustände der konkreten und der abstrakten Aktivitäten, bestimmt und die Historie für die Instanz-View entsprechend angepasst werden. Beispiele sind in Abbildung 4.31a (Reduktion) und Abbildung 4.31b (Aggregation) dargestellt.



(a) Reduktion (b) Aggregation

Abbildung 4.31: View-Bildung unter Berücksichtigung von Aktivitätenzuständen

Die folgende Funktion *VNS* (*view node state*) berechnet den Zustand einer abstrakten Aktivität basierend auf den Zuständen der zugrunde liegenden Aktivitäten $x \in X$ der Originalinstanz. Die Funktion ergibt sich intuitiv aus den in X vorkommenden Aktivitätenzuständen. Die vollständige Tabelle aller möglichen Zustandskombinationen, aus der sich die Funktion *VNS* ableitet, ist Anhang A zu entnehmen.

$$VNS(X) = \begin{cases} NotActivated & \forall x \in X : NS(x) \notin \{Activated, Running, \\ & Completed\} \wedge \exists x \in X : NS(x) = NotActivated \\ Activated & \exists x \in X : NS(x) = Activated \wedge \\ & \forall x \in X : NS(x) \notin \{Running, Completed\} \\ Running & \exists x \in X : NS(x) = Running \vee \\ & \exists x_1, x_2 \in X : NS(x_1) = Completed \wedge \\ & (NS(x_2) = NotActivated \vee NS(x_2) = Activated) \\ Completed & \forall x \in X : NS(x) \notin \{NotActivated, Running, \\ & Activated\} \wedge \exists x \in X : NS(x) = Completed \\ Skipped & \forall x \in X : NS(x) = Skipped \end{cases}$$

Definition 4.14 (View auf einer Prozessinstanz) Sei $I = (P, NS, \mathcal{H})$ eine Instanz des Prozessschemas $P = (N, E, EC, NT, ET)$. Sei weiter $V(P) = (N', E', EC', NT', ET')$ eine View auf P mit $AggrNodes$ und $RedNodes$ als zugehörige Aggregations- und Reduktionsmengen (vgl. Definition 4.7).

Eine View $V(I)$ auf Instanz I ist dann ein Tupel $(V(P), NS', \mathcal{H}')$ mit

- $V(P)$ ist die View auf P.
- $NS' : N' \rightarrow ExecutionStates$ mit $NS'(n') = VNS(VNode^{-1}(n'))$ weist jeder View-Aktivität den entsprechenden Ausführungszustand zu.
- $\mathcal{H}'$ ist die reduzierte/aggregierte Historie der Instanz bezogen auf V. Sie wird aus $\mathcal{H}$ abgeleitet, indem (1) alle Einträge e_i entfernt werden, die zu Aktivitäten in $RedNodes$ gehören und indem (2) (für alle j) das erste (letzte) Auftreten eines Start-Events (Ende-Events) einer Aktivität aus $AggrNodes_j$ durch das Start-Event (Ende-Event) der abstrakten Aktivität ersetzt und die verbleibenden Start-(Ende-)Events entfernt werden.

Beispiele von Instanz-Views mit zugehörigen Historien sind in Abbildung 4.31 dargestellt.

Algorithmisch verläuft die View-Bildung auf Prozessinstanzen identisch zu der auf Prozessschemata. Zusätzlich sind die Zustände aggregierter Aktivitäten sowie die aggregierte Historie zu berechnen. Dazu wird innerhalb der Funktion `createAggrNode(X)` die oben definierte Zustandszuordnungsfunktion $VNS(X)$ aufgerufen, die den Zustand für die Menge von aggregierten Aktivitäten X berechnet. Außerdem wird die Historie wie in Definition 4.14 beschrieben angepasst.

4.3.3.2 Instanzbezogene Eigenschaften der View-Operationen

Abbildung 4.32 zeigt dasselbe Beispiel wie Abbildung 4.30, hier allerdings angereichert um Zustände. Dabei fällt auf, dass bei Anwendung von `AggrShiftOut` ein aus Sicht eines operationalen Ausführungsmodells „inkonsistenter“ Gesamtzustand auftritt, da sich zwei aufeinander folgende Aktivitäten im Zustand *Running* befinden. Für die Visualisierung von Prozessen ist dies akzeptabel, solange dadurch kompaktere und übersichtlichere Prozessdarstellungen möglich sind. Zudem tritt diese Situation nur sehr selten ein und fällt dann nur dem geschulten Auge von Prozessexperten auf. Zur genaueren Beschreibung der Situation führt Definition 4.15 zunächst zwei Arten der Zustandskonsistenz für Instanzen ein.

Definition 4.15 (Zustandskonsistenz einer Prozessinstanz) Seien $I = (P, NS, \mathcal{H})$ eine laufende Prozessinstanz (d.h. in I wurde mindestens eine Aktivität aktiviert und die Instanz ist noch nicht abgeschlossen).

- I ist **streng zustandskonsistent** $\iff$ Für alle Pfade in I, die vom Start- zum Endeknoten führen und die keine Aktivitäten im Zustand *Skipped* enthalten, befindet sich genau eine Aktivität im Zustand *Activated* oder *Running*.
- I ist **zustandskonsistent** $\iff$ Für alle Pfade in I, die vom Start- zum Endeknoten führen und die keine Aktivitäten im Zustand *Skipped* enthalten, befindet sich nicht mehr als eine Aktivität im Zustand *Activated* oder *Running*.

Die Zustandskonsistenz einer Prozessinstanz ist eine zustandsabhängige Eigenschaft, d.h. bei Veränderung des Prozesszustandes kann sich auch dessen Zustandskonsistenz ändern. Gemäß unserer Definition von Prozessinstanzen (vgl. Definition 4.6 auf Seite 51) sind Instanzen immer streng zustandskonsistent. Durch die View-Bildung kann es jedoch zu nicht-streng zustandskonsistenten bzw. sogar zustandsinkonsistenten Instanzen kommen. Abbildung 4.33a zeigt ein Beispiel für eine Instanz, die nach Reduktion der Aktivitäten B und C nur noch zustandskonsistent, aber nicht mehr streng zustandskonsistent ist.

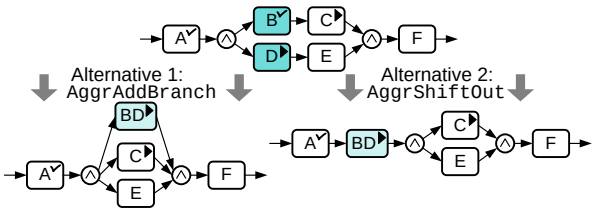


Abbildung 4.32: Alternativen bei der Aggregation auf Prozessinstanzen

Zur genaueren Charakterisierung der Zustandskonsistenz im Zusammenhang mit der View-Bildung dient Definition 4.16.

Definition 4.16 (Zustandskonsistenz) Seien $I = (P, NS, \mathcal{H})$ eine Prozessinstanz und $V(I)$ eine durch Anwendung der View-Bildungsoperation erzeugte View auf I. Dann:

- V ist **streng zustandskonsistent** $\iff$ Für alle möglichen Ausführungen von I ist $V(I)$ streng zustandskonsistent.
- V ist **zustandskonsistent** $\iff$ Für alle möglichen Ausführungen von I ist $V(I)$ zustandskonsistent.

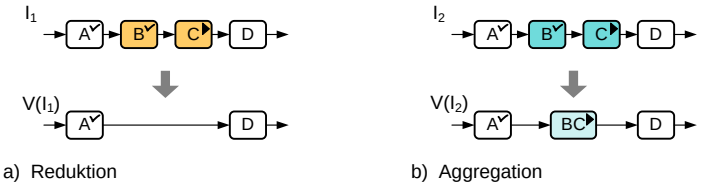
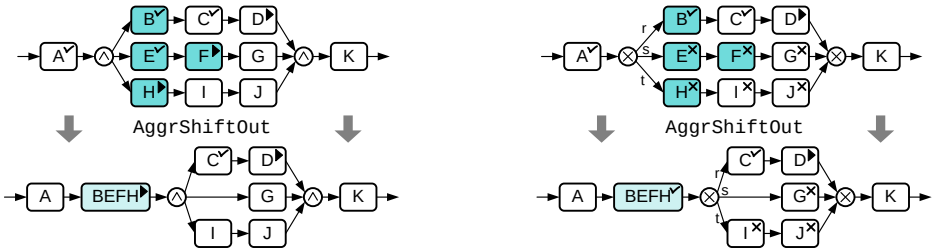


Abbildung 4.33: Auswirkungen der View-Bildung auf die Zustandskonsistenz

Wie aus Abbildung 4.32 ersichtlich, kann eine Aggregation mittels `AggrShiftOut` einen inkonsistenten Ausführungszustand erzeugen (Alternative 2), wohingegen `AggrAddBranch` einen konsistenten Zustand bewahrt (Alternative 1). Wann dies der Fall ist, soll anhand von Abbildung 4.34 erläutert werden. Die einzige Operation, durch die eine derartige Zustandsinkonsistenz hervorgerufen werden kann, ist `AggrShiftOut`. D.h. die Abhängigkeitserzeugung ist eine notwendige Voraussetzung für das Auftreten einer Inkonsistenz, wie sie Abbildung 4.34a zeigt. In diesem Beispiel werden die zu aggregierenden Aktivitäten in Form der abstrakten Aktivität vor den AND-Split-Knoten gezogen. Dadurch kann eine Situation auftreten, in der die zu aggregierenden Aktivitäten eines der parallelen Pfade abgeschlossen und die auf diesem Pfad nachfolgenden Aktivitäten bereits aktiviert bzw. laufend sind. Gleichzeitig befinden sich die

zu aggregierenden Aktivitäten eines der anderen parallelen Pfade noch in Ausführung. Im Gegensatz dazu tritt eine Inkonsistenz bei alternativen Verzweigungen (XOR) nicht auf, da in diesem Fall immer nur ein Pfad zur Ausführung kommt (vgl. Abbildung 4.34b). Für den Fall einer OR-Verzweigung ist die Möglichkeit des Auftretens einer Zustandsinkonsistenz abhängig von den Verzweigungsbedingungen. Wenn mehr als ein Pfad gleichzeitig ausgeführt werden kann, ist theoretisch wieder eine Zustandsinkonsistenz möglich.



(a) `AggrShiftOut` in Verbindung mit AND-Verzweigung

(b) `AggrShiftOut` in Verbindung mit XOR-Verzweigung

Abbildung 4.34: Zustandsinkonsistenzen bei Aggregation durch `AggrShiftOut`

Abbildung 4.33 zeigt ein Beispiel, in dem es durch Reduktion zu einer nur zustandskonsistenten Instanz kommt. Dies ist eine grundsätzliche Eigenschaft der Reduktion, da diese Aktivitäten aus dem Kontrollflussgraphen entfernt und dadurch „Löcher“ im Prozess entstehen. Das Entfernen von Aktivitäten aus dem Kontrollflussgraphen führt in analoger Art und Weise dazu, dass die Operation `AggrAddBranch` ebenfalls nicht streng zustandskonsistent ist. Bei der View-Bildung über Schleifen ist die strenge Zustandskonsistenz sichergestellt, solange nicht `AggrAddBranch` angewendet wird. Durch das in Abschnitt 4.2.1 beschriebene Zustandsverhalten von Schleifen, bei dem der Zustand aller Aktivitäten innerhalb einer Schleife bei Beginn einer neuen Iteration auf *NotActivated* gesetzt wird, treten keine Inkonsistenzen auf.

Es bleibt zu diskutieren, warum View-Operationen, die zu einer Zustandsinkonsistenz führen, überhaupt zugelassen werden. Alle bisher aus der Literatur bekannten View-Bildungsmechanismen verbieten derartige Operationen. Gleichzeitig existieren jedoch Workflow-Management-Systeme, die einen vorzeitigen Start von aufeinander folgenden Aktivitäten unterstützen und somit zu denselben „Inkonsistenzen“ bei der Ausführung führen [GCG04, GCG06, Beu03]. Offensichtlich gibt es also selbst auf operationaler Ebene Anwendungen für diesen Fall. Ungeachtet dessen ist eine derartige Inkonsistenz in einer Visualisierungskomponente kein Problem, da diese lediglich die Ausführungsinformationen der zugrunde liegenden Systeme widerspiegelt. Diese sind auch für die Steuerung der Prozessinstanzen zuständig. Für den Betrachter eines Prozesses dagegen ist intuitiv verständlich, was mit einem derartigen Zustand gemeint ist. Auch Beispiel 4-4 verdeutlicht die praktische Relevanz einer solchen View-Bildung. Sollten für eine spezielle Anwendung konsistente Zustände dennoch wichtig sein, erlaubt der Proviado-View-Mechanismus diese durch entsprechende Parametrisierung der View-Bildungsoperationen sicherzustellen. Details dieser Parametrisierung werden in Abschnitt 5.2.2 vorgestellt.

4.3.4 Vereinfachungsoperation (Simplify)

Sowohl Reduktion als auch Aggregation hinterlassen gegebenenfalls obsolete Kontrollflussstrukturen (z.B. leere Kanten, überflüssige Verzweigungen, etc.), die für die Darstellung nicht nötig sind und die zudem irrelevante Informationen darstellen. Aufgabe der Proviado-Vereinfachungsoperationen ist es, diese überflüssigen Konstrukte im Anschluss an eine Reduktion oder Aggregation aus dem Prozessmodell zu entfernen. Abbildung 4.35 zeigt beispielhaft ausgewählte Vereinfachungsoperationen. Dabei handelt es sich jeweils um primitive Graphumformungen, weshalb an dieser Stelle auf eine formale Darstellung verzichtet wird.

Ein Fall, der detaillierter betrachtet werden soll, ist die Vereinfachung von alternativen Verzweigungen. Hier müssen die Kantenbedingungen gegebenenfalls neu berechnet werden. In Abbildung 4.35 sind zwei Beispiele angegeben. Beim Zusammenfassen leerer Zweige in alternativen Verzweigungen mittels `SimEmptyBranch` werden die Kantenbedingungen EC der ursprünglichen Kanten mit $'\vee'$ konkateniert. Bei der Vereinfachung durch Zusammenfassen von Verzweigungsknoten (`SimObsoleteSplit`) wird die Bedingung des ersten Knotens mit den Bedingungen des nachfolgenden Verzweigungsknotens mittels $'\wedge'$ kombiniert.

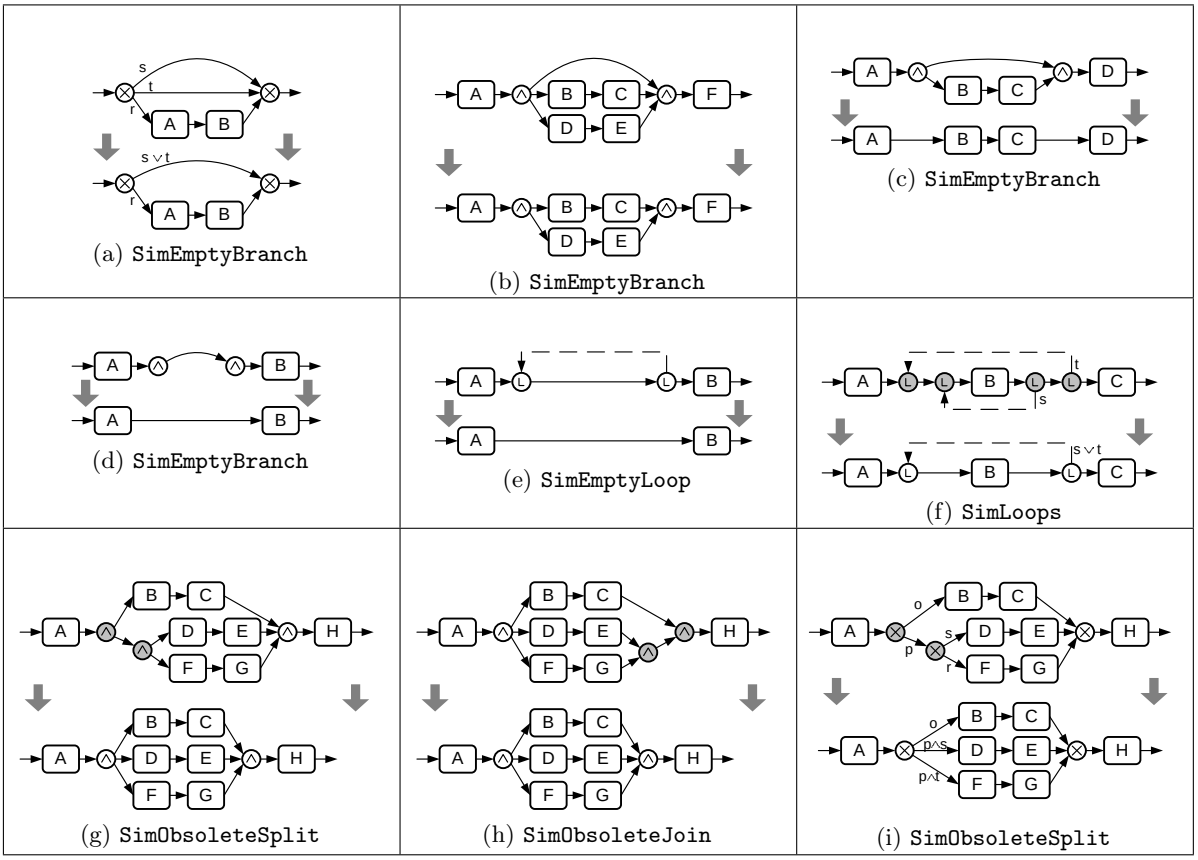


Abbildung 4.35: Vereinfachungsoperationen

4.4 View-Operationen für Attribute von Prozesselementen

Bei der Betrachtung der View-Operationen sind wir bislang davon ausgegangen, dass alle Prozesselemente atomare Objekte sind, die bei der Aggregation miteinander „verschmelzen“. In der Realität ist jedes Prozesselement ein komplexes Objekt mit einer Reihe von Attributen (vgl. Abbildung 4.36). Diese Attribute können zum einen prozessinterne Daten sein (z.B. Name des Elements, Startzeit, Endezeit), zum anderen aber auch anwendungsspezifische Daten (z.B. Ausführungskosten einer Aktivität). Bei der View-Bildung müssen all diese Daten mitberücksichtigt werden.

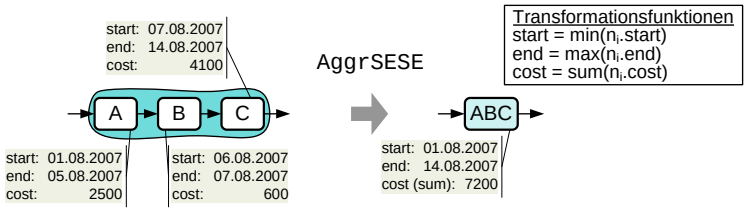


Abbildung 4.36: Beispiele für die Aggregation von Attributwerten

Prinzipiell existieren zwei Anwendungsformen für Attributoperationen bei der View-Bildung:

1. **Integriert in andere View-Operationen:** Dies ist vor allem für die Aggregation relevant, bei der die Attribute und deren Werte auf die abstrakte Aktivität übertragen werden sollen. Dazu werden entsprechende Attribut-Transformationsfunktionen definiert. Bei der Reduktion spielen Attribute keine Rolle, da hier die Prozesselemente mitsamt ihren Attributen aus dem Prozess entfernt werden.
2. **Eigenständige Operationen:** Unabhängig von der Aggregation sollten Attributwerte aus einem Modell entfernt (Projektion) oder zu neuen Werten verschmolzen werden können (Transformation).

Abbildung 4.37 illustriert sowohl für eine Attributoperation im Zusammenhang mit einer Aggregation als auch für unabhängige Attributoperationen wichtige Anforderungen:

1. In Kombination mit der Aggregation von Kontrollflusselementen (hier Aktivitäten B und C) sollen deren zugehörige Attribute aggregiert werden. Dazu wird beispielsweise eine Transformationsfunktion t_x definiert, die Attribut x der Aktivitäten B und C zu einem neuen Attribut x der abstrakten Aktivität BC aggregiert (analog für t_y und y).
2. Unabhängig von der Kontrollflussaggregation sollen die Attribute x und y mittels einer Transformationsfunktion h zu einem neuen Attribut w kombiniert werden. Ein Beispiel wäre die Berechnung der Arbeitskosten als Produkt aus Arbeitszeit und Stundensatz.
3. Das Attribut z von Aktivität A soll explizit entfernt werden, da es vertrauliche Daten enthält.

In relationalen Datenbanksystemen existieren ähnliche Aggregationsoperatoren zur Berechnung von Attributwerten [ÖV99, EN02]. Bevor wir die für die Verarbeitung von Prozessattributen benötigten Operationen vorstellen, erweitern wir das in Definition 4.1 vorgestellte, formale Prozessmodell um den Aspekt der Attribute.

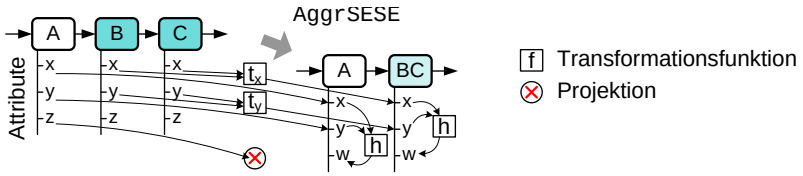


Abbildung 4.37: Beispiele für Attributoperationen

4.4.1 Erweiterung des Prozessmodells um Attribute

$\mathcal{A}$ bezeichne die Menge der verfügbaren Attribute. Dann ergibt sich folgende erweiterte Definition eines Prozessschemas:

Definition 4.17 (Prozessschema mit Attributen) Ein Prozessschema ist ein Tupel $P = (N, E, EC, NT, ET, attrib, val)$ mit

- N, E, EC, NT, ET sind wie in Definition 4.1 festgelegt.
- $attrib : N \cup E \to \mathfrak{P}(\mathcal{A})$ ordnet jedem Prozesselement eine Menge von Attributen zu.
- $val : (N \cup E) \times \mathcal{A} \to valueDomain(\mathcal{A})$ weist einem Attribut a eines Prozesselements c den entsprechenden Wert zu.

$$val(c, a) = \begin{cases} \text{Wert von } a, & \text{if } a \in attrib(c) \\ \bot, & \text{if } a \notin attrib(c) \end{cases}$$

Für eine View erfolgt die Erweiterung analog, d.h.

$$V(P) = (N', E', EC', NT', ET', attrib', val')$$

Als Notation für ein Attribut x eines Prozesselements A verwenden wir im Folgenden $A.x$. Aus dem jeweiligen Kontext erschließt sich dabei, ob es sich um das Attribut oder dessen Attributwert (d.h. $val(A.x)$) handelt.

4.4.2 Projektion

In Anlehnung an SQL [EN02] dient die Projektion dazu, über die Sichtbarkeit von Attributen zu entscheiden. Attribute können für eine Menge von Prozessobjekten eingeblendet (alle anderen Attribute werden entfernt) oder ausgeblendet (d.h. alle anderen Attribute sind sichtbar) werden. Die Operation kann auf beliebige Prozesselemente angewendet werden. Prinzipiell lassen sich zwei Möglichkeiten unterscheiden, wie die Attributmenge für die Projektion spezifiziert wird. Wir bezeichnen die Operation, die ein Attribut löscht, mit `reduce` und die Operation, die ein Attribut bei der View-Bildung erhält bzw. in die View übernimmt, mit `grant`.

destruktive Spezifikation der Projektion Sie definiert, welche Attribute entfernt werden sollen (alle anderen Attribute bleiben implizit erhalten)

- `reduce` $A.x$, $A.y$

- (`grant` *) implizit

konstruktive Spezifikation der Projektion Sie definiert, welche Attribute erhalten bleiben sollen (alle anderen Attribute werden implizit entfernt)

- `grant` $A.x$, $A.y$
- (`reduce` *) implizit

Für Proviado-Views verwenden wir die destruktive Variante, da standardmäßig möglichst viele Attribute erhalten bleiben sollen. Die konstruktive Variante würde viel Definitionsaufwand bedeuten, da der Verbleib jedes einzelnen Attributs explizit spezifiziert werden muss. Attribute, die erst zur Laufzeit hinzukommen, sollen standardmäßig in die View übernommen werden. Bei einer konstruktiven Vorgehensweise müssten diese jedoch im Voraus spezifiziert werden.

Mittels der Operation `ProjectAttr` kann spezifiziert werden, welche Attribute eines Prozesselements entfernt werden sollen (vgl. Abbildung 4.37). D.h. für `ProjectAttr(n,X)` mit $n \in N \cup E$ und $X \in \mathfrak{P}(\mathcal{A})$ gilt: $attrib'(n) = attrib(n) \setminus X$

4.4.3 Transformation

Wie in Abbildung 4.36 angedeutet, dient die Transformation dazu, die Werte mehrerer Attribute auf einen neuen Wert abzubilden. Sie ist in Verbindung mit der Aggregation relevant. Im Beispiel aus Abbildung 4.36 sollen die Attributwerte der Aktivitäten A, B und C auf die abstrakte Aktivität ABC übertragen werden. Zu diesem Zweck kommen Transformationsfunktionen zum Einsatz, die gemäß einer Abbildungsvorschrift den neuen Attributwert aus den existierenden Attributwerten berechnen. Beispielsweise ergibt sich der Startzeitpunkt einer abstrakten Aktivität als das Minimum der Startzeitpunkte der aggregierten Aktivitäten. Für ein Kostenattribut werden im einfachen Fall die Kosten der einzelnen Aktivitäten summiert. Je nach Anwendungsfall kann hier auch eine Durchschnittsbildung sinnvoll sein.

Definition 4.18 (Transformationsfunktion) Eine Transformationsfunktion berechnet aus einer Menge von Werten einen neuen Wert. Die Wertemengen der abzubildenden Attribute und des Zielattributs seien $\mathbb{W}_i$ und $\mathbb{W}$.

$$f : \mathbb{W}_1 \ldots \mathbb{W}_n \rightarrow \mathbb{W}$$

Je nach Typ des Attributs sind verschiedenen Transformationsfunktionen anwendbar bzw. erforderlich. Eine Transformationsfunktion für die Aggregation von Ausführungszuständen (*VNS*) haben wir in Abschnitt 4.3.3 vorgestellt. Beispiele für weitere gängige Transformationsfunktionen sind in Tabelle 4.4 zusammengestellt.

Numerische Attribute lassen sich durch einfache Aggregationsfunktionen transformieren. Bei textuellen Attributen ist die Transformation schwieriger. Die einfachste Lösung besteht in einem Aneinanderhängen von Zeichenfolgen (CONCAT). Die Funktionen FIRST und LAST hängen von der Reihenfolge der Prozesselemente im Prozess ab. Bei Elementen auf parallelen Pfaden kann die Reihenfolge nicht eindeutig bestimmt werden. In solchen Fällen kann nur ein beliebiges Element ausgewählt werden.

Funktionen für numerische Attribute	
SUM	Summe der Attributwerte
AVG	Durchschnitt der Attributwerte
MIN	Minimum der Attributwerte
MAX	Maximum der Attributwerte
COUNT	Anzahl der Attributwerte
Funktionen für textuelle Attribute	
CONCAT	Aneinanderhängen der Attributwerte (u.U. mit Trennzeichen, z.B. ', ')
FIRST	erster Wert aus der Liste der zu transformierenden Attributwerte
LAST	letzter Wert aus der Liste der zu transformierenden Attributwerte
MAXFREQ	Wert mit der größten Häufigkeit
MINFREQ	Wert mit der geringsten Häufigkeit
RANDOM	beliebiger Wert

Tabelle 4.4: Transformationsfunktionen für Attributwerte

Eine besondere Herausforderung stellt die Transformation von Attributen dar, deren zugehörige Aktivitäten auf verschiedenen Pfaden einer XOR-/OR-Verzweigung liegen. Hier werden Aktivitäten nicht in allen Fällen ausgeführt. Bei deren Aggregation führt dies zu Schwierigkeiten in Bezug auf die Berechnung von Attributwerten, vor allem wenn die Aktivitäten noch nicht ausgeführt worden sind. Anstelle exakter Attributwerte für aggregierte Aktivitäten muss hier mit Erwartungswerten gearbeitet werden. Dies setzt voraus, dass die Ausführungswahrscheinlichkeiten für die einzelnen Zweige einer XOR-/OR-Verzweigung bekannt sind [EGP00, EP02, EPGN03]. Mittels der Kontrollflussstruktur, angereichert um die Ausführungswahrscheinlichkeiten, lassen sich die Erwartungswerte der Attribute berechnen. In der Praxis sind solche Ausführungswahrscheinlichkeiten meist nicht verfügbar. Oftmals reicht es hier für eine Visualisierung aber aus, eine fixe Formel zu definieren, welche die Attributwerte transformiert. Für die große Mehrheit der Fälle, bei denen textuelle Attribute transformiert werden sollen, genügt es, einen spezifischen Wert auszuwählen oder alle Werte zu konkatenieren.

Abbildung 4.36 zeigt ein Beispiel für die Transformation der Standardattribute *Start-* und *Endezeit* sowie *Kosten*. In Abbildung 4.38 werden Attribute mit Applikationsdatenwerten transformiert. Hierbei kommen die Transformationsfunktionen LAST und CONCAT zum Einsatz.

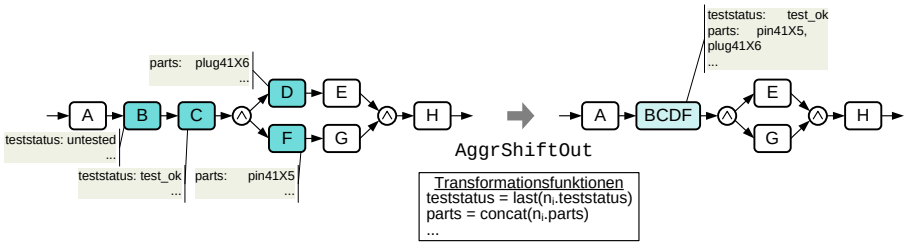


Abbildung 4.38: Beispiel für Attributtransformationen

Wir klassifizieren nun die verschiedenen Arten von Attributen, die in einer Prozessapplikation vorhanden sind. Als Illustration dient Abbildung 4.39.

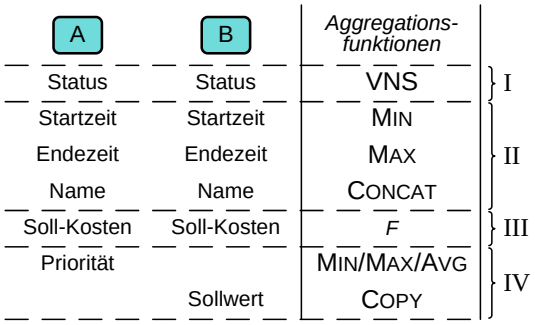


Abbildung 4.39: Aggregationsfunktionen für verschiedene Attribute

1. Klasse I repräsentiert das Attribut für den Aktivitätenzustand. Dieses wurde zunächst aus Darstellungsgründen im Zusammenhang mit der View-Bildung auf Prozessinstanzen eingeführt. Eigentlich handelt es sich hierbei jedoch um ein gewöhnliches Attribut der Aktivität, für das eine spezielle Transformationsfunktion *VNS* definiert wird.

2. Klasse II umfasst die Standardattribute, die jede Aktivität im System besitzt und deren Semantik bekannt ist. Somit können für diese Attribute auch Standard-Transformationsfunktionen angeboten werden, die nicht explizit für jede Operation deklariert werden müssen (wie dies erfolgt wird in Abschnitt 5.5 beschrieben). Ein Beispiel für ein Attribut dieser Klasse ist die Startzeit. Um den Startzeitpunkt einer aggregierten (abstrakten) Aktivität zu berechnen, wird die MIN-Funktion verwendet, sodass sich als Gesamtwert der erste und damit kleinste Startzeitpunkt aller aggregierten Aktivitäten ergibt.

3. Zu Klasse III zählen alle Attribute, die applikationsspezifisch sind, aber dennoch für alle Objekte eines bestimmten Typs (z.B. Aktivitäten) zur Verfügung stehen. Für die Transformation ist die Kenntnis der Semantik der Attribute wichtig. Die Attribute können daraufhin durch spezielle Transformationsfunktionen verrechnet werden. Beispielsweise sollen bei der Aggregation der Aktivitäten die Soll-Kosten aufsummiert werden. Wegen der angesprochenen Problematik bei XOR-/OR-Verzweigungen muss hierfür eine spezielle Transformationsfunktion (F) definiert werden, welche die Struktur des Prozesses und die Ausführungswahrscheinlichkeiten der einzelnen Zweige berücksichtigt und zu einem Erwartungswert kombiniert.

4. In Klasse IV fallen alle Attribute, die nur für einen Teil der Objekte bzw. Aktivitäten eines Prozesses relevant sind. Dieser Fall kann z.B. dann auftreten, wenn die Aktivitäten des darzustellenden Prozesses von verschiedenen Basissystemen kontrolliert bzw. ausgeführt werden. Bei der Aggregation der Aktivitäten sollen, falls vom Benutzer keine anderen Vorgaben existieren, diese Attributwerte standardmäßig für die abstrakte Aktivität übernommen werden (COPY).

Der Proviado-View-Mechanismus sieht vor, dass für Attribute entsprechende Standard-Transformationsfunktionen bereitgestellt werden. Bei Bedarf können in Einzelfällen benutzerdefinierte Transformationsfunktionen spezifiziert werden, die das beschriebene Standardverhalten überschreiben. Wie schon bei der Projektion verwenden wir auch hier eine destruktive Deklaration.

D.h. standardmäßig bleiben alle Attribute bei einer Aggregation erhalten. Einzelne Attribute können anschließend durch entsprechende Projektionsoperationen entfernt werden.

Neben der Transformation von Attributen in Zusammenhang mit der Aggregation von Aktivitäten existieren weitere Anwendungsfälle wie Abbildung 4.40 verdeutlicht. Der bislang diskutierte Fall ist in Abbildung 4.40a wieder zu erkennen. Hierbei werden die Werte eines bestimmten Attributs mittels der Transformationsfunktion auf den Attributwert der abstrakten Aktivität abgebildet. Bei den weiteren Fällen handelt es sich um eigenständige Transformationen, die unabhängig von einer Kontrollflussoperation ausgeführt werden: Abbildung 4.40b zeigt den Fall, dass ein neues Attribut aus mehreren lokalen Attributen, d.h. aus Attributen desselben Prozesselements berechnet werden soll. Die Berechnung eines neuen Attributs aus beliebigen anderen Attributen ist in Abbildung 4.40c illustriert. Im Gegensatz zum vorhergehenden Fall können auch Attribute anderer Prozesselemente einbezogen werden.

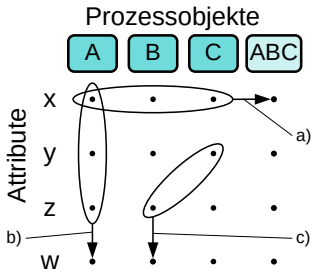


Abbildung 4.40: Beispiele für Attributoperationen

Im Proviado-View-Mechanismus stellen wir eine Elementaroperation für alle Formen der Attributtransformation zur Verfügung. Die Attributoperation `TransformAttr` bildet eine Menge von Ausgangsattributen auf ein Zielattribut ab, wobei eine bestimmte Transformationsfunktion zur Anwendung kommt.

`TransformAttr`$(\{A.x, B.y, \ldots\}, f, N.z)$ mit

- $\{A.x, B.y, \ldots\}$: Menge der Quellattribute, die transformiert werden.
- f: Transformationsfunktion zur Berechnung des neuen Attributwertes aus den Quellattributen.
- $N.z$: Zielattribut, in dem der neue Wert abgelegt wird.

Die Transformation der Attribute wird in den in Abschnitt 4.3.2.1 vorgestellten Algorithmen der Aggregationsoperationen für den Kontrollfluss innerhalb der Funktion `createAggrNode(X)` aufgerufen. Alle anderen Attributfunktionen können unabhängig von Kontrollflussoperationen jederzeit ausgeführt werden, um beispielsweise ein Attribut aus dem Prozess zu entfernen.

4.5 View-Operationen für Datenelemente und weitere Prozessaspekte

Bei der Prozessmodellierung werden neben den Aktivitäten und dem Kontrollfluss zwischen ihnen weitere Aspekte abgebildet. Besonders wichtig ist der Datenaspekt. Er stellt dar, wie Datenelemente durch den Prozess „fließen“ (vgl. Abbildung 4.41). Eine Möglichkeit der Realisierung solcher Datenflüsse, die wir auch im Folgenden anwenden, besteht in der Verknüpfung von Aktivitäten mit (globalen) Datenelementen über Schreib- bzw. Lesekanten [RD98]. Im Allgemeinen können Datenelemente sowohl einfache Datentypen (z.B. Nummer eines Änderungsantrages) als auch komplexe Datentypen (z.B. Beschreibung eines Änderungsantrages oder zugehöriger CAD-Datensatz) besitzen. In diesem Abschnitt stellen wir View-Bildungsoperationen für den Datenaspekt vor. Diese Konzepte können auf alle anderen Prozessaspekte übertragen werden.

Der Proviado-View-Mechanismus unterscheidet zwei Arten von View-Operationen: Reduktion und Aggregation (bei Attributen als Projektion und Transformation bezeichnet). Diese Operationen können analog zum Kontrollfluss auch für den Datenflussaspekt definiert werden, um Datenelemente gezielt aus dem Prozessmodell zu entfernen oder zusammenzufassen. Für einen umfassenden View-Mechanismus müssen die View-Operationen für den Datenfluss natürlich nicht nur einzeln sondern auch in Kombination mit den vorgestellten Kontrollflussoperationen anwendbar sein. Wir unterscheiden folgende Fälle:

1. Expliziter Aufruf der Datenfluss-View-Operation, z.B. zwecks Aggregation einer Menge von Datenelementen
2. Impliziter Aufruf der Datenfluss-View-Operation, z.B. aus einer Kontrollflussaggregation heraus

Die auf den folgenden Seiten dieses Kapitels vorgestellten Operationen decken zunächst den ersten Fall ab. Im nachfolgenden Kapitel gehen wir dann in Abschnitt 5.3 darauf ein, wie die Operationen mit Kontrollflussoperationen kombiniert werden können (Fall 2).

4.5.1 Erweiterung des Prozessmodells um den Datenflussaspekt

Zunächst erweitern wir unser in Abschnitt 4.2.1 eingeführtes formales Prozessmodell (vgl. Definition 4.1) um den Datenaspekt (siehe Abbildung 4.41). Wir führen dazu einen neuen Knotentyp ein, der Datenelemente repräsentiert.

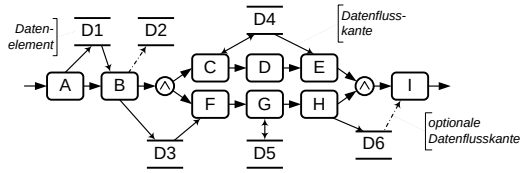


Abbildung 4.41: Beispiel für einen Prozess mit Datenfluss

Definition 4.19 (Prozessschema erweitert um Datenaspekt) Ein Prozessschema ist ein Tupel $P = (N, E, EC, NT, ET)$ mit

- $N = C \dot{\cup} D$ bezeichnet die Knotenmenge von P. Dabei entsprechen C der Menge der Kontrollflussknoten (d.h. Aktivitäten und Strukturknoten) und D der Menge der Datenflussknoten.
- $E = CE \dot{\cup} DE$ ist die Kantenmenge von P. $CE \subseteq C \times C$ bezeichnet die Menge der Kontrollflusskanten und $DE \subseteq C \times D$ die Menge der Datenflusskanten.
- $EC : E \rightarrow Conds \cup \{\text{TRUE}\}$ ordnet Kontrollflusskanten optional eine Transitionsbedingung zu.
- $NT : N \rightarrow NodeTypes$ ordnet jedem Knoten einen Knotentyp zu, wobei gilt $NodeTypes = NodeTypes_{CF} \cup \{DataElement\}$. Somit ergibt sich die Menge der Datenelemente D als $D = \{x \in N | NT(x) = DataElement\}$
  Dabei umfasst $NodeTypes_{CF} = \{Activity,\ ANDSplit,\ ANDJoin,\ ORSplit,\ ORJoin,\ XORSplit,\ XORJoin,\ LoopSplit,\ LoopJoin\}$ alle möglichen Knotentypen für Elemente aus C (vgl. Definition 4.1).
- $ET : E \rightarrow EdgeTypes$ ordnet jeder Kante einen Kantentyp zu, wobei $EdgeTypes = \{ControlFlow, DataFlow\}$

$$ET(e) = \begin{cases} ControlFlow & e \in CE \\ DataFlow & e \in DE \end{cases}$$

  Entsprechend definieren wir:

  - $CET : CE \rightarrow \{ControlFlowEdge, LoopEdge\}$ ordnet jeder Kontrollflusskante eine Typ zu.
  - $DET : DE \rightarrow \{r, w\} \rightarrow \{always, optional, never\}$ beschreibt für jede Datenkante, welcher Typ von Datenzugriff durch sie repräsentiert wird.

Datenelemente werden mit Aktivitäten mittels Datenflusskanten verbunden. Eine Kante repräsentiert einen lesenden und/oder schreibenden Zugriff der Aktivität auf das Datenelement. Des Weiteren besteht die Möglichkeit, dass ein Datenelement bei der Ausführung einer Aktivität nicht sicher, d.h. lediglich optional, gelesen bzw. geschrieben wird. In unserem formalen Prozessmodell bilden wir diesen Umstand mittels der Abbildung DET ab. Abbildung 4.42 verdeutlicht, wie die Datenkantenattribute (DET) graphisch interpretiert werden können.

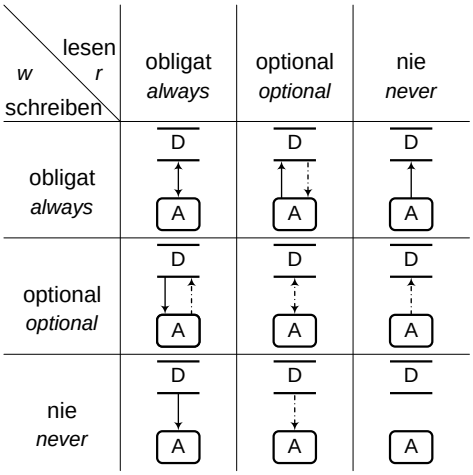


Abbildung 4.42: Graphische Interpretation der Attribute einer Datenkante

Optionale Datenkanten beschreiben einen Datenzugriff, der nicht immer erfolgt. Im Zusammenhang mit der Aggregation können diese Kanten unterschiedlich interpretiert werden, was entsprechend zu unterschiedlichen Ergebnissen führt. Der Unterschied wird aus der in Abbildung 4.43 dargestellten Situation (Aggregation mit optional lesenden Datenkanten) ersichtlich.

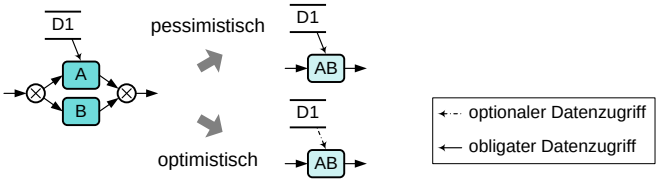


Abbildung 4.43: Unterschiedliche Semantiken von Datenflusskanten

Pessimistisch Die Versorgung der konsumierenden Aktivitäten hat höchste Priorität. Somit werden auch Lesekanten in einem Zweig einer Verzweigung zu obligaten aggregiert, da die entsprechende Aktivität unter Umständen das Datum benötigen könnte und es bei einer Nicht-Versorgung zu unerwünschten Effekten kommen kann.

Optimistisch Die Datenflusskanten geben genau das wieder was geschieht: Wird ein Datum nicht unbedingt benötigt oder nur in einem Teil der Fälle gelesen bzw. geschrieben, ist die Kante entsprechend optional.

Die pessimistische Variante ist zum Beispiel in [RD98, Rei00] zu finden. Für die Visualisierung verwenden wir die optimistische Variante, da diese für die Prozessvisualisierung intuitiver und auch ausreichend ist.

Für die Visualisierung wird im Folgenden eine Definition der Semantik von optionalen und obligaten Datenkanten vorgestellt. Stimmt diese Definition der Semantik von Datenkanten nicht mit den Anforderungen einer speziellen Anwendung überein, kann eine andere Definition

verwendet werden. Unter Umständen ist dann aber eine Anpassung der im weiteren Verlauf dieses Abschnitts vorgestellten Funktionen nötig.

Semantik	Kantenattribute	Erklärung
obligat schreibend	*w:always*	Datenelement wird sicher geschrieben
obligat lesend	*r:always*	Datenelement wird sicher gelesen
optional schreibend	*w:optional*	Datenelement wird nicht sicher geschrieben
optional lesend	*r:optional*	Datenelement wird nicht sicher gelesen
nicht schreibend[1]	*w:never*	Datenelement wird nicht geschrieben
nicht lesend[1]	*r:never*	Datenelement wird nicht gelesen

[1]Eine Kombination von nicht schreibend und nicht lesend ergibt keinen Sinn

Tabelle 4.5: Definition der Semantik von Datenkanten

Datenelemente in Prozessmodellen repräsentieren nicht nur einfache, sondern auch komplexe Datentypen (z.B. Formulare, CAD-Datensätze). Je nach Art der Prozesssystems (Prozessmodellierungswerkzeug, Workflow-Management-System) werden diese Daten entweder im System, zusammen mit den Prozessinformationen, oder in externen Anwendungssystemen verwaltet. Wie alle anderen Objekte besitzen auch Datenelemente wieder Attribute. Zum einen gibt es interne Attribute, die für die Prozessmodellierung relevant sind (z.B. Name des Datenelements). Zum anderen gibt es wieder die Nutzdaten, d.h. Applikationsdaten (manchmal in Form eines Verweises auf die entsprechenden Daten im Quellsystem). Trotz dieses Unterschieds bilden wir beide Arten von Datentypen durch Attribute ab. Für die View-Bildung wird dies relevant, sobald Datenelemente aggregiert werden sollen (siehe Abschnitt 4.5.3).

4.5.2 Reduktion von Datenelementen

Mittels Reduktion können Datenelemente entfernt werden. Adjazente Datenflusskanten müssen dann ebenfalls gelöscht werden. In Abbildung 4.44 wird das Datenelement $D1$ reduziert, wodurch gleichzeitig die adjazenten Datenflusskanten zwischen $D1$ und A bzw. B gelöscht werden.

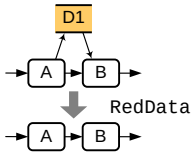


Abbildung 4.44: Reduktion von Datenelementen mittels `RedData`

Die Elementaroperation `RedData` reduziert ein einzelnes Datenelement mitsamt seinen adjazenten Datenkanten wie in Abbildung 4.44 dargestellt. In Abschnitt 5.2.4 werden wir höherwertige Operationen definieren, die auf Basis von `RedData` ganze Mengen von Datenelementen reduzieren.

4.5.3 Aggregation von Datenelementen

Wir stellen nun vor, wie Datenelemente mittels Elementaroperationen aggregiert werden können. Dazu behandeln wir in Abschnitt 4.5.3.1 zunächst den Fall, dass eine Menge von Datenelementen zu einem abstrakten Datenelement aggregiert werden soll (Abbildung 4.45 zeigt zwei solche Aggregationen). Anschließend beschreiben wir in Abschnitt 4.5.3.2, wie bei der Aggregation von Aktivitäten die adjazenten Datenflusskanten adaptiert werden müssen. Auf die Aggregation von Aktivitäten in Kombination mit Datenfluss Elementen gehen wir in Kapitel 5, Abschnitt 5.3 ein (aspekteübergreifende Operationen).

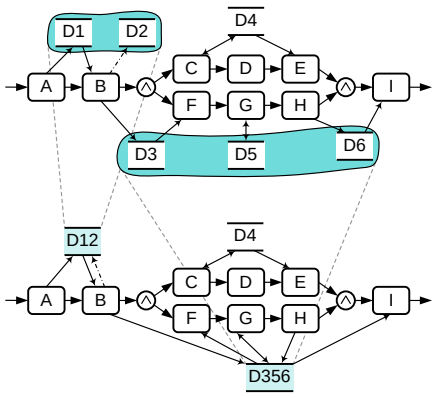


Abbildung 4.45: Aggregation von Datenelementen `AggrData`

4.5.3.1 Aggregation von Datenelementen ohne Änderungen am Kontrollfluss

`AggrData` aggregiert eine Menge von Datenelementen. Bei der Aggregation müssen die adjazenten Datenflusskanten geeignet angepasst werden, wie Abbildung 4.46 beispielhaft darlegt: Aus der obligaten Schreibkante $(A, D1)$ und der optionalen Schreibkante $(A, D2)$ wird bei der Aggregation eine obligate Schreibkante $(A, D12)$. Analog wird aus einer obligaten und einer optionalen Lesekante eine obligate Lesekante (vgl. Kante $(B, D12)$). Die Lese- und Schreibkanten $(C, D1)$ und $(C, D2)$ werden zu einer Lese-Schreib-Kante $(C, D12)$.

Abbildung 4.46: Aggregation von Datenelementen `AggrData`

Die Herausforderung besteht nun darin, eine Berechnungsfunktion für die Datenkantenattribute aufzustellen. Da letztere unabhängig von der Kontrollflussstruktur sind, können wir die Attribute für jede adjazente Aktivität einzeln berechnen. Ein entsprechendes Beispiel zeigt Abbildung 4.47. Ferner können die Attribute für Schreib- und Lesezugriffe getrennt voneinander betrachtet werden. Tabelle 4.6 illustriert, wie sich das Datenflusskantenattribut für lesenden Zugriff auf das

abstrakte Datenelement ($DET(A, D_{1...n})(r)$) aus den Datenflusskantenattributen der aggregierten Datenelemente ergibt. Identische Aussagen gelten auch für die Datenflusskantenattribute bei schreibendem Zugriff.

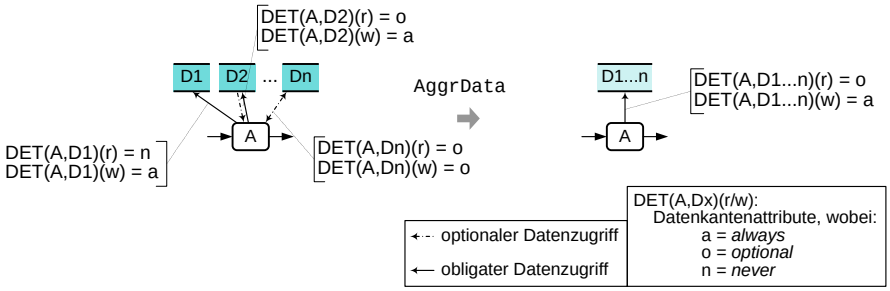


Abbildung 4.47: Beispiel für die Berechnung der Datenkantenattribute bei `AggrData`

Aggregation der Datenelemente $D_1, \ldots, D_n$ mit $\exists i \in \{1, \ldots, n\}$ mit $DET(A, D_i)(r)$				$DET(A, D_{1...n})(r)$
always	optional	never		
		+	→	never
	+		→	optional
	+	+	→	optional
+			→	always
+		+	→	always
+	+		→	always
+	+	+	→	always

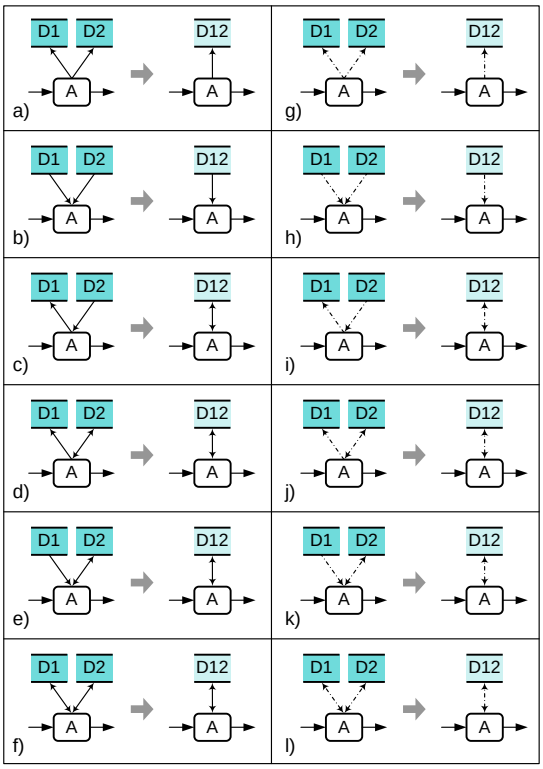
Tabelle 4.6: Aggregation von Datenelementen, die von einer Aktivität A gelesen werden

Aus Tabelle 4.6 kann die allgemeine Definition der Datenkantenattribute DET (vgl. Definition 4.19) für eine Menge zu aggregierender Datenelemente $\{D_1, \ldots, D_n\}$, die in Verbindung mit einer Aktivität A stehen und zu einem abstrakten Datenelement D' aggregiert werden (x sei die Datenzugriffsart r/w), wie folgt abgeleitet werden:

$$DET(A, D')(x) = \begin{cases} always & \exists i \in \{1, \ldots, n\} : DET(A, D_i)(x) = always \\ optional & \nexists i \in \{1, \ldots, n\} : DET(A, D_i)(x) = always \quad \wedge \\ & \qquad \exists i \in \{1, \ldots, n\} : DET(A, D_i)(x) = optional \\ never & \text{sonst} \end{cases}$$

Abbildung 4.48 illustriert verschiedene Beispiele einer Datenelementaggregation in einem Prozess mit einer Aktivität und zwei dazu adjazenten Datenelementen. Die Berechnung der Datenflusskantenattribute kann aus der nebenstehenden Tabelle entnommen werden.

Einhergehend mit der Aggregation von Datenelementen müssen deren transformierte Attribute berechnet werden. Für die Standardattribute (Klasse II in Abschnitt 4.4.3) können entsprechend Standardfunktionen definiert werden. Beispielsweise kann eine Transformationsfunktion angegeben werden, mit der der Name aller Datenelemente transformiert wird. Um auch komplexe Datenelemente, etwa die Ergebnisse der Stellungnahmen im Änderungsmanagement-Prozess,



$DET(A, D1)$ r	$DET(A, D1)$ w	$DET(A, D2)$ r	$DET(A, D2)$ w		$DET(A, D12)$ r	$DET(A, D12)$ w	Abb.
a	a	a	a	→	a	a	(f)
a	a	a	o	→	a	a	
a	a	a	n	→	a	a	
a	a	o	a	→	a	a	
⋮	⋮	⋮	⋮		⋮	⋮	
a	n	a	a	→	a	a	(e)
a	n	a	o	→	a	o	
a	n	a	n	→	a	n	(b)
⋮	⋮	⋮	⋮		⋮	⋮	
o	o	o	a	→	o	a	
o	o	o	o	→	o	o	(l)
⋮	⋮	⋮	⋮		⋮	⋮	
o	n	o	a	→	o	a	
o	n	o	o	→	o	o	(k)
o	n	o	n	→	o	n	(h)
o	n	n	a	→	o	a	
o	n	n	o	→	o	o	
n	a	a	a	→	a	a	(d)
n	a	a	o	→	a	a	
n	a	a	n	→	a	a	(c)
n	a	o	a	→	o	a	
n	a	o	o	→	o	a	
n	a	o	n	→	o	a	
n	a	n	a	→	n	a	(a)
⋮	⋮	⋮	⋮		⋮	⋮	
n	o	o	o	→	o	o	(j)
n	o	o	n	→	o	o	(i)
n	o	n	a	→	n	a	
n	o	n	o	→	n	o	(g)

a = always, o = optional, n = never

Abbildung 4.48: Beispiele für Aggregation von Datenkanten bei `AggrData`

aggregieren zu können, müssen spezielle Transformationsfunktionen bereitgestellt werden. In den meisten Fällen genügen standardmäßig bereitgestellte Funktionen nicht, da sie von der (semantischen) Bedeutung der Datenelemente abhängig und daher anwendungsspezifisch sind.

4.5.3.2 Aggregation von Kontrollfluss und Anpassung der Datenkanten

Werden Aktivitäten, die auf Datenelemente zugreifen, zu einer abstrakten Aktivität aggregiert, müssen die inzidenten Datenflusskanten der aggregierten Aktivitäten auf die abstrakte Aktivität übertragen werden (vgl. Abbildung 4.49). Für diesen Anwendungsfall bietet Proviado die Elementaroperation `AdaptDE`. Sie berechnet für ein Datenelement die Datenflusskantenattribute der inzidenten Datenkante, die aus der Aggregation adjazenter Aktivitäten entstehen.

Die Berechnung der Attribute der neuen Kante führt in vielen Fällen zum gleichen Ergebnis wie die Elementaroperation `AggrData`. Zum Beispiel wird aus einer Kante die obligat liest und einer Kante die obligat schreibt eine Kante die obligat liest und schreibt (vgl. Abbildung 4.49a). In einigen Fällen jedoch ist die Bestimmung der Kantenattribute wesentlich komplizierter, da sie von der Struktur der zum Datenelement adjazenten Aktivitäten im Prozess abhängig sind.

Dieser Sachverhalt ist in Abbildung 4.49b und c illustriert. Diese Abbildungen zeigen je ein Prozessmodell, das sich nur in der Art der Verzeigungsknoten unterscheidet (AND und XOR). Bei einer Aggregation der parallelen Aktivitäten ergibt sich jedoch in Beispiel b eine optionale und in Beispiel c eine obligate Datenflusskante. Dies rührt daher, dass in Beispiel b abhängig von der Verzweigungsentscheidung am XOR-Split-Knoten nicht immer ausgeführt und damit auch das Datenelement D1 nicht sicher geschrieben wird. Greifen beide Aktivitäten obligat auf D1 zu (vgl. Abbildung 4.49d), ergibt sich für die abstrakte Aktivität AB ein obligater Schreibzugriff. Die Vorgehensweise ist aber für Lese- und Schreibzugriffe identisch, weshalb wir hier beispielhaft Schreibzugriffe betrachten.

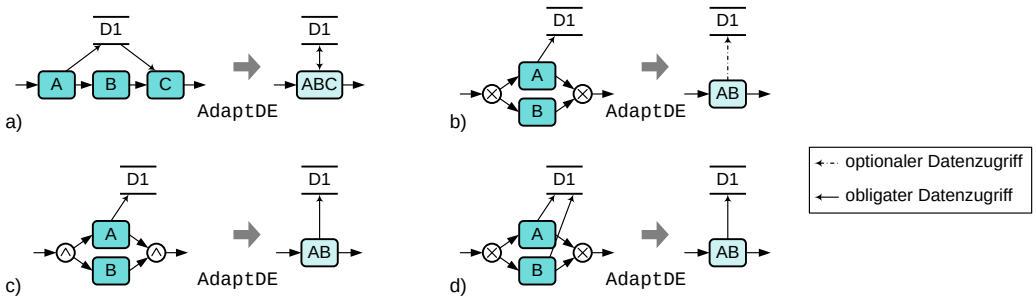


Abbildung 4.49: Neuberechnung der Datenkanten durch `AdaptDE`

Ausgangspunkt für die erforderliche Analyse des Kontrollflusses ist die Menge von Aktivitäten, die aggregiert werden sollen. Wir nehmen nun an, dass einige der Aktivitäten schreibend auf ein Datenelement zugreifen. Dies kann obligat oder optional erfolgen. Bei der Aggregation ist folglich zu entscheiden, ob der Schreibzugriff der abstrakten Aktivität auf das Datenelement obligat oder optional ist. Ohne weitere Analyse kann man immer von einem optionalen Zugriff ausgehen, solange mindestens eine der aggregierten Aktivitäten schreibt. Die Datenflusskante ist genau dann obligat, wenn für alle möglichen Ausführungen der aggregierten Aktivitätenmenge ein obligater Schreibzugriff erfolgt. Wir können nun verschiedene Varianten der Analyse unterscheiden, die hinsichtlich des erforderlichen Berechnungsaufwands divergieren:

1. Ein pragmatischer Ansatz würde von vornherein auf die Unterscheidung zwischen optionalem und obligatem Datenzugriff verzichten. Dadurch wäre bei einer Datenkante zwar nicht mehr klar, ob zur Laufzeit letztlich ein Datenzugriff stattfindet oder nicht, jedoch würden die Algorithmen für die Aggregation von Aktivitäten und adjazenten Datenelementen wesentlich vereinfacht. Ein Aspekt, der für diesen pragmatischen Ansatz spricht, ist die ohnehin fehlende Unterscheidung zwischen den Zugriffsarten in existierenden Prozessmanagementsystemen. Bei der Visualisierung von Prozessen, die auf diesen Systemen basieren, kann diese Unterscheidung zwischen den Zugriffsarten nur mit erheblichem zusätzlichem Modellierungsaufwand eingebracht werden. Im Allgemeinen ist daher der Verzicht auf eine Unterscheidung zwischen obligatem und optionalem Datenzugriff durchaus akzeptabel und stellt keinen übermäßigen Informationsverlust dar.

2. Für alle abhängigkeitserhaltenden Aggregationsoperationen (`AggrSESE`, `AggrShiftLoop`, `AggrComplBranches`) kann einfach entschieden werden, ob ein Datenelement für alle möglichen Ausführungen der adjazenten, zu aggregierenden Aktivitäten stets geschrieben wird. Komplexer gestaltet sich diese Analyse für die anderen Operationen. Insbesondere bei

`AggrAddBranch` müssen bei der Analyse die Aufsetzpunkte und Verzweigungsbedingungen berücksichtigt werden.

3. Alternativ kann man in den für die Analyse aufwendigen Fällen stets nur optionale Datenzugriffe als Ergebnis liefern. Dadurch geht zwar Information verloren, dies ist aber für viele Anwendungen akzeptabel.

In den Algorithmen der Aggregationsoperationen findet die Berechnung der Datenflusskante (`AdaptDE`) beim Anlegen der neuen, abstrakten Aktivität (`createAggrNode(`X`)`) statt.

4.5.4 Ausweitung des Konzepts auf weitere Prozessaspekte

Wir haben in den beiden vorangegangenen Abschnitten View-Operationen für den Datenflussaspekt vorgestellt. Neben dem Kontroll- und Datenflussaspekt sind für die Prozessvisualisierung noch weitere Aspekte relevant [JB96]: Der *Organisationsaspekt* ordnet Personen bzw. organisatorische Rollen den einzelnen Aktivitäten zu. Der *Systemaspekt* beschreibt die Zuordnung von IT-Systemen zu Aktivitäten. Für diese anderen Prozessaspekte müssen entsprechende View-Operationen definiert werden. Dies gestaltet sich meist einfacher als im Fall des Zusammenspiels von Kontroll- und Datenflussaspekt, da keine Unterscheidungen zwischen lesendem und schreibendem bzw. obligatem und optionalem Zugriff gemacht werden müssen.

Die Reduktion wird wie bei Datenelementen realisiert. Bei der Aggregation muss zunächst die Semantik einer Aggregation auf Bearbeitern bzw. Systemen geklärt werden. Festzulegen ist, was beispielsweise das Ergebnis einer Aggregation von mehreren Bearbeitern sein soll: deren gemeinsamer Vorgesetzter oder die Abteilung. Hier kann keine allgemeingültige Lösung angeboten werden, sondern es sind für verschiedene Anwendungen unterschiedliche Lösungen erforderlich. Ist diese Frage der Aggregationssemantik geklärt, können die Operationen ähnlich wie bei Datenelementen definiert werden. Auf Details verzichten wir an dieser Stelle.

4.6 Überblick über die View-Operationen

In diesem Kapitel wurden eine Reihe von Elementaroperationen für die Bildung von Prozess-Views bezogen auf Kontrollfluss, Datenfluss und Attribute vorgestellt. Tabelle 4.7 fasst die Kontrollflussoperationen mit ihren Eigenschaften zusammen.

Die Eigenschaften sind zustandsunabhängig definiert, d.h. ob eine View eine bestimmte Eigenschaft besitzt oder nicht, hängt nicht vom aktuellen Ausführungszustand ab. Selbstverständlich könnte man ähnliche Eigenschaften auch zustandsabhängig festlegen. Wie wir in Kapitel 5 erklären werden, verwenden wir die Eigenschaften später, um höherwertige Operationen zu definieren, die aufgrund der Struktur des Kontrollflusses und der vom Benutzer vorgegebenen Anforderungen (in Form von Eigenschaften) automatisch entscheiden, welche elementare View-Operationen anzuwenden sind. Basiert diese Auswahl auf zustandsabhängigen Eigenschaften, würde dies bedeuten, dass potentiell mit jeder Zustandsänderung des Basisprozesses eine andere Elementaroperation zu wählen ist. Das bedeutet für den Benutzer einer Visualisierungskomponente, dass je nach aktuellem Ausführungszustand eine andere View gebildet wird und somit ein anderes Modell dargestellt wird. Dies wäre im Sinne einer Erhaltung der „Mental Map“

Operation \ Eigenschaft	streng ordnungs-erhaltend	ordnungs-erhaltend	streng zustands-konsistent	zustands-konsistent	abhängigkeits-erhaltend	abhängigkeits-löschend	abhängigkeits-erzeugend	Iterations-einbeziehend	Iterations-ausschließend
`RedActivity`	+	+	-	+	-	+	-	-	-
`AggrSESE`	+	+	+	+	+	-	-	-	-
`AggrComplBranches`	+	+	+	+	+	-	-	-	-
`AggrShiftOut`	+	+	+/-[1]	+/-[1]	-	-	+	-	-/+[2]
`AggrShiftLoop[In]`[3]	+	+	+	+	+	-	-	+	-
`AggrShiftLoop[Out]`[3]	+	+	+	+	+	-	-	-	+
`AggrAddBranch`	-	+	-	+	-	+	-	-	+

[1] je nach Verzweigungstyp kann `AggrShiftOut` zustandskonsistent oder nicht sein (siehe Abschnitt 4.3.3.2)
[2] in besonderen Fällen kann `AggrShiftOut` Iterations-ausschließend sein (siehe Abschnitt 4.3.2)
[3] Werte des Parameters *`LoopShift`* (siehe Abschnitt 4.3.2.1 auf Seite 66)

Tabelle 4.7: Übersicht der Eigenschaften der elementaren View-Bildungsoperationen

nicht akzeptabel. Deshalb verwendet der Proviado-View-Mechanismus zustandsunabhängige Eigenschaften.

Die vorgestellte Menge von elementaren View-Operationen ist in dem Sinne vollständig, als dass für jede beliebige Menge von selektierten Objekten in einem Prozess stets eine View gebildet werden kann. Garantiert wird dies in erster Linie durch die Operation `AggrAddBranch`, die beliebig verteilte Aktivitäten zu einer abstrakten Aktivität auf einem parallelen Pfad aggregieren kann. Wie erwähnt, kann der View-Mechanismus jederzeit durch weitere Elementaroperationen ergänzt werden, um spezielle Anforderungen wie beispielsweise besondere Konstellationen von zu aggregierenden Aktivitäten, abdecken zu können.

Kommutativität von View-Operationen

Bei der Betrachtung der Hintereinanderausführung mehrerer elementarer View-Bildungsoperationen ergibt sich die interessante Frage, ob die Anwendung der Operationen in einer anderen Reihenfolge möglich ist und zu dem gleichen Ergebnis führt.

Definition 4.20 (Kommutativität von View-Operationen) Seien Op_1 und Op_2 zwei View-Bildungsoperationen. Op_1 und Op_2 sind *kommutativ* $\iff$

$$V_{12}(P) = Op_2 \circ Op_1(P) \equiv Op_1 \circ Op_2(P) = V_{21}(P)$$

Wann zwei View-Operationen nun kommutativ sind, beschreibt der folgende Satz.

Satz 4.2 (Bedingung für die Kommutativität) Sei $V_{12}(P) = Op_2 \circ Op_1(P)$ eine View bestehend aus 2 Operationen. Op_1 und Op_2 sind *kommutativ*, wenn sich die Bereiche des Kontrollflusses, auf die Op_1 und Op_2 angewendet werden, nicht überlappen. D.h. dann gilt:

$$V_{12}(P) = Op_2 \circ Op_1(P) \equiv Op_1 \circ Op_2(P) = V_{21}(P)$$

Satz 4.2 beschreibt ein hinreichendes Kriterium für die Kommutativität von View-Operationen.

Beispiel 4-8: Beispiel einer View-Bildung mit Elementaroperationen
In dem in Abbildung 4.2 auf Seite 40 dargestellten Prozess sollen gemäß Beispiel 4-3 und Beispiel 4-6 alle Aktivitäten, die durch Partnerunternehmen durchgeführt werden, aggregiert, und alle automatisch (ohne Beteiligung von Menschen) durchgeführten Aktivitäten entfernt werden. Mit den Operationen, die in diesem Kapitel vorgestellt wurden, sieht eine entsprechende View-Definition wie folgt aus:

```
AggrAddBranch({F1,H1,J1,M1,N1,O1})
AggrAddBranch({F2,H2,J2,M2,N2,O2})
AggrAddBranch({F3,H3,I1,M3,N3,O3})
RedActivity(C)
RedActivity(D)
RedActivity(L1)
RedActivity(L2)
RedActivity(L3)
RedActivity(Q)
```

```
RedData(d2)
RedData(d3)
RedData(d5)
RedData(d6)
RedData(d8)
RedData(d9)
RedData(d12)
```

Das Ergebnis der View-Bildung wurde bereits in Abbildung 4.3 auf Seite 42 dargestellt. In dem Beispiel wurden nur die Kontroll- und Datenflussoperationen aufgelistet. Dennoch ergibt sich für diesen, verglichen mit realen Anwendungsszenarien, kleinen Prozess eine beachtliche Anzahl von Einzeloperationen, welche der Visualisierungs-Designer alle (korrekt) definieren müsste. Hinzu kommt, dass die Definition eine genaue Kenntnis der Semantik der einzelnen Operationen voraussetzt. Für eine einfache Definition und damit bessere Benutzbarkeit werden wir in Kapitel 5 höherwertige Operationen definieren, welche die Komplexität der Einzeloperationen verbergen.

4.7 Diskussion

In diesem Kapitel haben wir die Grundlagen des Proviado-View-Mechanismus vorgestellt, mit dessen Hilfe komplexe Prozessmodelle kompaktifiziert werden können, so dass sie nur noch diejenigen Aspekte enthalten, die für den jeweiligen Benutzer relevant sind. Der Begriff *Views* ist aus dem Bereich relationaler Datenbanken entliehen [EN02]. Hier dienen Views dazu, nicht benötigte Spalten und Tupel einer Relation (d.h. Tabelle) auszublenden, mehrere Relationen zu kombinieren (Join) bzw. durch Aggregation die Attributwerte mehrerer Tupel zu kombinieren, so dass letztlich eine übersichtliche virtuelle Relation entsteht. Bei verteilten Datenbankapplikationen spielt zudem die Optimierung und somit Reduktion der zu übertragenden Datenmenge eine wichtige Rolle, was bei Prozessmodellen eher irrelevant ist [Dad96, ÖV99].

Zum Thema Prozess-Views existiert eine überschaubare Menge von Arbeiten mit verschiedenen Zielsetzungen. Einen grobe Kategorisierung der Anwendungsbereiche und der dabei zum Einsatz kommenden Techniken zeigt Abbildung 4.50. Im Folgenden werden die drei Klassifikatoren Techniken, Basismodelle und Anwendungen vorgestellt und die wichtigsten Repräsentanten der existierenden Arbeiten anhand ihrer Anwendung diskutiert.

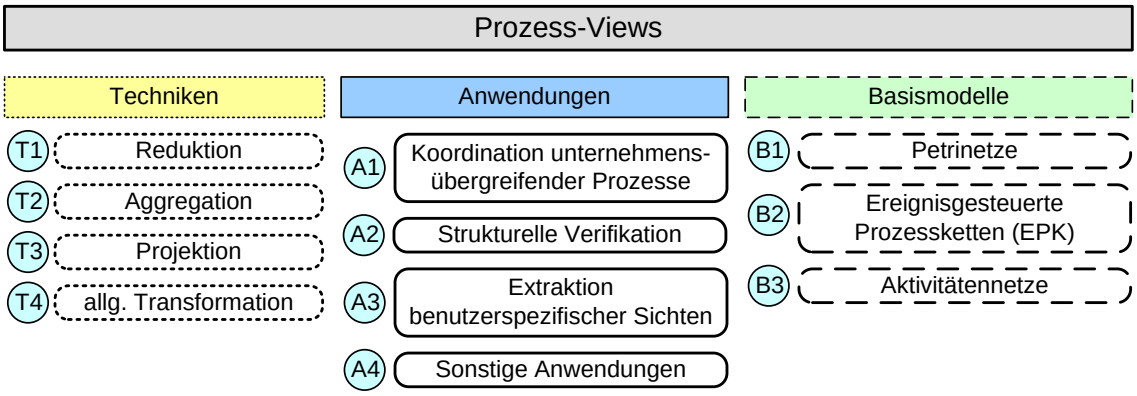


Abbildung 4.50: Überblick über die Kategorien der verwandten Arbeiten

Techniken

Zur Erzeugung von Views werden im Bereich des Prozessmanagements vier verschiedene Techniken eingesetzt. Die Prinzipien der Reduktion (T1) und der Aggregation (T2) wurden in diesem Kapitel ausführlich vorgestellt. Vor allem für die Reduktion existieren verschiedene Ansätze, die alle ähnliche Operationen definieren, aber unterschiedlichen Anwendungsfokus haben. Daneben existieren andere Ansätze, die eine Aggregation definieren, dabei jedoch nie über das Zusammenfassen zusammenhängender Bereiche hinausgehen. Eine Aggregation nicht zusammenhängender Bereiche, wie in Proviado mittels der Operation `AggrAddBranch` möglich, ist in der Literatur nicht zu finden.

Ein weiterer Ansatz zur View-Bildung ist die Projektion (T3). Hierbei werden Elemente auf Ebene des Prozessmetamodells entfernt (z.B. Datenelemente und Datenflusskanten). Durch Projektion eines Prozessmodells auf dieses reduzierte Metamodell erhält man eine Sicht, bei der gewisse Prozessaspekte (z.B. Datenfluss) ausgeblendet sind [BDK07, SL06]. Andere Arbeiten definieren für ihre Zwecke spezielle Transformationen auf Prozessgraphen, die sich keiner der drei vorangegangenen Kategorien zuordnen lassen und die hier unter dem Begriff allgemeine Transformationen (T4) zusammengefasst werden.

Basismodelle

Als zweites Unterscheidungskriterium für View-Mechanismen in Prozessmanagementsystemen kann man das zugrunde gelegte Basismodell heranziehen. Viele Ansätze basieren auf Petrinetzen [Aal98], die wegen ihrer formalen bzw. mathematischen Eigenschaften oftmals für theoretische Betrachtungen herangezogen werden (B1). In [PS05] wird, in Analogie zur Relationenalgebra, eine Workflow-Algebra basierend auf Petrinetzen definiert. Operationen (Selektion, Differenz, Projektion, Join und Union) werden formal definiert und erlauben es, Petrinetze strukturell zu verändern. Das Ergebnis wird als View bezeichnet. Die Betrachtung beschränkt sich jedoch auf statische Petrinetze ohne Token, d.h. alle dynamischen Aspekte werden explizit ausgeschlossen.

Ein Formalismus, der strukturelle Ähnlichkeiten mit Petri-Netzen aufweist und der in der Prozessmodellierung häufig Anwendung findet, sind ereignisgesteuerte Prozessketten (EPK [KNS92, Sch00]) (B2). Auch hierfür werden View-Mechanismen vorgeschlagen, wobei jedoch die

Besonderheiten der bipartiten Graphen (abwechselnd Ereignisse und Funktionen) berücksichtigt werden müssen.

Aktivitätennetze kommen in diversen Variationen (mit und ohne andere Prozessaspekte) zum Einsatz (B3). Viele Ansätze beschränken das Basismodell auf blockstrukturierte Prozessgraphen [KHB00, WW96b]. Die Vereinfachung des Modells resultiert in einfacheren Operationen, die weniger Fälle zu berücksichtigen haben. Durch die einfache Struktur sind zudem sehr effiziente Algorithmen möglich. Aus den mannigfaltigen Kombinationen von Kontrollflusskonstrukten bei nicht-blockstrukturierten Prozessen resultiert zusätzliche Komplexität. In [Klo04] haben wir einen View-Mechanismus für blockstrukturierte Graphen entwickelt, der jedoch wegen des eingeschränkten Basismodells für die geplante Anwendung zur Visualisierung allgemeiner Prozesse nur eingeschränkt anwendbar ist. Durch Restriktionen der verfügbaren Operationen (`AggrAddBranch`, `AggrShiftOut`, `Simplify`) lässt sich der Proviado-View-Mechanismus allerdings problemlos derart umgestalten, dass er die Eigenschaften blockstrukturierter Modelle beibehält. Die dazu erforderlichen Änderungen sind ohne großen Aufwand realisierbar.

Anwendungen

Prozess-Views werden in der Literatur für verschiedene Zwecke herangezogen. Eine der häufigsten Anwendungen für Views ist die Koordination unternehmensübergreifender Prozesse (A1). Es wird davon ausgegangen, dass ein Unternehmen die Details seiner Prozesse für Kooperationspartner nicht sichtbar machen will (diese werden daher als privat bezeichnet). Stattdessen werden auf Basis der privaten Prozesse so genannte öffentliche (public) Sichten erstellt, welche die für die Interaktion erforderlichen Details beinhaltet und private Aspekte ausblendet. In [GLK+06a, GLK+06b] werden mögliche Reihenfolgen für die Definition der privaten und öffentlichen Modelle diskutiert. Existierende Ansätze gehen meist davon aus, dass zuerst die privaten Prozesse existieren, die dann über die öffentlichen Versionen choreographiert werden. In [AW01, Aal02] wird der umgekehrte Weg eingeschlagen: d.h. ausgehend von den Interaktionsprozessen leitet jeder Partner seine privaten Prozesse ab. Dies erfolgt mit Hilfe so genannter „Inheritance Preserving Transformation Rules“. Sie sollen sicherstellen, dass die Konsistenz der Modelle bei jedem Erweiterungsschritt bewahrt bleibt. Insgesamt entspricht dieses Vorgehen einer Umkehrung von Aggregation und Reduktion, da die Aktivitäten schrittweise verfeinert bzw. neue Aktivitäten eingefügt werden.

In AHEAD [HW06a, HW06b, HW06c] werden Prozesssichten verwendet, um kooperative Software-Entwicklungsprozesse in der chemischen Industrie zu koordinieren. Die Sichten beschreiben einen öffentlichen Teil eines Prozesses, der von einem Partnerunternehmen abonniert werden kann. Die View-Bildung erfolgt bei diesem Ansatz durch explizite Spezifikation der Knoten und Kanten, die in der View erscheinen sollen.

[CDT06] definiert, basierend auf Petrinetzen, Vereinfachungsregeln, um den öffentlichen Prozess aus dem privaten Prozess abzuleiten. Der öffentliche Prozess ist dabei quasi der Subgraph der Aktivitäten, die für eine Interaktion mit dem Partner benötigt werden. Die verwendeten Vereinfachungsregeln entsprechen in Proviado einer Reduktion mit anschließender Anwendung der Vereinfachungsoperationen.

[ZLY05] erklärt, wie die für den Partner „wahrnehmbaren“ Teile von Prozessen aus dem Originalprozess erzeugt werden können. Dazu werden zunächst alle Aktivitäten, die mit dem Partner interagieren und daher nach außen sichtbar sind, identifiziert und anschließend die umliegenden

zusammenhängenden Aktivitätenbereiche (d.h. SESEs) aggregiert. Diese wahrnehmbaren Sichten des Partnerprozesses werden mit dem interagierenden, internen Prozess zu dem so genannten „relativen Prozess“ kombiniert. Eine Reduktion wird von diesem Ansatz nicht unterstützt.

Die umfangreichsten Arbeiten zu Prozess-Views bei unternehmensübergreifenden Prozessen stammen von Dickson K.W. Chiu et al. [CKL01, KCK01, CCK02a, CCK+02b, CKLK02, CCKL03, CCL03, CCT+04, CSHL04, CSH+05, SCL05, CHCK06, LSH+06, CHCK07]. Eine View ist hier als strukturell korrekte Teilmenge des Ausgangsprozesses definiert [CKLK02]. Darauf aufbauend wurden Ansätze publiziert, die Views zum Beispiel im Kontext von Web-Services untersuchen [CCK+02b, CKLK02, CCT+04]. In [CCKL03, CCL03] wird gezeigt, wie sich Views zur Adaption von Abläufen auf Geräten mit eingeschränkten Interaktionsfunktionen (z.B. Mobiltelefone) einsetzen lassen. Eine Realisierung dieses View-Ansatzes, basierend auf dem Deputy-Object Modell [KP96], wird in [SLLP04, SLLP05] vorgestellt. Wegen der eingeschränkten praktischen Einsetzbarkeit (das Deputy-Object Modell setzt SmallTalk voraus), wird in [SYL+06] ein weiterer Implementierungsansatz präsentiert. Dieser fokussiert auf die Verwaltung der Views unter der Voraussetzung, dass die Views vorhanden sind, d.h. modelliert wurden. Er befasst sich nicht mit der Frage, wie man aus Prozessen automatisch Views ableiten bzw. wie man Views basierend auf Prozessen definieren kann.

Es existieren noch einige weitere Arbeiten, die im Kontext unternehmensübergreifender Prozesse eine Art von Prozess-Views verwenden [Som03, SO04, LGB05, GLK+06a, GLK+06b]. Allen ist jedoch gemein, dass die View auf dem privaten Prozess von Hand „unter Beachtung gewisser Regeln“ nachmodelliert werden muss. In der Praxis stößt dieser Ansatz schnell an seine Grenzen, wenn für mehrere Partner jeweils unterschiedliche Views auf die Prozesse bereitgestellt und diese Modelle bei jeder Änderung von Hand angepasst werden müssen.

Eine weitere Anwendung von Prozess-Views ist die Verifikation von Prozessmodellen bzw. ihrer strukturellen Eigenschaften (A2). Die bekannten Ansätze verwenden Reduktionsregeln, um das Prozessmodell zu vereinfachen [SO99, SO00, LZLC02, DAV05]. Diese Regeln sind strukturerhaltend, d.h. eventuell vorhandene Modellierungsfehler bleiben erhalten. Kann der Prozess auf einen einzigen Knoten reduziert werden, ist dieser korrekt. Einige der Regeln entsprechen den in unserem Ansatz verwendeten Vereinfachungsoperation, etwa dem Zusammenfassen adjazenter Verzweigungsknoten oder dem Löschen leerer Kanten in Verzweigungen. [LZLC02] präsentiert eine verbesserte Version der in [SO99, SO00] vorgestellten Algorithmen, in der einige Fälle ergänzt wurden. [DAV05] verwendet als Basismodell ereignisgesteuerte Prozessketten. Lassen sich diese mittels der Reduktionsregeln nicht zu einem trivialen Prozess vereinfachen, wird das bis dahin vereinfachte Prozessmodell in ein Petrinetz transformiert. Auf der Grundlage des Petrinetzes wird anschließend verifiziert, ob es sich um eine korrektes, ein eventuell korrektes oder ein inkorrektes Prozessmodell handelt. Eine endgültige Entscheidung über die Korrektheit liefert das vorgestellte Verfahren allerdings nicht in allen Fällen. Basieren auf Petrinetztheorien werden in [WVA+06b, WVA+06a] Reduktionsregeln für YAWL (Yet Another Workflow Language [AH05]) bzw. Reset-Workflow Nets vorgestellt. Die Regeln sind so konzipiert, dass wichtige Korrektheitskriterien („soundness“, „liveness“, „boundedness“, „saveness“) der Basismodelle erhalten bleiben. Das Ziel all dieser Ansätze ist, die Verifikation eines Prozessmodells effizienter zu realisieren. Dazu wird versucht, die Größe der zu analysierenden Modelle, soweit möglich, zu reduzieren, ohne evtl. enthaltene Inkonsistenzen zu entfernen.

Die dritte Kategorie von Anwendungen für Prozess-Views ist die Generierung von benutzerspezifischen Sichten auf Prozesse (A3). Eines der ersten Projekte, das sich mit dem Thema Views auf Prozessen auseinandergesetzt hat, ist OPSIS [AC95, AC96, ACF96]. Der dort entwickelte View-Ansatz basiert auf Petrinetzen und definiert eine View als Projektion gemäß eines bestimmten charakteristischen Merkmals. Damit ist beispielsweise der Subprozess gemeint, der alle Aktivitäten umfasst, die durch eine bestimmte Person verrichtet werden oder die mit einem bestimmten System interagieren. Gebildet werden die Views durch eine Menge von Vereinfachungsoperationen, ähnlich der Reduktion bzw. Simplify.

[BMG05, MBGM05a, MBGM05b, BMG06] verwenden eine Art von Views, um durch Entfernen (Reduktion) von einzelnen Schritten eine Service-Spezifikation an die Bedürfnisse der Benutzer anzupassen.

Der Ansatz konfigurierbarer Referenzprozesse unterstützt, neben der zuvor beschriebenen (Metamodell-)Projektion, auch eine Reduktion, die als Modellprojektion bezeichnet wird [BK02, BDK07]. In entsprechenden Annotationen der Modellelemente von ereignisgesteuerten Prozessketten hinterlegt der Modellierer, welche Elemente für welchen Anwender relevant bzw. nicht relevant sind. Aufgrund dieser Inforation können die Modelle dann benutzerspezifisch angepasst werden. Dieses Vorgehen setzt voraus, dass der Modellierer über das Wissen bzgl. der Relevanz der Elemente verfügt und dieses vollständig im Modell hinterlegt. Weiter müssen dem Modellierer alle möglichen Darstellungen des Prozesses bekannt sein, um diese im Modell zu hinterlegen. Letztlich resultiert das Verfahren in sehr komplexen Annotationen der Prozesselemente, die langfristig schwer wartbar sind.

[EG07, EG08] präsentiert einen zweistufigen View-Mechanismus, um Prozessbeschreibungen an die Bedürfnisse der Benutzer anzupassen. In Schritt 1 werden aggregierte Views angelegt, die Implementierungsdetails verbergen. In Schritt 2 könne diese Views weiter aggregiert werden, um die Anforderungen des Benutzers zu erfüllen. Der Ansatz verwendet ein blockstrukturiertes Modell und definiert Aggregationsoperationen, welche die Funktionalität von `AggrSESE` und `AggrComplBranches` besitzen. Die aggregierten Bereiche müssen also immer ein SESE oder vollständige Äste einer Parallelität sein (keine Verletzung der Blockstruktur). Da dieser Ansatz keine Reduktion unterstützt, werden Teile des Prozesses, die der Benutzer nicht sehen darf oder nicht sehen will als „HIDDEN“ angezeigt, d.h. sie können nicht wirklich aus dem Modell entfernt werden.

Aus dem Bereich der wissenschaftlichen Workflows (engl. Scientific Workflows) stammt der Ansatz ZOOM*UserViews [CBD06, BCBD07, CBBCD08]. Bei den dort modellierten Workflows geht es um die massenhafte Verarbeitung von wissenschaftlichen Daten (z.B. aus Messreihen). Die Zuführung der Daten zu verschiedenen Bearbeitungsschritten wird durch Workflows koordiniert. Dementsprechend steht bei diesen Anwendungen der Datenfluss im Zentrum des Interesses. Der Ansatz ermöglicht es dem Benutzer, sich mit Hilfe einer Aggregation von Datenaufbereitungsschritten einen personalisierten Workflow anzeigen zu lassen.

Der in [KLRZ94] beschriebene Ansatz stammt aus dem Bereich der Software-Visualisierung und befasst sich mit der Reduktion der visuellen Komplexität von Graphen. Es werden sehr ähnliche Anforderungen wie in Proviado beschrieben. Als Lösung schlägt dieser Ansatz ebenfalls die Reduktion von Graphknoten und deren Abstraktion (d.h. Aggregation) vor. Eine weitere Lösungsidee besteht darin, die betroffenen Knoten in der Zeichnung „auszugrauen“ und dadurch in den Hintergrund zu setzen. Leider wird in [KLRZ94] nur die Idee beschrieben. Ein Vorschlag für

die operative Umsetzung fehlt. Prozess-spezifische Aspekte, wie die Integration des Datenflusses, bleiben außen vor.

Der neben Proviado für die Visualisierung geeignetste Ansatz stammt von Shen und Liu [LS01, SL01, LS03, SL03, LS04, SL04]. In [LS03] wird eine Aggregationsoperation definiert, die eine Menge von Aktivitäten zu einer so genannten virtuellen Aktivität aggregiert. Eine virtuelle Aktivität muss immer atomar ausgeführt werden, weshalb eine Aggregation nur für SESEs erlaubt ist. Dieses Papier führt zudem den Begriff der Ordnungserhaltung ein, was in Proviado der strengen Ordnungserhaltung entspricht. Weiter wird auch ein Algorithmus präsentiert, um ausgehend von einer beliebigen Menge von Aktivitäten durch Hinzufügen weiterer Aktivitäten die kleinste Menge zu bestimmen, die die Eigenschaften der Atomarität und Ordnungserhaltung erfüllen. In [LS04] wird dieser Ansatz auf unternehmensübergreifende Prozesse übertragen. Zusätzlich wird eine Zustandsabbildungsfunktion angegeben, und es werden verschiedene Strategien vorgestellt, wie mit adjazenten Datenelementen der Aggregationsmenge verfahren werden kann. In [SL03, SL04] wird schließlich ein Algorithmus vorgestellt, der zeigt, wie mit Hilfe einer Relevanzfunktion, welche die Wichtigkeit der einzelnen Aktivitäten für einen Bearbeiter beschreibt, benutzerspezifische Views automatisch generiert werden können. Eine Reduktion wird nicht angeboten.

Ein ähnlicher Ansatz, der auf Reduktion und Vereinfachungsregeln basiert, findet sich in [SPB05]. Auch dieser Ansatz verwendet eine Relevanzfunktion zur Bestimmung der zu reduzierenden Aktivitätenmenge. Wir werden auf diesen und den vorangegangenen Ansatz in der Diskussion von Kapitel 5 noch einmal genauer eingehen.

Fazit

Alle in der Literatur dokumentierten Ansätze zur Bildung von Views auf Prozessmodellen sind für eine Anwendung in der Prozessvisualisierung nicht weit reichend genug. Der Proviado-View-Mechanismus hebt sich in vielen Aspekten von den existieren Ansätzen zu Prozess-Views ab:

- Die View-Bildung ist auf einem sehr allgemeinen Prozessmodell definiert, d.h. es bestehen keine signifikanten Einschränkungen hinsichtlich des zugrunde liegenden Prozessformalismus.
- Der Proviado-View-Mechanismus berücksichtigt alle Prozessaspekte (Kontrollfluss, Attribute, Datenfluss, etc.).
- Um bessere Ergebnisse bei der Visualisierung zu erhalten, erlaubt Proviado die Bildung von Views, die nicht abhängigkeitserhaltend sind.

Mit dem in dieser Arbeit vorgestellten Ansatz erreichen wir die Flexibilität und Mächtigkeit, die für eine benutzerspezifische Visualisierung erforderlich sind. In Kapitel 5 werden wir höherwertige Operationen zur Bildung von Prozess-Views präsentieren, die auf den Elementaroperationen aus diesem Kapitel aufbauen.

4.8 Zusammenfassung

In diesem Kapitel haben wir die Grundlagen von Prozess-Views eingeführt. Mit deren Hilfe können Prozessmodelle strukturell modifiziert werden, indem Elemente entfernt (Reduktion) oder zusammengefasst werden (Aggregation). Dabei decken die vorgestellten View-Bildungsoperationen alle Prozessaspekte ab. Die Eigenschaften der View-Operationen wurden formal untersucht und entsprechende Kriterien definiert. Somit ist es möglich, anhand der angewendeten View-Bildungsoperationen über die Eigenschaften des resultierenden Prozessmodells zu argumentieren.

Mittels der Prozess-Views können Prozessmodelle für die Visualisierung an die Bedürfnisse der Betrachter angepasst werden, indem Prozesselemente, die in der jeweiligen Situation irrelevant sind, entfernt oder abstrahiert dargestellt werden. Damit erreichen wir eine entscheidende Vereinfachung der Modelle, was diese für die Betrachter besser verständlich macht.

Nachdem in diesem Kapitel die theoretischen Grundlagen für die View-Bildung gelegt wurden, werden wir im folgenden Kapitel 5 die praktischen Aspekte genauer betrachten und aufbauend auf den Elementaroperationen dieses Kapitels höherwertige Operationen vorstellen. Diese ermöglichen im Gegensatz zur alleinigen Verwendung von Elementaroperationen eine wesentlich einfachere Definition von Views.

5

Prozess Views: Höherwertige Operationen und Anwendung

Im vorangehenden Kapitel haben wir als Grundlage des Proviado-View-Mechanismus die benötigten Elementaroperationen für die Bildung von Views auf Prozessschemata und -instanzen eingeführt. Neben Operationen für die Bildung von Views auf Kontrollflussgraphen haben wir weitere Prozessaspekte (Attribute und Daten) berücksichtigt. Jedoch wurde auch deutlich, dass die Definition einer View mit Hilfe dieser primitiven Elementaroperationen zu aufwendig und umständlich wäre. In diesem Kapitel führen wir Operationen ein, die auf den in Kapitel 4 vorgestellten Elementaroperationen aufbauen, intern allerdings mehr Logik aufweisen. Dadurch soll die Definition von Views entscheidend vereinfacht werden.

Abschnitt 5.1 motiviert die Problematik im Detail und gibt einen Überblick über die Konzeption der höherwertigen View-Bildungsoperationen in Proviado. Die Abschnitte 5.2 bis 5.4 stellen diese Operationen im Detail vor. Für die benutzerfreundliche und flexible Definition von Views beschreiben wir in Abschnitt 5.5 eine View-Definitionssprache. Abschnitt 5.6 geht auf Implementierungsaspekte ein und zeigt Optimierungspotentiale auf. Schließlich diskutieren wir den Gesamtansatz einer View-Bildung in Abschnitt 5.7 und stellen einen Vergleich mit existierenden Ansätzen an.

5.1 Motivation

In Abschnitt 4.6 haben wir gezeigt, wie sich die Beispiele 4-3 und 4-6 (siehe Seite 39) mit Hilfe der vorgestellten elementaren View-Operationen realisieren lassen. Dabei wird deutlich, dass der Aufwand für die Definition einer View relativ hoch ist und Detailwissen erfordert. Gemäß der Anforderungen 4-9 und 4-10 (siehe Seite 43) sollte eine View-Definition aber auf möglichst hoher Abstraktionsebene und mit Hilfe von semantisch reichhaltigen Operationen erfolgen

können. Die Herausforderung besteht darin, eine Methode zu entwickeln, die auf Grundlage der Elementaroperationen aus Kapitel 4 eine einfache Definition von Prozess-Views ermöglicht. Dadurch soll möglichst viel Komplexität vor dem Benutzer verborgen werden.

Diese Anforderungen werden in Proviado erfüllt, indem höherwertige View-Bildungsoperationen in einem mehrschichtigen Ansatz organisiert werden (vgl. Abbildung 5.1). Die Operationen in höher liegenden Schichten greifen dabei auf die darunter liegenden Operationsschichten zu, um komplexere Funktionen zu realisieren. Zur Definition einer View kann der Benutzer entsprechenden Gebrauch von Operationen aus allen Schichten machen. Typischerweise wird er aber auf Operationen aus höheren Schichten zurückgreifen.

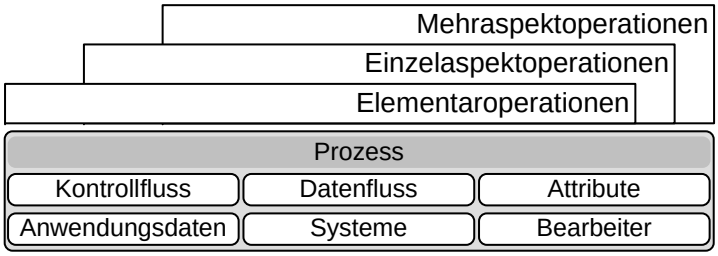


Abbildung 5.1: Schichtenmodell der Proviado-Views

Im Einzelnen legt das Schichtenmodell aus Abbildung 5.1 die folgenden Operationstypen fest:

Elementaroperationen sind auf eine spezielle Konstellation der involvierten Elemente im Prozessmodell zugeschnitten und können nur die jeweilige Struktur verarbeiten. Die Elementaroperationen wurden in Kapitel 4 vorgestellt.

Einzelaspektoperationen (Single-Aspect Operations) erhalten als Eingabe eine Menge von Prozesselementen desselben Typs (Kontrollfluss oder Datenfluss), die verarbeitet werden sollen. Sie analysieren die Struktur der Eingabemenge im Prozessmodell und wählen daraufhin die passende(n) Elementaroperation(en) aus. Mit Hilfe von Parametern kann die Auswahl gesteuert und somit Einfluss auf die Eigenschaften der resultierenden View genommen werden.

Mehraspektoperationen (Multi-Aspect Operations) ermöglichen eine integrierte View-Bildung über mehrere Prozessaspekte hinweg. So können beispielsweise Aktivitäten zusammen mit deren adjazenten Datenelementen verarbeitet werden. Dazu bilden die Mehraspektoperationen die View-Bildung auf verschiedene Einzelaspektoperationen ab.

5.2 Einzelaspektoperationen

Aufgabe der Einzelaspektoperationen ist es, für eine gegebene Menge von Prozesselementen desselben Typs (z.B. Aktivitäten im Kontrollfluss) die passenden Elementaroperationen auszuwählen. Dazu definieren wir für jeden Prozessaspekt je eine Einzelaspektoperation für die Reduktion und eine für die Aggregation. In den folgenden Abschnitten werden diese Operationen für alle Prozessaspekte betrachtet. Wir beginnen mit den Operationen für den Kontrollfluss (ReduceCF und AggregateCF). Die Auswahl der Elementaroperationen wird über Parameter

gesteuert. Mit ihrer Hilfe lässt sich festlegen, welche Anforderungen (d.h. Eigenschaften) die resultierende View erfüllen soll.

5.2.1 Reduktion

Ziel der Operation ReduceCF ist es, eine gegebene Menge von Aktivitäten aus dem Prozessmodell zu entfernen. Abbildung 5.2 zeigt ein Beispiel. Im ersten Schritt werden die zu reduzierenden Aktivitäten (hier: $\{B, F, G, J, M, N, O, P, Q, R\}$) nacheinander durch Anwendung der Elementaroperation `RedActivity` gelöscht. Das resultierende Prozessmodell enthält unter Umständen überflüssige Kontrollflussstrukturen (leere Kanten, unnötige Verzweigungsknoten, etc.). Diese werden im zweiten Schritt durch die Operation SimplifyCF erkannt und entfernt (für Details siehe Abschnitt 5.2.3).

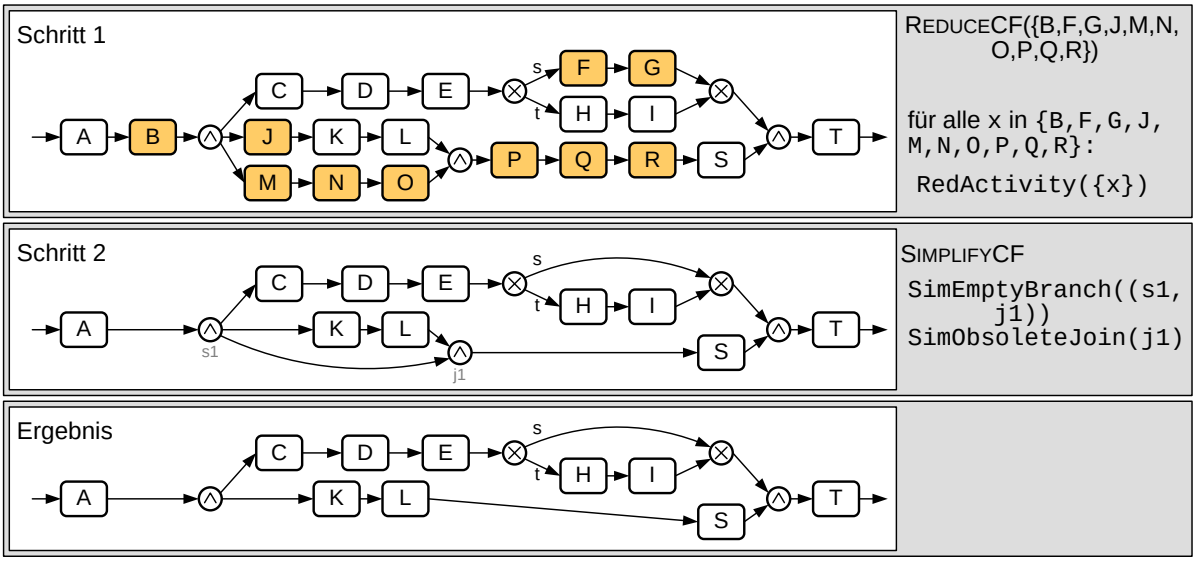


Abbildung 5.2: Reduktion von Aktivitäten durch ReduceCF

5.2.2 Aggregation

Während die Reduktion recht einfach aufgebaut ist, sind bei der Aggregation mehr Faktoren zu berücksichtigen. Die Idee der Operation AggregateCF ist im Prinzip dieselbe: wähle für eine gegebene Menge zu aggregierender Aktivitäten die passende(n) Elementaroperation(en). Im Gegensatz zur Reduktion muss nun aber zwischen mehreren Elementaroperationen ausgewählt werden. Des Weiteren existieren, wie in Kapitel 4 erörtert, für manche Situationen mehrere mögliche Aggregationsoperationen. Umgekehrt bestehen, je nach Anwendung, verschiedene Anforderungen an die Konsistenz bzw. Präzision einer View. In Abschnitt 4.3.2 wurden die Elementaroperationen für die Aggregation vorgestellt und deren Eigenschaften analysiert. An dieser Stelle nutzen wir das Wissen über diese Eigenschaften, um die Einzelaspektoperation AggregateCF zu parametrisieren. Abbildung 5.3 illustriert ein entsprechendes Beispiel eines Prozesses, in dem die Aktivitätenmenge $\{B, C, F, H, I\}$ aggregiert wird. Soll beispielsweise die

resultierende View zustandskonsistent sein, etwa weil diese später automatisch ausgewertet werden soll und das System einen konsistenten Zustand voraussetzt, spezifizieren wir mittels eines Parameters (*states=consistent*), dass bei der View-Bildung keine Operationen angewendet werden dürfen, die zu einer Zustandsinkonsistenz führen können. Dies führt zur Anwendung der Elementaroperation `AggrAddBranch` (vgl. Fall 1). Spielt Zustandskonsistenz keine Rolle (*states=default*), so wird die Operation `AggrShiftOut` angewendet. Es resultiert für den Beispielprozess die in Fall 2 dargestellte View. Werden die Anforderungen (d.h. Parameter) zu strikt gewählt, kann dies für bestimmte Kontrollflussstrukturen bedeuten, dass keine Elementaroperation angewendet werden kann (vgl. Fall 3 in Abbildung 5.3).

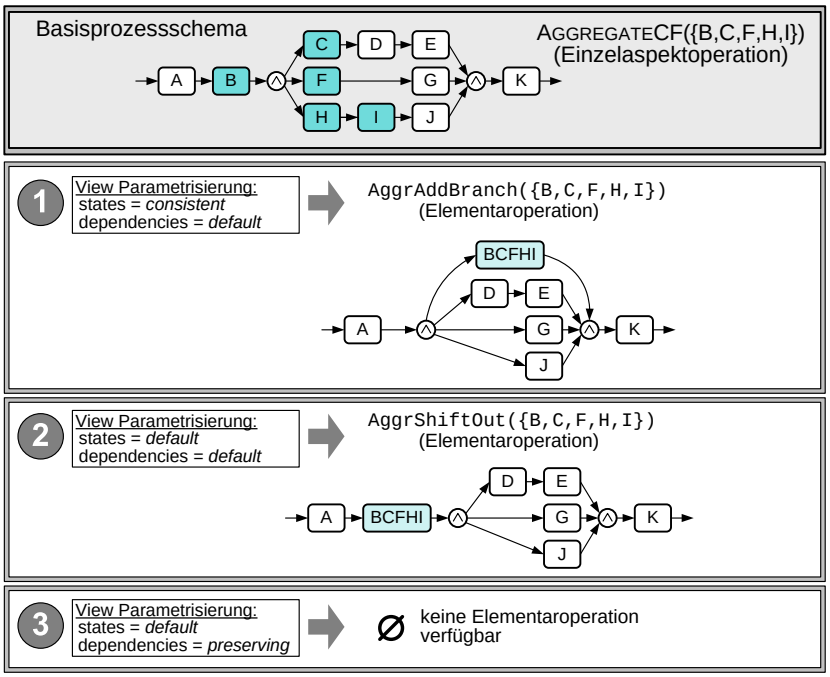


Abbildung 5.3: Auswahl verschiedener Aggregationsoperationen durch Parametrisierung

Die Parameter stellen somit quasi die Beschreibung einer Anforderung an die View-Bildung dar. Der resultierende Prozess muss (und wird) diese Anforderungen (Eigenschaften) erfüllen. Die Auswahl der Operationen basiert auf den in Kapitel 4 identifizierten Operationseigenschaften (vgl. Tabelle 4.7 auf Seite 93). Der Algorithmus zur Auswahl der Operationen wird später auf Seite 107 beschrieben.

Die Auswahl aufgrund der Operationseigenschaften ist jedoch nur ein Freiheitsgrad der Operation AGGREGATECF. Neben der Möglichkeit die Eigenschaft der View durch Ausschluss bestimmter Elementaroperationen bei der View-Bildung zu beeinflussen, können wir weiter die zu aggregierende Menge von Aktivitäten verändern. Hierfür existieren zwei Optionen:

1. Die Menge der Aktivitäten wird durch Hinzunehmen weiterer Aktivitäten vergrößert.
2. Die Menge der Aktivitäten wird so unterteilt, dass die View-Bildung auf Submengen eventuell zu besseren Views (d.h. Views mit mehr zugesicherten Eigenschaften) führt.

Eine denkbare weitere Option, die Menge der Aktivitäten zwecks Erzielung eines besseren Ergebnisses zu verkleinern, schließen wir aus. Gerade für Anwendungen, in denen Views zum Verbergen vertraulicher Details verwendet werden, muss sichergestellt sein, dass (mindestens) die Menge der selektierten (vertraulichen) Aktivitäten aggregiert wird.

Zusammen mit der Option, keine Änderungen an der gegebenen Aktivitätenmenge vorzunehmen, ergeben sich somit drei mögliche Basisstrategien zur Aggregation. Proviado bildet diese durch den Parameter *strategy* für die Operation AggregateCF ab.

- *as-is* Die selektierte Aktivitätenmenge wird ohne Unterteilung oder Vergrößerung aggregiert.
- *subdivide* Die Selektionsmenge wird unterteilt. Je nach Anforderungen an die View-Eigenschaften ist eine mehr oder weniger starke Aufteilung nötig, im Extremfall solange, bis die Submengen SESEs bilden.
- *expand* Die Selektionsmenge wird zu einem SESE erweitert, so dass eine Aggregation durch die Elementaroperation `AggrSESE` erfolgen kann.

Die Auswirkungen dieses Parameters auf die Aggregation illustriert Abbildung 5.4. In diesem Beispiel sollen die Aktivitäten C, D, G und H aggregiert werden. Bei Aggregation mit der Strategie *as-is* werden die Aktivitäten mittels `AggrAddBranch` zu einer neuen, abstrakten Aktivität $CDGH$ zusammengefasst. Die Anwendung der Strategie *subdivide* führt zu zwei eigenständigen Aggregationen mittels `AggrSESE`, woraus die abstrakten Aktivitäten CD und GH resultieren. Die Strategie *expand* erweitert die Aggregationsmenge zu einem SESE und aggregiert diese zur abstrakten Aktivität $BCDEFGHI$.

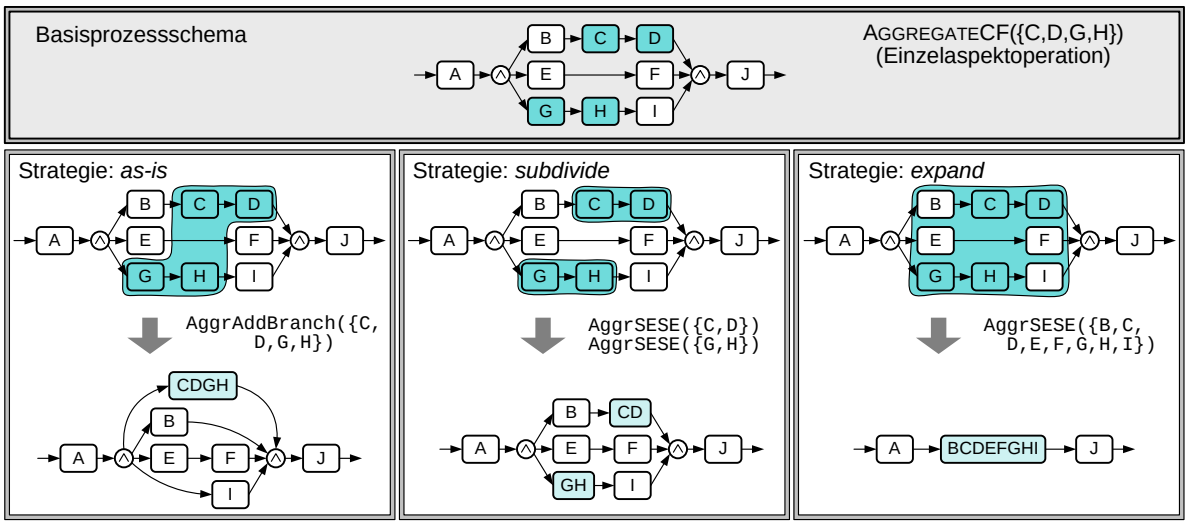


Abbildung 5.4: Auswirkungen der verschiedenen Strategien bei Aggregation

Die Strategien lassen sich zu beliebig komplexen Vorgehensweisen kombinieren. Beispielsweise könnte die Menge der selektierten Aktivitäten zunächst in mehrere Mengen unterteilt werden, welche dann durch Hinzufügen einzelner Aktivitäten für eine optimale Aggregation aufbereitet werden. Derartige Methoden sind zum einen sehr komplex und zum anderen sind die Ergebnisse

für Benutzer, die ein klares Bild von der gewünschten Prozessdarstellung haben, kaum vorhersehbar. Dies macht sie für die praktische Anwendung schwer nutzbar, weshalb wir hier auf die drei Basisstrategien fokussieren.

Gleichermaßen stellt das Vergrößern der Aktivitätenmenge (Strategie *expand*) im Allgemeinen keine adäquate Lösung für unsere Anwendungszwecke dar. Bei der Visualisierung sollten genau die selektierten Aktivitäten aggregiert werden und nicht mehr oder weniger. Dennoch berücksichtigen wir diese Option, da sie bei einigen Ansätzen aus der Literatur [WW96a, WW96b, LS03, EG07, EG08] existiert. Diese Ansätze verfügen meist nur über eine Operation, die vergleichbar mit `AggrSESE` ist. Daher ist das Aufblähen der Aggregationsmenge in diesen Fällen der einzig gangbare Weg, dennoch eine Aggregation durchführen zu können.

Tabelle 5.1 gibt eine Übersicht über die Parameter der Einzelaspektoperation AggregateCF. Sie setzen sich aus den Qualitätsparametern, welche die Eigenschaften der resultierenden View beeinflussen (*dependencies*, *order preserving*, *states*), einzelnen Parametern von Elementaroperationen (*branching*, *loopInSESE*, *iteration*) und dem Parameter zur Aggregationsstrategie (*strategy*) zusammen. Welche Qualitätseigenschaften die einzelnen Elementaroperationen erfüllen, ist in Tabelle 4.7 auf Seite 93 zusammengefasst.

Parameter	Parameterwerte[1]	Beschreibung
dependencies	non-erasing, non-generating, preserving, **any**	Form der geforderten Abhängigkeitserhaltung (keine Abhängigkeiten löschen, keine erzeugen, alle erhalten bzw. beliebig verfahren)
order preserving	strong, **any**	Form der geforderten Ordnungserhaltung
states	consistent, **any**	Form der geforderten Zustandskonsistenz
loopInSESE	**yes**, no	Aggregation oder Erhaltung von Schleifenkante und Schleifenknoten
branching	**tight**, wide	Art der Kantenführung von `AggrAddBranch` (eng bzw. weit)
iteration	**in**, out	Hinein- bzw. Herausziehen der Aktivitäten in die Schleife bei Aggregation durch `AggrShiftLoop`
strategy	**as-is**, subdivide, expand	Strategie im Umgang mit der selektierten Aktivitätenmenge: keine Veränderung erlaubt; Erweiterung der Aggregationsmenge; Unterteilung der Aggregationsmenge (d.h. es werden mehrere unabhängige Aggregationen durchgeführt)

[1] default-Werte sind durch fette Schrift gekennzeichnet

Tabelle 5.1: Übersicht über die Parameter der Einzelaspektoperation AggregateCF

5.2.2.1 Bestimmung der elementaren Aggregationsoperation

Die Herausforderung besteht darin, für eine gegebene Menge von selektierten Aktivitäten S im Prozessschema P die passende Aggregationsoperation aus der Menge der verfügbaren Elementaroperationen auszuwählen. Passend bedeutet in diesem Kontext, dass die resultierende View die mittels der Qualitätsparameter *param* geforderten Eigenschaften aufweisen soll. Wir verwenden

hierfür die Operation SelectAggregationOperation. Ihr Vorgehen ist in Algorithmus 5-5 detailliert dargestellt. Er prüft für jede Elementaroperation, ob die Vorbedingungen, wie in Kapitel 4 angegeben, für die Menge S erfüllt sind. Das bedeutet, dass die Struktur der Aktivitätenmenge S im Prozess P analysiert wird, um zu verifizieren, ob eine Elementaroperation in der Lage ist, die gegebene Menge von Aktivitäten zu aggregieren. Ist dies der Fall, prüft ein zweiter Schritt, ob die betreffende Operation aufgrund der geforderten Eigenschaften auch zulässig ist. Sobald eine Elementaroperation beide Bedingungen (Struktur und Qualitätseigenschaften) erfüllt, wird diese als Ergebnis von SelectAggregationOperation zurückgeliefert.

Algorithmus 5-5 : SelectAggregationOperation($P, S, param$)

Input :
P Prozessschema
S Menge von Aktivitäten, die aggregiert werden
param Steuerungsparameter
Output :
Elementaroperation für die Aggregation

```
1  if CheckPrecondition(AggrSESE,P,S) then
2      return AggrSESE
3  else if CheckPrecondition(AggrComplBranches,P,S) and
           CheckCompliance(param, AggrComplBranches,P,S) then
4      return AggrComplBranches
5  else if CheckPrecondition(AggrShiftOut,P,S) and
           CheckCompliance(param, AggrShiftOut,P,S) then
6      return AggrShiftOut
7  else if CheckPrecondition(AggrShiftLoop,P,S) and
           CheckCompliance(param, AggrShiftLoop,P,S) then
8      return AggrShiftLoop
9  // Vorbedingung von AggrAddBranch ist immer erfüllt
10 else if CheckCompliance(param, AggrAddBranch,P,S) then
11     return AggrAddBranch
12 else
13     return Error: no operation could be found conforming to parameters param
```

Die Reihenfolge, in der die Elementaroperationen innerhalb des Algorithmus überprüft werden, ist nicht zufällig, sondern das Ergebnis einer gezielten Sortierung. Ziel ist, für jede mögliche Menge von Aktivitäten die bestmögliche und damit passende Elementaroperation zu ermitteln, um so ein optimales Ergebnis für die nachfolgende Visualisierung zu erlangen. Daher werden im Algorithmus die Operationen, die ein qualitativ hochwertigeres Ergebnis liefern, weiter oben angeordnet (z.B. `AggrSESE`). Im Gegenzug wird beispielsweise die Operation `AggrAddBranch`, die durch die Erzeugung eines parallelen Pfades viele Abhängigkeiten löscht und daher ein qualitativ schlechteres Ergebnis liefert, als letzte Option ausgewählt.

Nicht jede Elementaroperation kann und darf zum Einsatz kommen. Zum einen sind die Elementaroperationen auf eine bestimmte Kontrollflussstruktur zugeschnitten und können auf andere Situationen nicht angewendet werden. Zum anderen schränken die vom Benutzer vorgegebenen Qualitätsparameter die zur Auswahl stehende Operationsmenge ein. Diese zwei Aspekte werden im Algorithmus durch die jeweiligen Überprüfungen vor Rückgabe der anzuwendenden Elementaroperation berücksichtigt.

Anforderung 4-7 fordert die Erweiterbarkeit des Ansatzes. Für Elementaroperationen haben wir diesen Punkt bereits in Abschnitt 4.3.2 angesprochen. Zusätzliche Elementaroperationen können jederzeit in den Algorithmus von SelectAggregationOperation aufgenommen werden. Dazu müssen die Vorbedingung der neuen Operation und deren Eigenschaften bekannt sein. Innerhalb des Algorithmus 5-5 wird eine neue Operation dann analog zu den vorhandenen Operationen eingefügt (tendenziell direkt vor `AggrAddBranch`).

5.2.2.2 Ablauf der Einzelaspekt-Aggregation

Abbildung 5.5 zeigt das Ablaufschema der Operation AggregateCF. Wir erklären dieses Schema entlang eines Beispiels. In diesem Beispiel sollen die im Basisprozessschema markierten Aktivitäten aggregiert werden. Je nach Parametrisierung ergibt sich ein anderes Ergebnis (vgl. Abbildung 5.9). Aus Gründen der Übersichtlichkeit werden in den folgenden Grafiken nur die Operationsparameter aufgeführt, die für das jeweilige Ergebnis bzw. die Unterscheidung der Varianten relevant sind.

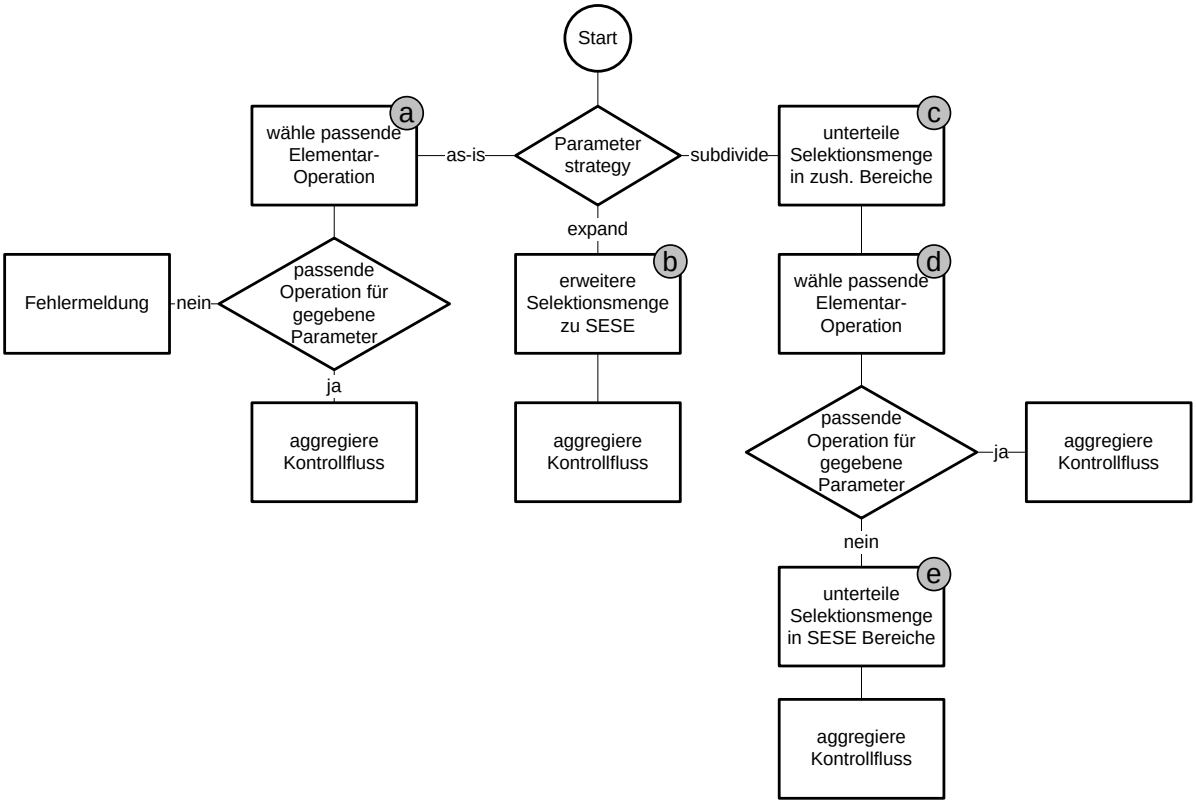


Abbildung 5.5: Ablaufschema der Operation AggregateCF

Der Algorithmus der Operation AggregateCF aus Abbildung 5.5 wertet zunächst den Strategie-Parameter aus und wählt abhängig von der (den) resultierenden Menge(n) selektierter Aktivitäten die passenden Elementaroperation(en). Im Detail verfährt er dabei wie folgt.

Strategie *as-is*

1. Wähle Elementaroperation gemäß SELECTAGGREGATIONOPERATION (Schritt (a) in Abbildung 5.5).
2. Falls eine Operation gefunden wurde, führe sie aus. Andernfalls melde einen Fehler.

Im Beispiel (vgl. Abbildung 5.6) wird bei der gegebenen Parametrisierung die Elementaroperation `AggrAddBranch` ausgewählt. In dieser View werden alle Aktivitäten zu einer abstrakten Aktivität aggregiert. Diese liegt auf einem parallelen Pfad. Werden die Qualitätsparameter strenger gewählt (z.B. *`dependencies = preserving`*), ist eine View-Bildung nicht möglich und die Operation bricht mit einer Fehlermeldung ab.

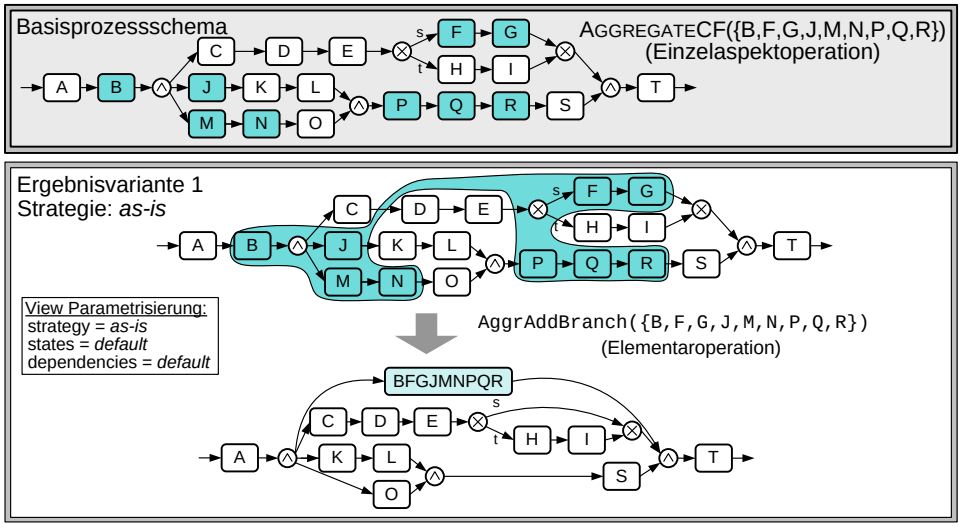


Abbildung 5.6: Aggregation mit Strategie *as-is*

Strategie *expand*

1. Erweitere Selektionsmenge S mittels der in [LS03, EG07, EG08] angegebenen Algorithmen zu einem SESE S' (Schritt (b) in Abbildung 5.5).
2. Wende die Elementaroperation `AggrSESE` auf S' an.

Diese Strategie führt in unserem Beispiel (vgl. Abbildung 5.7) dazu, dass alle Aktivitäten von B bis unmittelbar vor T zur Selektionsmenge hinzugefügt werden. Die neue Menge S' stellt nun einen SESE dar und kann daher durch die Elementaroperation `AggrSESE` aggregiert werden. Vom ursprünglichen Prozessmodell ist in der View allerdings nicht mehr viel zu erkennen. Da die Strategie *expand* automatisch eine SESE bildet und bei der View-Bildung durch die zugehörige Elementaroperation alle Eigenschaften erhalten bleiben, haben die Qualitätsparameter keinen Einfluss auf das Ergebnis.

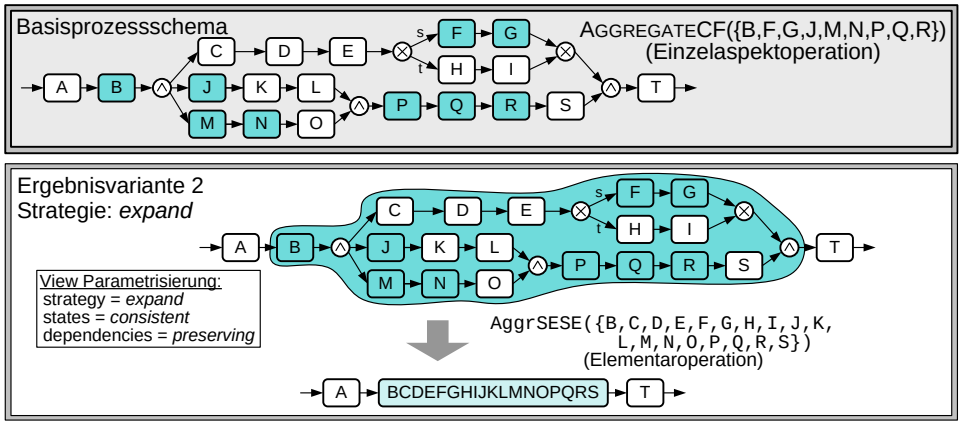


Abbildung 5.7: Aggregation mit Strategie *expand*

Strategie *subdivide*

1. Unterteile Selektionsmenge S in maximale Teilmengen S_i, sodass jeweils jedes S_i^* zusammenhängend ist (Schritt (c) in Abbildung 5.5).
2. Wähle für jedes S_i eine Operation gemäß SELECTAGGREGATIONOPERATION (Schritt (d) in Abbildung 5.5).
3. Aggregiere diejenigen S_i, für die eine Operation gefunden wurde.
4. Die verbleibenden S_i werden in SESE-Blöcke S_{i_j} zerlegt (Schritt (e) in Abbildung 5.5).
5. Aggregiere die neuen S_{i_j} mit `AggrSESE`.

Abbildung 5.8 illustriert mögliche Ergebnisse der Strategie *subdivide*. Zunächst wird die Aktivitätenmenge in Schritt (c) des Ablaufschemas aus Abbildung 5.5 in zusammenhängende Submengen unterteilt. Wir fordern für Ergebnisvariante 3.1 keine speziellen Formen der Abhängigkeitseigenschaften (vgl. Parameter ***dependencies=default***). Allerdings führt eine unterschiedliche Belegung des Parameters ***states*** zu verschiedenen Views. In Ergebnisvariante 3.1a sollen die Zustände konsistent abgebildet werden, weshalb die Elementaroperation `AggrAddBranch` zur Aggregation der Menge $\{B, J, M, N\}$ ausgewählt wird. Sind inkonsistente Zustände erlaubt, wird in Schritt (d) des Ablaufschemas die Operation `AggrShiftOut` ausgewählt, was wiederum zu Ergebnisvariante 3.1b führt.

Für Ergebnisvariante 3.2 wird die Erhaltung der Abhängigkeiten gefordert. Damit ist eine Anwendung der Operationen `AggrAddBranch` und `AggrShiftOut` ausgeschlossen. Der Algorithmus fährt daher mit der Unterteilung der Menge $\{B, J, M, N\}$ fort, bis die Teilmengen nur noch SESEs bilden. Danach ist eine Aggregation durch `AggrSESE` möglich.

Abbildung 5.9 zeigt nochmals im Überblick, abhängig von der Parametrisierung der Einzelaspektoperation AGGREGATECF, die möglichen Ergebnisse für unser Beispiel. Für die Visualisierung wird in den meisten Fällen die Strategie ***as-is*** gewählt werden, da genau die spezifizierten Aktivitäten zu einem Knoten aggregiert werden sollen. In Abschnitt 5.4 werden wir noch Fälle kennen lernen, in denen die Aggregationsmenge nicht von Hand festgelegt wird, sondern sich

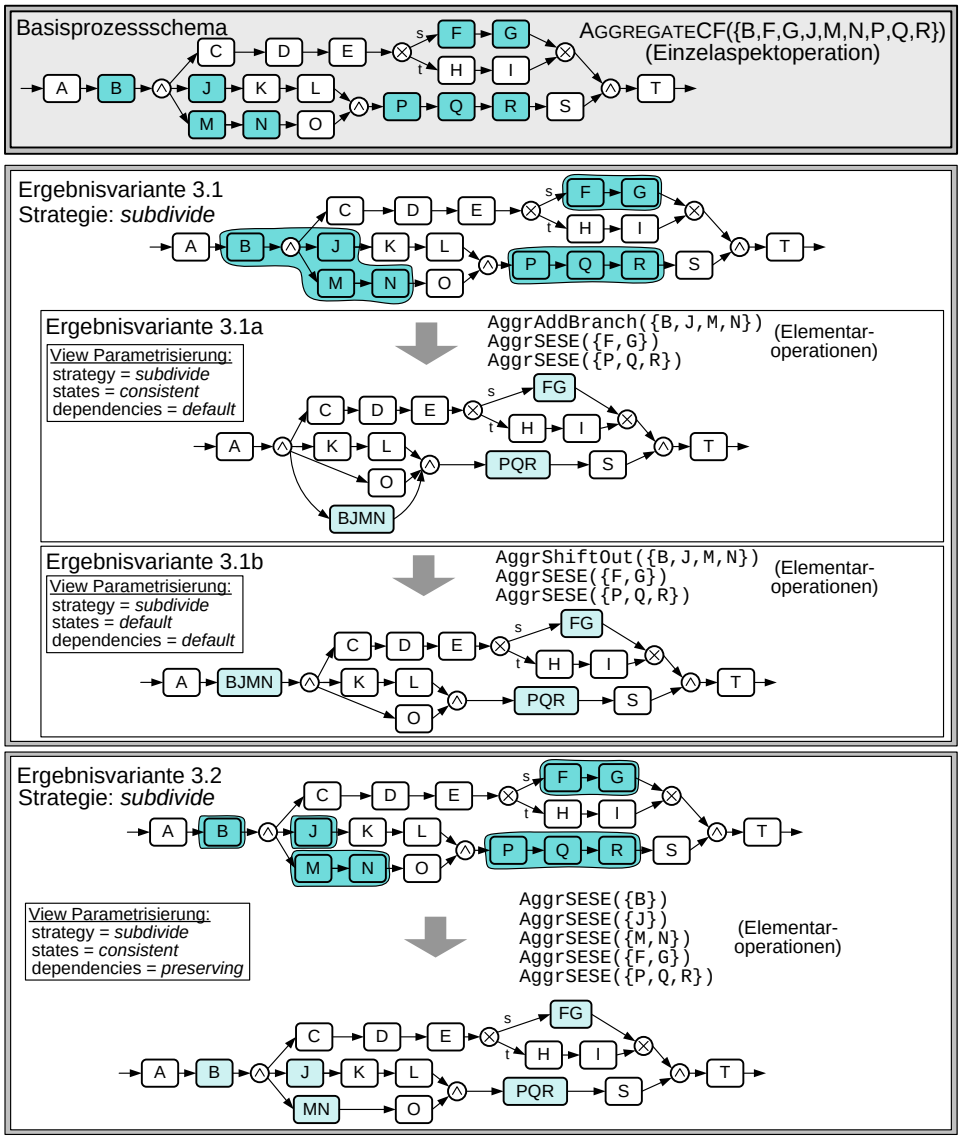


Abbildung 5.8: Aggregation mit Strategie *subdivide*

aus der Auswertung eines Prädikats ergibt. Für diese dynamische View-Bildung kann es wünschenswert sein, die Menge der selektierten Aktivitäten vor der View-Bildung in Submengen zu unterteilen. Im vorgestellten Algorithmus wird für die Unterteilung eine zweistufige Vorgehensweise vorgeschlagen. Diese erzeugt für die erwartete Anwendung akzeptable Ergebnisse. Generell existieren unzählige Möglichkeiten, die Aktivitätenmenge zu unterteilen. Zum Beispiel kann die Aktivitätenmenge derart partitioniert werden, dass die Aktivitäten der Submengen jeweils in einer Sequenz liegen (ohne Strukturknoten, möglicherweise aber mit anderen Aktivitäten dazwischen), um sie anschließend mittels `AggrAddBranch` zu aggregieren. Alternativ kann man Partitionen aus Aktivitäten bilden, die jeweils nicht weiter als n Schritte voneinander entfernt sind.

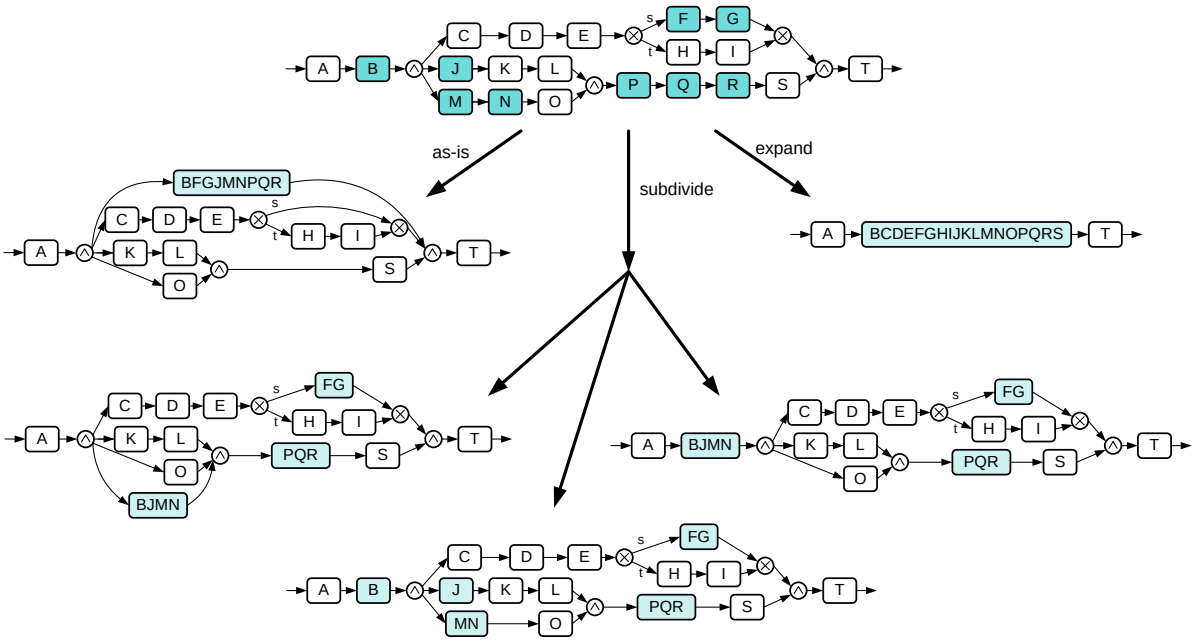


Abbildung 5.9: Ergebnisvarianten der Aggregation abhängig von der Parametrisierung

5.2.3 Simplify

Die Operation SimplifyCF analysiert das gegebene Prozessmodell und wählt aus den vorhandenen Vereinfachungsoperationen (vgl. Abschnitt 4.3.4) die passenden aus. Dazu ist zunächst eine strukturelle Analyse des Prozessgraphen erforderlich, um die Strukturen zu identifizieren, die mittels einer der elementaren Vereinfachungsoperationen verarbeitet werden können. Die Operation SimplifyCF wird prinzipiell nach jeder Reduktion oder Aggregation ausgeführt, um überflüssige Kontrollflussstrukturen zu entfernen.

In der hier vorgestellten Form wirkt SimplifyCF global auf den gesamten Prozess. Als Optimierung bzw. zur Steigerung der Effizienz könnte man den Untersuchungsbereich von SimplifyCF auf die von der View-Bildung betroffenen Knoten und deren direkte Umgebung beschränken, um so den Analyseaufwand zu senken. Unter Umständen werden durch SimplifyCF Strukturen im Prozess vereinfacht, die zwar kompakter modelliert hätten werden können, vom Prozessmodellierer jedoch explizit so abgebildet wurden (z.B. zwei aufeinander folgende AND-Splits). Durch spezielle Markierungen der Knoten kann man in diesen Fällen eine Vereinfachung verhindern.

Durch die Anwendung der Vereinfachungsoperationen kann bei blockstrukturierten Prozessen die Struktur verletzt werden. Dies geschieht beispielsweise, wenn durch `SimObsoleteSplit` bzw. `SimObsoleteJoin` (vgl. Abschnitt 4.3.4) Verzweigungsknoten zusammengefasst werden. Durch eine entsprechende Restriktion der Operation kann dem vorgebeugt werden. In Proviado arbeiten wir jedoch mit nicht-blockstrukturierten Graphen, weshalb dies hier nicht weiter vertieft wird.

5.2.4 Einzelaspektoperationen für andere Prozessaspekte

Datenfluss

Die Elementaroperation `RedData` aus Abschnitt 4.5.2 ist in der Lage, ein einzelnes Datenelement aus einem Prozessmodell zu entfernen. Darauf aufbauend definieren wir ReduceDF, um eine Menge von Datenelementen zu reduzieren. ReduceDF iteriert über die Menge der Datenelemente und verarbeitet jedes einzeln durch einen Aufruf von `RedData` (vgl. Abbildung 5.10).

Die Einzelaspektoperation zur Aggregation von Datenelementen AggregateDF wird hier lediglich der Vollständigkeit halber aufgeführt. Ein Aufruf der Operation wird direkt an die Elementaroperation `AggrData` weitergeleitet, welche die gegebenen Datenelemente zu einem abstrakten Datenelement aggregiert (vgl. Abbildung 5.10).

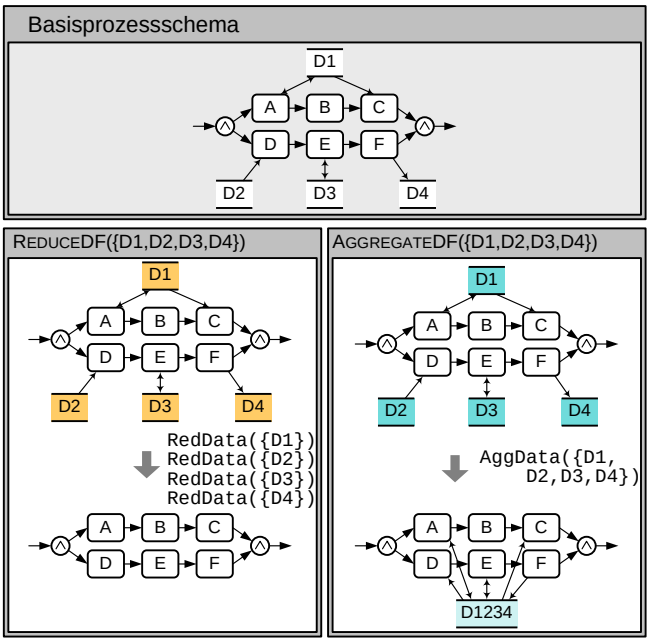


Abbildung 5.10: Einzelaspektoperationen für Datenfluss

Attribute

Für die Verarbeitung von Attributen, stellen wir Einzelaspektoperationen zur Verfügung, die jeweils eine Menge von Attributen wegprojizieren respektive transformieren.

Für eine gegebene Menge von Prozessobjekten S und eine Menge von Attributen A ruft Project jeweils die Elementaroperation `ProjectAttr` (vgl. Abschnitt 4.4.2)auf, um die Attribute in A von den Prozessobjekten S zu entfernten (vgl. Abbildung 5.11). D.h. $\forall s \in S$: `ProjectAttr`(s, A)

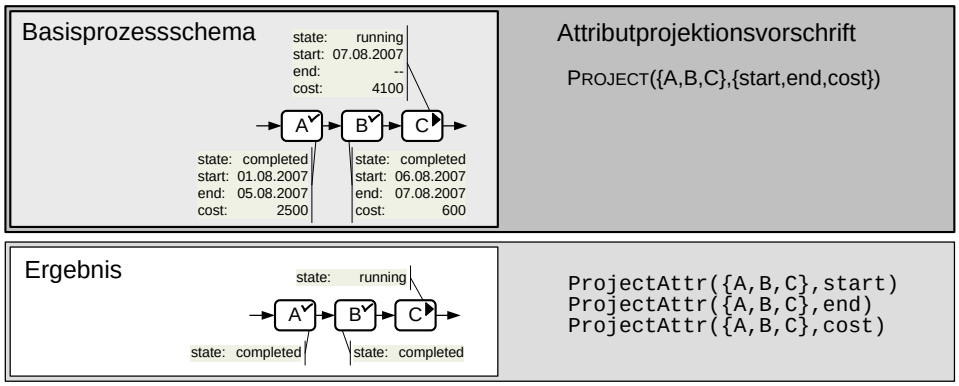


Abbildung 5.11: Projektion von Attributen mittels PROJECT

Die Einzelaspektoperation TRANSFORM bietet die Möglichkeit, mehrere Transformationen gleichzeitig für eine bestimmte Menge von Ausgangsobjekten sowie ein Zielobjekt durchzuführen. Dabei wird intern wieder auf die Elementaroperation zur Transformation (`TransformAttr`) (vgl. Abschnitt 4.4.3) zurückgegriffen.

Gegeben sei eine Menge von Prozessobjekten S deren Attribute durch eine Menge von Transformationsbeschreibungen $\mathcal{T}$ auf ein Prozessobjekt n übertragen werden sollen. Eine Transformationsbeschreibung ist hier ein Tupel (a, f), das definiert, welche Transformationsfunktion f für ein Attribut a verwendet werden soll. TRANSFORM ruft dazu für jede Transformationsbeschreibung aus $\mathcal{T}$ die Elementaroperation `TransformAttr` auf. D. h.

$$\forall (a, f) \in \mathcal{T} : \texttt{TransformAttr}(\{s.a | s \in S\}, f, n.a)$$

Abbildung 5.12 illustriert dies anhand eines Beispiels.

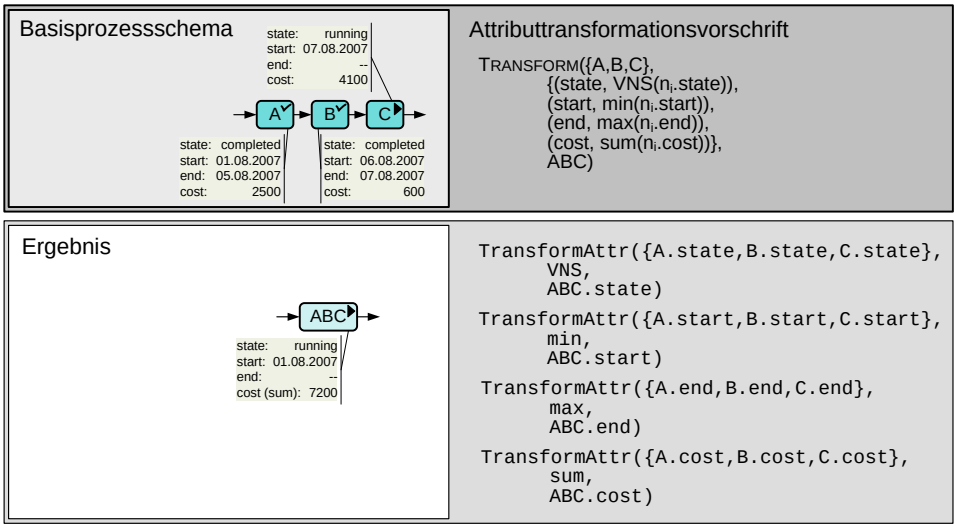


Abbildung 5.12: Transformation von Attributen mittels TRANSFORM

Die Elementaroperationen stehen dem Benutzer natürlich weiterhin zur Verfügung, so dass über diesen Weg spezielle Transformationen mittels `TransformAttr` direkt definiert werden

können. Mit der Elementaroperation `TransformAttr` können einem bestehenden Objekt auch neue Attribute hinzugefügt werden. Der Attributwert kann das Ergebnis einer Berechnung oder ein konstanter Wert (ohne Verwendung einer Transformationsfunktion) sein.

5.3 Mehraspektoperationen

Die View-Operationen der vorangegangenen Abschnitte arbeiten jeweils nur auf einem Prozessaspekt. Aufgabe der Proviado-Mehraspektoperationen ist es, die automatische View-Bildung über alle Aspekte (Kontroll-, Datenfluss und Attribute) hinweg zu ermöglichen. Proviado definiert in dieser Operationsschicht daher genau zwei Operationen: Reduce für die aspektübergreifende Reduktion und Aggregate für die aspektübergreifende Aggregation. Bevor wir auf die Realisierung dieser Operationen näher eingehen, werfen wir einen Blick auf die Interaktion der View-Operationen in den unterschiedlichen Schichten.

In Abbildung 5.13 sind die zwei Mehraspektoperationen Reduktion und Aggregation dargestellt. Diese interagieren mit den Operationen der drei dargestellten Aspekte (Kontrollfluss, Datenfluss und Attribute). Ausnahmen bilden die elementaren Aggregationsoperationen, die jeweils zur Transformation ihrer Attribute die entsprechende Attributoperation aufrufen. Betrachtet man die Interaktion zwischen den Schichten, so ruft jede Operation prinzipiell die Operationen der darunter liegenden Schicht vom selben Typ (Reduktion oder Aggregation) auf. Die Darstellung zeigt die in dieser Arbeit detailliert betrachteten Aspekte. Wie in Abschnitt 4.5.4 beschrieben, kann der Proviado-View-Mechanismus jederzeit um weitere Aspekte (Bearbeiterzuordnungen, Systeme) ergänzt werden. Dazu müssen lediglich entsprechende Operationen bereitgestellt werden und wenn gewünscht in die Interaktion integriert werden.

Bei der View-Bildung auf Prozessen ist für die Visualisierung der in Proviado untersuchten Prozesse der Kontrollflussaspekt (und damit die Aktivitäten) der wichtigste Aspekt. Daher sind die beiden hier vorgestellten Mehraspektoperationen entsprechend aktivitätenzentriert. Das bedeutet, dass die Eingabe aus einer Menge von Aktivitäten besteht, die reduziert bzw. aggregiert werden soll. Ausgehend von dieser Menge werden die notwendigen Einzelaspektoperationen konzertiert. Analoge Operationen, die zum Beispiel ausgehend vom Datenfluss die View-Operationen auf den anderen Aspekten steuern, sind denkbar, werden hier aber nicht weiter betrachtet. Wir werden in Abschnitt 5.4 die notwendigen Mittel vorstellen, um eine solche Vorgehensweise im Proviado-View-Mechanismus zu realisieren.

5.3.1 Reduktion

Die Mehraspektoperation Reduce entfernt eine Menge von Aktivitäten aus einem Prozess und bei Bedarf deren adjazente Datenelemente. Wir werden diese Operation wieder anhand eines Beispiels erläutern. In Abbildung 5.14 sollen die Aktivitätenmenge $S = \{C, D, E, F, G, H\}$ aus dem Prozessmodell entfernt werden. Mittels der bekannten Einzelaspektoperation ReduceCF werden im ersten Schritt die entsprechenden Aktivitäten gelöscht. Zu beachten ist, dass die dazu verwendete Operation `RedActivity` alle inzidenten Kanten entfernt, darunter auch die Datenkanten. Im zweiten Schritt ergeben sich drei Möglichkeiten, wie mit den adjazenten Datenelementen der Reduktionsmenge S weiter verfahren werden soll.

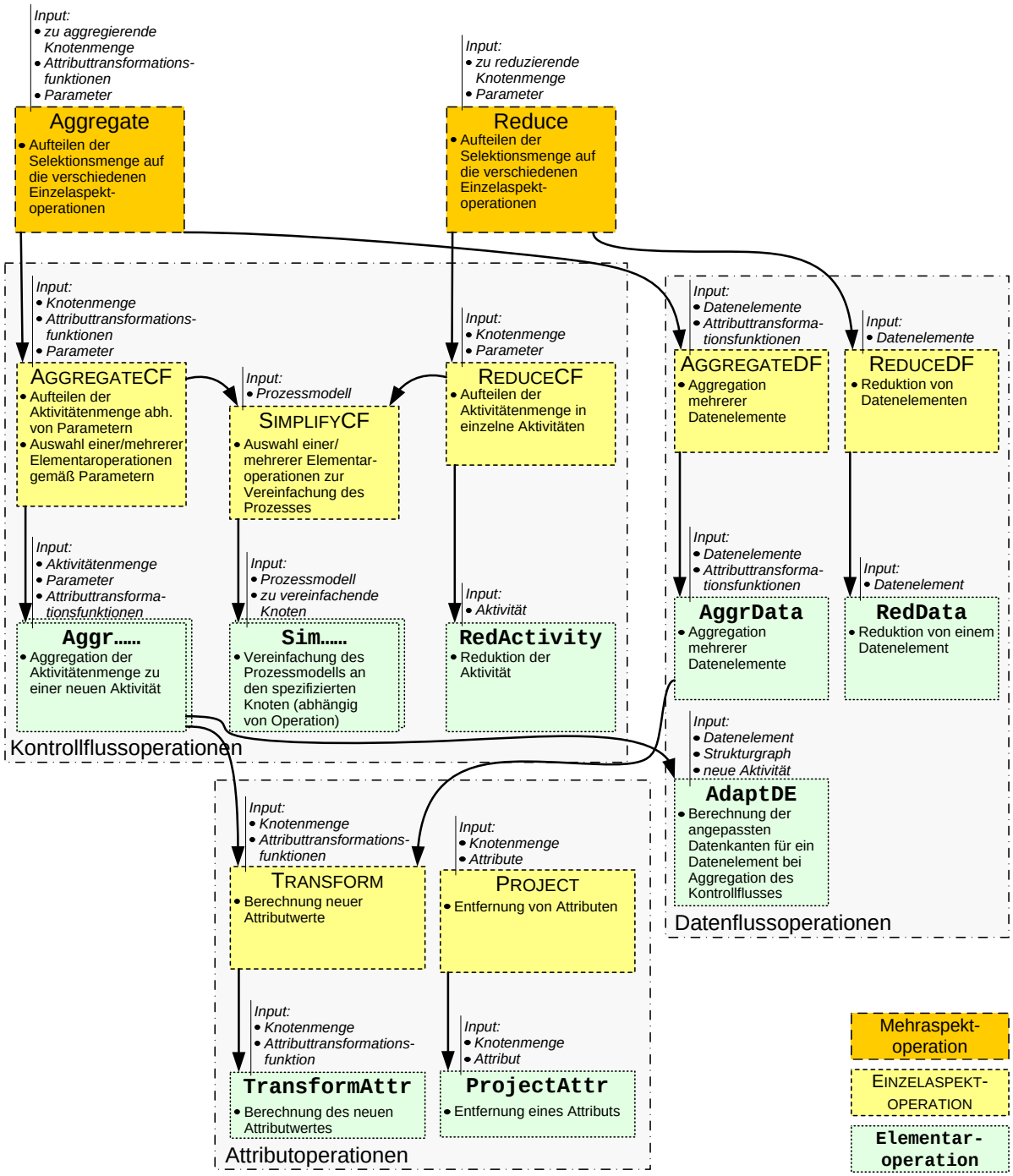


Abbildung 5.13: Interaktion der Schichten der unterschiedlichen View-Operationen

(i) Alle Datenelemente bleiben erhalten.

(ii) Die verwaisten Datenelemente (d.h. diejenigen, die zu keiner anderen Aktivität adjazent sind) werden entfernt, die anderen bleiben erhalten.

(iii) Alle adjazenten Datenelemente werden entfernt.

In Proviado steuert ein Parameter *reduceData* mit den möglichen Werten (i) *maintain* (ii) *reduceOrphaned* und (iii) *reduceAll* das Verhalten der Operation (vgl. Tabelle 5.2). Standardmäßig werden nur die verwaisten Datenelemente gelöscht (*reduceOrphaned*).

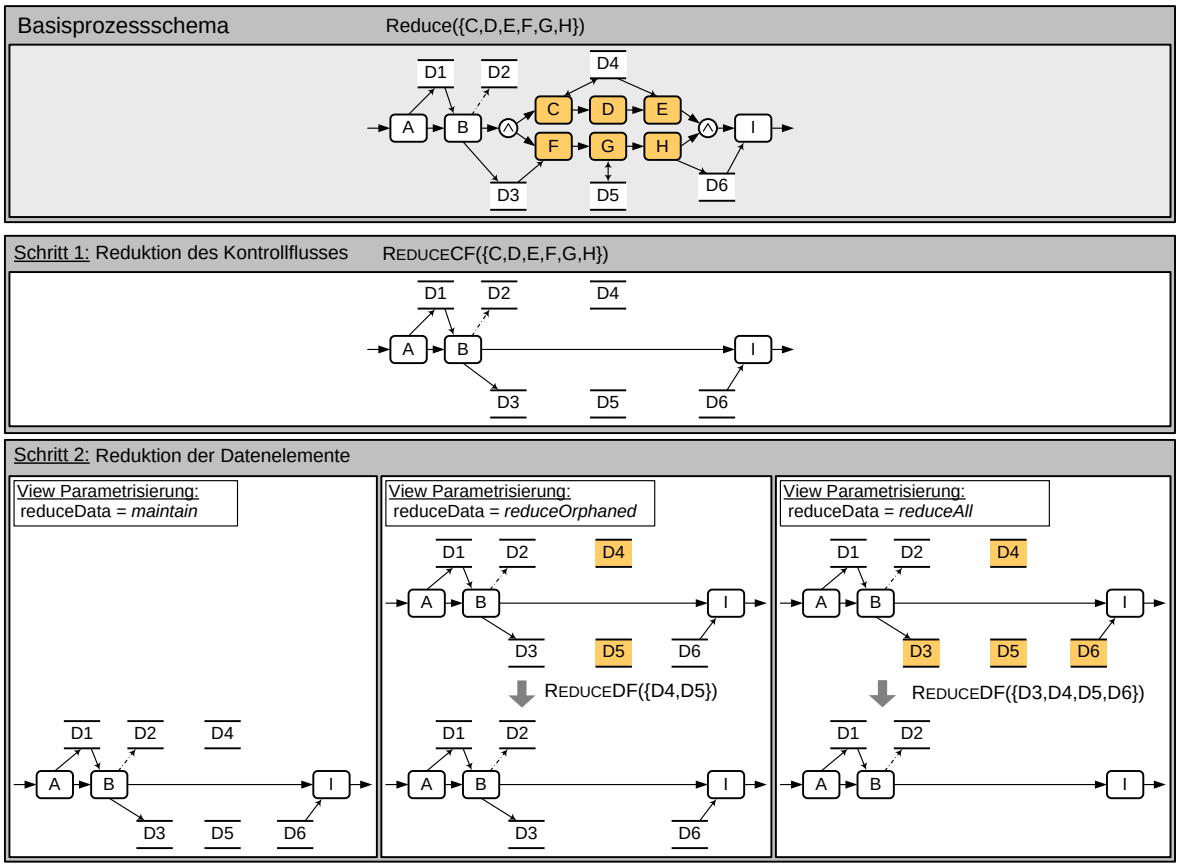


Abbildung 5.14: Reduktion von Kontrollflusselementen und Datenelementen

Parameter	Parameterwerte[1]	Beschreibung
reduceData		Steuerung der Datenflussreduktion:
	maintain,	Erhaltung aller Datenelemente
	reduceOrphaned,	Reduktion von verwaisten Datenelementen
	reduceAll	Reduktion aller adjazenten Datenelemente

[1] default-Werte sind durch fette Schrift gekennzeichnet

Tabelle 5.2: Parameter der Operation Reduce

Beispiel 5-1: Mehraspektoperation Reduce
In Abschnitt 4.6 auf Seite 92 haben wir gezeigt, wie sich das Beispiel 4-3 auf Seite 39 kombiniert mit Beispiel 4-6 mit Hilfe von Elementaroperationen umsetzen lässt. Durch die in diesem Kapitel eingeführte Einzelaspektoperation ReduceCF und die Mehraspektoperation Reduce vereinfacht sich das Beispiel 4-3 wesentlich. Anstelle der vielen einzelnen Elementaroperationen zwecks Reduktion von Aktivitäten und Datenelementen (vgl. Spalte 1 in Tabelle 5.3), genügen nun die in Spalte 2 bzw. 3 von Tabelle 5.3 aufgeführten Anweisungen für Einzel- bzw. Mehraspektoperationen. Der direkte Vergleich macht deutlich, in welchem Maße sich der Aufwand für die Definition einer View (und damit auch die Komplexität) durch Verwendung von Einzel- bzw. Mehraspektoperationen reduzieren lässt.

Realisierung des Beispiel 4-3 mit Elementaroperationen	Einzelaspektoperationen	Mehraspektoperationen

`RedActivity(C)`
`RedActivity(D)`
`RedActivity(L1)`
`RedActivity(L2)`
`RedActivity(L3)`
`RedActivity(Q)`
	ReduceCF($\{C, D, L1, L2, L3, Q\}$)	Reduce($\{C, D, L1, L2, L3, Q\}$, *[reduceData=reduceAll]*)

`RedData(d2)`
`RedData(d3)`
`RedData(d12)`
	ReduceDF($\{d2, d3, d12\}$)	

`RedData(d5)`	`RedData(d5)`	`RedData(d5)`
`RedData(d6)`	`RedData(d6)`	`RedData(d6)`
`RedData(d8)`	`RedData(d8)`	`RedData(d8)`
`RedData(d9)`	`RedData(d9)`	`RedData(d9)`

Tabelle 5.3: Benötigte Operationsaufrufe zur Realisierung von Beispiel 4-3

5.3.2 Aggregation

Die Mehraspektoperation Aggregate kombiniert, wie schon bei der Reduktion gezeigt, die View-Operationen der unterschiedlichen Aspekte. Bevor wir den Ablauf der Operation beschreiben, werfen wir einen Blick auf Anwendungen aus der Praxis und auf das, was sich hinter den Abbildungsfunktionen für Datenelemente verbirgt. Es muss die Frage gestellt werden, was die Aggregation von Datenelementen in Kombination mit Kontrollflusselementen für eine Bedeutung hat. Wir haben es in Geschäftsprozessen häufig mit komplexen, strukturierten Datenelementen zu tun (z.B. Antragsformulare, CAD-Datensätze). Eine Aggregation derartiger Datenstrukturen bietet sich nur in wenigen, ausgewählten Sonderfällen an, etwa wenn mehrere Objekte desselben oder ähnlichen Typs vorliegen, die dann in sinnvoller Art und Weise zusammengefasst werden können. In den meisten Anwendungen ergibt die automatische Aggregation der Datenelemente mittels Aggregate keinen Sinn. Wenn Datenelemente aggregiert werden sollen, wird dies meist durch explizite Operationen erfolgen.

Ein Beispiel für einen Fall, in dem eine automatische Aggregation der Datenelemente vorstellbar ist, ist die Antragsbearbeitung. Sie erfolgt durch verschiedene Abteilungen parallel (vgl. Phase II und III „Begutachtung“ des Änderungsmanagement-Prozesses siehe Seite 12). Bei der Aggregation

der parallelen Arbeitsschritte können die Antragsdokumente (im Prozessmodell durch adjazente Datenelemente repräsentiert) mit aggregiert werden. Hierbei geht die Operation Aggregate in zwei Schritten vor. Im ersten wird der Kontrollfluss durch AggregateCF aggregiert und die Datenflusskanten werden angepasst. Der zweite Schritt aggregiert die Datenelemente.

Der prinzipielle Ablauf ist exemplarisch in Abbildung 5.15 dargestellt. Für die adjazenten Datenelemente der zu aggregierenden Aktivitätenmenge S ergeben sich drei Alternativen. Im Gegensatz zur Reduktion werden bei der Aggregation jedoch keine Kanten zwischen Aktivitäten und Datenelementen gelöscht, sondern durch `AdaptDE` angepasst. Der Parameter *`aggregateData`* steuert den Umgang mit den adjazenten Datenelementen (vgl. Tabelle 5.4).

(i) *`maintain`* behält die Datenelemente bei.

(ii) *`aggregateOrphaned`* aggregiert nur die Datenelemente, die adjazent zu den Aktivitäten aus S sind. Alle anderen Datenelemente bleiben erhalten (der Begriff der verwaisten (orphanded) Datenelemente wird in Analogie zur Reduktion beibehalten).

(iii) *`aggregateAll`* aggregiert alle zu S adjazenten Datenelemente.

Im Standardfall werden die Datenelemente beibehalten (d.h. *`aggregateData = maintain`*).

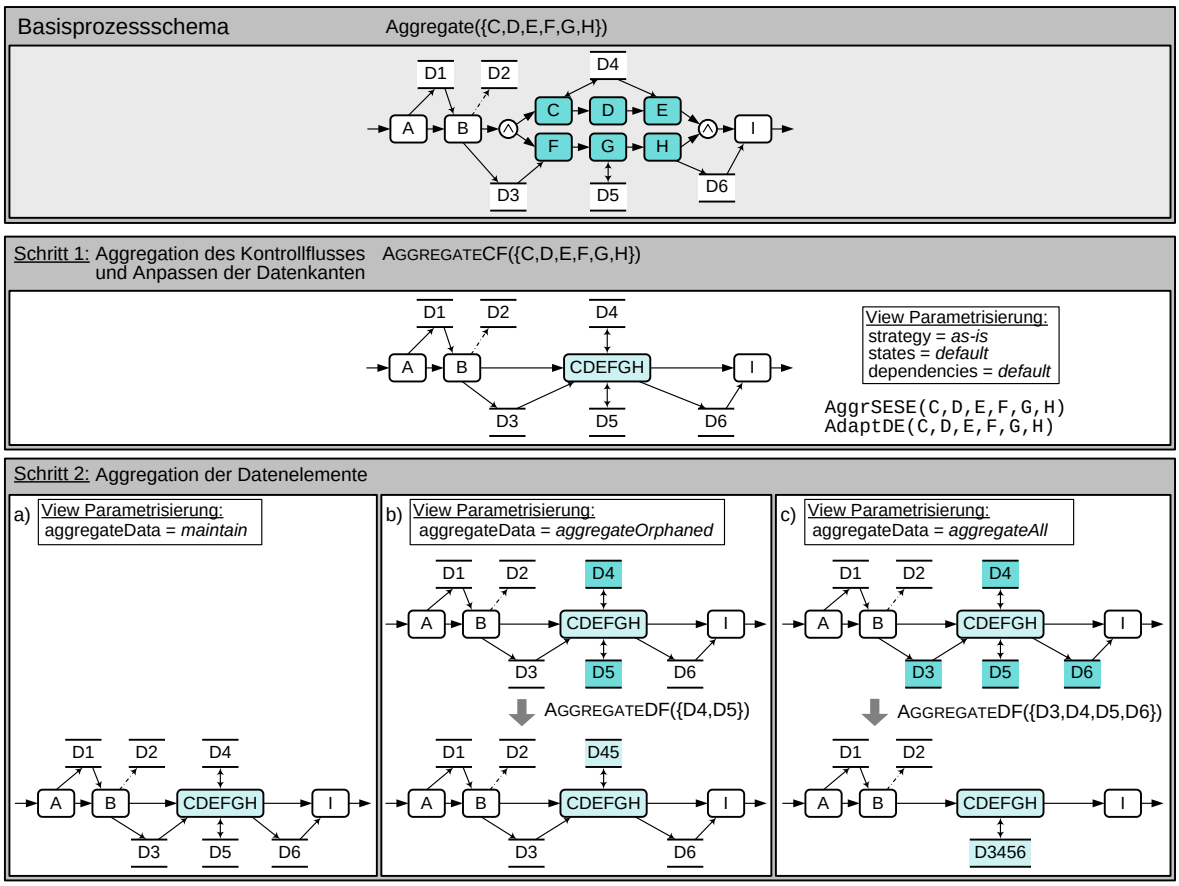


Abbildung 5.15: Zweistufiges Vorgehen bei Aggregation von Kontrollfluss- und Datenelementen

Zur Steuerung der Kontrollflussaggregation erlaubt die Operation Aggregate dieselben Parameter, wie die Operation AggregateCF (vgl. Tabelle 5.1). Diese werden beim Aufruf der Kontrollflussoperationen weitergeleitet.

Parameter	Parameterwerte[1]	Beschreibung
Kontrollfluss		
Parameter wie bei der Einzelaspektoperation AggregateCF *(vgl. Tabelle 5.1)*		
Datenfluss		
`aggregateData`		Steuerung der Datenflussaggregation:
	maintain,	Erhaltung aller Datenelemente
	aggregateOrphaned,	Aggregation der Datenelemente, die nur mit aggregierten Aktivitäten in Verbindung stehen
	aggregateAll	Aggregation aller Datenelemente

[1] default-Werte sind durch fette Schrift gekennzeichnet

Tabelle 5.4: Parameter der Operation Aggregate

Betrachtet man die kombinierte Aggregation von Daten- und Kontrollfluss im Detail, zeigt sich, dass die Reihenfolge, in der die Aspekte verarbeitet werden, Einfluss auf das Ergebnis der View-Bildung hat. Abbildung 5.16 zeigt ein Beispiel, in dem das Vertauschen der Ausführungsreihenfolge von Kontroll- und Datenflussoperationen zu unterschiedlichen Datenkanten ((a) optional, (b) obligat) führt. Die in diesem Abschnitt vorgestellte Operation realisiert Variante (a), d.h. zuerst wird der Kontrollfluss und danach der Datenfluss aggregiert. Zu der Frage, welche der beiden in Abbildung 5.16 dargestellten Alternativen die bessere Lösung darstellt, existieren unterschiedliche Ansichten. Zum einen kann man argumentieren, dass für jede mögliche Ausführung entweder A oder B ausgeführt und somit $D1$ oder $D2$ geschrieben werden. Daher wäre eine obligate Datenflusskante zu dem aggregierten Datenelement gerechtfertigt, was für Variante (b) sprechen würde. Zum anderen wird aber, egal welche der beiden Aktivitäten letztlich ausgeführt wird, immer nur entweder $D1$ oder $D2$ geschrieben und somit niemals $D12$ vollständig. Bevorzugt man Variante (b), bleibt es dem Benutzer freigestellt, die entsprechenden (wenigen) Datenelementaggregationen vor dem Aufruf von Aggregate explizit anzustoßen.

Beispiel 5-2: Mehraspektoperation Aggregate
Wir betrachten nun wieder Beispiel 4-3 kombiniert mit Beispiel 4-6. Beide Beispiele hatten wir in Abschnitt 4.6 mit Hilfe von Elementaroperationen umgesetzt. Für die Reduktion ergab sich für dieses Beispiel eine beachtliche Reduzierung der Anzahl der notwendigen Operationen. Bei der Aggregation im Beispiel 4-6 kann die Anzahl der Operationen nicht weiter minimiert werden (vgl. Tabelle 5.5). Der Vorteil der Einzel- bzw. Mehraspektoperationen liegt aber darin, dass die Komplexität für die Definition der View reduziert wird, da keine Detailkenntnisse zur Menge der verfügbaren Elementaroperationen erforderlich sind. Vielmehr werden die Elementaroperationen auf Grundlage der Struktur der Aktivitäten im Prozess und der Parameter automatisch ausgewählt. Für das hier gezeigte Beispiel verwenden wir die Standardwerte der Parameter (vgl. Tabelle 5.4), weshalb diese in den Operationsdefinitionen in Tabelle 5.5 nicht vorhanden sind. In Abschnitt 5.4 werden wir höherwertige Operationen einführen, die in der Lage sind, die View-Definition für Beispiel 4-3 weiter zu vereinfachen.

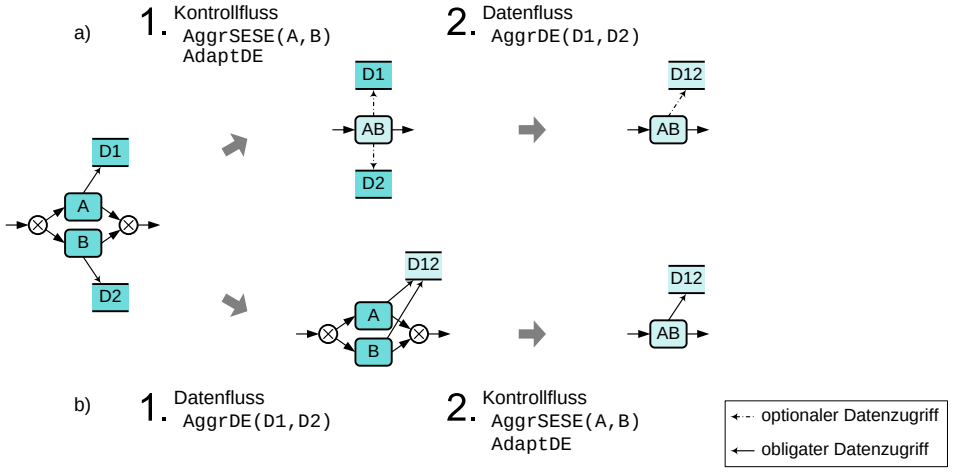


Abbildung 5.16: Einfluss der Reihenfolge der Operationen bei der Aggregation von Kontrollfluss- und Datenelementen

Realisierung des Beispiels 4-6 mit Elementaroperationen	Einzelaspektoperationen	Mehraspektoperationen
`AggrAddBranch`$(\{F1, H1, J1, M1, N1, O1\})$	AggregateCF$(\{F1, H1, J1, M1, N1, O1\}, [])$	Aggregate$(\{F1, H1, J1, M1, N1, O1\}, [])$
`AggrAddBranch`$(\{F2, H2, J2, M2, N2, O2\})$	AggregateCF$(\{F2, H2, J2, M2, N2, O2\}, [])$	Aggregate$(\{F2, H2, J2, M2, N2, O2\}, [])$
`AggrAddBranch`$(\{F3, H3, I1, M3, N3, O3\})$	AggregateCF$(\{F3, H3, I1, M3, N3, O3\}, [])$	Aggregate$(\{F3, H3, I1, M3, N3, O3\}, [])$

Tabelle 5.5: Benötigte Operationsaufrufe zur Realisierung von Beispiel 4-6

5.4 Höherwertige Operationen

5.4.1 Vor- und Nachbehandlung für die View-Bildung

Die aus einer View-Bildung resultierenden Prozessmodelle stellen immer einen strukturell korrekten Prozess im Sinne der in Abschnitt 4.2.1 aufgestellten Korrektheitskriterien dar. Für bestimmte Anwendungen sind jedoch Prozessvisualisierungen gefragt, die diesen Anforderungen nicht genügen. Wir illustrieren diesen Aspekt anhand eines einfachen Beispiels. Wir haben bereits das Szenario motiviert, bei dem sich ein Prozessbeteiligter einen Überblick über den aktuellen Zustand eines Prozesses sowie die nächsten anstehenden Aufgaben verschaffen will. Bei großen und komplexen Prozessen sind dabei die Aktivitäten, die in ferner Zukunft liegen, nicht relevant. Hieraus könnte zum Beispiel die Forderung resultieren, dass die Aktivitäten, die mehr als drei Schritte in der Zukunft liegen, „abgeschnitten“ werden sollen. Damit ist allerdings nicht das „Abschneiden“ mittels Reduktion gemeint. Diese ist zwar in der Lage, die betroffenen Aktivitäten zu entfernen, die Kontrollflusskanten und Strukturknoten bleiben aber erhalten. Für die vorliegende Anwendung wird jedoch gewünscht, dass der Prozess jenseits des genannten „Horizonts“ komplett abgeschnitten wird (vgl. Abbildung 5.17). Genauer gesagt sollen alle folgenden Aktivitäten und Kanten gelöscht und damit einhergehend gegebenenfalls offene Zweige

nicht mehr zusammen geführt werden. Auf der einen Seite verletzt eine solche Vorgehensweise zwar die Struktur unseres Prozesses, auf der anderen Seite führt es zu sehr übersichtlichen Darstellungen. Ist dies gefordert, so ist auch eine entsprechende Operation notwendig.

Der Proviado-View-Mechanismus unterstützt solche strukturverletzende Operationen im Kern nicht. Um dennoch Visualisierungen der oben skizzierten Art realisieren zu können, bietet er primitive Manipulationsoperationen an, mit denen die geforderte Darstellung erreicht werden kann. Mittels dieser primitiven Operationen ist es möglich,

- einzelne Knoten explizit zu löschen
- einzelne Kanten explizit zu löschen
- neue Knoten und Kanten beliebigen Typs (Kontrollfluss, Datenfluss, etc.) zu erzeugen
- Knoten- und Kantenattribute zu manipulieren (Hinzufügen, Löschen, Ändern)

Die Operationen können als Vor- oder Nachbehandlung in Kombination mit den View-Bildungsoperationen eingesetzt werden. Während einer Vorbehandlung lassen sich nicht modellkonforme Prozesse mittels der Manipulationsoperationen in ein konformes Modell umwandeln, um anschließend die bekannten View-Operationen darauf auszuführen. Abbildung 5.17 zeigt wie beispielsweise im Anschluss an eine View-Bildung im Rahmen der Nachbehandlung Knoten und Kanten gelöscht werden können, um das oben motivierte Szenario zu ermöglichen.

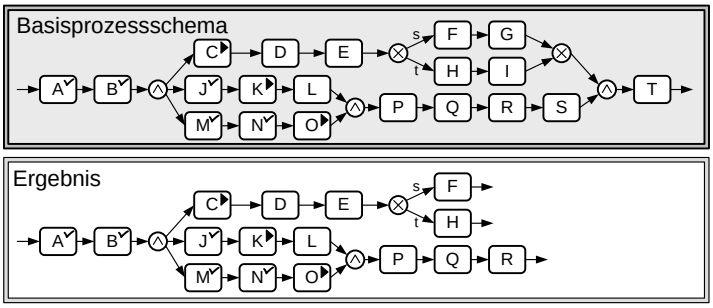


Abbildung 5.17: Nachbehandlung eines Prozesses mittels Manipulationsoperationen

5.4.2 Höherwertige View-Operationen

Mit den beschriebenen Mehraspektoperationen (siehe Abschnitt 5.3) lassen sich, in Kombination mit den View-Operationen der unteren Schichten (Einzelaspekt- und Elementaroperationen) beliebige Views bilden. Generell gibt es auch Views, die in praktischen Anwendungen besonders häufig auftreten. In solchen Fällen kann es lohnend sein, weitere spezifische Operationen anzubieten, die je einen speziellen Anwendungsfall abdecken. Unsere Anforderungsanalysen haben zum Beispiel ergeben, dass häufig eine View auf einen Prozess benötigt wird, bei der alle „technischen“ Aktivitäten (z.B. Datenbankzugriffe oder Datentransformationsschritte) ausgeblendet werden (vgl. Beispiel 4-3). Ebenso gefragt ist in der Praxis eine View auf eine Prozessinstanz, die alle abgearbeiteten (d.h. beendeten) Schritte zusammenfasst. Um diese und ähnliche Anforderungen abzudecken (vgl. Anforderung 4-9), ermöglicht Proviado in einer zusätzlichen Schicht die Definition derartiger Operationen. Diese sind nicht ausschließlich auf

Aggregation oder Reduktion beschränkt, sondern kombinieren beliebige Aggregations- und Reduktionsoperationen der niedrigeren Schichten miteinander. Die im vorangehenden Abschnitt beschriebenen Manipulationsoperationen (z.B. Löschen von Knoten und Kanten) werden in unserem Ansatz in keiner weiteren Schicht zusammengefasst, sondern können als Vor- bzw. Nachbehandlung bei der Definition von höherwertigen Operationen mit verwendet werden. Abbildung 5.18 zeigt das Gesamtschichtenmodell des Proviado-View-Bildungsansatzes.

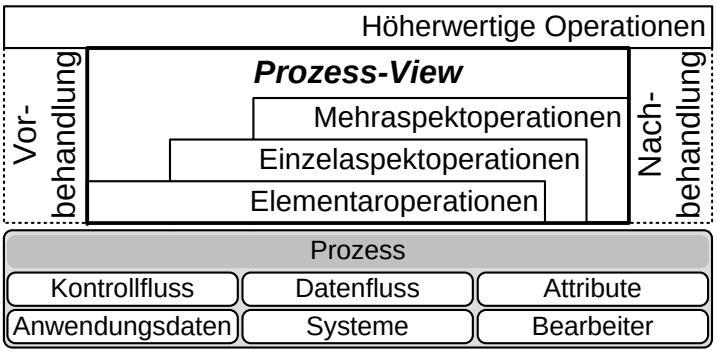


Abbildung 5.18: Erweitertes Schichtenmodell der Proviado-Views

Im Folgenden präsentieren wir verschiedene höherwertige View-Operationen, die in Proviado zur Verfügung stehen. Ausgehend von der Grundmenge der in dieser Arbeit vorgestellten Operationen können weitere Operationen aller Schichten definiert werden, um spezielle Anforderungen einer Anwendung abzudecken. Eine solche Erweiterung sind beispielsweise die oben bereits erwähnten Operationen, die ausgehend von selektierten Datenelementen alle adjazenten Aktivitäten aggregieren.

Im Zusammenhang mit der flexiblen Definition von Prozess-Views wird häufig eine Funktion benötigt, die eine Menge relevanter Prozesselemente mit speziellen Eigenschaften bestimmt. Proviado stellt hierfür die Hilfsfunktion `Select` zur Verfügung. Ähnlich zu Objekt-Anfrage-Sprachen (Object-Query-Languages) erlaubt sie eine prädikative Beschreibung einer Menge von Prozesselementen. Die Funktion kann mit beliebigen View-Operationen kombiniert werden, um flexible Views abhängig von beliebigen Eigenschaften zu definieren.

Tabelle 5.6 gibt einen Überblick über höherwertige View-Bildungsoperationen in Proviado.

Operation	**Beschreibung**
ShowActivitiesOfUser	Aktivitäten, die nicht durch einen bestimmten Benutzer (bzw. dessen Rolle) ausgeführt werden bzw. worden sind, werden reduziert.
AggrExecutedPart	Alle bereits ausgeführten oder übersprungenen Aktivitäten werden zu einer abstrakten Aktivität aggregiert.
ShowExecutedPath	Aktivitäten, die nicht ausgeführt worden sind (d.h. abgewählte Zweige in (X)OR-Verzweigungen) werden aus dem Prozess mittels Reduktion entfernt. Es verbleiben nur abgeschlossene und zukünftige Aktivitäten.
GroupedAggregation	Aktivitäten werden nach einem festgelegten Kriterium gruppiert (z.B. Unternehmenszugehörigkeit des Bearbeiters). Anschließend werden die Gruppen einzeln aggregiert.
ViewByRelevance	Abhängig von einer Relevanzfunktion werden Aktivitäten selektiert und aggregiert bzw. reduziert (siehe [SPB05] oder [SL03] für ähnliche Funktionen)

ViewByPredicate	Aktivitäten mit einer bestimmten Eigenschaft werden mittels eines Prädikats unter Zuhilfenahme der Funktion `Select` selektiert und anschließend abhängig von einem Parameter reduziert oder aggregiert.
Subgraph[1]	Ausgehend von einer selektierten Aktivitätenmenge wird der durch die Menge induzierte Subgraph gebildet (mittels Reduktion des Komplements und weiteren Nachbehandlungsoperationen). Dies entspricht einem SELECT in relationalen Datenbanken und ist damit quasi eine Negation der Reduktion.
SubgraphRange[1]	Analog zur Operation *Subgraph* werden alle Aktivitäten reduziert, die nicht zwischen zwei vorgegebenen Knotenmengen liegen.
CutProcess[1]	Der Prozess wird in einem definierten Abstand nach einer Aktivitätenmenge abgeschnitten. Durch Parametrisierung kann der Abstand und die Richtung (*in* oder *gegen* die Ablaufrichtung des Kontrollflusses) spezifiziert werden.

[1] Die Operation beinhaltet eine Nachbehandlung mittels der Manipulationsoperationen aus Abschnitt 5.4.1

Tabelle 5.6: Übersicht über die verfügbaren höherwertigen Operationen

Alle höherwertigen Operationen werden auf Basisoperationen abgebildet, wodurch ihre Semantik festgelegt ist. Wir verzichten hier auf eine formale Darstellung der Operationen und erläutern deren Prinzip anhand ausgewählter Beispiele.

Beispiel 5-3: Reduktion mittels Prädikat durch eine höherwertige Operation
Abbildung 5.19 zeigt ein Beispiel einer Prozess-View basierend auf einer höherwertigen Operation, die intern ein Prädikat verwendet. Die View soll ein vereinfachtes Prozessmodell generieren, das nur diejenigen Aktivitäten enthält, an denen ein bestimmter Benutzer beteiligt ist. Zu diesem Zweck erzeugt die höherwertige Operation *ShowActivitiesOfUser* ein Prädikat für einen bestimmten Benutzernamen. Dieses Prädikat identifiziert alle Aktivitäten, die ohne Beteiligung des Benutzers ablaufen. D.h. es erfolgt eine Negation des ursprünglichen Ausdrucks (vgl. Schritt 1 in Abbildung 5.19). Zur Auswertung des Prädikats wird die oben beschriebene Hilfsoperation `Select` verwendet. Schritt 2 ruft dann die Mehraspektoperation Reduce auf, welche die Reduktion wiederum an die Einzelaspektoperation ReduceCF delegiert. Daraufhin werden die selektierten Aktivitäten mittels der Elementaroperation `RedActivity` aus dem Prozessmodell entfernt. Durch die abschließende Vereinfachung des Prozessmodells in Schritt 3 (SimplifyCF) resultiert der dargestellte Prozess.

Beispiel 5-4: Aggregation abgeschlossener Aktivitäten mittels einer höherwertigen Operation
In Beispiel 4-4 wird ein beliebtes Anwendungsbeispiel für Prozess-Views angesprochen: alle bereits abgeschlossenen Aktivitäten sollen zu einer abstrakten Aktivität aggregiert werden, damit der Fokus auf die relevanten, anstehenden Aufgaben gelegt werden kann. Die höherwertige Operation *AggrExecutedPart* realisiert genau diese Funktionalität. Wie in Abbildung 5.20 dargestellt, wird zunächst die Menge der abgeschlossenen oder ausgelassenen Aktivitäten ($NS = Completed \vee Skipped$) mittels eines Prädikats und `Select` bestimmt (Schritt 1). Diese werden dann mit der Operation Aggregate aggregiert (Schritt 2). Da in dem Beispiel kein Datenfluss vorhanden ist, wird der Aufruf direkt an AggregateCF weitergeleitet (Schritt 3). Diese Operation wählt nun die passende Elementaroperation (Schritt 4) für die gegebene Parametrisierung aus. Intern werden dabei die Parameter `strategy=as-is, states=default` und `dependencies=default` verwendet. Dies führt dazu, dass die Einzelaspektoperation AggregateCF versucht, den selektierten Bereich als Ganzes zu einer neuen

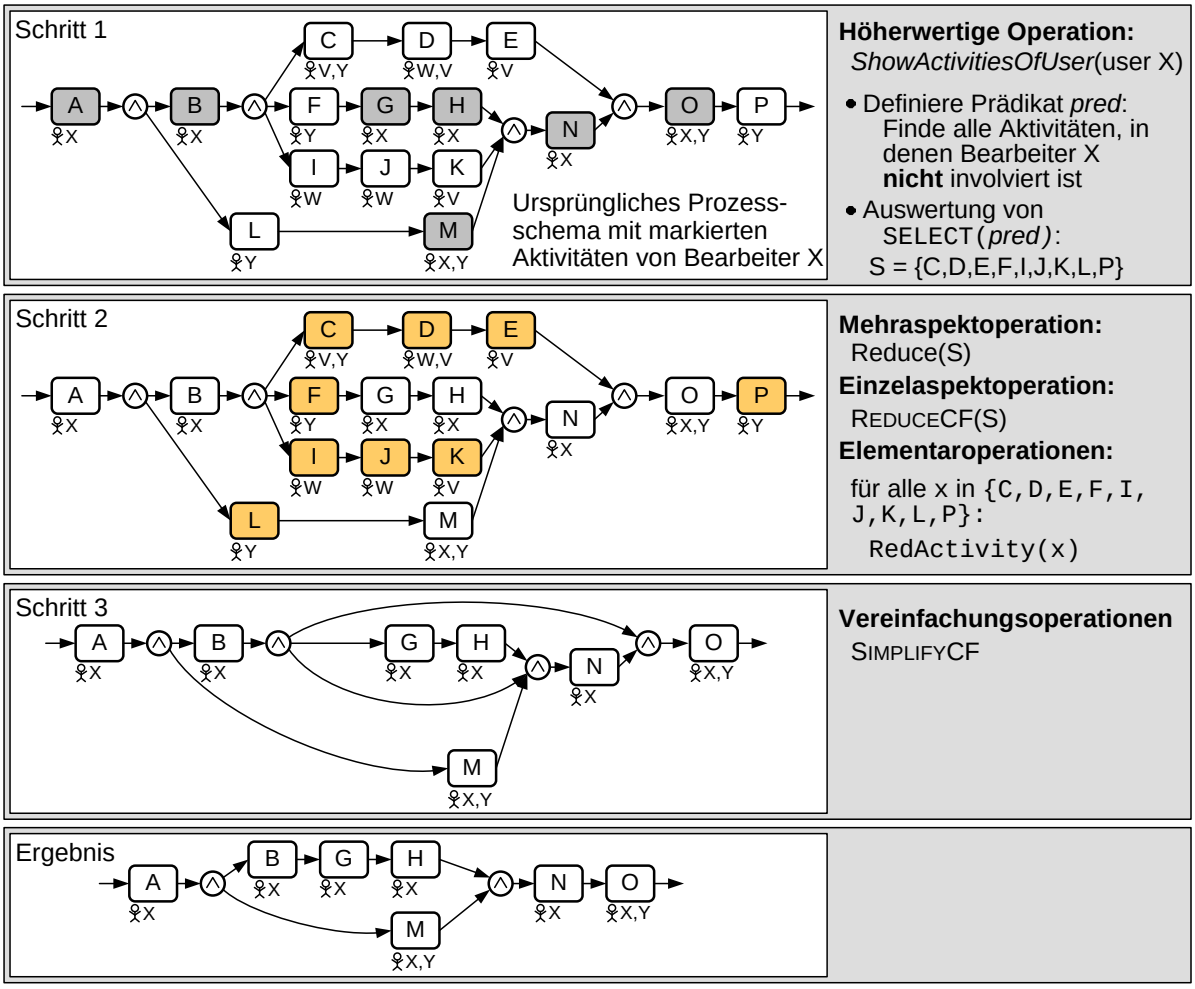


Abbildung 5.19: *ShowActivitiesOfUser*

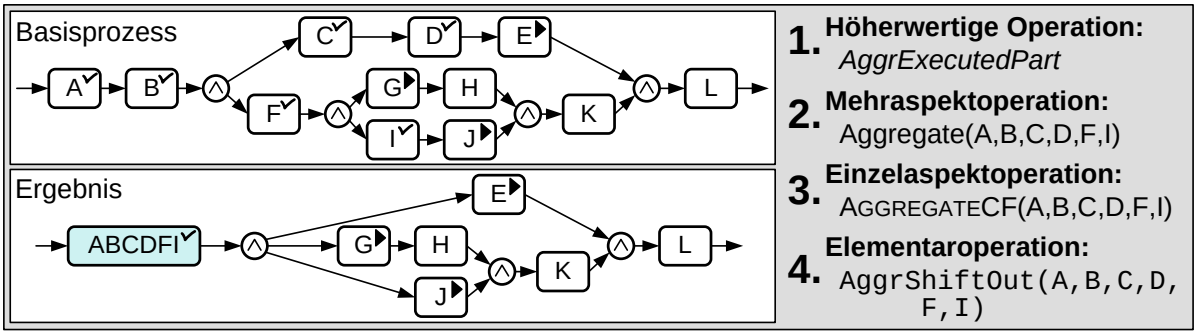


Abbildung 5.20: *AggrExecutedPart*

abstrakten Aktivität zu aggregieren. Da alle bereits ausgeführten Schritte immer am „Anfang“ des Prozesses liegen, bilden diese einen Verzweigungsbaum mit Stamm (vgl. Definition 4.5). Somit wird für die Aggregation die Elementaroperation `AggrShiftOut` verwendet, was zu dem dargestellten Ergebnis führt.

Beispiel 5-5: Höherwertige Operation mit Nachbehandlung
Proviado erlaubt auch die Definition höherwertiger Operationen, die eine Vor- bzw. Nachbehandlung beinhalten. Ein Beispiel ist die Operation *SubgraphRange*. Diese Operation dient dazu, einen Bereich aus einem großen Prozessmodell herauszulösen, indem alle umgebenden Prozesselemente entfernt werden. Zur Illustration dient Abbildung 5.21. Zunächst werden zwei Aktivitätenmengen A und B definiert, welche die relevanten Aktivitäten des Prozesses nach vorne (A) bzw. hinten (B) im Prozess abgrenzen. Ausgehend von diesen Aktivitäten bestimmt die Operation *SubgraphRange* die Menge S aller Aktivitäten, die nicht zwischen A und B im Kontrollfluss liegen (Schritt 1). Schritt 2 entfernt alle Aktivitäten in S mittels einer Reduktion. In Schritt 3 werden anschließend die umgebenden Kontrollflussknoten (d.h. Strukturknoten) und -kanten gezielt mittels der Nachbehandlungsoperationen `DeleteNode` bzw. `DeleteEdge` gelöscht.

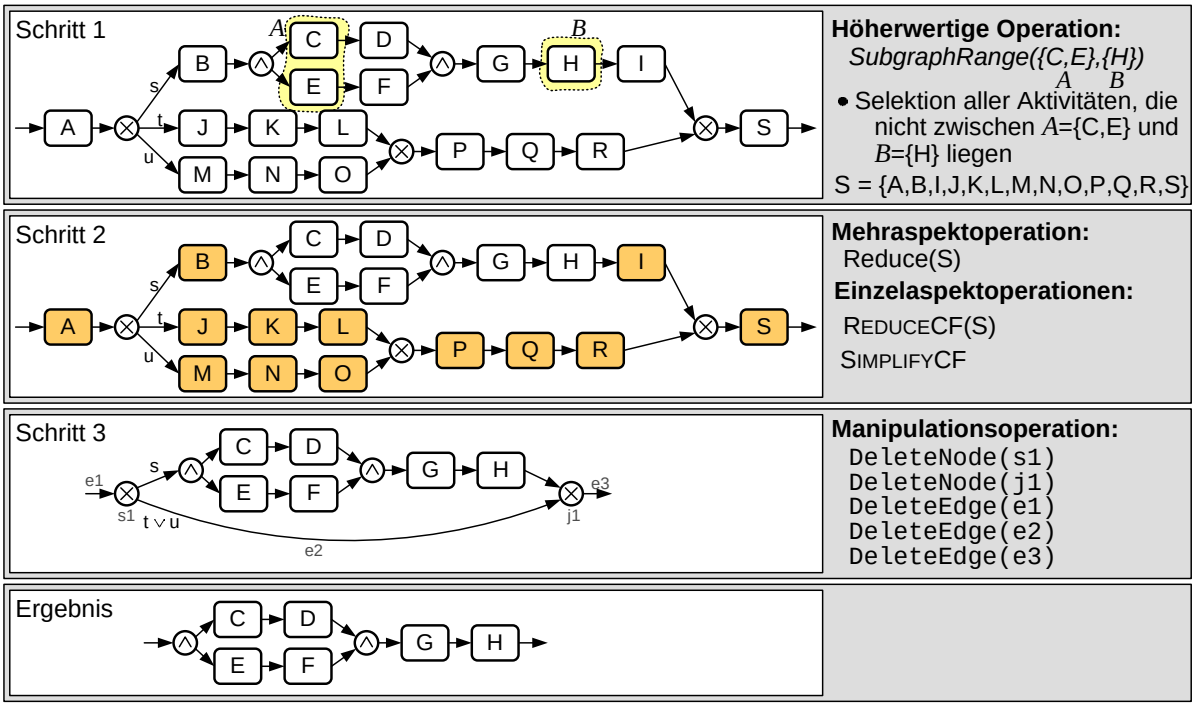


Abbildung 5.21: Ablauf der Operation *SubgraphRange* mit Nachbehandlung

Beispiel 5-6: View-Bildung durch eine Kombination verschiedener Operationen
Im Laufe dieser Arbeit haben wir mehrfach die Beispiele 4-3 und 4-6 zu Illustrationszwecken herangezogen. In dem Beispiel werden sowohl Reduktion als auch Aggregation verwendet. In Beispiel 4-8 auf Seite 94 hatten wir gezeigt, wie dieses Beispiel mit Hilfe von Elementaroperationen realisiert werden kann, jedoch auch gesehen, dass dies relativ viele Operationsaufrufe erfordert (vgl. linke Spalte in Tabelle 5.7). Später haben wie dasselbe Beispiel mit Hilfe von Einzel- und Mehraspektoperationen realisiert (vgl. Beispiel 5-1 auf Seite 118 bzw. Beispiel 5-2 auf Seite 120). Nachdem nun der volle Umfang des Proviado-View-Mechanismus vorgestellt worden ist, wird der Unterschied zwischen der Definition einer View mittels reiner Elementaroperationen und der Definition unter Verwendung von Operationen der unterschiedlichen, in diesem Kapitel eingeführten Schichten erkennbar (vgl. Tabelle 5.7). Das Ergebnis ist eine deutlich reduzierte Anzahl von Operationsaufrufen.

Elementaroperationen	Höherwertige Operationen
`AggrAddBranch`($\{F1, H1, J1, M1, N1, O1\}$) `AggrAddBranch`($\{F2, H2, J2, M2, N2, O2\}$) `AggrAddBranch`($\{F3, H3, I1, M3, N3, O3\}$)	*GroupedAggregation([groupby='BU' where 'BU != OEM'])*

`RedActivity(C)`
`RedActivity(D)`
`RedActivity(L1)`
`RedActivity(L2)`
`RedActivity(L3)`
`RedActivity(Q)`
`RedData(d2)`
`RedData(d3)`
`RedData(d12)`
`RedData(d5)`
`RedData(d6)`
`RedData(d8)`
`RedData(d9)`
	Reduce($\{C, D, L1, L2, L3, Q\}$, *[reduceData=reduceAll]*)
	`RedData(d5)`
	`RedData(d6)`
	`RedData(d8)`
	`RedData(d9)`

Tabelle 5.7: Gegenüberstellung benötigter Operationsaufrufe zur Realisierung von Beispiel 5-6

5.5 View-Definitionssprache

Die View-Bildung ist ein vorbereitender Schritt der Prozessvisualisierung, um die Komplexität der Modelle zu reduzieren und ihre Darstellung an die Bedürfnisse des Benutzers anzupassen. Die View-Definition stellt somit einen Teil der in Kapitel 3 skizzierten Darstellungsbeschreibung dar. Bislang ist jedoch eine View nur eine Folge von Operationen, die über eine Programmierschnittstelle angesprochen werden kann. Im Allgemeinen benötigen wir auch eine Syntax für die View-Beschreibung, d.h. eine View-Definitionssprache (VDL). Eine VDL beschreibt im Prinzip, welche Operationen bei der View-Bildung ausgeführt werden sollen. Dabei müssen verschiedenartige Parameter an die View-Operationen übergeben werden. Die folgende Liste führt die benötigten Informationen am Beispiel einer Aggregationsoperation auf:

- Betroffene Knotenmenge (explizit oder über ein Prädikat mittels der Hilfsfunktion `Select`)
- Parameter für die View-Bildung
- Transformationsfunktionen für alle Attribute
- Namen für aggregierte Aktivitäten
- Operationen für die Datenelemente (Aggregation/Reduktion)

Abbildung 5.22 zeigt eine Beispielsyntax für die Definition einer View. Die erste Operation (Zeilen 2–7) stellt eine Reduktion dar. Die Menge der zu reduzierenden Aktivitäten ist hier explizit deklariert (Zeile 6). Mittels eines Parameters ist festgelegt, dass verwaiste Datenelemente gelöscht werden sollen. Die zweite Operation (Zeilen 9–18) in dieser View ist ebenfalls eine Reduktion, deren Knotenmenge nun aber durch ein Prädikat spezifiziert wird. Hierfür verwenden wir eine Syntax, die an existierende Objektanfragesprachen angelehnt ist (vgl. Object Query Language [CBB+00], Hibernate-QL [BK06]), und die alle technischen Aktivitäten auswählt,

die später entfernt werden sollen. Die dritte Operation ist eine Aggregation (Zeilen 20–37). Zusätzlich zu den Angaben, die für eine Reduktion benötigt werden, muss hier spezifiziert werden, wie die resultierende abstrakte Aktivität benannt werden soll und wie die Attribute der ursprünglichen Aktivitäten auf die abstrakte Aktivität übertragen werden sollen.

```
1  <view>
2     <reduce opId="red1">
3        <parameterSet>
4           <parameter name="reduceData" value="reduceOrphaned" />
5        </parameterSet>
6        <nodeSet>A,B,P,Q,R</nodeSet>
7     </reduce>
8
9     <reduce opId="red2">
10       <parameterSet>
11          <parameter name="reduceData" value="reduceAll" />
12       </parameterSet>
13       <nodeSet>
14          <Select>
15             SELECT DISTINCT a FROM a in Activity WHERE a.type = 'technical'
16          </Select>
17       </nodeSet>
18    </reduce>
19
20    <aggregate opId="agg1">
21       <parameterSet>
22          <parameter name="aggregateData" value="maintain" />
23          <parameter name="strategy" value="as-is" />
24          <parameter name="depdency" value="default" />
25          <parameter name="states" value="consistent" />
26       </parameterSet>
27       <nodeSet>C,D,E,F,G,H</nodeSet>
28       <newNode nodeID="CDEFGH" type="Activity">
29          <name>parallele Begutachtung</name>
30       </newNode>
31       <transformAttr sourceAttr="starttime" destAttr="starttime" function="MIN" />
32       <transformAttr sourceAttr="endtime" destAttr="endtime" function="MAX" />
33       <transformAttr sourceAttr="costs" destAttr="costs">
34          <function class="de.proviado.aggregation.CostAggrFunc" />
35       </transformAttr>
36    </aggregate>
37 </view>
```

Abbildung 5.22: Beispiel für die Definition einer View

Schon anhand dieses einfachen Beispiels wird deutlich, dass für eine Aggregation wesentlich mehr Angaben erforderlich sind als für eine Reduktion. Fast vollständig unberücksichtigt sind in dem Beispiel Datenelemente. Durch zusätzliche Angaben für Aggregation und Transformationen von Datenelementen wächst die View-Definition weiter. Um die Definition einer View zu vereinfachen, sollte in einem nächsten Schritt ein graphisches Werkzeug angeboten werden, mit dessen Hilfe die Views zusammengestellt werden können. Die entsprechende View-Definition würde dann im Hintergrund erzeugt und könnte bei Bedarf manuell angepasst werden.

5.6 Implementierungsaspekte

Dieser Abschnitt diskutiert ausgewählte Aspekte, die es bei der Implementierung des Proviado-View-Mechanismus zu beachten gilt. Insbesondere werden verschiedene Implementierungsalternativen für die interne Repräsentation einer View angesprochen und Optimierungspotentiale aufgezeigt.

5.6.1 Interne Repräsentation von Views

Die Bildung einer View erfordert komplexe Berechnungen auf Graphen. Ein Ziel der Implementierung des Proviado-View-Mechanismus war es, die Berechnungen (d.h. die Zwischenergebnisse der View-Bildung) wieder zu verwenden, um so die Effizienz des Verfahrens zu erhöhen. Wir gehen von dem in Kapitel 3 beschriebenen Szenario aus, bei dem die Geschäftsprozessmodelle aus den Quellsystemen extrahiert und auf unser Prozessmetamodell abgebildet werden. Diese Modelle werden anschließend mittels einer großen Anzahl elementarer Reduktions- und Aggregationsoperationen aufbereitet. Die höherwertigen Operationen werden dazu, wie in diesem Kapitel beschrieben, in Elementaroperationen aufgelöst. Der resultierende Graph bzw. das Prozessmodell bildet die Grundlage für die anschließende Visualisierung. Für die Realisierung der View-Operationen existieren unterschiedliche Möglichkeiten, die im Folgenden kurz vorgestellt und diskutiert werden.

Variante 1: Eine Realisierungsvariante für die vorgestellten View-Operationen implementiert diese direkt auf Grundlage der Prozessmodelle mit Hilfe von Änderungsoperationen. Dazu müssen die Elementaroperationen auf entsprechende Einfüge- und Löschoperationen abgebildet werden. Diese Lösung hat den Nachteil, dass der ursprüngliche Prozess modifiziert und somit verloren geht. Dem kann durch zusätzliches Erstellen einer Kopie begegnet werden. Die View wird dann auf der Kopie des ursprünglichen Prozesses gebildet. In Abbildung 5.23 ist diese Realisierungsvariante anhand von zwei Operationen (`AggrSESE`, `RedActivity`) illustriert, ist aber gleichermaßen auf alle anderen Elementaroperationen anwendbar.

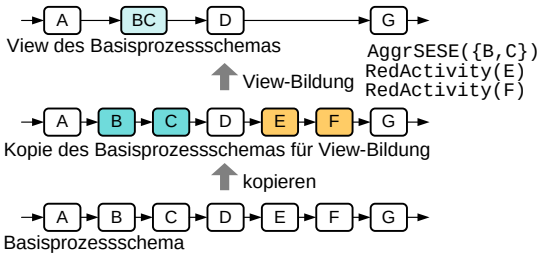


Abbildung 5.23: Realisierung der View-Bildung als Änderungsoperationen auf einer Kopie

Der Nachteil Realisierungsvariante 1 besteht darin, dass die Verbindung zwischen Originalprozess und View verloren geht. Das bedeutet für die View, dass nicht nachvollziehbar ist, aus welchen Daten zum Beispiel ein Attributwert entstanden ist. Soll eine Prozessvisualisierung nach Änderung eines Attributwertes (z.B. Zustand einer Aktivität) aktualisiert werden, muss

die gesamte View neu berechnet werden. Mit dem folgenden Ansatz lässt sich dies in vielen Fällen vermeiden.

Variante 2: Die optimierte Realisierungsvariante verwendet ein Ebenenmodell, wie es in Abbildung 5.24 dargestellt ist (nicht zu verwechseln mit dem Schichtenmodell der View-Operationen aus Abbildung 5.18). Eine View-Operation legt dazu zunächst eine Kopie des Basisprozesses an, in der jedes Objekt mit dem Original verknüpft ist. Auf dieser verknüpften Kopie arbeiten anschließend die View-Operationen. Dadurch bleibt die Verbindung zum Basisprozess gewahrt. Für jede einzelne View-Operation wird in diesem Ansatz eine neue Ebene erzeugt. Für eine Aggregationsoperation enthält der aggregierte Knoten Referenzen auf alle durch ihn aggregierte Knoten der darunter liegenden Ebene. Demgegenüber werden bei der Reduktion keine Referenzen hinterlegt. Dies ist auch nicht erforderlich, da im Falle einer Aktualisierung der Darstellung die reduzierten Daten keine Rolle mehr spielen und deshalb nicht aktualisiert werden müssen.

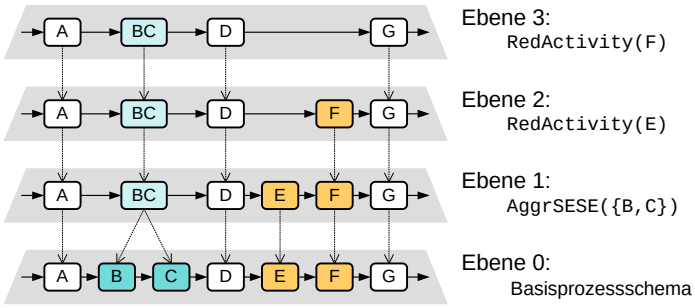


Abbildung 5.24: Implementierung einer View mittels verknüpfter Ebenen

Der Hauptvorteil dieser Realisierungsvariante liegt darin, dass bei statischen Views, die nicht von Status- oder Anwendungsdaten abhängen (vgl. Abschnitt 4.2), die View bei Änderung eines Attributwertes nicht neu berechnet werden muss. Der Weg der Daten ist durch die Referenzierung nachvollziehbar und somit kann der neue (aggregierte) Attributwert entlang des Weges durch die Ebenen neu berechnet werden.

Durch dieses Ebenenmodell ergeben sich weitere Optimierungsmöglichkeiten. Auf einige davon gehen wir im folgenden Abschnitt ein. Bei der Proof-of-Concept Implementierung der Proviado-Views (siehe Kapitel 8) kommt der beschriebene Ebenenansatz zum Einsatz [EH07].

5.6.2 Weitere Optimierungsmöglichkeiten für die View-Berechnung

Durch die interne Repräsentation von Views in mehreren Ebenen eröffnen sich verschiedene Wege, die View-Berechnung effizient zu gestalten. Wie in Abschnitt 5.6.1 erwähnt, können Prozessattribute aktualisiert werden, ohne die vollständige View erneut berechnen zu müssen. Hier werden nun zusätzliche Optimierungspotentiale aufgezeigt, die bei Bedarf weiter untersucht werden können.

- Bei Updates von dynamischen Views (d.h. Views, die von Applikationsdaten abhängig sind; vgl. Abschnitt 4.2) kann analysiert werden, welche Ebenen von der Änderung betroffen

sind. Eine Neuberechnung muss dann nur für diese und die darüber liegenden Ebenen erfolgen.

- Sind View-Operationen kommutativ (vgl. Abschnitt 4.6), können die entsprechenden Ebenen vertauscht werden. Dadurch sind, wie bei relationalen Datenbanken, entsprechende Optimierungen des Query-Plans möglich. Im Fall von dynamischen Views sollten beispielsweise Operationen, die von keinen Applikationsdaten abhängen bzw. von Daten, die sich nur selten ändern, möglichst zu Beginn, d.h. in einer unteren Ebene angeordnet werden. View-Operationen, die von Daten mit hoher Änderungsfrequenz abhängen, sollten entsprechend die obersten Ebenen bilden. So erreicht man, dass die in vielen Fällen nur die obersten Ebenen vollständig neu berechnet werden müssen, während ein Großteil der berechneten View-Ebenen wieder verwendet werden kann.
- Existieren mehrere Views, die in einigen Operationen übereinstimmen, können deren Ebenen gemeinsam verwendet werden. Beispielsweise werden für zwei Views A und B zunächst die technischen Aktivitäten ausgeblendet sowie anschließend in View A die abgeschlossenen Aktivitäten aggregiert und in View B alle Aktivitäten von Partnerunternehmen entfernt. In diesem Fall kann für die Reduktion der technischen Aktivitäten eine gemeinsame Ebene verwendet werden.
- Wegen der intrinsischen Komplexität der Aggregation, ist es sinnvoll etwaige Reduktionen vor der Aggregation durchzuführen, um dadurch die Größe des Prozesses für die Aggregation zu reduzieren.

Weiteres Optimierungspotential birgt die gewählte Form der Materialisierung. Je nach Anwendung kann die Definition einer View oder das resultierende Prozessmodell persistiert werden. Ebenso möglich ist es, verschiedene Zwischenstufen der View-Bildung zu speichern. Entscheidend für die Wahl der Persistierungsstrategie ist zum einen die Art der View. Bei hoch dynamischen Views bringt eine Speicherung von Zwischenprodukten keinen Gewinn. Zum anderen kann eine hohe Anfragefrequenz einer auf einer View basierenden Darstellung es auch erforderlich machen, dass Zwischenprodukte der View-Bildung oder das resultierende Prozessmodell vollständig materialisiert werden. Wie beschrieben liegt dann die einzige Effizienzverbesserung in der Optimierung der Reihenfolge der View-Bildungsoperationen.

5.7 Diskussion

In Kapitel 4 haben wir existierende Ansätze zu Prozess-Views detailliert diskutiert. Nachfolgend greifen wir einige dieser Arbeiten wieder auf und betrachten sie bezüglich ihrer praktischen Anwendbarkeit im Hinblick auf die Visualisierung von Prozessen.

Es existieren einige wenige Ansätze, die mittels View-ähnlichen Mechanismen auf den Benutzer zugeschnittene Prozessmodelle aus den Basisprozessen extrahieren und dies durch Operationen unterstützen. Das bedeutet, dass die View auf den Prozess automatisch erzeugt und nicht von einem Modellierer anhand der Vorlage manuell nachmodelliert wird. Einer dieser Ansätze ist in [SL03, SL04] beschrieben. Eine Relevanzfunktion definiert die Wichtigkeit der Aktivitäten für einen Benutzer und aggregiert unwichtige Aktivitäten. Wie bereits in Abschnitt 4.7 beschrieben,

unterstützt dieser Ansatz nur die Aggregation zusammenhängender SESE-Bereiche. Daher werden immer so viele irrelevante Aktivitäten zusammengefasst, bis die Relevanz des entstandenen SESE-Bereichs einen bestimmten Schwellenwert erreicht. Um die Relevanz einer Aktivität für einen Benutzer zu bewerten, verwendet der Ansatz (a) Informationen aus der Rechteverwaltung über die für einen Benutzer erlaubten Operationen (z.B. GetActivityState) auf einer Aktivität, (b) Informationen über die zulässigen Operationen auf einer Aktivität und (c) die Relevanz einer Operation für eine bestimmte Benutzerrolle.

Ein ähnlicher Ansatz verwendet Reduktions- und Vereinfachungsoperationen, um nicht relevante Aktivitäten aus dem Prozess zu entfernen [SPB05]. Hier werden zwei Relevanzfunktionen angeboten. Eine bewertet die strukturelle Bedeutung von Knoten, die andere verwendet eine Volltextsuche auf den Knotenbeschreibungen. Fällt ein Knoten unter einen bestimmten Schwellwert, wird er aus dem Modell entfernt.

Der Nachteil des erstgenannten Ansatzes ist, dass die Art der Bestimmung der Relevanz auf ein sehr spezielles Modell angepasst und nicht einfach auf beliebige Prozesssysteme übertragbar ist. Die Relevanzfunktionen des zweiten Ansatzes verfolgen einen pragmatischeren und allgemeingültigeren Weg. Generell ist es bei Verwendung von Relevanzfunktionen allerdings nur schwer vorhersagbar, welche Knoten letztlich als relevant identifiziert werden. Eine gezielte Vereinfachung des Prozessmodells zwecks Darstellung für einen bestimmten Benutzer ist nicht möglich. In Proviado stellt die höherwertige Operation *ViewByRelevance* vergleichbare Funktionalität zur Verfügung. Zusätzlich lässt sich hier mittels eines Parameters spezifizieren, ob nicht-relevante Aktivitäten reduziert oder aggregiert werden sollen.

Die Vielzahl der existierenden View-Ansätze ist für unsere Zwecke praktisch nicht anwendbar. Sie bieten zum einen keine Operationen für die Berechnung von Views, sondern erfordern deren Modellierung von Hand. Zum anderen bieten sie mit ihren beschränkten Möglichkeiten der View-Bildung (z.B. werden von allen Ansätzen bestenfalls SESE-Bereiche aggregiert) zu wenige Möglichkeiten für die Visualisierung.

Auch wenn in dieser Arbeit der Anwendungsfokus auf der Visualisierung von Prozessen liegt, ist der Proviado-View-Mechanismus nicht auf diese Anwendungsdomäne beschränkt:

- Die Abbildung realer Geschäftsprozesse aus der Praxis ergibt oftmals riesige Modelle, die nicht zu Unrecht oftmals auch als „Wandtapeten“ bezeichnet werden. Um die Komplexität der Gesamtmodelle beherrschbar zu machen, werden diese häufig in hierarchisch strukturierte Teilmodelle unterteilt. Die Unterteilung erfolgt fix nach einem vorab definierten Schema (z.B. nach zuständiger Organisationseinheit). Dadurch geht jedoch der Zusammenhang des Gesamtprozesses verloren. Mit Hilfe von Prozess-Views ist es möglich diese Hierarchisierung durch Aggregationsoperationen flexibel abzubilden. Dies hat den Vorteil, dass bei Bedarf mehrere unterschiedliche Hierarchisierungen definiert werden können.
- In [BRBB07] haben wir untersucht, wie sich Prozess-Views für die Zugriffskontrolle nach dem RBAC-Modell [FK92] einsetzen lässt. Wie in Beispiel 4-2 motiviert, eignet sich gerade die Reduktion, um vertrauliche Prozesse vor unbefugtem Zugriff zu schützen.

Der vorgestellte Ansatz kann noch in verschiedenerlei Hinsicht erweitert werden. Beispielsweise kann bei Einbeziehung temporaler Aspekte die Reduktion bzw. Aggregation so gestaltet werden, dass nur Aktivitäten innerhalb eines bestimmten Zeitfensters dargestellt werden. Der vorgestellte

Proviado-Ansatz bietet die Basisfunktionalität, um solche fortschrittlichen View-Bildungen realisieren zu können.

5.8 Zusammenfassung

Geschäftsprozessmodelle aus der Praxis sind in der Regel zu groß und komplex, um in ihrer Rohform visualisiert zu werden. Für eine benutzerspezifische Prozessvisualisierung benötigen wir einen flexiblen und mächtigen Mechanismus, um die Prozesse auf die Bedürfnisse der Nutzer bei der Visualisierung anzupassen. In den Kapiteln 4 und 5 haben wir den Proviado-View-Mechanismus vorgestellt. Er ermöglicht es, die Prozesse exakt an die Nutzerbedürfnisse durch Reduktion bzw. Aggregation von Prozessobjekten anzupassen. Auf Grundlage der in Kapitel 4 eingeführten Elementaroperationen haben wir in Kapitel 5 mehrere aufeinander aufbauende Schichten von View-Operationen eingeführt. Sie erlauben es, komplexe Views auf eine einfache und komfortable Art und Weise zu definieren. Während andere Ansätze zur Vermeidung von Ungenauigkeiten, wie dem Löschen von Abhängigkeiten (vgl. Definition 4.12), das zugrunde liegende Basismodell oder die Operationen entsprechend einschränken, werden diese in Proviado bei Bedarf in Kauf genommen, um die Prozessmodelle besser für deren Darstellung aufzubereiten. Durch die verschiedenen Parameter der View-Bildungsoperationen bleibt jedoch kontrollierbar, welche Art von Ungenauigkeiten zugelassen werden, bzw. welchen Qualitätsanforderungen die resultierenden View-Modelle genügen sollen.

Der Proviado-View-Mechanismus bildet die Basis für eine benutzerspezifische Darstellung von Prozessen.

6

Konfiguration der graphischen Darstellung

Die in den vorangehenden Kapiteln vorgestellten Konzepte zur Personalisierung von Prozessdarstellungen basieren auf strukturellen Veränderungen der Prozessgraphen, indem Prozesselemente entfernt oder zu abstrakten Elementen zusammengefasst werden. Dies entspricht einem der in Kapitel 3 vorgestellten Freiheitsgrade in Bezug auf die Konfiguration von Prozessvisualisierungen. In diesem Kapitel behandeln wir einen weiteren Freiheitsgrad — die graphische Adaption der Prozessvisualisierung durch Definition und flexible Anwendung einer Prozessnotation. Dieser Freiheitsgrad deckt in Proviado alle graphischen Aspekte einer Prozessvisualisierung ab, vom „Aussehen“ der Symbole für die verschiedenen Prozesselemente bis hin zur Beschreibung des Verwendungskontexts dieser Symbole [BBR06].

Einer Motivation in Abschnitt 6.1 folgt in Abschnitt 6.2 die Darlegung konkreter Anforderungen an die Konfigurierbarkeit der graphischen Darstellung von Prozessen. Abschnitt 6.3 stellt einen flexiblen Mechanismus vor, der es erlaubt, beliebige Prozessnotationen zu entwerfen und Regeln für ihre Verwendung zu definieren. Die parametrisierbare Konfiguration einer konkreten Prozessdarstellung wird in Abschnitt 6.4 thematisiert. Es folgen die Diskussion verwandter Arbeiten in Abschnitt 6.5 und eine kurze Zusammenfassung in Abschnitt 6.6.

6.1 Motivation

Die Prozessnotation ist in der Wahrnehmung der Endbenutzer ein wesentlicher Aspekt. Eine als unpassend empfundene Notation wirkt sich entsprechend negativ auf die Akzeptanz von Prozessdarstellungen aus [ISO95, ISO98, May99, RC02, SBH+05]. Daher ist es wichtig, für die jeweilige Zielgruppe oder sogar für jeden Benutzer eine geeignete Prozessnotation wählen zu können. Existierende Werkzeuge für die Prozessmodellierung bzw. -visualisierung, wie WBI Modeller [IBM06, WAM+07] oder ARIS Web Publisher [IDS07b], erweisen sich in dieser Hinsicht als zu unflexibel. Hier erfolgt die Darstellung von Prozessen mit exakt denselben Symbolen,

die zuvor vom Prozessmodellierer gewählt bzw. gezeichnet wurden. Um mit diesen Werkzeugen einen Prozess in einer anderen als bei der Modellierung des Prozesses verwendeten Notation darzustellen, müssten die verwendeten Symbole manuell ausgetauscht werden, d.h. es muss jedem einzelnen Element in der Prozessdarstellung ein anderes Symbol von Hand zugewiesen werden. Dies stellt jedoch bei großen Prozessbibliotheken einen nicht vertretbaren Aufwand dar und verhindert oftmals den breiten Einsatz dieser Werkzeuge.

Damit die Betrachter einen möglichst hohen Nutzen aus den Prozessdarstellungen ziehen können, sollten neben den Prozesselementen (d.h. Aktivitäten, Datenelementen, Bearbeiterzuordnungen etc.) weitere Applikationsdaten angezeigt werden können. Fügt man beispielsweise einer Instanzdarstellung des Änderungsmanagement-Prozesses eine Beschriftung mit der Änderungsnummer hinzu, hilft dies dem betroffenen Ingenieur bei der Einordnung der Darstellung. Des Weiteren sollen Applikationsdaten auch innerhalb der Symbole angezeigt werden können, um beispielsweise während der Prozessausführung entstehende Daten in die Prozessvisualisierung einfließen zu lassen. Die Prozessdarstellung existierender Werkzeuge unterstützt dies nicht.

Beispiel 6-1: Realisierung eines Änderungsvorhabens
Als Beispiel dient wieder unser Änderungsmanagement-Prozess aus Abschnitt 2.2.1. Abbildung 6.1 zeigt die Verfeinerung des Schrittes „Realize Change“ (vgl. Seite 13) in abstrakter Darstellung. Sobald entschieden worden ist, die Änderung zu realisieren, wird deren Umsetzung angestoßen (1). Es folgen drei parallele Ablaufpfade: Erstens werden die Produktionsanlagen geplant, die später für die Fertigung des geänderten Teils benötigt werden (2). Zweitens werden die Testumgebungen zur Absicherung der Funktion des vom Änderungsvorhaben betroffenen Teils definiert (3). Drittens werden die Änderungen zunächst im CAD-Entwurf detailliert (4) sowie ein Prototyp aufgebaut (5) und getestet (6). Im Anschluss wird die Integration des geänderten Teils (Prototyps) mit anderen Teilen und Werkzeugen getestet (7). Sind diese Schritte erfolgreich durchlaufen, werden die neuen Teile beim Zulieferer bestellt (8). Während letzterer die Teile produziert (10), werden beim Fahrzeughersteller die nötigen Anpassungen an der Produktionslinie vorgenommen (9). Der (Sub-)Prozess endet mit dem Einsatz der neuen Teile in der Produktion (11). Abbildung 6.2 zeigt denselben Prozess in einer ansprechenderen, benutzergerechten Darstellung für Prozessbeteiligte. Dazu wird eine adaptierte Notation verwendet, die nur die wichtigsten Daten enthält und nicht relevante Details ausblendet. Weitergehende Informationen erhält der Betrachter über Tooltipps (vgl. Aktivität “Prototype Test“ in Abbildung 6.2). Diese werden angezeigt, sobald sich der Mauszeiger über einem Objekt befindet für das Daten hinterlegt sind.

6.2 Anforderungen

Bevor wir zeigen, wie sich Prozessdarstellungen wie in Abbildung 6.2 dargestellt realisieren lassen, diskutieren wir Anforderungen an eine graphische Konfiguration von Prozessvisualisierungen.

Anforderung 6-1 (Verwendung beliebiger Symbole für Prozesselemente): Eine Prozessvisualisierungskomponente muss beliebige Symbole für die Darstellung von Prozesselementen zulassen. Einfache Symbole, die nur den Namen der Aktivität enthalten, müssen ebenso darstellbar sein, wie komplexe Symbole mit einer Vielzahl von Datenwerten. Zur ansprechenden Gestaltung der Symbole sollten beliebige Graphiken einbindbar sein, beispielsweise um eine automatisch ausgeführte Aktivität durch das Symbol eines Rechners zu repräsentieren.

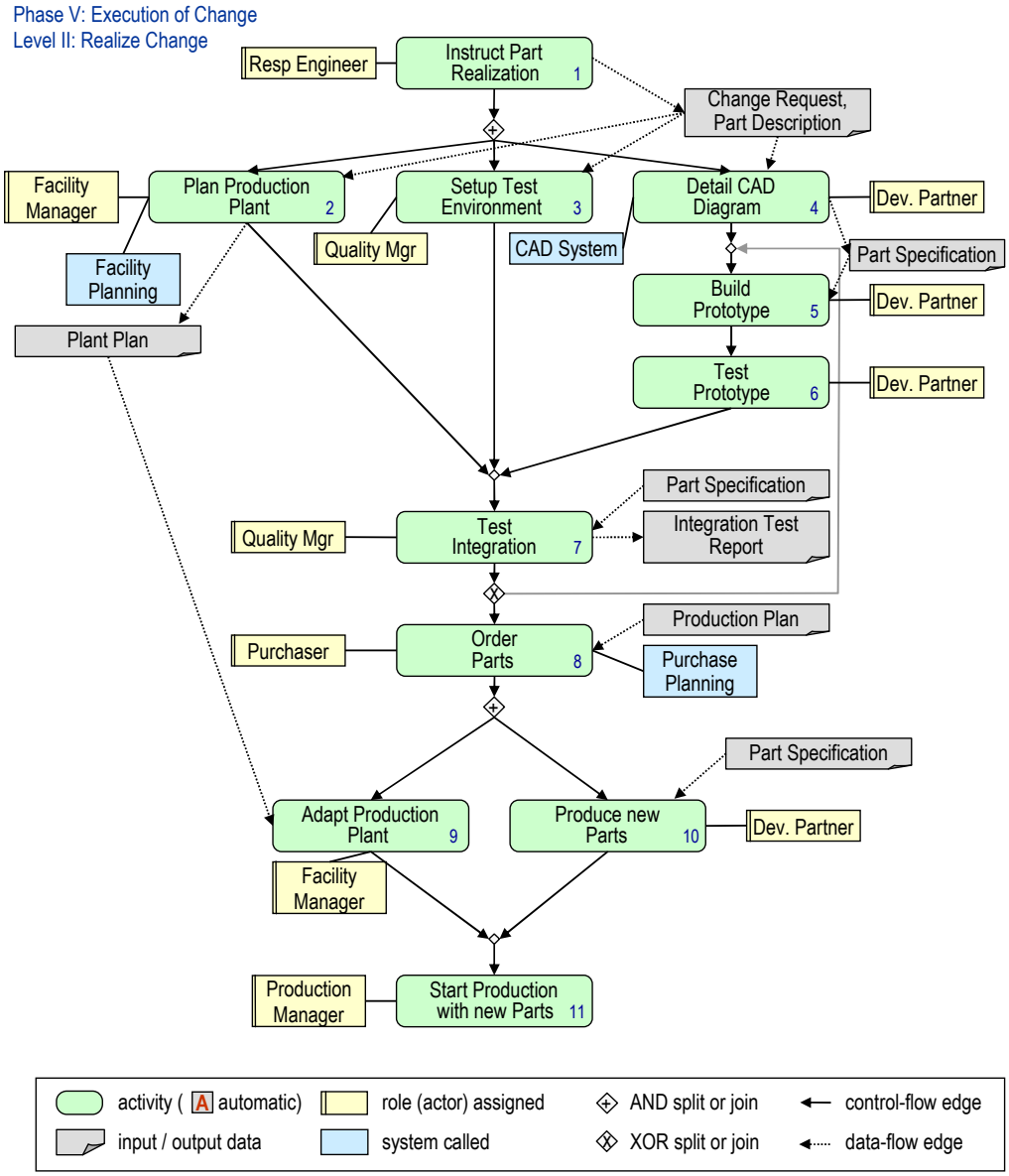


Abbildung 6.1: Abstrakte Darstellung des Änderungsmanagement-Prozesses

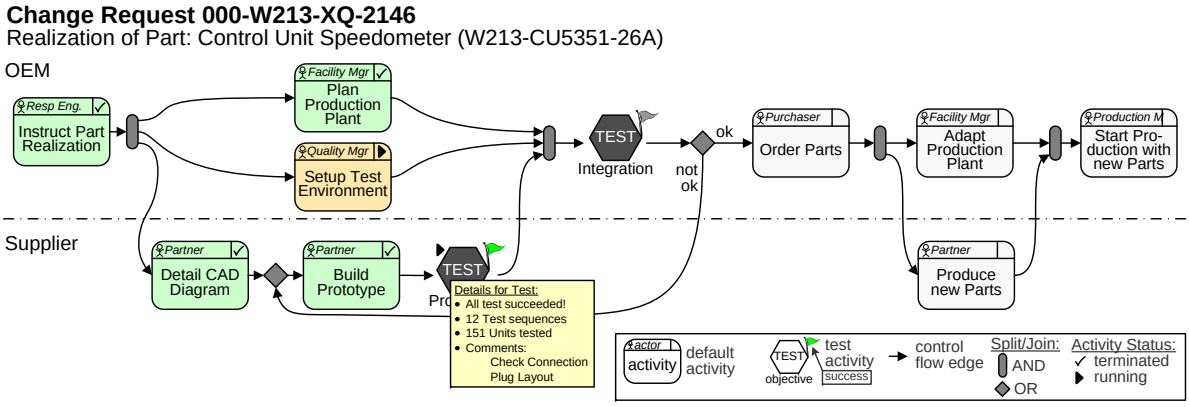


Abbildung 6.2: Darstellung des Änderungsmanagement-Prozesses für den Endbenutzer

Anforderung 6-2 (Symbolauswahl abhängig von Attributwerten): Im Allgemeinen muss die Verwendung eines Symbols abhängig von Attributwerten des Prozesselements gestaltet werden können. Die Verwendungsregeln sollten dabei beliebige Prozessdaten referenzieren können. Welches Symbol für die Repräsentation eines Prozesselements zum Einsatz kommt, ist somit nicht allein durch den Typ des Prozesselements (Aktivitätenknoten, Bearbeiterknoten, Kontrollflusskante, etc.) vorgegeben sondern auch durch Attribute. Beispiel 6-1 etwa verwendet für die Darstellung von Test-Aktivitäten ein spezielles Symbol (vgl. Abbildung 6.2).

Anforderung 6-3 (Dynamische Verwendung von Darstellungssymbolen): Die Darstellung eines bestimmten Prozesselements muss dynamisch adaptierbar sein. Im Verlauf der Visualisierung einer Prozessinstanz etwa sollte die Verwendung der Symbole mit fortschreitender Ausführung des Prozesses anpassbar sein. Beispielsweise könnten für abgeschlossene Aktivitäten andere Symbole verwendet werden als für sich in Ausführung befindliche Prozessschritte. Gleichermaßen sollten in einem anderen Szenario die Dokumente in einem Prozess abhängig von ihrem Fertigstellungsgrad unterschiedlich visualisiert werden können. Eine solche dynamische Auswahl der Symbole sollte auch unter Einbeziehung von Laufzeitdaten erfolgen können.

Anforderung 6-4 (Dynamische Symbole): Neben statischen Symbolen, die Datenwerte in Textfeldern darstellen können, müssen auch dynamische Symbole unterstützt werden. Diese verändern ihr Aussehen mit den sich ändernden Prozessdaten. Typische Beispiele sind Kostenattribute oder Zufriedenheitswerte, die visuell aufbereitet als Balken- bzw. Tachodiagramme dargestellt werden sollen.

Anforderung 6-5 (Personalisierung): Eine zentrale Anforderung für die Visualisierung von Prozessen ist die Personalisierung der Darstellung für den jeweiligen Benutzer. Die in Kapitel 2 skizzierten Fallstudien haben gezeigt, dass Mitarbeiter oftmals durch die in ihrer Abteilung eingesetzten Prozesswerkzeuge bereits an eine bestimmte Notation gewohnt sind. Die Verwendung einer hiervon abweichenden Notation bei der Prozessvisualisierung führt deshalb zu einer verringerten Benutzerakzeptanz. Eine personalisierte Darstellung kann, neben dem vollständigen Austausch einer Notation, auch die Anpassung von Farben, Schriften und Linienformaten erfordern.

Anforderung 6-6 (Präzise Spezifikation der Verwendung von Symbolen): Die Spezifikationen zur Verwendung von Symbolen in der Prozessdarstellung müssen präzise und eindeutig sein. Sollte eine Auswertung entsprechender Regeln für ein Prozesselement mehrere mögliche Symbole ergeben, muss klar definiert sein, wie mit derart konfliktären Anweisungen umgegangen werden soll. Alternativ muss durch die Art der Spezifikation sichergestellt werden, dass derartige Situationen nicht auftreten können.

Anforderung 6-7 (Einfache Neuerstellung von Prozessnotationen und -symbolen): Der Aufwand für die Erstellung einer neuen Prozessnotation sollte so gering wie möglich gehalten werden. Anzustreben ist, dass in einem Graphikprogramm gezeichnete Symbole direkt in die Symboldefinition übernommen werden können. Weiterhin sollten keine Änderungen an der Beschreibung des Graphikobjekts mehr erforderlich sein, um beispielsweise zu definieren, an welcher Stelle des Symbols gewisse Daten angezeigt werden sollen. Andernfalls würde jede graphische Symbolanpassung dazu führen, dass die erforderlichen Änderungen erneut durchzuführen sind. Dies ist zum Beispiel der Fall, wenn in die Graphikbeschreibung ein Platzhalter für später anzuzeigende Daten eingefügt wird. Solche Platzhalter innerhalb einer Graphik werden von Graphikprogrammen in der Regel nicht akzeptiert und müssen vor einer Änderung entfernt und anschließend wieder eingefügt werden.

Anforderung 6-8 (Einblendung von Detailinformationen): Prozessdaten, die entweder nicht wichtig genug oder für die Darstellung im Symbol zu lang sind, müssen bei Bedarf eingeblendet werden können (z.B. durch Tooltipps).

Anforderung 6-9 (Anzeige von Zusatzelementen): Eine benutzergerechte Prozessdarstellung erfordert neben dem Prozessmodell (bzw. der Prozessinstanz) die Anzeige weiterer für den Benutzer relevanter Daten. Beispielsweise erlaubt eine aussagekräftige Überschrift eine bessere Einordnung des Prozessmodells. Die Darstellung von Prozess- oder Applikationsdaten erleichtert es zudem, betroffene Geschäftsvorfälle schnell zu identifizieren, ohne zum Beispiel mit dem Prozess verknüpfte Dokumente öffnen zu müssen. Diese globalen Zusatzelemente einer Prozessdarstellung müssen in ihrer graphischen Erscheinung wieder frei konfigurierbar sein. Weiter müssen beliebige im Prozess oder in Applikationssystemen vorhandene Daten darstellbar bzw. verknüpfbar sein.

Anforderung 6-10 (Parametrisierbare Beschreibung einer Prozessdarstellung): Eine Visualisierungskomponente muss die Gesamtheit aller möglichen bzw. erlaubten Darstellungen als Kombinationen aus Prozessen, Instanzen, Prozess-Views, verschiedenen Darstellungsformen und Notationen verwalten. D.h. ein entsprechend parametrisierbares Modell ist zu entwerfen, das zum Zeitpunkt der Darstellung interpretiert wird. Hieraus wird die Darstellung des Prozesses generiert.

Die bisher beschriebenen Anforderungen beziehen sich auf die funktionalen Eigenschaften einer Prozessnotation, um möglichst ansprechende und nützliche Prozessvisualisierungen erzeugen zu können. Darüber hinaus beschreibt Anforderung 6-10 eine Anforderung an eine Visualisierungskomponente als Ganzes, die für die Darstellung vieler Prozessmodelle mit jeweils unterschiedlichen Notationen verwendet werden soll.

Neben diesen funktionalen Anforderungen existiert eine Reihe nicht-funktionaler Anforderungen, die für die Einsetzbarkeit einer Visualisierungskomponente in der Praxis wichtig sind.

Anforderung 6-11 (Einfacher Zugang zu Prozessvisualisierungen über Intranet): Um den Mitarbeitern im Unternehmen Prozessdarstellungen einfach zugänglich zu machen, müssen diese im Intranet verfügbar sein.

Anforderung 6-12 (Verfügbarkeit von Viewern für verwendete Graphikformate): Für das bei der Prozessdarstellung verwendete Graphikformat müssen passende Anzeigeprogramme (Viewer) zur Verfügung stehen. Gerade in großen Unternehmen ist die Verteilung zusätzlicher Software mit organisatorischen Hürden versehen. Dem vorzuziehen sind Graphikformate, für die auf jedem Arbeitsplatz standardmäßig die passende Software installiert ist.

Anforderung 6-13 (Skalierbarkeit der Graphiken): Ein wichtiger Aspekt, der die Wiederverwendbarkeit und damit den Nutzen einer Prozessdarstellung erhöht, ist die Skalierbarkeit der visualisierten Prozessdiagramme. Sollen die Darstellungen etwa für Dokumentationen (z.B. für ISO9001:2000) oder als Diskussionsgrundlage eingesetzt werden, ist es wichtig, dass die Grafik verlustfrei auf verschiedenen Medien und in unterschiedlichen Größen darstellbar ist.

Anforderung 6-14 (Verwendung von standardisierten Formaten): Um die Unabhängigkeit von Produkten und deren Versionen zu sichern, sollen bei der Implementierung möglichst standardisierte Formate zum Einsatz kommen. Durch die breite Nutzerbasis ist für solche Standardformate am ehesten eine langfristige Verfügbarkeit gegeben. Zudem entfällt der Wartungsaufwand, der für Eigenentwicklungen einzuplanen ist.

Die hier genannten Anforderungen werden von derzeitigen Werkzeugen und Prozessvisualisierungsansätzen nicht in ausreichendem Maße erfüllt. Die meisten Werkzeuge definieren das Aussehen statisch zur Entwurfszeit des Prozessmodells (siehe Abschnitt 6.5). Tabelle 6.1 fasst die vorgestellten Anforderungen nochmals zusammen.

Anforderung	Beschreibung
Anforderung 6-1	Verwendung beliebiger Symbole für Prozesselemente
Anforderung 6-2	Symbolauswahl abhängig von Attributwerten
Anforderung 6-3	Dynamische Verwendung von Darstellungssymbolen
Anforderung 6-4	Dynamische Symbole
Anforderung 6-5	Personalisierung
Anforderung 6-6	Präzise Spezifikation der Verwendung von Symbolen
Anforderung 6-7	Einfache Neuerstellung von Prozessnotationen und -symbolen
Anforderung 6-8	Einblendung von Detailinformationen
Anforderung 6-9	Anzeige von Zusatzelementen
Anforderung 6-10	Parametrisierbare Beschreibung einer Prozessdarstellung
Anforderung 6-11	Einfacher Zugang zu Prozessvisualisierungen über Intranet
Anforderung 6-12	Verfügbarkeit von Viewern für verwendete Graphikformate
Anforderung 6-13	Skalierbarkeit der Graphiken
Anforderung 6-14	Verwendung von standardisierten Formaten

Tabelle 6.1: Anforderungen an die Konfigurierbarkeit der graphischen Darstellung

6.3 Festlegung der Notation einer Prozessvisualisierung

Um die verwendete Prozessnotation flexibel konfigurieren zu können, ist eine strikte Trennung von graphischer Darstellung und dargestellten Daten vonnöten. Der nun vorgestellte Ansatz *beschreibt* die Visualisierung unabhängig vom jeweiligen Prozessmodell. Dies folgt dem Konzept der Trennung von Daten und Darstellung [KP88]. Hierzu führen wir einen *Template-Mechanismus* ein, d.h. einen Mechanismus, mit dem Symbole einmalig abstrakt beschrieben werden (Template) und dann für beliebig viele konkrete Prozesselemente innerhalb desselben oder verschiedener Prozessmodelle instanziierbar sind. Ein Algorithmus wertet bei Anforderung einer Visualisierung die Notationskonfiguration aus und weist den Prozesselementen die entsprechenden Symbole zu. Die Konfiguration einer Notation mit Hilfe dieses Template-Mechanismus besteht aus zwei Komponenten.

1. Definition der für die Darstellung zu verwendenden Symbole (vgl. Anforderung 6-1)
2. Festlegung der Symbole, die für die Darstellung bestimmter Prozesselemente verwendet werden sollen (vgl. Anforderung 6-2).

Diese beiden Bestandteile werden in den Abschnitten 6.3.1 und 6.3.2 im Detail erörtert. Zur Beschreibung relevanter Konfigurationen verwenden wir eine allgemeine XML-Syntax [YCP+06] ohne Festlegung auf einen bestimmten Standard. Abschnitt 6.3.3 beschreibt den Algorithmus, der einem Prozesselement ein graphisches Symbol zuordnet. In Abschnitt 6.3.4 wird das Konzept auf mehrere existierende Standards abgebildet (darunter SVG als Graphikformat). In diesem Kontext diskutieren wir Realisierungsdetails, die für die eingesetzten Technologien spezifisch sind.

6.3.1 Definition von Symbolen

Dieser Abschnitt beschreibt im Detail, wie Symbole für die Verwendung in der Proviado-Prozessvisualisierung definiert werden. Dabei gehen wir zunächst in Abschnitt 6.3.1.1 auf den statischen Fall ein, in dem das Aussehen der Symbole nicht von Prozessdaten abhängt. Abschnitt 6.3.1.2 erläutert dann, was bei der Definition dynamischer Symbole, deren Form und Aussehen sich abhängig von Prozessdaten ändern kann, zusätzlich zu beachten ist. In Abschnitt 6.3.1.3 gehen wir darauf ein, wie Symbole für die Verwendung in einer Prozessvisualisierung referenziert und geschachtelt verwendet werden können. Abschließend beschreibt Abschnitt 6.3.1.4 ein Konzept zum dynamischen Ein- und Ausblenden von Detailinformationen.

6.3.1.1 Statische Templates

Bevor wir beschreiben, wie statische Templates definiert werden, geben wir eine kurze informelle Definition von Templates und Notation.

Definition 6.1 (Template und Notation) Ein Symbol, das bei der Generierung einer Prozessvisualisierung für die Darstellung eines konkreten Prozesselements verwendet wird, bezeichnen wir als *Template* (engl. Schablone). Für die Darstellung wird es mit konkreten Prozessdaten des entsprechenden Prozesselements „befüllt“. Eine *(Prozess-)Notation* ist dem entsprechend über eine Menge von Templates definiert.

Ein Beispiel zeigt das Aktivitäten-Template aus Abbildung 6.3. Ein solches Template kann im Zusammenhang mit einer bestimmten Prozessdarstellung mehrfach Verwendung finden. Templates werden bei der Prozessvisualisierung typischerweise für die Darstellung von Aktivitäten, Datenelementen, Bearbeitern, Strukturknoten (z.B. AND-Splits) und diversen Ausprägungen von Kanten (z.B. Kontrollfluss- oder Datenflusskanten) verwendet.

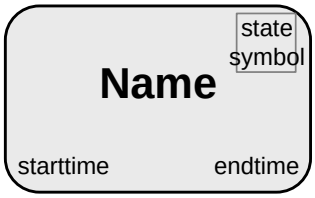


Abbildung 6.3: Symbol für eine Aktivität

Die Definition eines Templates wirft drei grundlegende Fragen auf:

1. Welches Prozesselement (z.B. Aktivität, Datenelement) soll dargestellt werden?
2. Wie soll die graphische Darstellung des Symbols aussehen?
3. Welche Attribute des Prozesselements (z.B. Aktivitätenname, Aktivitätenzustand, Aktivitätenstartzeit etc.) sollen an welcher Stelle und in welcher Form dargestellt werden?

Den prinzipiellen Aufbau von Templates und grundlegende Begriffe des Proviado-Template-Mechanismus illustriert Abbildung 6.4. Vereinfacht ausgedrückt kann man eine Template-Definition mit einer Methodendeklaration vergleichen. Sie besitzt einen Namen, Eingabeparameter und einen Methodenrumpf. Der Methodenrumpf besteht hier aus der Definition des darzustellenden Symbols und der Parameterdefinition. Er beschreibt, wie die übergebenen Parameter (z.B. anzuzeigende Daten der Prozesselemente) mit den konstanten Bestandteilen des Templates aus der Symboldefinition zu einem Rückgabewert, dem fertigen Symbol, kombiniert werden.

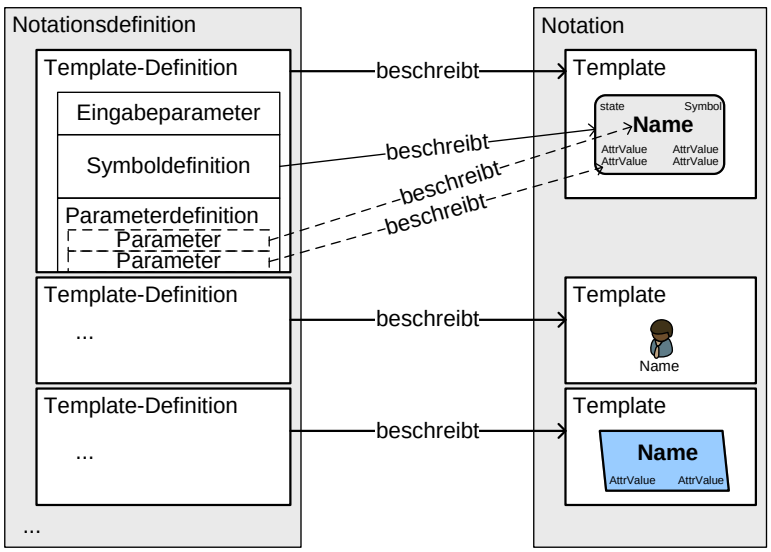


Abbildung 6.4: Struktureller Aufbau einer Template-Definition

Konkret stellen wir die Definition eines Templates in einer XML-Sprache dar (vgl. Abbildung 6.5). Die Definition besteht aus drei zentralen Teilen, um die oben formulierten Aspekte zu handhaben. Im ersten Teil (Zeilen 4–6) werden Variablen deklariert, auf die das Template zugreifen kann und deren Datenfelder somit für die Visualisierung zur Verfügung stehen (Aspekt 1). Zur Laufzeit wird an das Template das darzustellende Prozesselement übergeben. Die Graphik, die das Prozesselement später repräsentieren soll, entspricht in unserer Metapher der Methodendeklaration den Konstanten. Teil 2 der Template-Definition beschreibt die Grafik in einer XML-Syntax (vgl. `symbol`-Tag in Zeilen 10–20 bzw. Aspekt 2). Sie besteht im vorliegenden Beispiel aus einem umschließenden Rechteck und mehreren Textzeilen als Platzhalter für Aktivitätennamen (a), Startzeit (b) und Endezeit. Zusätzlich ist ein Feld erkennbar, in dem später der Zustand der Aktivität dargestellt werden soll (c).

Nach Definition der Eingabeparameter und des Symbols müssen diese miteinander in Verbindung gebracht werden. Dazu muss spezifiziert werden, welcher Datenwert an welcher Stelle der Symboldefinition eingetragen werden soll (Aspekt 3). Beispielsweise muss für den Namen einer Aktivität, der als Attribut des Prozesselements vorliegt, festgelegt werden können, dass dieser in dem zentral gelegenen Textfeld visualisiert werden soll (vgl. Abbildung 6.3). Diese Aufgabe übernimmt Teil 3 der Template-Definition (vgl. Abbildung 6.5), der zugleich der umfangreichste und komplexeste ist. Jeder so genannte *Parameter* beschreibt dabei ein Datenfeld des Symbols, z.B. für den Namen der Aktivität (vgl. (a) bzw. Zeile 13). Das Attribut `location` verweist mittels eines Pfadausdrucks auf die betroffene Stelle der Symboldefinition. Im Beispiel des Aktivitätennamens entspricht dies (a') bzw. Zeile 23. Die Pfadausdrücke erlauben es, entweder auf ganze XML-Tags oder gezielt auf einzelne Attribute zu verweisen (vgl. Parameter `id` in Zeile 29). An dieser Stelle trägt der Template-Mechanismus anschließend die Ausgabewerte ein. Für deren Berechnung stehen die im `inputs`-Abschnitt der Template-Definition deklarierten Variablen zur Verfügung. Sie können alle Elemente umfassen, die im Prozessmodell bzw. in der Prozessinstanz vorhanden sind.

Für die Berechnung der Ausgabewerte existieren zwei Alternativen:

Alternative 1 Der Ausgabewert wird durch einen Skriptausdruck im Attribut `value` des Parameters angegeben. In unserem Beispiel ist dies `act.getName()`, also der Attributwert des Namensfeldes der betreffenden Aktivität (vgl. Zeile 24). Die Eingabevariablen, die an ein Template im Abschnitt `inputs` übergeben werden, umfassen alle Elemente des Prozessmodells bzw. der Prozessinstanz. Um neben solchen Prozessattributen auch Daten aus angebundenen Applikationen darstellen zu können, erlaubt das `value`-Attribut die Angabe von Ausdrücken einer Skriptsprache (z.B. JavaScript). Damit wird es möglich, Berechnungen durchzuführen, etwa die Konvertierung eines Datums in das gewünschte Format (vgl. Zeilen 26 und 28). Auch eine Abstrahierung von Attributwerten (z.B. anstelle eines konkreten Kostenwerts erfolgt die Zuordnung einer Kategorie „hoch“, „mittel“ oder „niedrig“) kann auf diese Weise realisiert werden. Durch die Verwendung von Skriptausdrücken erhöht sich die Mächtigkeit des Template-Mechanismus signifikant, da auf beliebige Prozessdaten zugegriffen werden kann, die dem System zugänglich sind.

Alternative 2 Für komplexere Parameter, die entweder größere Ausgabewerte ergeben (z.B. weil sie zusätzliche graphische Elemente enthalten) oder deren Berechnung sich nicht in einem

```
<template id="default_activity">
  <!-- Part 1: input section -->
  <inputs>
    <input variable="act" type="activity">
      <descr>activity node</descr>
    </input>
  </inputs>
  <graphic>
    <!-- Part 2: symbol section -->
    <symbol>
      <object pv:name="activity" position="(0,0)" id="">
        <rect pv:name="frame" class="activity" width="80" height="50" />
        <text pv:name="act_name" position="(5,5)" size="(60,35)/>
        <text pv:name="starttime" position="(3,41)" size="(35,8)/>
        <text pv:name="endtime" position="(43,41)" size="(35,8)/>
        <object pv:name="state" position="(66,2)">
          <rect width="12" height="12"/>
        </object>
      </object>
    </symbol>
    <!-- Part 3: parameter section -->
    <parameter name="activity_name"
      location="object/text[@pv:name='act_name']"
      value="act.getName()" />
    <parameter name="starttime" location="object/text[@pv:name='starttime']"
      value="formatDate(act.getStarttime(),'dd/mm/yyyy')" delete="empty"/>
    <parameter name="endtime" location="object/text[@pv:name='endtime']"
      value="formatDate(act.getEndtime(),'dd/mm/yyyy')" delete="empty"/>
    <parameter name="id" location="object/@id" value="act.getId()"/>
    <parameter name="position" location="object/@position"
      value="'('+act.getX()+','+act.getY()+')'"/>
    <parameter name="state" location="object/object[@pv:name='state']" >
      <choose>
        <case test="act.getState()='activated'">
            <polygon pv:name="activated" points="2,10 10,10 6,2" />
        </case>
        <case test="act.getState()='running'">
            <polygon pv:name="running" points="2,2 2,10 10,6" />
        </case>
        <case test="act.getState()='completed'">
            <polyline pv:name="completed" points="2,6 4,10 10,2" />
        </case>
        <otherwise>
          <line points="2,6 10,6">
        </otherwise>
      </choose>
    </parameter>
  </graphic>
</template>
```

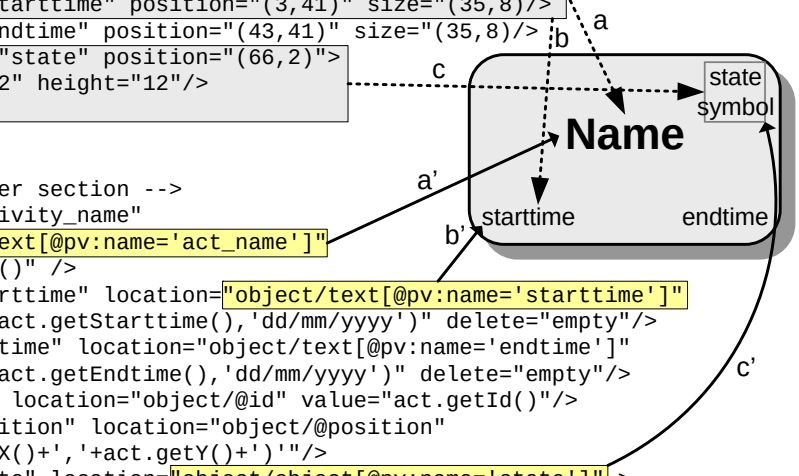


Abbildung 6.5: Template-Mechanismus: Definition eines Symbols

Ausdruck darstellen lässt, bietet der Template-Mechanismus eine weitere Alternative. Sie wertet den Rumpf der XML-Parameterdefinition aus, falls das Attribut `value` nicht vorhanden ist. Prinzipiell wird der gesamte Inhalt als Ausgabewert übernommen, mit Ausnahme der `script`- und `choose`-Elemente. Durch `script`-Elemente werden Skripte gekennzeichnet, die komplexe Berechnungen auf Grundlage der Eingabeparameter durchführen können. Beispielsweise können damit gewisse Attribute abhängig von ihrem Wert durch unterschiedliche graphische Elemente dargestellt werden. Das `choose`-Element ermöglicht es, mehrere Bedingungen nacheinander abzuprüfen und so aus verschiedenen Ausgabewerten auszuwählen. Unser Beispiel verwendet diesen Mechanismus für das Statusattribut einer Aktivität. Dieses soll je nach Ausführungsstatus durch unterschiedliche Symbole repräsentiert werden. Zeilen 33–46 zeigen das `choose`-Element. Für jeden möglichen Fall (durch `case`-Elemente gekennzeichnet) enthält das `test`-Attribute die jeweilige Bedingung (als Skriptausdruck). Die Bedingungen werden sequentiell ausgewertet und der Inhalt des `case`-Elements, dessen `test`-Attribut zu `true` evaluiert, wird als Ausgabewert übernommen. Das `otherwise`-Tag fängt die Fälle ab, für die keiner der vorangegangenen `test`s erfolgreich war. Als Rumpf der `case`- bzw. `otherwise`-Tags sind neben graphischen Elementen auch Skripte zulässig (markiert durch `script`).

Alle Details zum Proviado-Template-Mechanismus und dessen prototypische Implementierung sind in [KH06] dokumentiert.

6.3.1.2 Dynamische Templates

Die bisher vorgestellte Beschreibung von Templates ist für Standardfälle, in denen ein Prozesselement durch ein einfaches, statisches Symbol dargestellt werden soll, ausreichend (vgl. Anforderung 6-3). Statisch bedeutet, dass die geometrische Form des Symbols unabhängig von den zu visualisierenden Daten ist. Bei dynamischen Symbolen passt sich dagegen die Form des Symbols (oder Teile davon) mit Änderung der darzustellenden Daten an.

Ein Beispiel für die Verwendung von dynamischen Templates ist die Darstellung von Kennzahlen in Prozessmodellen. Abbildung 6.6 zeigt dazu Tachodiagramme. Sie erlauben es, schnell einen Überblick zu gewinnen, ohne die genauen Werte der Kennzahlen kennen zu müssen. Solche sich dynamisch verändernden Darstellungen sind mit Hilfe der bislang vorgestellten Methoden des Template-Mechanismus prinzipiell abbildbar (z.B. Berechnung der Neigung der Tachonadel mittels Skript). Dies führt im Allgemeinen aber zu komplizierten und damit schwer wartbaren Template-Definitionen. Die Generierung dynamischer Symbole wird in Proviado deshalb in externe Komponenten ausgelagert, die vom Template-Mechanismus referenziert werden können. Sie erhalten Eingabeparameter und liefern eine Graphikbeschreibung zurück. Durch diesen Ansatz geht zwar der Vorteil des einfachen Bearbeitens von Symbolen verloren, gleichzeitig

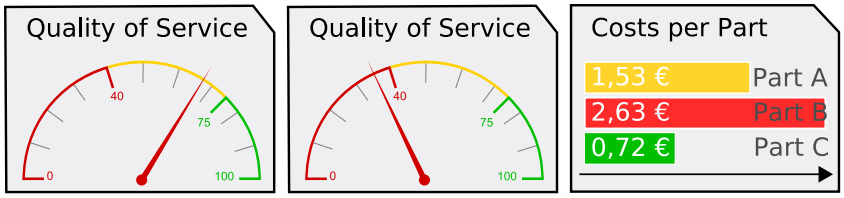


Abbildung 6.6: Beispiele für dynamische Templates

können in den externen Komponenten aber komplexe Berechnungen zur Generierung der Symbole vorgenommen werden.

In der Umsetzung werden für die dynamischen Templates Java-Klassen verwendet, die für die Generierung der Symbole auf existierende Bibliotheken zurückgreifen können. Ein Beispiel ist die Bibliothek JFreeChart [JFr08], mit deren Hilfe ansprechende Graphiken für viele verschiedene Diagrammarten (z.B. Balken-, Linien- oder Tachodiagramme) generiert werden können.

6.3.1.3 Referenzierung und Schachtelung

Dynamische Templates, wie im vorgehenden Abschnitt vorgestellt, finden ihren Einsatz meist in Kombination mit statischen Templates. Beispielsweise lässt sich in direkter Nachbarschaft zu einem Aktivitätensymbol ein kleines Diagramm platzieren, um Attributwerte der betreffenden Aktivität visuell ansprechend aufzubereiten (vgl. Abbildung 6.7). Das bedeutet, dass das Template für das Diagramm vom eigentlichen Aktivitäten-Template referenziert wird. Wie dies in unserem Template-Mechanismus realisiert wird, ist Abbildung 6.8 zu entnehmen. Sie zeigt eine Parameterdefinition, die auf das dynamische Template verweist. Dabei ist auch der Mechanismus zur Übergabe der Eingabeparameter zu erkennen (Zeilen 20–24). Allgemein lässt sich die Referenzierung von Templates dazu benutzen, Symbole zu modularisieren und somit die Wiederverwendung von Template-Definitionen zu fördern.

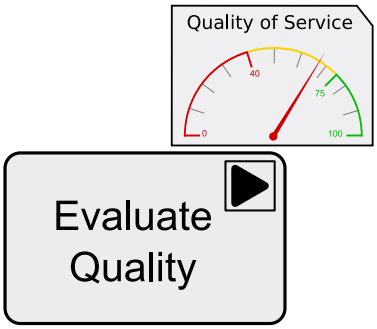


Abbildung 6.7: Aktivität mit referenziertem dynamischen Template

6.3.1.4 Dynamisches Ein- und Ausblenden von Detailinformationen

Prozesse und damit verbundene Applikationssysteme enthalten eine Vielzahl von Daten, die für den Benutzer visualisiert werden sollen (vgl. Anforderung 6-10). Diese Daten alle auf einmal anzuzeigen überfrachtet meist die Visualisierung. Eine mögliche Methode, um Detailinformationen in die Visualisierung zu integrieren, sie aber nur bei Bedarf anzuzeigen, besteht in der Verwendung von *Tooltipps*, d.h. kleinen Fenstern, die nur erscheinen, solange der Mauszeiger auf einem Symbol verweilt. Dadurch kann Hintergrundinformation in die Visualisierung einbezogen werden, die standardmäßig nicht angezeigt wird.

Mit dem vorgestellten Template-Mechanismus lassen sich Tooltipps mittels zusätzlicher Templates realisieren. Auf dem Client werden diese speziellen Templates dann bei der Anzeige

```
<template id="activity_withQoS">
  ...
    <!-- Part 3: parameter section -->
    ...
    <parameter name="QoS-diagramm"
      location="object/object[@pv:name='QoS']" >
      <template ref="compass_diagramm">
        <inputs>
          <input name="value" value="act.getAttributValue('QoS')" />
          <input name="limits" value="0,40,75,100" />
          <input name="heading" value="'Quality of Service'" />
        </inputs>
      </template>
    </parameter>
    ...
  </graphic>
</template>

<template id="compass_diagramm">
  <inputs>
    <input variable="value" type="Integer" />
    <input variable="limits" type="String" default="0,50,80,100" />
    <input variable="heading" type="String" />
  </inputs>
  <graphic classname="de.proviado.templates.CompassDiagramm"/>
</template>
```

Abbildung 6.8: Beispiel für die Referenzierung eines dynamischen Templates

dynamisch ein- und ausgeblendet. Eine andere Realisierungsvariante besteht darin, nur die anzuzeigenden Daten in der Symboldefinition versteckt abzulegen und daraus bei der Anzeige den Tooltipp per Skript zusammenzusetzen.

6.3.2 Verwendung von Symbolen

Um eine Prozessvisualisierung automatisch generieren zu können, muss spezifiziert werden, welches Prozesselement durch welches Template repräsentiert werden soll. Üblicherweise wird jedem Prozesselementtyp (z.B. Aktivität) ein bestimmtes Symbol zugeordnet. D.h. Aktivitäten werden durch ein vordefiniertes Aktivitätssymbol, Bearbeiter durch ein bestimmtes Bearbeitersymbol, usw. dargestellt. Eine solche statische Zuordnung von Prozesselementtyp und Symbol wird typischerweise von existierenden Systemen angeboten.

Eine flexible Visualisierung, wie sie in Proviado gefordert ist (vgl. Anforderung 6-2), lässt sich damit allein aber nicht realisieren. Im Kontext von Prozessvisualisierungen gibt es viele Anwendungsfälle, die eine dynamische und flexible Auswahl der darzustellenden Symbole erfordern, etwa:

1. Automatische und interaktive Aktivitäten sollen durch unterschiedliche Symbole dargestellt werden.
2. Bestimmte Aktivitäten (z.B. Aktivitäten vom Typ „Test“ in Beispiel 6-1) sollen durch spezielle Symbole von anderen Aktivitäten abgehoben werden.

3. Externer Partner bzw. Bearbeiter sollen durch spezielle Bearbeitersymbole dargestellt werden.
4. Die Darstellung einer Aktivität soll abhängig vom Zustand der Aktivität variiert werden (vgl. Anforderung 6-3).
5. Sobald Dokumente eines Prozesses unterzeichnet sind, soll dies durch Verwendung eines besonderen Dokumentensymbols hervorgehoben werden.

Während die ersten drei Beispiele die Verwendung von Symbolen in Abhängigkeit von beliebigen, zur Modellierzeit feststehenden Prozessdaten erfordern, gehen die letzten beiden Fälle einen Schritt weiter. Wann welches Symbol zum Einsatz kommen soll, ist hier abhängig von Laufzeit- bzw. Applikationsdaten. Abbildung 6.9 illustriert die unterschiedlichen Abhängigkeiten.

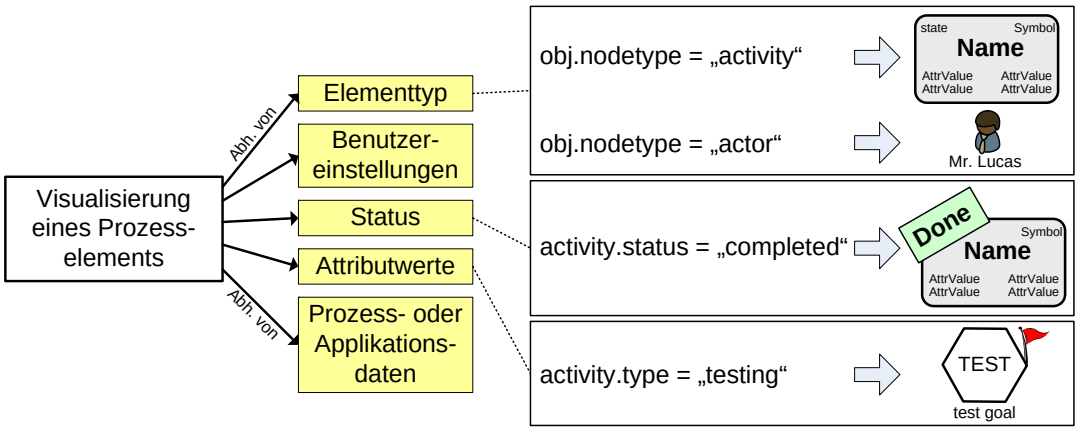


Abbildung 6.9: Zusammenhang zwischen Prozessdaten und Templates

Zur Realisierung einer flexiblen Symbolauswahl müssen die Verwendungsvorschriften für die verschiedenen Symbole mit dem Proviado-Template-Mechanismus einmalig statisch definiert werden (vgl. Abbildung 6.10). Dies ergibt die *Verwendungsdefinition*, in der festgeschrieben wird, welches Template in Abhängigkeit welcher Prozesselementtypen bzw. -daten für die Visualisierung eines Prozesselements jeweils verwendet werden soll. Zur Laufzeit wird diese Definition bei der Generierung einer Prozessvisualisierung interpretiert und erlaubt dadurch die dynamische Auswahl von Templates auch anhand von Laufzeitdaten.

Wir werden nun zwei alternative Ansätze zur Beschreibung der Verwendungsvorschriften diskutieren.

Ansatz I Ein naiver Ansatz zur Festlegung der Verwendungsvorschriften besteht darin, jede Vorschrift als Regel auszudrücken. Nun existiert für jedes Template mindestens eine Regel, so dass bei diesem Ansatz für eine Prozessnotation ein ganzer Satz von Regeln resultieren würde. Obgleich dieser direkte Ansatz einfach zu implementieren ist, führt er in der Praxis zu Problemen. So kann a priori nicht ausgeschlossen werden, dass widersprüchliche Regeln existieren.

Beispiel 6-2: Konfliktäre Regeln
Angenommen Regel 1 spezifiziert, dass für eine von einem externen Bearbeiter ausgeführte

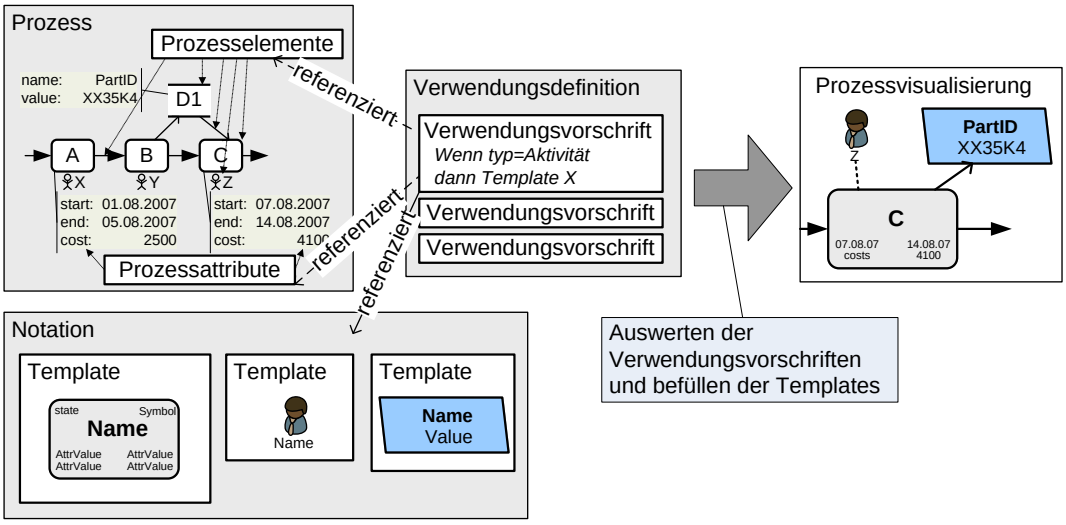


Abbildung 6.10: Prinzip der Verwendung von Symbolen

Aktivität das Template `Act_external` verwendet werden soll, und Regel 2 fordert für abgeschlossene Aktivitäten das Template `Act_finished`. Dann resultiert ein Konflikt bzgl. der Visualisierung, sobald ein externer Bearbeiter eine Aktivität abschließt (vgl. Abbildung 6.11). Ein ähnlicher Konflikt kann in Verbindung mit Beispiel 6-1 auftreten, wenn zwei Regeln sowohl für Testaktivitäten als auch für Aktivitäten in abgewählten Zweigen besondere Symbole fordern.

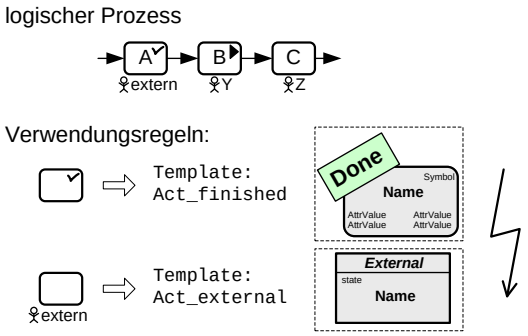
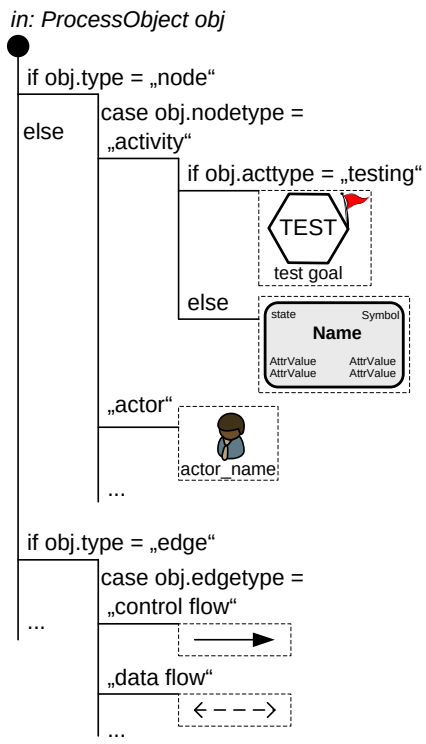


Abbildung 6.11: Konfliktäre Verwendungsvorschriften für Symbole

Zur Vermeidung solcher Darstellungskonflikte sind umfangreiche Konsistenzprüfungen erforderlich. Außerdem leidet mit zunehmender Größe der Regelmenge die Wartbarkeit des Systems. D.h. sobald die Verwendungsregeln eine gewisse Komplexität erreichen, ist es nicht trivial herauszufinden, wann welche Templates auszuwählen sind, und etwaige Widersprüche zu erkennen.

Ansatz II Der Template-Mechanismus von Proviado verfolgt einen anderen Ansatz, der per Konstruktion die eindeutige Auswahl von Templates bei der Prozessvisualisierung garantieren soll. Zu diesem Zweck bilden wir die Verwendungsregeln auf einen Entscheidungsbaum ab, dessen Blätter auf Template-Definitionen verweisen (vgl. Abbildung 6.12). Die inneren Knoten

des Baumes enthalten Verzweigungsbedingungen. Sie spezifizieren, welcher Zweig verfolgt werden soll. Um einem Prozesselement das entsprechend der Verwendungsvorschriften zugehörige Template zuzuordnen, wird der Baum durchlaufen. Dabei wird an jedem inneren Knoten die Verzweigungsbedingung ausgewertet. Diese Bedingungen beziehen sich auf Daten des Prozesselements oder auf beliebige andere Prozess- oder Applikationsdaten. Schließlich gelangt man zu einem Blatt, von dem auf ein Template referenziert wird. Dieses Template wird daraufhin für die Darstellung des Prozesselements verwendet. Blätter können auch leer sein, was dazu führt, dass dem Prozesselement kein Template zugeordnet wird und dieses Element folglich nicht in der Prozessvisualisierung erscheint. Wohlgemerkt sind die Daten solcher nicht dargestellter Elemente nach wie vor im Prozessmodell enthalten und können daher von Templates anderer Prozesselemente angezeigt werden. Der Weg von der Wurzel des Baumes ist immer eindeutig. Da der Baum endlich ist, terminiert dieser Algorithmus stets. Konflikte, wie sie bei dem auf Regelmengen basierenden Ansatz aufgetreten sind, können durch die Vorgehensweise des Algorithmus in Kombination mit der Konstruktionsweise des Baumes nicht länger auftreten. Für die Implementierung bilden wir den Entscheidungsbaum auf ein XML-Format ab (vgl. Abbildung 6.12). Den skizzierten Algorithmus zur Auswertung der Verwendungsdefinition erklären wir im folgenden Abschnitt anhand eines Gesamtbeispiels, das auch die zuvor in Abschnitt 6.3.1 beschriebene Template-Auswertung umfasst.




```
1  <if test="obj.nodetype=ACTOR">
2    <template id="actor">
3      <inputs>
4        <input name="actor" value="obj"/>
5      </inputs>
6    </template>
7  </if>
8  <if test="obj.nodetype=ACTIVITY">
9    <choose>
10     <case test="obj.acttype='testing'">
11       <template ref="testing_act">
12         <inputs>
13           <input name="act" value="obj"/>
14         </inputs>
15       </template>
16     </case>
17     <otherwise>
18       <template ref="default_act">
19         <inputs>
20           <input name="act" value="obj"/>
21         </inputs>
22       </template>
23     </otherwise>
24   </choose>
25 </if>
```

Abbildung 6.12: Verwendungsdefinition für Templates in abstrakter Form und als XML

6.3.3 Gesamtablauf

Wir haben bisher dargelegt, wie man Symbole für die Prozessvisualisierung definieren kann und wie sich deren Verwendung flexibel festlegen lässt. In diesem Abschnitt beschreiben wir den Gesamtablauf zur Generierung einer graphischen Prozessdarstellung aus einem gegebenen logischen Prozessmodell. Wir erklären diesen Ablauf anhand eines Beispiels.

Zunächst erläutern wir den Aufbau einer vollständigen Prozessnotationsdefinition. Neben den Templates und deren Verwendungsvorschriften gehören dazu globale Elemente. Hierunter verstehen wir die Teile einer Prozessnotation, die für die Darstellung benötigt werden, die aber keinem Prozesselement direkt zugeordnet werden können. Als Beispiel seien die zur Realisierung der Tooltipps benötigten Skripte genannt. Die für das kontextabhängige Ein- und Ausblenden von Prozessinformationen benötigten Skripte sind eng mit bestimmten Templates gekoppelt und müssen daher zusammen mit den Template-Definitionen verwaltet werden. Bei der Erzeugung einer Prozessvisualisierung werden die globalen Elemente direkt in die Prozessgraphik übernommen.

Von diesen Vorbetrachtungen ausgehend verstehen wir unter einer Notationsdefinition Folgendes.

Definition 6.2 Eine *Notationsdefinition* ist ein Tupel bestehend aus einer Menge von Template-Definitionen, einer Verwendungsdefinition für die Templates und einer Menge globaler Elemente.

Eine vollständige Notationsdefinition wird in Proviado als XML-Datei gespeichert.

Im Folgenden stellen wir Schritt für Schritt dar, wie man von einem logischen Prozessmodell zu einer Prozessvisualisierung gelangt. Für die Beschreibung verwenden wir das folgende Beispiel.

Beispiel 6-3: Aggregierte Darstellung des Änderungsmanagement-Prozesses
Für die Führungskräfte soll eine vereinfachte Darstellung des Änderungsmanagement-Prozesses aus Abschnitt 2.2.1 erstellt werden. Dazu wird eine aggregierende View definiert, die genau fünf abstrakte Aktivitäten erzeugt. Diese entsprechen logisch den fünf Phasen des Änderungsmanagement-Prozesses. Das resultierende abstrahierte Prozessmodell, welches als Basis der Visualisierung dient, zeigt Abbildung 6.13.

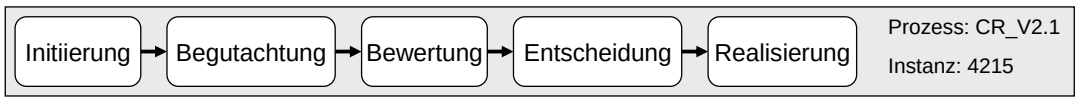


Abbildung 6.13: Logisches Prozessmodell für die Visualisierung

Basierend auf diesem vereinfachten Modell wird die in Abbildung 6.18 dargestellte Visualisierung gewünscht. Sie enthält neben Prozess- und Instanzdaten weitere Zusatzobjekte (z.B. Überschriften). Die dazu benötigten Elemente werden dem zu visualisierenden Prozessmodell in einem vorbereitenden Schritt hinzugefügt und können dann bei der Generierung der Visualisierung in gleicher Art und Weise wie die „normalen“ Prozesselemente und -daten verarbeitet bzw. behandelt werden. Wir kommen auf dieses Detail im Rahmen der Beschreibung der Gesamtkonfiguration (siehe Abschnitt 6.4) nochmals zurück.

Algorithmus zur Auswertung einer Prozessnotation

Der Algorithmus AssignTemplates erhält als Eingabe das logische Prozessmodell (vgl. Abbildung 6.14). Der Ansatz funktioniert gleichermaßen für Prozessinstanzen, jedoch mit dem Unterschied, dass über die Instanz noch Zugriff auf weitere Prozessdaten besteht (z.B. Ausführungszustände von Aktivitäten oder Laufzeitdaten aus Applikationssystemen), die für die Visualisierung zur Verfügung stehen. Im vorliegenden Fall soll eine Prozessinstanz dargestellt werden, d.h. anstelle des Prozessschemas referenzieren wir auf die Prozessinstanz. Des Weiteren werden dem Algorithmus die oben angesprochenen Zusatzobjekte, sowie die Notationsdefinition, bestehend aus Template-Definition und Verwendungsdefinition, übergeben. Abbildung 6.15 zeigt graphisch die für dieses Beispiel benötigten Template-Definitionen.

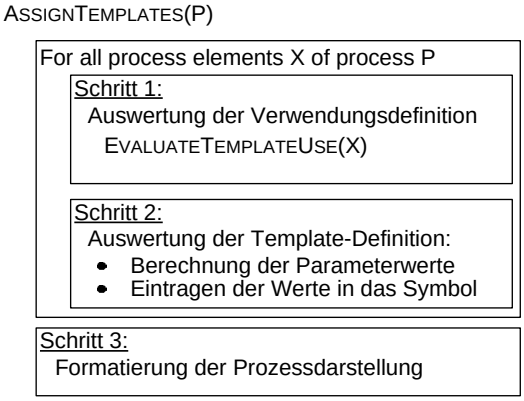


Abbildung 6.14: Algorithmus zur Auswertung einer Prozessnotationsdefinition

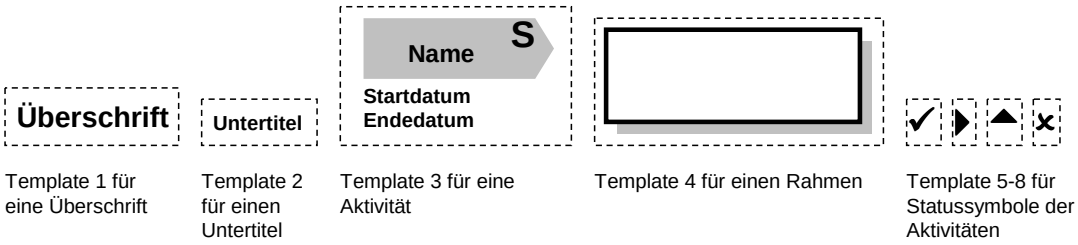


Abbildung 6.15: Zur Verfügung stehende Templates

Schritt 1 (Auswertung der Verwendungsdefinition): In Schritt 1 des Algorithmus wird für alle Prozesselemente die Template-Zuordnung, wie in Abschnitt 6.3.2 beschrieben, ausgewertet. D.h. es wird für jedes Prozesselement bestimmt, durch welches Template es dargestellt werden soll. Bezogen auf unser Beispiel wird jeder Aktivität des logischen Prozessmodells ein Aktivitätenpfeil zugeordnet. Analog wird der Überschrift (als Zusatzelement vorhanden) ein entsprechend großes Textfeld zugewiesen usw. (vgl. Abbildung 6.16).

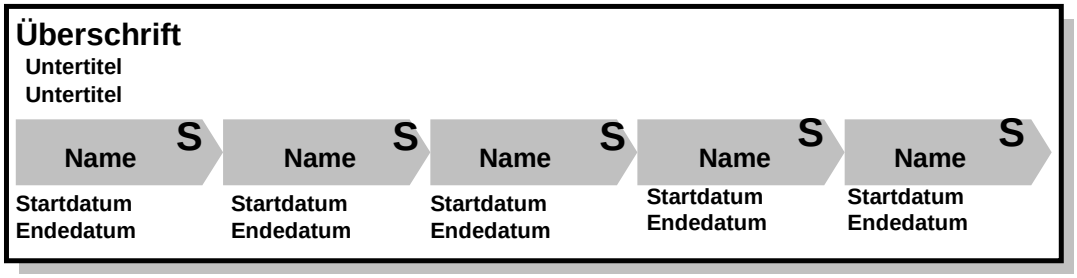


Abbildung 6.16: Nach der Zuordnung der Templates zu den Prozessobjekten

Schritt 2 (Auswertung der Template-Definition): In Schritt 2 müssen die Templates mit Daten der logischen Prozesselemente befüllt werden. Die dazu nötigen Angaben sind in den Parametern der Template-Definition abgelegt (vgl. Abschnitt 6.3.1.1). Schritt 2 des Algorithmus wertet die Parameterdefinitionen aus. Enthält der Parameter ein `value`-Attribut, wird dieses als Skript-Ausdruck ausgewertet. Das Ergebnis wird an die durch das `location`-Attribut definierte Stelle in der Symbolbeschreibung des Templates kopiert. Das genaue Vorgehen bei der Auswertung der Parameter wurde in Abschnitt 6.3.1.1 beschrieben. Bezogen auf unser Beispiel werden der Name der jeweiligen Aktivität sowie ihr Start- und Endedatum in das betreffende Aktivitäten-Template eingetragen.

Der Zustand der Visualisierung nach diesem Schritt ist in Abbildung 6.17 illustriert.

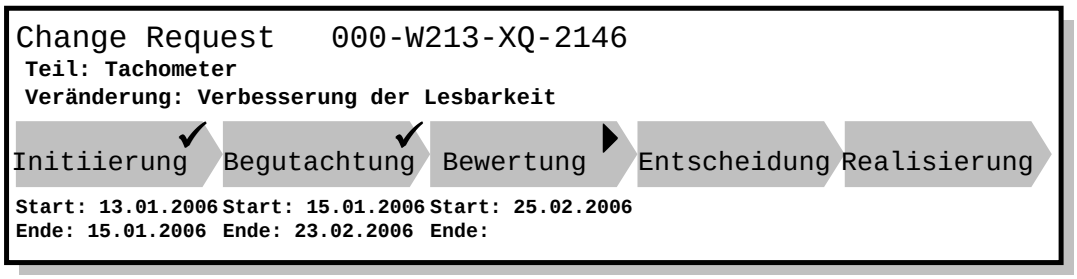


Abbildung 6.17: Auflösung der Template-Parameter: Darstellung mit Daten

Schritt 3 (Formatierung der Prozessdarstellung): Für das endgültige Aussehen der Prozessvisualisierung, wie es in Abbildung 6.18 dargestellt ist, müssen die Symbole noch formatiert werden, d.h. Farbgebung, zu verwendende Schriftarten und Linienformate müssen festgelegt werden. Die Trennung der Formatierungsinformation von den darzustellenden Prozesssymbolen ermöglicht ohne großen Aufwand eine weitere Personalisierung der Darstellung (vgl. Abbildung 6.19). Spezielle Attribute beschreiben die Funktionen der einzelnen Symbole bzw. Symbolbestandteil in der Template-Definition (z.B. Attribut `pv:name` in Zeile 11 in Abbildung 6.5). Zur Formatierung können diese Attribute referenziert und dadurch beispielsweise festgelegt werden, welche Farben für den Rahmen der Aktivitätensymbole verwendet werden soll. Die Realisierung erfolgt mittels *Stylesheets* [LJBL98]. Die für die Formatierung der Prozessvisualisierung benötigten Stylesheets werden als globale Elemente bei der Generierung der Darstellung in die resultierende Graphikdatei eingebettet. Ist eine flexible Formatierung der Prozessgraphiken nicht erforderlich, kann das Format der dargestellten Objekte auch bereits in der Symbol-Definition erfolgen. Ein Stylesheet ist dann nicht mehr erforderlich.

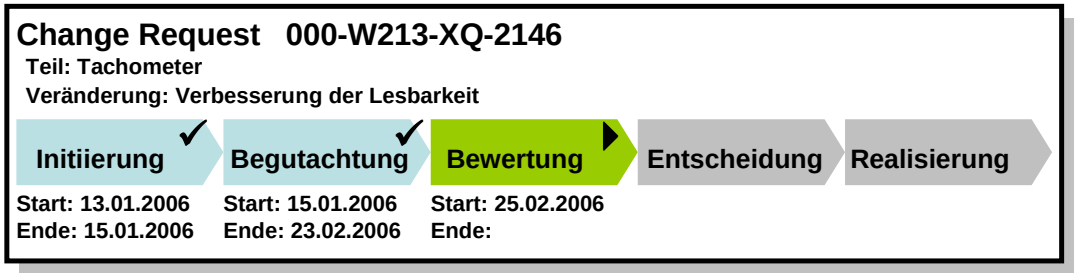


Abbildung 6.18: Form der Visualisierung, wie sie dem Benutzer dargestellt wird

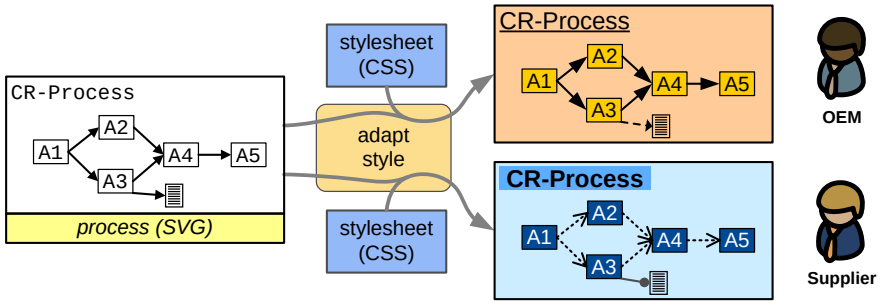


Abbildung 6.19: Formatierung der Darstellung durch Stylesheets

Standardmäßig wird der beschriebene Algorithmus AssignTemplates jedes Mal angestoßen, wenn eine Visualisierung benötigt wird. Damit ermöglichen wir eine hohe Dynamik in der Verwendung der Templates. Auch dynamische Änderungen der Symbolik, abhängig vom Aktivitätenzustand oder anderen Laufzeitdaten, sind mit dem vorgestellten Ansatz kein Problem.

Abhängig vom Einsatzszenario der Visualisierung kann der Ablauf bei Bedarf optimiert werden. Hängt die Verwendung der Templates beispielsweise nicht von Laufzeitdaten ab, kann die Auflösung der Template-Verwendung in einen Vorverarbeitungsschritt ausgelagert werden. Zu Laufzeit müssen dann nur noch die Template-Parameter ausgewertet werden.

6.3.4 Realisierung auf Grundlage von Standardformaten

Der Proviado-Template-Mechanismus wurde bislang konzeptionell beschrieben, ohne Bezug auf existierende Technologien und Datenformate zu nehmen. Dieser Abschnitt legt dar, auf welche Standardformate wir bei der Implementierung des Template-Mechanismus zurückgreifen.

Im Allgemeinen ist es sinnvoll, den zu visualisierenden Prozess in ein Standardgraphikformat zu transformieren. Dies bietet die bekannten Vorteile, dass Standards zum einen langfristig verfügbar sind und zum anderen von vielen Softwareprodukten unterstützt werden. Zudem sind für viele Standardformate Bibliotheken für deren Erzeugung bzw. Verarbeitung verfügbar, die den Aufwand einer Implementierung signifikant reduzieren. Da diese Bibliotheken von einer großen Anzahl von Nutzern verwendet werden, werden Fehler in der Implementierung schneller gefunden und ebenso schnell behoben. Aus diesen Gründen wird zur Realisierung des Proviado-

Template-Mechanismus soweit möglich und sinnvoll auf verfügbare Standards zurückgegriffen (vgl. Anforderung 6-14).

Für die Repräsentation von persistierbaren Daten wie Prozessmodellbeschreibungen oder Konfigurationsdateien wird XML (Extensible Markup Language [YCP+06]) verwendet, so auch für die Beschreibung der Template-Definitionen. Für XML sind ausgereifte Parser und Bibliotheken verfügbar, die einen komfortablen Zugriff auf die Inhalte erlauben [Har02, DOM08].

Als Datenformat für die generierten Prozessdarstellungen (inkl. ihrer Symbole) setzen wir SVG (Scalable Vector Graphics) ein. SVG ist ein XML-basiertes Format zur Beschreibung von Vektorgraphiken und wurde als Standard vom World Wide Web Consortium (W3C) etabliert [FFJ03, JN05, DNB+06]. SVG bietet als Vektorgraphikformat eine Reihe von Vorzügen gegenüber Rastergraphikformaten (z.B. BMP, PNG oder JPEG). So sind Vektorgraphiken im Allgemeinen und SVG-Graphiken im Besonderen skalierbar und lassen sich deshalb auf beliebigen Medien ohne Qualitätsverlust reproduzieren. Gleichzeitig ist die Größe einer SVG-Datei im Vergleich zu Rastergraphiken unabhängig von der dargestellten Bildgröße, d.h. eine kompakte Repräsentation der darzustellenden Daten wird möglich. SVG ist ausdrucksmächtig genug, um beliebig komplexe Graphiken zu definieren. Weiterhin werden Animationen unterstützt, von denen wir hier allerdings keinen Gebrauch machen. Die verfügbaren Darstellungsprogramme und Programmierbibliotheken [HK01, Bat08] für SVG nehmen einem das Rendering der Graphiken (d.h. die Interpretation der SVG-Daten und die Generierung der entsprechenden Graphik) ab und vereinfachen somit die Implementierung weiter. Als XML-basiertes Format ist SVG zudem mit anderen Web-Standards kombinierbar und lässt sich beispielsweise in HTML einbetten. Dieser Umstand ist in Kombination mit den Darstellungsprogrammen, die z.B. mit Adobe Acrobat ausgeliefert werden, ein wichtiger Vorteil für den praktischen Einsatz. Durch die Verwendung von SVG lassen sich Proviado-Prozessdarstellungen einfach in Web-Portale und Web-Anwendungen integrieren (vgl. Anforderungen 6-11 und 6-12).

Wie mehrfach erwähnt, werden im Proviado-Template-Mechanismus die Symbole durch reines SVG beschrieben. Die Pfadausdrücke, die die Verbindung zwischen Symbol und Parameter herstellen, basieren auf XPath [BBC+07]. XPath ist ein Standard zur Navigation innerhalb von XML-Dokumenten. Auch hierfür sind zahlreiche Implementierungen vorhanden, die in vielen XML-Bibliotheken standardmäßig integriert sind.

Durch die Trennung von Symbolen und Parametern in der Template-Definition lassen sich die Symbole unter Verwendung von SVG-Editoren sehr einfach definieren (vgl. Anforderung 6-7). Zu diesem Zweck wird der entsprechende SVG-Code in den `symbol`-Bereich der Template-Definition kopiert (vgl. Abbildung 6.5). Durch die nicht-invasive Beschreibung der Parameter (d.h. die Parameter werden spezifiziert, ohne Änderungen am SVG-Code vorzunehmen) ist zudem ein Round-Trip-Engineering möglich. Die Symboldefinitionen können aus der Template-Definition in einen SVG-Editor kopiert, dort bearbeitet und anschließend wieder zurückkopiert werden, ohne dass manuelle Anpassungen am erstellten SVG-Code vorgenommen werden müssen. Lediglich die XPath-Ausdrücke müssen bei strukturellen Änderungen am Symbol angepasst werden.

Der Proviado-Template-Mechanismus verwendet Skriptausdrücke, um auf Prozessattribute zuzugreifen oder um komplexe Berechnungen innerhalb der Templates zu ermöglichen. Für die Realisierung dieser Ausdrücke wird auf ECMAScript [ECM99] zurückgegriffen, das unter dem Namen JavaScript geläufiger ist. Die Skriptsprache besitzt eine hohe Ausdruckmächtigkeit und kann durch die gute Integration in Java auf das Java-Objektmodell zugreifen [Rhi06]. Eine

weitere Anwendung findet ECMAScript in den Tooltipps. Hier werden in SVG eingebettete Skripten benutzt, um auf dem Client das Ein- und Ausblenden der Tooltipps zu steuern.

Bei der abschließenden Formatierung der Prozessvisualisierung kommt ein weiterer Standard des W3C zum Einsatz. Mit Hilfe von Cascading Style Sheets (CSS) [LJBL98] lässt sich die Formatierung der SVG-Objekte unabhängig von der Form der Symbole beschreiben. In erster Linie zählen hierzu die Farbgebung, Linienformate, Schriftgrößen und Schriftarten. Auf diese Art und Weise kann die Gestalt einer Prozessgraphik einfach an die Anforderungen angepasst werden.

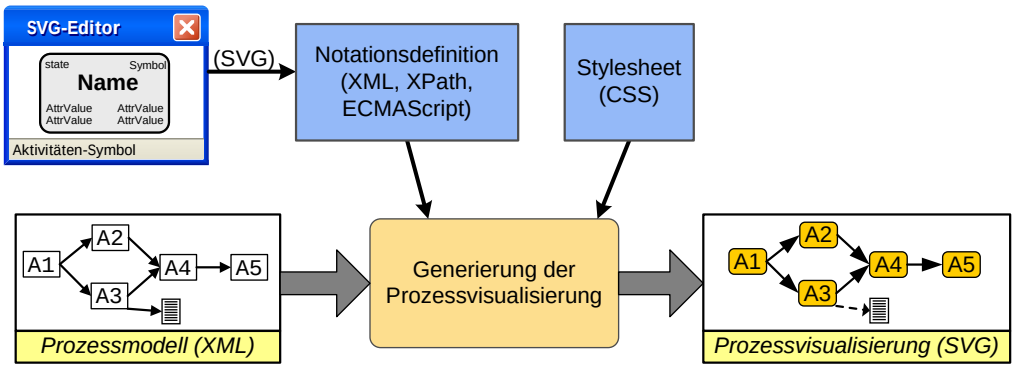


Abbildung 6.20: Zusammenwirken der Standardformate im Template-Mechanismus

Abbildung 6.20 illustriert das Zusammenspiel der verschiedenen im Template-Mechanismus verwendeten Standardformate. Zusammenfassend lässt sich festhalten, dass durch die kombinierte Verwendung von zahlreichen Standards in Proviado in geschickter Art und Weise deren jeweilige Stärken ausgenutzt und zugleich die Komplexität der Implementierung reduziert werden konnte. In Template-Definitionen ist der Anteil an Elementen, die auf einem Standard beruhen, sehr hoch. Lediglich die Verarbeitungslogik des Templates Mechanismus musste selbst implementiert werden [KH06]. Weitere Details zur Implementierung des Template-Mechanismus und des Gesamtsystems sind in Kapitel 8 zu finden.

6.4 Konfiguration der Gesamtdarstellung

Wir haben vorangehend gezeigt, wie die Notation einer Prozessvisualisierung definiert und deren Verwendung konfiguriert werden kann. Um zu einer personalisierten Prozessdarstellung zu gelangen, müssen neben der Notationsdefinition noch weitere Konfigurationsparameter zu einer Gesamtkonfiguration zusammengefasst werden, etwa das darzustellende Prozessmodell und gegebenenfalls die darauf anzuwendende Prozess-View zur strukturellen Anpassung des Prozesses (vgl. Kapitel 4 und 5).

Definition 6.3 (Visualisierungsmodell) Ein *Visualisierungsmodell* ist ein logisches Modell, das alle Informationen zusammenführt, die für die Darstellung eines Prozessmodells (bzw. einer Prozessinstanz) benötigt werden.

Die konkreten Bestandteile eines Visualisierungsmodells zeigt Tabelle 6.2.

Prozessmodell	Das der Darstellung zugrunde liegende Prozessmodell.
Prozessinstanz	Darzustellende Instanz.
View-Definition	Beschreibung der zu bildenden View. Grundlage für die View-Bildung ist das Prozessmodell bzw. die Prozessinstanz.
Darstellungsform	Gewünschte Darstellungsform (z.B. Prozessgraph oder Tabelle).
Notation	Zu verwendende Notation bestehend aus Template-Definitionen und deren Verwendungsvorschrift.
Stylesheet	Ein zur Notation passendes Stylesheet.
Layout-Parameter	Parameter, die für die Berechnung des Prozessgraph-Layouts benötigt werden.
Zusatzobjekte	Objekte, die neben den im Prozessmodell (Prozessinstanz) vorhandenen Objekten dargestellt werden sollen.

Tabelle 6.2: Bestandteile eines Visualisierungsmodells

Bei der Prozessvisualisierung sollen neben den Elementen des Prozessmodells selbst (Aktivitäten, Datenelemente, Bearbeiter, etc.) oftmals noch Zusatzobjekte dargestellt werden (vgl. Anforderung 6-9). Typische Beispiele hierfür sind Überschriften, Beschriftungsfelder, Legenden und Applikationsdaten. Diese Objekte werden ebenfalls im Visualisierungsmodell definiert. Mit den Methoden des Proviado-Template-Mechanismus besteht zum einen Zugriff auf beliebige Prozess- und Applikationsdaten, zum anderen lässt sich definieren, wie die Darstellung erfolgen soll. Zusätzliche Templates und Verwendungsvorschriften können spezifiziert werden, welche bei der Darstellungsgenerierung der vorhandenen Notationsdefinition hinzugefügt und mit verarbeitet werden. Die Zusatzobjekte ermöglichen es, die Prozessvisualisierung durch die Darstellung zusätzlicher Informationen aufzuwerten.

Ein Grundprinzip, das wir bei allen Überlegungen in Proviado berücksichtigt haben, ist die Trennung von Daten und deren Präsentation. Das Visualisierungsmodell stellt die Verbindung zwischen ihnen her, indem es beschreibt, wie die Daten eines Prozessmodells dargestellt werden sollen. Dementsprechend lassen sich mit verschiedenen Visualisierungsmodellen angewandt auf denselben Quellprozess unterschiedliche Darstellungen erzeugen (siehe Abbildung 6.21).

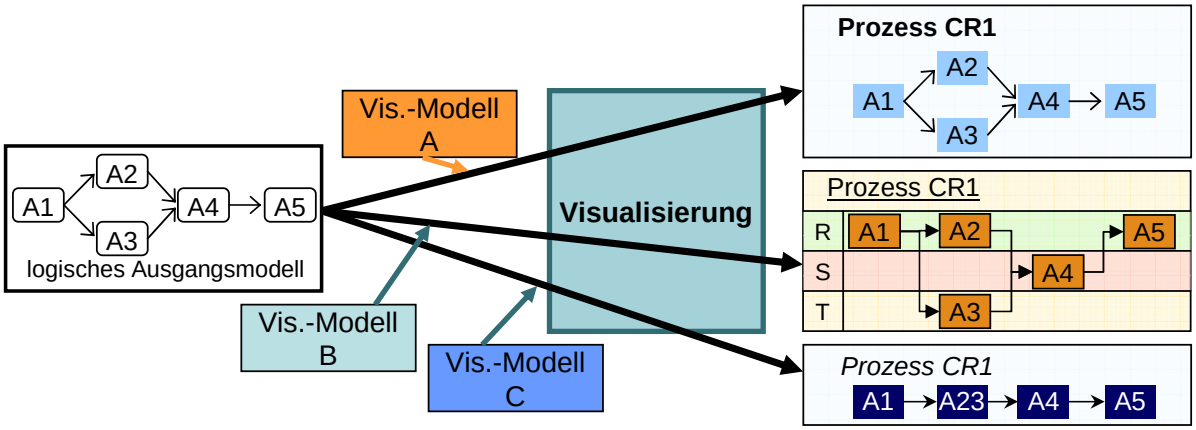


Abbildung 6.21: Trennung von Daten und deren Präsentation durch das Visualisierungsmodell

6.4.1 Parametrisierung des Visualisierungsmodells

In der Praxis werden Visualisierungen von einer großen Anzahl von Prozessen in unterschiedlichsten Formen gefordert. Das bislang beschriebene Visualisierungsmodell wäre hierfür noch zu unflexibel, um diesen dynamischen Anfragen gerecht zu werden. In der jetzigen Form muss für jede mögliche Darstellung ein komplettes Visualisierungsmodell definiert werden. Selbst die Darstellung mehrerer Instanzen desselben Prozessschemas erfordert ein neues Visualisierungsmodell.

Die benötigte Flexibilität erhält man durch die Einführung von Parametern im Visualisierungsmodell. Die Parameter werden zur Laufzeit mit konkreten Werten belegt, die entweder direkt aus der Anfrage des Benutzers stammen oder auf Laufzeitdaten basieren. Aufgrund dieser Werte kann im Visualisierungsmodell zwischen mehreren vorgesehenen Varianten gewählt werden. Beispielsweise lassen sich so unterschiedliche Benutzer(-gruppen) mit verschiedenen Stylesheets versorgen. In einer anderen Ausgestaltung der Parametrisierung übergibt ein Parameter die Instanz-ID, so dass nun ein Visualisierungsmodell für die Darstellung beliebig vieler Instanzen des Prozesses genügt.

6.4.2 Semantisches Zoomen durch Parametrisierung des Visualisierungsmodells

Ein Konzept aus dem Bereich der Informationsvisualisierung (Information Visualization) ist der semantische Zoom [PF93, FB95, WLS98]. Hierbei geht es darum, den Detaillierungsgrad einer Darstellung abhängig von der jeweiligen Zoom-Stufe zu variieren. D.h. beim Herauszoomen aus einer Graphik wird die Anzahl der angezeigten Attribute reduziert und die verbleibenden Elemente werden relativ zu den anderen Elementen vergrößert. Dadurch bleiben die wichtigsten Attribute selbst bei sehr niedrigen Zoom-Stufen lesbar und der Betrachter kann sich einen Überblick über die Darstellung verschaffen. Umgekehrt werden beim Heranzoomen immer mehr Details dargestellt. Übertragen auf die Prozessvisualisierung wird man bei geringer Zoom-Stufe die Aktivitäten nur durch kleine Rechtecke andeuten, wodurch aber die Gesamtstruktur des Prozesses gut erkennbar wäre. Je weiter der Benutzer in die Darstellung hineinzoomt, desto mehr Details einer Aktivität würden dargestellt: zunächst nur der Titel, nach und nach auch Zustand, Start- und Endezeitpunkte und weitere Detaildaten.

Mit dem in diesem Kapitel vorgestellten Proviado-Template-Mechanismus lässt sich das Semantische Zoomen einfach realisieren. Dazu werden mehrere Notationen definiert, die jeweils verschieden viele Detaildaten zur Anzeige bringen. Als Parameter des Visualisierungsmodells verwendet man anschließend die Zoom-Stufe, um die Notation mit dem passenden Detaillierungsgrad auszuwählen. Eine alternative Realisierung des Semantischen Zooms mittels SVG ist in [MMT02] beschrieben.

Konzeptionell ist der Semantische Zoom mit den Prozess-Views vergleichbar. Beide erzeugten abstrahierte Darstellungen eines Prozesses. Im Detail unterscheiden sich die Konzepte jedoch fundamental. Prozess-Views entfernen Details aus dem Prozess, indem sie dessen Struktur verändern und Elemente entfernen (Reduktion) bzw. zusammenfassen (Aggregation). Beim Semantischen Zoomen hingegen bleibt das Prozessmodell strukturell dasselbe. Abhängig vom Zoom werden mehr oder weniger Details der einzelnen Prozesselemente dargestellt.

6.4.3 Reduzierung der Symbolanzahl durch Templates

Bei der Modellierung von Prozessen werden neben dem Kontrollfluss auch Datenfluss, Bearbeiterzuordnungen und weitere Aspekte erfasst. Abhängig von der verwendeten Modellierungsmethodik bzw. der Modellierungssprache (z.B. ARIS) wird jedes Element der verschiedenen Aspekte durch ein eigenständiges Prozesssymbol dargestellt. Abbildung 6.22 zeigt einen Ausschnitt aus einem mit EPKs (ereignisgesteuerten Prozessketten [Sch01]) modellierten Prozesses. Durch die große Anzahl an Symbolen wird die Prozessdarstellung sehr unübersichtlich (vgl. Fallstudie III aus Abschnitt 2.2.3 bzw. Abbildung 2.5).

Mit den Mitteln des Proviado-Template-Mechanismus kann die Anzahl der Symbole reduziert werden, indem die Daten der zu den Aktivitäten adjazenten Prozesselemente innerhalb des Aktivitätensymbols dargestellt werden (vgl. Abbildung 6.22). Konkret definieren wir dazu ein Template, dass mittels der in Abschnitt 6.3.1 beschriebenen Parameter und den darin enthaltenen Skriptausdrücken auf die gewünschten Daten der adjazenten Prozesselemente zugreift. Innerhalb der Verwendungsdefinition wird dann festgelegt, dass dieses Template für die Darstellung der Aktivitäten verwendet werden soll. Da für die adjazenten Datenelemente keine Templates hinterlegt werden, erscheinen diese in der Prozessvisualisierung nicht als eigenständige Symbole. In unserem Beispiel werden auf diese Art und Weise die Bearbeiterinformation oberhalb des Aktivitätennamens und die bearbeiteten Dokumente unterhalb des Namens dargestellt.

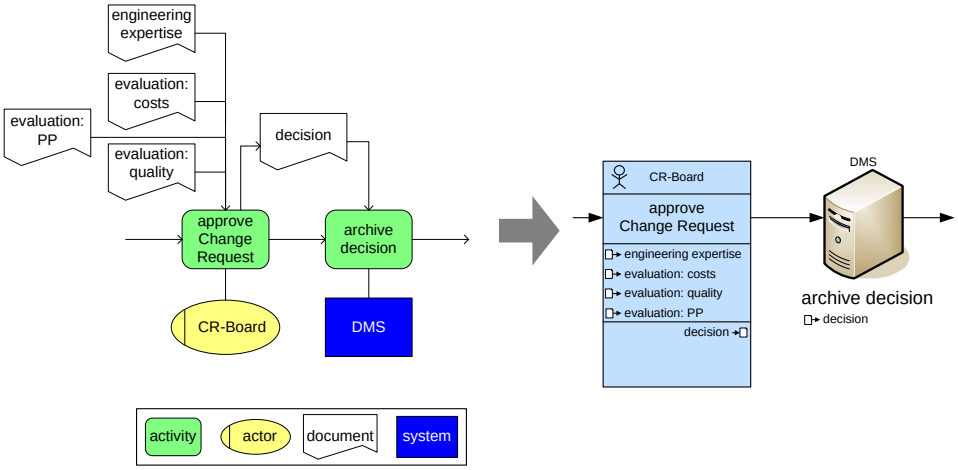


Abbildung 6.22: Integration von Daten verschiedener Prozesselemente in ein Symbol

6.5 Diskussion

Wir haben gezeigt, wie ein logisches Prozessmodell (inkl. seiner Daten) in eine konkrete graphische Repräsentation überführt werden kann. Das Grundprinzip hierbei ist die Trennung von Daten und deren Darstellung, wie es beispielsweise auch vom Model-View-Controler-Pattern (MVC) verfolgt wird [Ree79, GHJV95]. Der Fokus des Proviado-Ansatzes bildet die einfache Definition von Prozessnotationen und die flexible Konfiguration der Verwendung der Symbole. In der Literatur und in der Praxis existieren einige verwandte Ansätze, die nachfolgend diskutiert werden. Die Prinzipien, nach denen diese Ansätze vorgehen, sind in Abbildung 6.23 zusammengefasst.

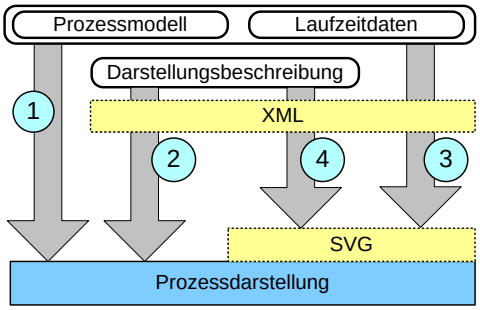


Abbildung 6.23: Unterschiedliche Konzepte zur Visualisierung von Prozessen

Variante 1 überführt einen Prozess direkt in ein graphisches Endformat, ohne Möglichkeiten der Konfiguration und damit der Einflussnahme auf das Aussehen der Prozessvisualisierung. Die Ausgangsdaten liegen meist in einem proprietären Format vor, welches direkt in ein Raster- oder Vektorgraphikformat überführt wird. Zu finden ist dieser Ansatz in den meisten Visualisierungskomponenten kommerzieller Prozesswerkzeuge (ARIS WebPublisher [IDS07b], MID Innovator [MID06], IBM WBI Modeller [IBM06, WAM+07]). Die Darstellung des Prozesses wird exakt so übernommen, wie sie vom Prozessdesigner angelegt wurde.

Bei Variante 2 kann die Generierung der Graphik durch eine Darstellungsbeschreibung, die in unserem Ansatz der Rolle des Visualisierungsmodells entspricht, beeinflusst werden. Dadurch erreicht man einen höheren Grad an Variabilität, was das Aussehen des fertigen Bildes anbetrifft. Ein Ansatz aus dieser Kategorie ist zum Beispiel der ARIS Business Publisher [IDS07a]. Da die Ausgangsdaten in vielen Fällen in XML verfügbar sind, fallen unter diese Kategorie aber auch viele allgemeine Visualisierungsrahmenwerke, die Basisdaten durch Programmbibliotheken in Bilder verwandeln. JViews [ILO04] ist ein Beispiel hierfür. Diese Visualisierungskomponente konvertiert Prozessgraphen unter Zuhilfenahme einer Konfigurationsdatei in graphische Darstellungen. Es bietet ähnlich dem hier vorgestellten Template-Mechanismus die Möglichkeit, Symbole zu definieren und diese in der Visualisierung einzusetzen. Die Graphiken können mit dem mitgelieferten Programm betrachtet oder in SVG exportiert werden. Nachteilig ist bei dieser Realisierung die aufwendige Konfiguration und Neuerstellung einer Notation. Die Verwendung von Graphikprogrammen für das Zeichnen der Prozesssymbole ist nicht möglich. Weiter gehende Personalisierungsmethoden, wie Views und Darstellungsformen, finden keine Berücksichtigung.

Variante 3 charakterisiert sich dadurch, dass die Prozesse gemäß einer festen Abbildungsvorschrift in SVG konvertiert werden. Die Notation wird im Voraus definiert und fix in den Code eingebettet. In [MBN04] wird ein Ansatz vorgestellt, der ARIS Modelle in SVG verwandelt. Ein ähnlicher Ansatz, der aus Petrinetz-Beschreibungen eine graphische Repräsentation derselben generiert, ist in [DV02] beschrieben. Liegen die Quelldaten in XML vor kann man alle Technologien, die XML-Formate ineinander überführen, zu dieser Variante zählen, da SVG ebenfalls ein XML-basiertes Format ist. [Jol05] gibt einen Überblick über gängige Technologien, wie XSLT (Extensible Stylesheet Language Transformations [Kay07]) und sXBL (SVG's XML Binding Language [Hic07]), und vergleicht diese mit dem Ansatz aus JViews. Bekanntester Vertreter ist XSLT, eine Sprache zur Konvertierung von XML-Formaten. Für die Zwecke der Prozessvisualisierung reicht die Mächtigkeit der Sprache zunächst aus. Jedoch ist die Definition einer solchen Transformation sehr aufwendig. Da eine saubere Trennung von Symbolen und

Logik wie im Template-Mechanismus nicht möglich ist, sind die resultierenden Skripte später nur schwer wartbar.

Variante 4 repräsentiert den in diesem Kapitel verfolgten Ansatz. Basierend auf den Quelldaten wird anhand einer Darstellungsbeschreibung eine Visualisierung des Prozesses generiert. Zielformat ist in unserem Fall SVG. Lösungen, die dieser Realisierungsvariante entsprechen, gibt es nach unserem Wissensstand bisher nicht.

Die Zuordnung einer graphischen Repräsentation zu einem abstrakten Modell sind Gegenstand der Visual Language Theory (VLT) [NH98]. Ziel der VLT ist, einer Visualisierung eine formale und exakte Semantik zu geben. [Fil06, FH06, FK06] versuchen die Erkenntnisse der VLT mit Hilfe einer Ontologie auf die Prozessvisualisierung zu übertragen. Abbildung 6.24 gibt die Idee des Ansatzes wider. Dabei stellt die Ontologie die Verbindung zwischen Modell und Visualisierungsobjekten her. Der Ansatz ist vom Ziel ähnlich zu dem hier vorgestellten Template-Mechanismus. In Bezug auf die Flexibilität der Verwendung von Templates und die Einfachheit der Definition von Notationen kann er aber einem Vergleich nicht standhalten, da beispielsweise eine dynamische Zuordnung von Symbolen zu Prozesselementen nicht angeboten wird.

6.6 Zusammenfassung

Prozessmodelle sind abstrakte Datenstrukturen, die erst durch die graphische Darstellung für den Benutzer greifbar werden. Eine graphische Prozessdarstellung wird definiert durch die verwendete Notation. Für eine Prozessvisualisierungskomponente ist es wichtig, unterschiedliche Notationen und deren Elemente flexibel einsetzen zu können. Mit dem vorgestellten Proviado-Template-Mechanismus haben wir in diesem Kapitel ein Konzept vorgestellt, das genau dies möglich macht. Beliebig definierte Symbole können zu einer Notation zusammengestellt werden. Die Inhalte der Symbole lassen sich frei konfigurieren. Dabei können Daten aus Applikationssystemen in die Symboldarstellung integriert werden. Der Proviado-Template-Mechanismus erlaubt es, dynamisch, (d.h. abhängig von Laufzeitdaten) zu entscheiden, wann welches Symbol zum Einsatz kommen soll. Diese Mächtigkeit deutet Abbildung 6.25 an, die zwei Darstellungen des Prozesses aus Beispiel 6-1 mit verschiedenen Notationen zeigt.

Mit Abschluss dieses Kapitels haben wir alle relevanten Mechanismen zur Personalisierung von Prozessdarstellungen kennengelernt.

- Änderung der Informationsdichte durch Prozess-Views
- Anpassung der Art der Präsentation durch verschiedene Darstellungsformen
- Adaption des Aussehens durch Anpassung der Notation

Der Template-Mechanismus, gepaart mit den Möglichkeiten der Views, versetzen uns in die Lage, beliebige Prozessdarstellungen nach den Anforderungen des Nutzers zu erzeugen. Prozesse lassen sich durch den Einsatz von Views und geeigneten Notationen in beliebigem Detaillierungsgrad darstellen.

Für die Darstellung von Ausführungsdaten innerhalb einer Prozessvisualisierung benötigen wir ein Konzept, um auf die Daten aus den unterschiedlichen prozessausführenden Systemen zu

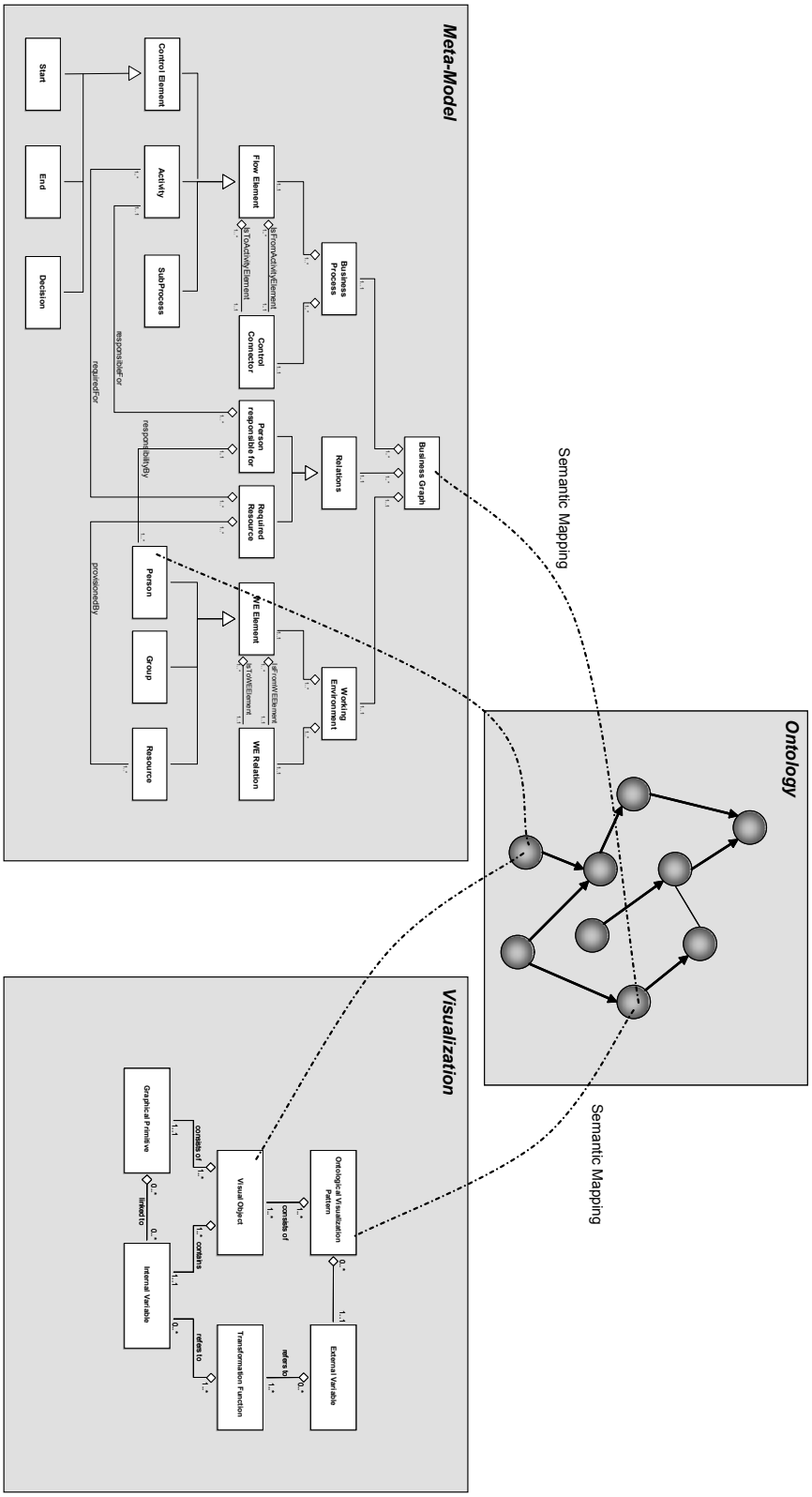
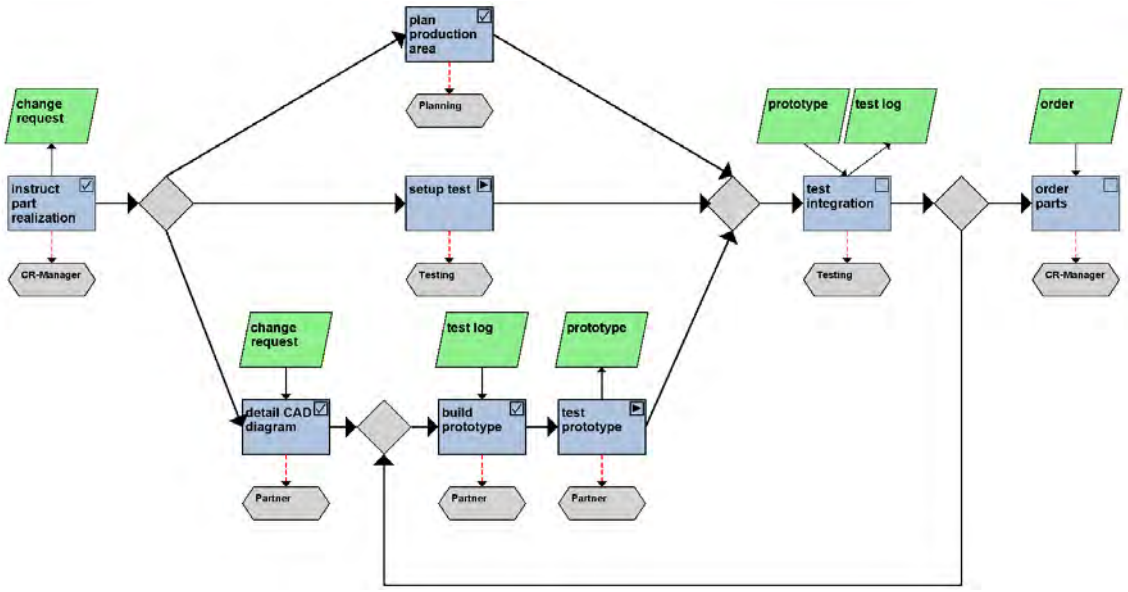
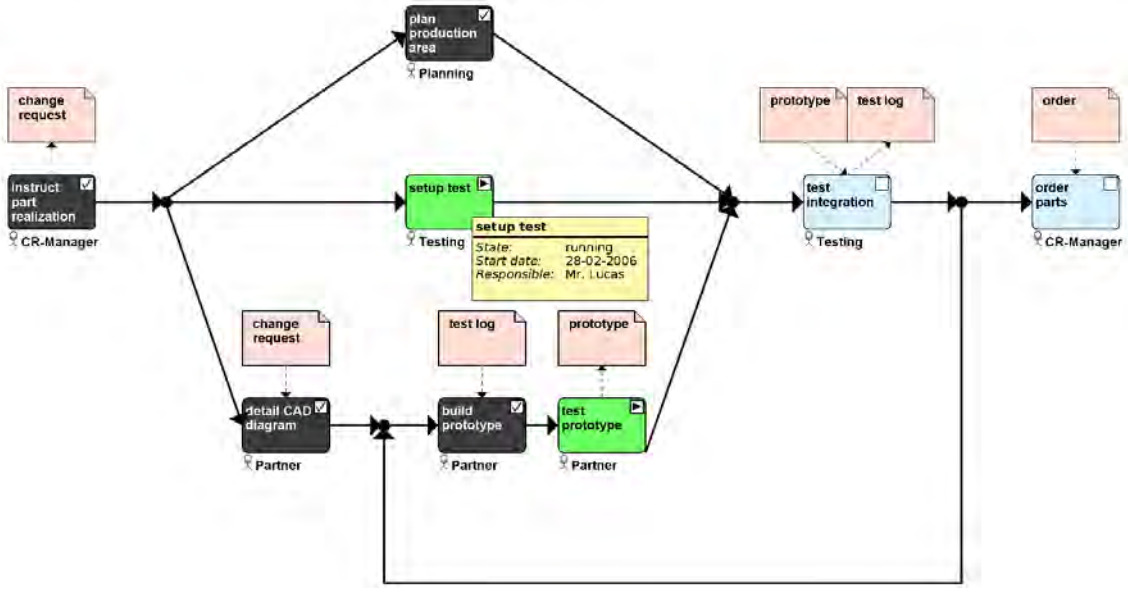


Abbildung 6.24: Semantisches Mapping von Prozessmetamodell und -notation [FK06]



(a)



(b)

Abbildung 6.25: Änderungsmanagement-Prozess mit zwei unterschiedlichen Notationen

zugreifen. Diesen und weitere Aspekte für einen umfassenden, praxistauglichen Visualisierungsansatz betrachten wir in Kapitel 7.

7

Weitere Aspekte der Prozessvisualisierung

7.1 Motivation

In den vorangehenden Kapiteln haben wir mächtige Konzepte zur Bildung von Prozess-Views sowie zur flexiblen Konfiguration der Prozessnotation entwickelt. Für einen umfassenden Ansatz zur Prozessvisualisierung, der für einen Einsatz in der Praxis in Frage kommt, fehlen jedoch noch Komponenten, die hier kurz beleuchtet werden.

Abschnitt 7.2 diskutiert verschiedene Aspekte der Prozessdatenintegration zur Modellierungs- bzw. Laufzeit. Sie bilden eine wichtige Grundlage für jeden Prozessvisualisierungsansatz, wenn dieser unabhängig von einem bestimmten Werkzeug sowie für die Visualisierung fragmentierter Prozesse zum Einsatz kommen soll. Anschließend präsentiert Abschnitt 7.3 die Herausforderungen, die sich bei der Berechnung der Position der Prozesselemente auf der Zeichenebene (Prozessgraph-Layout) ergeben, und skizziert hierzu zwei im Rahmen von Proviado realisierte Lösungsansätze.

7.2 Datenanbindung

Zu visualisierende Prozesse bzw. deren Implementierungen sind häufig über mehrere Systeme fragmentiert bzw. verteilt. Beispielsweise können Teile des Gesamtprozesses durch ein ERP-System (Enterprise-Resource-Planning) realisiert sein, während andere Anteile durch ein Workflow-Management-System oder ein prozessorientiertes Informationssystem implementiert sind. Typischerweise basieren die Schemadaten der verschiedenen Prozessfragmente nicht auf denselben Metamodellen. Zudem sind die Instanz- und Applikationsdaten des jeweiligen Prozesses auf verschiedene Systeme mit jeweils eigenen Datenformaten und Zugriffsmechanismen verteilt. Für eine integrierte Visualisierung von Prozessen (Schemata und Instanzen) müssen diese Daten integriert werden, d.h. die Heterogenität muss beseitigt werden.

7.2.1 Kanonisches Prozessmetamodell

Als Grundlage für die Visualisierung benötigen wir eine einheitliche Prozessdatenbasis auf Grundlage eines kanonischen Metamodells. Dieses Metamodell muss in der Lage sein, die Prozessmodelle der unterschiedlichsten Quellsysteme abzubilden. Da es sich um ein Metamodell für die Visualisierung von Prozessen handelt, kann auf eine streng formale Ausführungssemantik verzichtet werden.

Daneben existieren weitere Anforderungen, denen ein solches Prozessmetamodell gerecht werden sollte [Men04]. Ein wichtiger Aspekt betrifft die *Vollständigkeit* der angebotenen Modellierungskonstrukte. Sie ist erforderlich, um die in der Praxis auftretenden Abläufe möglichst exakt abbilden und darstellen zu können. Eine Visualisierungskomponente kann natürlich nur diejenigen Aspekte anzeigen, für deren Abbildung das Prozessmetamodell entsprechende Konstrukte bereitstellt. Neben der Vollständigkeit des Metamodells garantiert dessen *Allgemeingültigkeit*, dass es für alle möglichen Szenarien eingesetzt werden kann. Dabei ist es wichtig, dass die Konzepte etablierter Werkzeuge abgedeckt sind. Trotz der Mächtigkeit des Modells muss dessen *Verständlichkeit* gewahrt bleiben. Nur wenn die Benutzer den Aufbau eines Prozessmetamodells in kurzer Zeit verstehen und nachvollziehen können, bleibt es handhabbar und kann in der Praxis eingesetzt werden. Die *Eindeutigkeit* und klare Verständlichkeit der Bezeichner stellen sicher, dass es zu keinen Verwechslungen aufgrund von mehrdeutigen Namen kommt. Dazu muss die Semantik der verwendeten Elemente klar beschrieben sein. Zu guter Letzt ist die *Erweiterbarkeit* zu nennen. Sie ist erforderlich, wenn das Metamodell an sich ändernde Anforderungen angepasst werden können soll und wenn zusätzliche Attribute und Elemente aufgenommen werden.

In [Bob04b] werden existierende Prozessmetamodelle und -notationen systematisch untersucht. [Bob05] beschreibt das für die Prozessvisualisierung in Proviado entwickelte Prozessmetamodell im Detail. Abbildung 7.1 zeigt das Objektdiagramm dieses Metamodells. Eine detaillierte Darstellung mit Attributen der Modellelemente findet sich im Anhang B dieser Arbeit. Das Proviado-Metamodell orientiert sich an allgemeinen Graphen. Daher sind die zentralen Elemente des Modells durch *Knoten* und *Kanten* gegeben. Durch das Element *Prozess* wird ein vollständiges Prozessmodell repräsentiert. Zusammen bilden *Knoten*, *Kanten* und *Prozess* die drei Subtypen von *Prozesselement*.

Zur Abbildung der Elemente eines Prozesses dienen die verschiedenen Subtypen von *Knoten*. Das Proviado-Metamodell kennt die Knotentypen *Aktivität* und *Strukturknoten* für die Abbildung des Kontrollflussaspekts, *Datenelement* für den Datenflussaspekt sowie *Ausführungseinheit* für die Repräsentation der Bearbeiter von Aktivitäten. Ferner subsumiert *Ressource* alle Systeme oder sonstigen Ressourcen, die bei der Prozessausführung eine Rolle spielen. Ein *Scope* definiert einen Bereich, d.h. eine Menge von Knoten, innerhalb dessen beispielsweise auftretende Ausnahmesituationen durch *Ereignis*-Elemente aufgefangen werden. *Partition*en werden verwendet, um die Aktivitäten in Gruppen zu unterteilen, wie zum Beispiel von BPMN [Bus06] oder UML2 [UML04] bekannt. Zur Modellierung eines Informationsaustausches, auch über Prozessgrenzen hinweg, dienen *Nachricht*en.

Knoten werden durch Kanten zu einem Prozessmodell verknüpft: für den Kontrollfluss, d.h. die Verbindung von Aktivitäten, Strukturknoten und Ereignissen untereinander, dienen *Kontrollflusskant*en. Der Datenfluss wird durch *Datenflusskant*en abgebildet. Für die Zuordnung

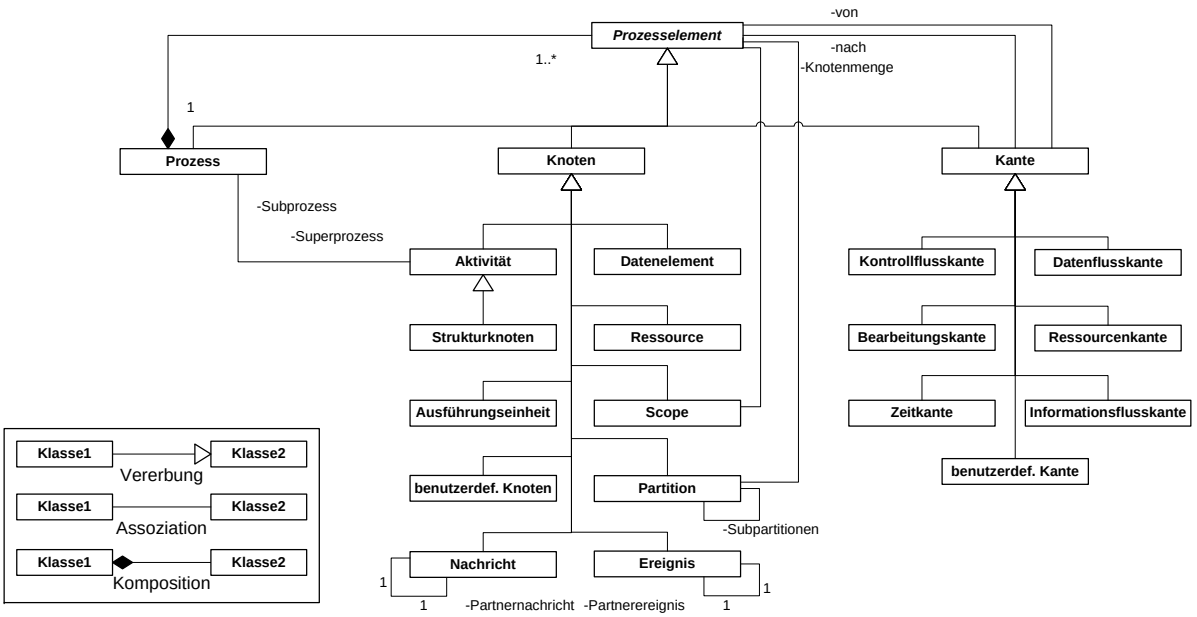


Abbildung 7.1: Kanonisches Metamodell für die Prozessvisualisierung (UML-Klassendiagramm)

der Ausführungseinheiten zu Aktivitäten verwenden wir *Bearbeiterkanten*, analog *Ressourcenkanten* für Ressourcen. Zeitliche Abhängigkeiten zwischen Prozesselementen können mit Hilfe von *Zeitkanten* modelliert werden. Zur Darstellung eines Informationsaustausches, beispielsweise zwischen Bearbeitern, werden diese mittels *Informationsflusskanten* und *Nachrichten* verbunden.

Das Metamodell ist in dieser Form für die Abbildung von Prozessschemata ausgelegt. Für Prozessinstanzen sind weitere Attribute für Laufzeitdaten erforderlich (siehe Anhang B).

Mit den vorgestellten Metamodellobjekten sind wir in der Lage, die gängigen Prozessmetamodelle in unserem Ansatz abzubilden. Darüber hinaus ist das Proviado-Metamodell mittels benutzerdefinierbarer Knoten und Kanten erweiterbar.

7.2.2 Prozessmodelltransformation und -integration

Wie erwähnt, ist eine Voraussetzung für die integrierte Visualisierung systemübergreifender Prozesse, dass die entsprechenden Prozessmodelle in einem einheitlichen Format vorliegen. Zu diesem Zweck haben wir in Abschnitt 7.2.1 das Proviado-Metamodell skizziert. Wir müssen nun die Prozessmodelle aus den verschiedenen Quellsystemen in dieses generische Format konvertieren. [Buc05, Mih05] befassen sich im Detail mit dieser Thematik, wir geben hier nur einen kurzen Überblick.

Ausgangspunkt des Proviado-Ansatzes sind die in verschiedenen Systemen verteilt ablaufenden Prozesse, deren Modell- und Laufzeitdaten für die Visualisierung in ein einheitliches Format übertragen werden müssen. Abbildung 7.2 fasst die dabei zu bewältigenden Aufgaben zusammen:

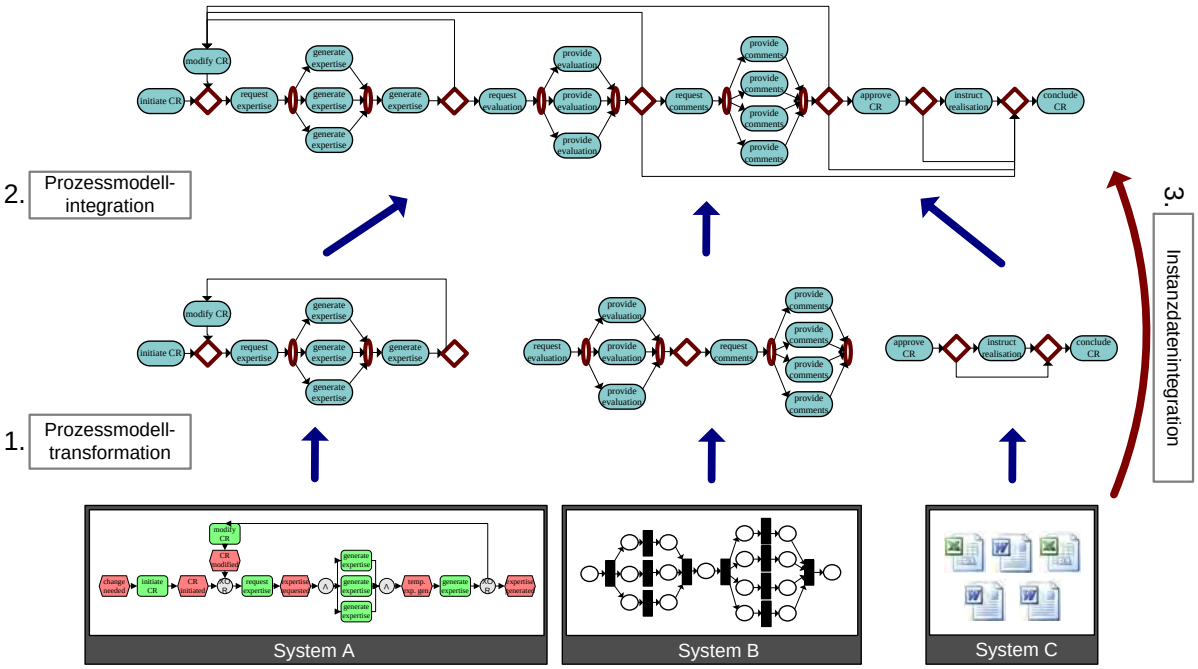


Abbildung 7.2: Prozessmodelltransformation und -integration

1. **Prozessmodelltransformation** Die Prozessmodelle der verschiedenen Quellsysteme müssen aus dem jeweiligen proprietären Format in das von Proviado vorgegebene kanonische Format transformiert werden.
2. **Prozessmodellintegration** Die einzelnen Prozessfragmente, die den Teilprozessen der verschiedenen Quellsysteme entsprechen, müssen in ein Gesamtmodell integriert werden.
3. **Instanzdatenintegration** Die Instanzdaten der Prozessfragmente müssen für eine Visualisierung integriert werden.

Die folgenden beiden Abschnitte diskutieren die speziellen Herausforderungen bei der Transformation und Integration von Prozessmodellen. Die Instanzdatenintegration wird in Abschnitt 7.2.3 genauer betrachtet.

7.2.2.1 Prozessmodelltransformation

Je nach Art des zugrunde liegenden Quellsystems bieten sich unterschiedliche Vorgehensweisen für die Prozessmodelltransformation an. Grundsätzliche Möglichkeiten sind in Abbildung 7.3 dargestellt.

Import bzw. direktes Mapping Das Prozessmodell des Quellsystems wird direkt importiert. Dabei werden einzelne Objekte des Quellmetamodells oder Kombinationen derer auf entsprechende Objekte des Zielmetamodells (d.h. des Proviado-Metamodells) abgebildet.

Extract Transform Load (ETL) Ist ein direkter Import nicht möglich, kann das Quellmodell auf geeignete Zwischenmodelle abgebildet werden, um schließlich im Proviado-Metamodell geladen zu werden.

Process Mining Bei manchen (Legacy-)Systemen existiert kein explizites Prozessmodell, sondern die Ablauflogik ist hart im Programm-Code verankert. Existieren aber detaillierte Log-Dateien, in denen Start und Ende der Prozessaktivitäten dokumentiert sind, so kann daraus mit Hilfe von Process-Mining Techniken ein Modell des abgelaufenen Prozesses erstellt werden [Her03, AWM04].

Nachmodellieren Im ungünstigsten Fall, wenn keine der vorangehenden Techniken zum Erfolg führt, muss das Prozessmodell von Hand nachmodelliert werden. Dies ist zum Beispiel auch der Fall, wenn manuelle Schritte im Prozess enthalten sind, die systemseitig nicht dokumentiert bzw. unterstützt sind.

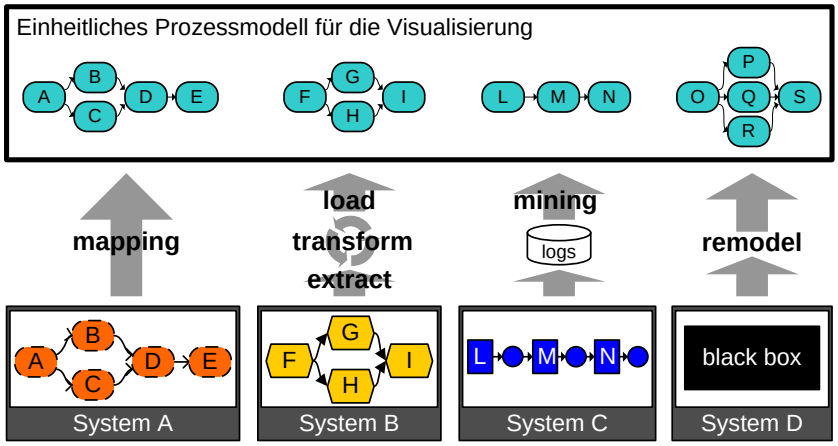


Abbildung 7.3: Arten der Transformation von Prozessmodellen

Bei allen Transformationsansätzen treten, sobald Daten automatisch konvertiert werden sollen, weitere Problemstellungen auf, wie sie aus dem Bereich der Schemaintegration bei Datenbanken bestens bekannt sind [NL06]. Dies betrifft zum Beispiel Homonyme (ein Bezeichner für unterschiedliche Konzepte) und Synonyme (unterschiedliche Bezeichner für ein Konzept), aber auch alle Probleme der Datenintegration, die aus heterogenen Formaten resultieren (z.B. Zeichenfolgen unterschiedlicher Codierung). Ein Beispiel für ein spezielles Problem der Prozesstransformation ist in Abbildung 7.4 dargestellt. Es zeigt eine alternative Verzweigung mit Verzweigungsbedingungen, wie sie mit ereignisgesteuerten Prozessketten (EPK) und unter Verwendung unseres Metamodells modelliert werden können. Bei einer angenommenen Transformation von EPKs müssen die Verzweigungsbedingungen aus den Ereignissen extrahiert und als Kantenbedingung in unser Modell übernommen werden. Ferner werden EPK-Funktionen auf Aktivitäten abgebildet. Im Allgemeinen ist die Transformation eines Prozessmetamodells auf ein anderes ein nicht triviales Problem, da für alle Elemente des Quellmetamodells Entsprechungen im Zielmetamodell definiert werden müssen. Da jedoch die Ausdrucksmächtigkeit der Metamodelle variiert, ist eine direkte Zuordnung der Metamodellelemente zueinander oftmals nicht möglich, sondern es müssen wie in obigem Beispiel mehrere Elemente (im Beispiel Ereignis und Funktion) auf ein Zielelement (im Beispiel Aktivität) abgebildet werden oder umgekehrt.

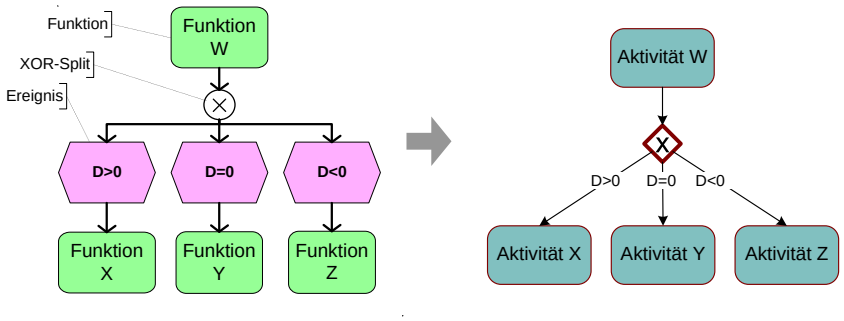


Abbildung 7.4: Transformation einer alternativen Verzweigung

Neben Arbeiten zur Transformation von Prozessmodellen im Allgemeinen [MK07] existiert in der Literatur eine Vielzahl von Arbeiten für verschiedenste Metamodelle. Sie beschreiben jeweils, wie ein Metamodell (bzw. eine Teilmenge eines Metamodells) auf ein anderes Metamodell abgebildet wird [MN03a, MNN04, HSS05, KV06, MMN06, HSS05, MZ05b, ODHA06, SMW07]. XML-Austauschformate für Prozessmodelle nehmen hierbei eine wichtige Rolle ein [NM04, NM05].

7.2.2.2 Prozessmodellintegration

Bei der Prozessmodellintegration werden die Prozessfragmente der verschiedenen Quellsysteme zu einem Gesamtprozess zusammengefügt. Im Idealfall lassen sich die Fragmente lückenlos (z.B. sequentiell) kombinieren. Abbildung 7.5 zeigt Beispiele für diese und weitere mögliche Beziehungen der Fragmente untereinander. So können sich die Prozessfragmente überlappen (b), verschränkt sein (c) oder in einer hierarchischen Beziehung stehen (d). Für alle Fälle gilt gleichermaßen, dass die Beziehungen bei der Integration zu einem Gesamtprozessmodell aufzulösen sind.

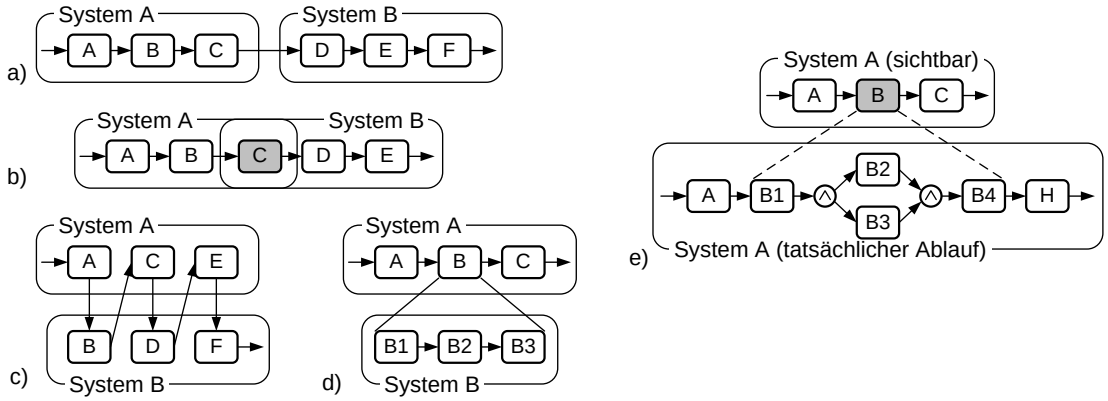


Abbildung 7.5: Beziehungen von Prozessfragmenten

Die Integration homogenisierter Prozessfragmente in ein Gesamtprozessmodell wird in der Praxis durch eine inkonsistente Modellierung, wie sie bei Beteiligung verschiedener Modellierer zustande kommt, erschwert. Da jedes Modell letztlich nur die Sichtweise des jeweiligen Modellierers

widerspiegelt, werden sich derartige Inkonsistenzen auch durch stringente Modellierungskonventionen nicht vollständig aus der Welt schaffen lassen. Die unterschiedliche Interpretation eines Ablaufs durch verschiedene Betrachter ist oftmals auch an der unterschiedlichen Granularität der Modellierung erkennbar.

Bei Black-Box-Systemen, bei denen der Prozess nicht bekannt ist und nur anhand der Ergebnisse beobachtet werden kann, lassen sich Diskrepanzen zwischen dem sichtbaren und tatsächlichen Prozess nicht vermeiden (vgl. Abbildung 7.5e). Umgekehrt treten in Prozessen mit einem hohen Anteil an Aktivitäten, die durch Menschen (und nicht automatisch) ausgeführt werden, so genannte versteckte Aktivitäten auf. Dies sind Aktivitäten, die zwar durch den jeweiligen Bearbeiter ausgeführt werden, aber im Prozessmodell nicht vorhanden sind.

7.2.3 Instanzdatenintegration

Neben den Prozessschemata sollen in Proviado auch Instanzen (inkl. ihrer Applikationsdaten) visualisiert werden. Die Workflow-Management-Coalition (WfMC) unterscheidet drei Arten von Daten, die bei einer Prozessausführung relevant sind [Hol95]:

1. *Workflow-Kontrolldaten* sind interne Statusdaten des Workflow-Management-Systems (WfMS), etwa zum Start/Ende von Aktivitäten.
2. *Workflow-relevante Daten* sind Anwendungsdaten, die dem WfMS bekannt sind und von diesem verwendet werden, um an alternativen Verzweigungen einen Ausführungspfad auszuwählen.
3. *Applikationsdaten* sind diejenigen Daten, die anwendungsspezifisch sind und die vom WfMS nicht interpretiert, sondern höchstens referenziert und an Applikationsprogramme weitergegeben werden.

Für die Visualisierung sind alle drei Datenarten interessant, das heißt sie müssen gegebenenfalls dargestellt bzw. zumindest verlinkt werden können. Konkret sind folgende Daten für eine Instanzvisualisierung relevant und müssen daher der Visualisierungskomponente zugänglich gemacht werden:

- *Start- und Endzeitpunkte von Aktivitäten und Prozessinstanzen*
- *Zustand von Aktivitäten*
- *Konkreter Bearbeiter der Aktivitäten:* Zur Modellierungszeit des Prozesses wird eine Bearbeiterformel im Modell hinterlegt, die die in Frage kommende Gruppe von ausführenden Personen beschreibt. Erst zur Laufzeit (in der Regel bei Start oder Auswahl der Aktivität) ergibt sich daraus der konkrete Bearbeiter.
- *Applikationsdaten:* Für eine aussagekräftige Visualisierung werden Applikationsdaten in die Prozessdarstellung integriert. Sie erlauben dadurch dem Betrachter beispielsweise durch die Darstellung der Teilenummer in einem Änderungsmanagement-Prozess eine schnelle Zuordnung der Instanzdarstellung zu einem konkreten Geschäftsvorfall. Während einfach strukturierte Datenwerte direkt in die Visualisierung eingebaut werden können, werden komplexere Daten (wie Dokumente oder CAD-Zeichnungen) üblicherweise verlinkt.

Bei Anbindung der verschiedenen Quellsysteme treten ebenfalls die vorangehend beschriebenen Probleme auf, die Folgen der Heterogenität der Datenformate sind. Wir fokussieren nachfolgend daher auf weitere Herausforderungen, die die Anbindung der Laufzeitdaten mit sich bringt.

7.2.3.1 Laufzeitdatenanbindung von verschiedenen Systemtypen

Zur Anbindung verschiedener Systeme benötigen wir eine generische Schnittstelle. Prinzipiell existieren zwei unterschiedliche Arten von Systemen, die getrennt von einander betrachtet werden müssen. Bevor wir eine Schnittstelle für jede der Systemarten vorstellen, beschreiben wir zunächst deren Eigenheiten.

1. **Workflow-Management-Systeme (WfMS):** WfMS verwalten Workflow-Kontrolldaten. Sie besitzen eine entsprechende API, über die diese Daten von anderen Programmen abgefragt werden können. Während der Ausführung eines Prozesses wird der Ablauf in einem Log dokumentiert. Über APIs können auch die Logs von externen Programmen abgerufen werden.
2. **Legacy-Systeme (LS):** Hierunter fallen alle Systeme bzw. Applikationen, die nicht unter der Kontrolle eines WfMS ausgeführt werden. Bei LS existiert kein expliziter Prozess, sondern die Ablauflogik ist meist in den Programmcode integriert. Somit erfolgt auch keine explizite Verwaltung des Zustandes. Daraus ergibt sich, dass Zustandsänderungen nicht immer klar abzulesen sind. Manchmal lassen sich Zustandsübergänge nur mittels Beobachtung von Daten feststellen (z.B. sobald Datei X geschrieben wurde, ist Zustand Y erreicht). Ein weiterer Unterschied zu WfMS ist, dass LS keine Historie mit Beginn und Ende der Einzelschritte schreiben. Schließlich besitzen LS auch keine API für Zugriffe auf die Ausführungsdaten.

Diese Unterschiede erfordern unterschiedliche Vorgehensweisen bei der Anbindung der Laufzeitdaten. Abbildung 7.6 zeigt eine schematische Darstellung der Schnittstellen.

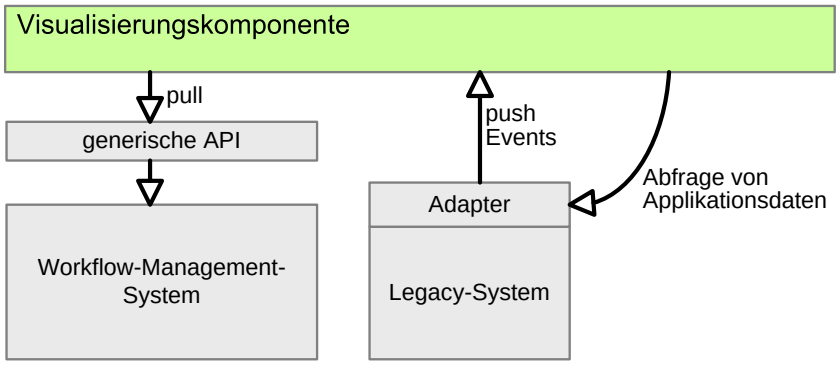


Abbildung 7.6: Anbindung von Laufzeitdaten

Fall 1 (Workflow-Management-Systeme): Die Anbindung eines WfMS erfolgt mittels einer generischen Schnittstelle. Diese Zwischenschicht übersetzt die Anfragen der Visualisierungskomponente in Befehle der WfMS-API und leitet sie an das WfMS weiter. Unter der Voraussetzung, dass das WfMS jederzeit verfügbar ist (wovon wir in unserem Szenario ausgehen), werden die benötigten Daten bei Bedarf abgefragt (Pull-Prinzip).

Fall 2 (Legacy-Systeme): Für die Anbindung eines LS verwenden wir einen Adapter, der die benötigten Daten aus dem LS zur Verfügung stellt. Auf der Seite der Visualisierungskomponente verfügt dieser Adapter über eine einheitliche Schnittstelle. Die Anbindung an das LS hängt dagegen entscheidend von dem konkret vorhandenen System ab und kann hier nicht allgemein beschrieben werden.

Da ein LS wie beschrieben in der Regel über keine Historiendaten verfügt und auch den Zustand einer Instanz nicht speichert, können die Daten in diesem Fall nicht nach dem Pull-Prinzip abgefragt werden. Stattdessen generiert der Adapter Events, welche die Visualisierungskomponente über relevante Zustands- oder Datenänderungen informiert (Push-Prinzip).

In Abschnitt 7.2.3.3 werden wir die Architektur der Laufzeitdatenschnittstelle vorstellen und erklären, wie hier ein Zugriff auf Laufzeitdaten abläuft.

7.2.3.2 Korrelation von Laufzeitdaten

Eine weitere Herausforderung bei der Anbindung von Laufzeitdaten stellt die Korrelation der Daten dar [SM04]. Wir wollen diesen Sachverhalt anhand von Abbildung 7.7 verdeutlichen. Bei einer verteilten Ausführung eines Prozesses sind in der Regel keine eindeutigen, systemübergreifenden IDs zur Identifikation zusammengehöriger Instanzen vorhanden. Im Beispiel aus Abbildung 7.7 etwa werden zwei Instanzen des Prozesses durch Systeme A und B ausgeführt. Die Zuordnung, welche Instanz von System A zu welcher Instanz von System B gehört, nennt man Korrelation. Im Beispiel ist dies durch Verwendung von Applikationsdaten (hier: Teilenummer) möglich.

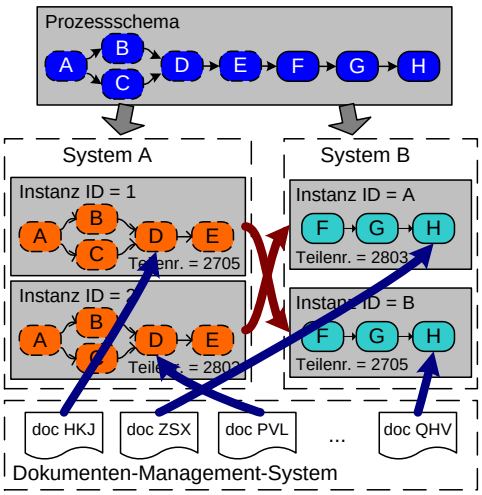


Abbildung 7.7: Korrelation von Prozessinstanzen

In Proviado korrelieren wir Instanzen anhand der in Cross-Reference-Tabellen (CRT) abgelegten Informationen (vgl. Abbildung 7.8). Der Administrator muss bei der Integration eines neuen Prozesses beschreiben, welche Applikationsdaten zur Korrelation verwendet werden sollen. In der Prozessausführungssprache BPEL etwa werden diese Zuordnungen unter Verwendung von

Applikationsdaten durch so genannte *Correlation Sets* beschrieben [OAS07]. Die CRT kommt in folgenden zwei Fällen zur Anwendung:

1. Korrelation von Events
2. Suche nach IDs für Anfrage an Quellsysteme

Zur Laufzeit wird die CRT nach und nach befüllt. Bei Legacy Systemen ist die Visualisierungskomponente für Events registriert und wird somit beim Anlegen einer Prozessinstanz über die neuen Instanz-IDs informiert. Sie kann dann einen entsprechenden Eintrag in der CRT vornehmen (vgl. Abbildung 7.8). Im Fall von WfMS bietet unsere Schnittstelle eine Operation an, die anhand von Applikationsdaten die Instanz-ID ermittelt. Später wird dann beim Eintreffen von Events in der Tabelle nachgeschlagen, welche interne ID zu der Prozessinstanz gehört. Bei Anfragen an das Quellsystem (WfMS) wird die externe ID mit Hilfe der internen ID und der CRT ermittelt.

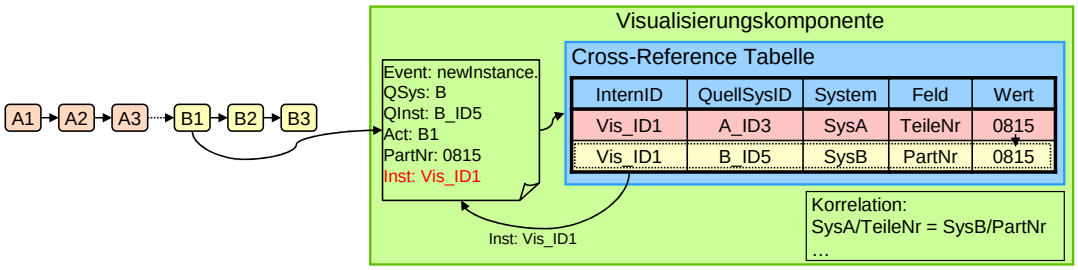


Abbildung 7.8: Cross-Reference Tabelle zur Korrelation von Instanzen

7.2.3.3 Architektur und Ablauf einer Datenanfrage

Abbildung 7.9 zeigt die Architektur der von Proviado bereitgestellten Schnittstelle für den Zugriff auf Laufzeitdaten. Dargestellt sind die beiden wichtigsten Operationen: (i) `getStatus` zur Abfrage des Aktivitätenstatus und (ii) `getAppData` zur Abfrage von Applikationsdaten. Die Informationen, welches System welche Daten bereitstellen kann, sind über das Visualisierungsmodell (vgl. Kapitel 6) zusammen mit den Prozessmodelldaten zugänglich.

Die Datenanfrage verläuft je nach Art des angebundenen Systems unterschiedlich.

Fall 1 (Workflow-Management-Systeme): Die Anfrage der Visualisierungskomponente wird an die generische API weitergeleitet (Pull). Dabei werden die internen IDs der Visualisierungskomponente in die IDs des Quellsystems übersetzt. Bei Bedarf können alle Daten zusätzlich mittels Caching-Mechanismen zwischengespeichert werden, so dass ein wiederholter Zugriff auf dieselben Daten effizient gestaltet werden kann.

Fall 2 (Legacy-Systeme): Da eine direkte Anfrage an Legacy-Systeme wegen der fehlenden Schnittstellen nicht möglich ist, verwenden wir für ihre Anbindung spezielle Adapter in Verbindung mit einem Message-Bus (Push-Prinzip). Der Adapter generiert für alle relevanten Datenänderungen (vgl. Abschnitt 7.2.3 auf Seite 171) ein Event, das über den Message Bus an die Visualisierungskomponente weitergeleitet wird. Da diese Event-Daten flüchtig sind, müssen

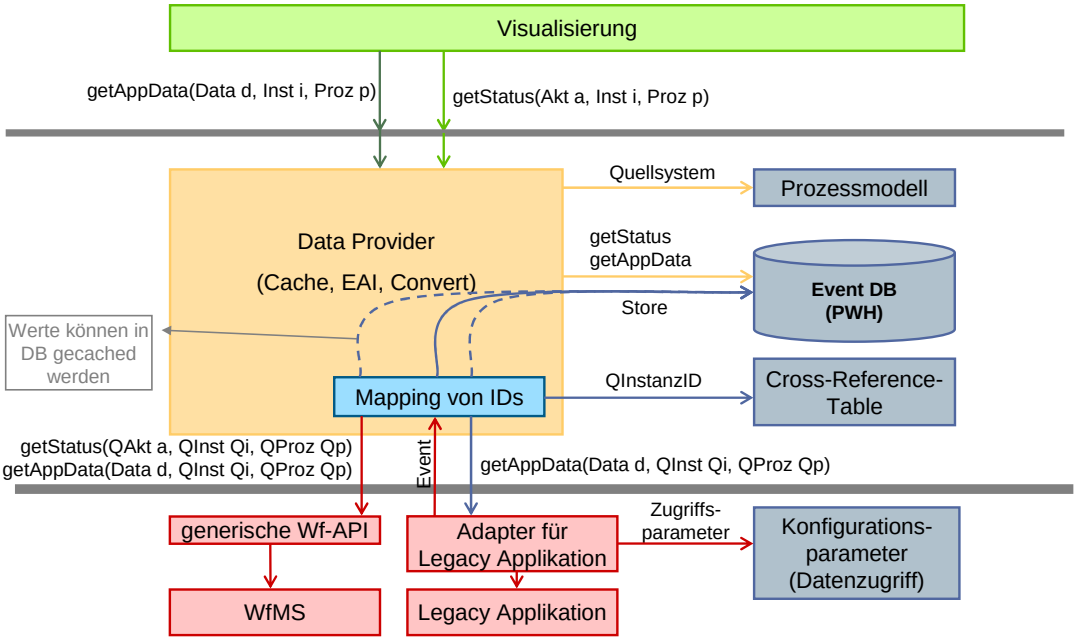


Abbildung 7.9: Architektur der Laufzeitdaten-Schnittstelle

sie in der Visualisierungskomponente gespeichert werden, um später darauf zugreifen zu können, d.h. um den Zustand von Instanzen darstellen zu können.

Ablauf einer Datenanfrage

Am Beispiel einer Zustandsanfrage (z.B. Anfrage nach dem Zustand einer Aktivität `a` in Instanz `i` vom Prozesstyp `p`, d.h. `getStatus(a,i,p)`) wollen wir nun aufzeigen, wie eine Datenanfrage in der vorgeschlagenen Architektur abläuft (vgl. Abbildung 7.9).

1. Visualisierung fragt Datum an: `getStatus(a,i,p)`
2. Der *Data-Provider* ermittelt das Quellsystem anhand der Konfigurationsdaten aus dem Visualisierungsmodell.
3. Fall Quellsystem ist WfMS:
   a) Über die Cross-Reference-Tabelle werden die IDs `Qa`, `Qi` und `Qp` für Aktivität `a`, Instanz `i` und Prozess `p` auf dem Quellsystem ermittelt.
   b) Über die generische Schnittstelle wird die Anfrage nach dem Status an das Quellsystem gestellt.

   Fall Quellsystem ist LS:
   a) Der Status der Aktivität `a` wird aus der Event-Datenbank gelesen. Ein Mapping der IDs ist nicht notwendig, da dies bei Eintreffen der Events geschieht und die Eventdaten unter der ID der Visualisierungskomponente in der Datenbank abgelegt werden.

Bei Anfrage eines Applikationsdatums werden im Fall von Legacy-Systemen die Zugriffsparameter (z.B. Datenbanktabelle, -felder, etc.) aus den Konfigurationsdaten im Visualisierungsmodell ermittelt.

7.2.4 Anbindung weiterer Datenquellen

Neben der Anbindung von prozessausführenden Systemen (WfMS und LS) kann für die Visualisierung die Anbindung weiterer Datenquellen nötig sein. Hier sind in die folgenden Systeme zu nennen:

- Systeme zur Verwaltung des Organisationsmodells, das Informationen zu den Bearbeitern der Prozessaktivitäten enthält.
- Systeme zum Enterprise-Architecture-Management, die Informationen über die IT-Infrastruktur und die in Prozessen verwendeten Systeme verwalten.
- Produktdatenmanagement-Systeme (PDM), die Informationen zu den Produktstrukturen enthalten.

Da die Schnittstellen zu diesen Systemen stark system- und anwendungsabhängig sind, gehen wir auf die weiteren Details hier nicht ein. Es sei jedoch darauf hingewiesen, dass für einige der in Abschnitt 2.2.4.2 angesprochenen Darstellungsformen die Anbindung dieser Systeme erforderlich ist (z.B. Matrixdarstellung mit Systemen und Bearbeitern als Dimensionen).

Eine andere Form der Integration, die weniger aufwendig ist als eine direkte Anbindung mittels expliziter Schnittstellen, ist die Verlinkung. Gerade bei der Visualisierung lässt sich dadurch eine einfache „Integration“ mehrerer Systeme realisieren, indem deren Visualisierungen gegenseitig verlinkt werden.

7.3 Layout von Prozessgraphen

Das Thema Graph-Layout befasst sich mit der Berechnung einer optimalen Positionierung der Graphelemente (Knoten und Kanten) auf der Zeichnungsebene [TBET98, KW01]. Für Prozessmodelle ist dies eine sehr anspruchsvolle Aufgabe. In existierenden Systemen zur Visualisierung von Prozessen, werden die Prozessgraphen meist in derselben Form dargestellt, wie sie zuvor vom Prozessmodellierer gezeichnet wurden, d.h. es treten keine Änderungen am Layout auf. Für eine statische Visualisierung von Prozessen, bei der sich die Modelle bzw. deren Darstellung nie oder nur sehr selten ändern, ist es durchaus möglich, die Layout-Daten manuell zu erzeugen und gegebenenfalls anzupassen.

In Proviado bilden derart statische Darstellungen eher die Ausnahme. Mit Hilfe der in Kapitel 4 und 5 vorgestellten Konzepte für die View-Bildung sowie der in Kapitel 6 dargestellten Möglichkeiten zur Anpassung der Templates verändern sich die Prozessmodelle in ihrer Struktur und Darstellung ständig. Beispielsweise wird durch Reduktion von Aktivitäten entsprechend Platz in der Zeichnungsebene frei, oder größere bzw. kleinere Templates machen Anpassungen am Layout nötig. In all diesen Fällen ist eine manuelle Zeichnung zu aufwendig und nicht

praktikabel. Für eine in der Praxis einsetzbare Visualisierungskomponente benötigen wir daher einen entsprechenden Algorithmus zur Berechnung von Prozessgraph-Layouts.

In der Literatur existieren für verschiedene Arten von Graphen (Graphklassen) unterschiedliche Algorithmen [BETT94, BJM97].

Prozessgraphen (Kontrollflussgraphen) können prinzipiell der Klasse der DAGs (Directed Acylic Graphs: gerichtete azyklische Graphen) zugeordnet werden. Probleme bereiten Schleifenkanten, die Zyklen in die Graphen einführen. Für DAGs existiert ein Standardverfahren, das das Problem der Zyklen-verursachenden Schleifenkanten elegant umgeht: der Sugiyama-Algorithmus [STT81, BM01]. Abbildung 7.10 zeigt die Phasen des Sugiyama-Algorithmus am Beispiel:

1. *Entfernen von Zyklen (Decycling)* In einem Vorbereitungsschritt werden zunächst alle Kanten, die einen Zyklus verursachen, entfernt. Später werden diese wieder in den Graphen eingefügt.
2. *Ebeneneinteilung* Die Knoten werden topologisch sortiert und auf Ebenen, beginnend mit dem Startknoten, verteilt. Aus den Ebenen ergeben sich später die x-Koordinaten (bei Zeichnung von links nach rechts).
3. *Kreuzungsreduzierung* Durch Vertauschen der Knoten einer Ebene lassen sich Kantenkreuzungen minimieren.
4. *Koordinatenzuweisung* Aus der Reihenfolge der Knoten auf einer Ebene können die y-Koordinaten abgeleitet werden.
5. *Zyklen wiederherstellen* Schleifenkanten, die im Schritt „Decycling“ entfernt wurden, werden nun wieder eingefügt.
6. *Kantenziehen* Durch die vorangegangenen Schritte wurden bislang erst die Koordinaten der Knoten berechnet. Nun werden die Kanten eingefügt und deren optimaler Verlauf berechnet. In der Zeichnung wurden der Übersichtlichkeit halber die Kanten schon in den vorangegangenen Schritten dargestellt.

Der Sugiyama-Algorithmus liefert, erweitert um einige Optimierungen [BK01], für Kontrollflussgraphen schöne Ergebnisse. Prozessmodelle umfassen jedoch neben dem Kontrollfluss weitere Aspekte, die bei der Zeichnung und damit auch bei der Berechnung berücksichtigt werden müssen. Wir bezeichnen diese Zusatzobjekte, die um die Aktivitäten „kreisen“ (z.B. Datenelemente oder Bearbeiterknoten) als Satellitenobjekte. Neben Satellitenobjekten müssen weitere Objekte (z.B. Überschriften, Beschreibungen, Applikationsdaten) in der Zeichnung positioniert werden. Eine automatische Berechnung der Position dieser Objekte ist ohne entsprechende Hinweise (evtl. auch genaue Positionsangaben) nicht möglich.

In der Literatur existieren einige wenige Ansätze, die sich mit dem Layout von Prozessgraphen befassen. [ST01] beschreibt einen Ansatz, in dem ein Prozess-Layout in linearer Zeit bestimmt wird, wozu auf existierende Partitionierungsinformationen (z.B. Swimlanes) zurückgegriffen wird. [YLS+04] beschreibt einen so genannten Force-Scan-Algorithmus, der die Veränderungen am Layout nach einer Prozessänderungen gering zu halten versucht. Für eine besser Übersicht und zur Vermeidung des „Wandtapeten“-Problems schlägt der Ansatz Fisheye-Techniken [Fur86] vor. [DDK+02] zeigen wie das Werkzeug JViews mit der Problematik umgeht. Keiner der

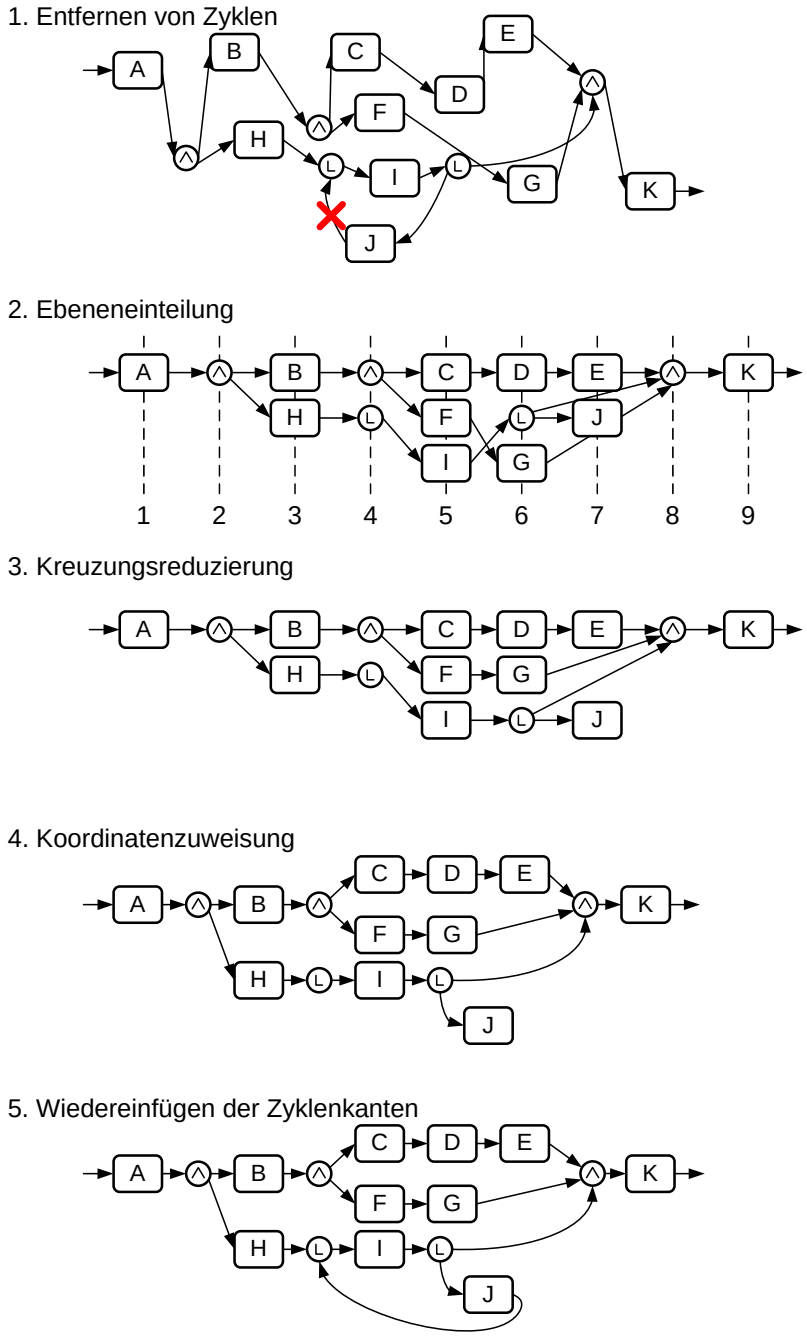


Abbildung 7.10: Phasen des Sugiyama-Algorithmus

Ansätze ist jedoch in der Lage ein zufriedenstellendes Layout für komplexe Prozessgraphen mit Satellitenobjekten zu berechnen.

Ein Umstand verkompliziert die Berechnung eines Prozessgraph-Layouts abermals. Die zentrale Anforderung an einen Layout-Algorithmus lässt sich kompakt formulieren: Gesucht ist ein schönes Layout. Was jedoch bedeutet „schön“ im vorliegenden Kontext. Verschiedene Untersuchungen wurden durchgeführt, um diesen relativen Begriff durch messbare Größen zu beschreiben

[PCJ96, Pur00, Pur02]. Ein Resultat ist zumindest eine große Zahl von Metriken, die jeweils getrennt voneinander optimiert werden können (z.B. minimale Anzahl Kantenkreuzungen, Kantenlänge, Fläche bzw. Größe der Zeichnung, Erhaltung von Symmetrien, gleiche Darstellung ähnlicher Strukturen, etc.)

Es können somit vielfältige Anforderungen formuliert werden, die sich in zwei Kategorien einteilen lassen:

Zeichenkonventionen: Kriterien, die vom Algorithmus in jedem Fall eingehalten werden müssen (Muss-Kriterien).

Ästhetikkriterien: Neben den Zeichenkonventionen sollen weitere Anforderungen erfüllt werden, die zu schöneren Zeichnungen führen (Nebenbedingungen bzw. Kann-Kriterien).

Dies führt zu einem Optimierungsproblem, bei dem häufig widersprüchliche Anforderungen auftreten. Zum Beispiel widerspricht sich oftmals die gleichzeitige Forderung nach Minimierung von Kantenkreuzungen und Kantenlängen. Welches Kriterium letztlich wichtiger ist, hängt zum Teil vom zu visualisierenden Graphen, zum Teil aber auch von den Vorstellungen der Betrachter ab (für entsprechende Untersuchung zu UML siehe [PAC02, PMN03]). Ein automatischer Algorithmus, der immer das optimale Ergebnis produziert, ist eher nicht realistisch. Durch die Vielzahl der Parameter, die Einfluss auf eine Darstellung nehmen und die optimiert werden sollen, sind effiziente Lösungen für das Gesamtproblem nicht zu erwarten [DPS02]. Der einzig gangbare Weg ist momentan, die vorhandenen Algorithmen so gut es geht zu verwenden und mit Hilfe von Heuristiken und wenn nötig manuellen Eingriffen ein möglichst optimales Layout zu erzielen (siehe [RBRB06] für einen ersten Ansatz).

7.3.1 Layout-Algorithmus für Proviado-Prozessgraphen

In [Zha07] stellen wir einen im Rahmen von Proviado entwickelten Algorithmus vor, der speziell für das Layout von Prozessgraphen entwickelt wurde und auf dem Sugiyama-Algorithmus basiert. Dieser Abschnitt gibt einen kurzen Überblick über die Vorgehensweise und Besonderheiten dieses Algorithmus, abstrahiert aber von Realisierungsdetails.

Für den Algorithmus definieren wir konfigurierbare Zeichenkonventionen. Zum Beispiel werden die Mindestabstände zwischen den verschiedenen Prozessobjekten festgelegt. Die folgende Liste stellt einen Auszug der vor der Berechnung des Layouts konfigurierbaren Abstände dar (vgl. Abbildung 7.11):

- Abstand zwischen zwei Aktivitäten (horizontal d_H und vertikal d_V)
- vertikaler Abstand zwischen Aktivitäten und adjazenten Datenelementen d_{AD1}
- vertikaler Abstand zwischen Aktivitäten und adjazenten Datenelementen, die zu mehr als einer Aktivität adjazent sind d_{AD2}
- vertikaler Abstand zwischen Aktivitäten und adjazenten Bearbeiterobjekten d_{AA}
- horizontaler Abstand zwischen Satellitenobjekten d_{SS}
- etc.

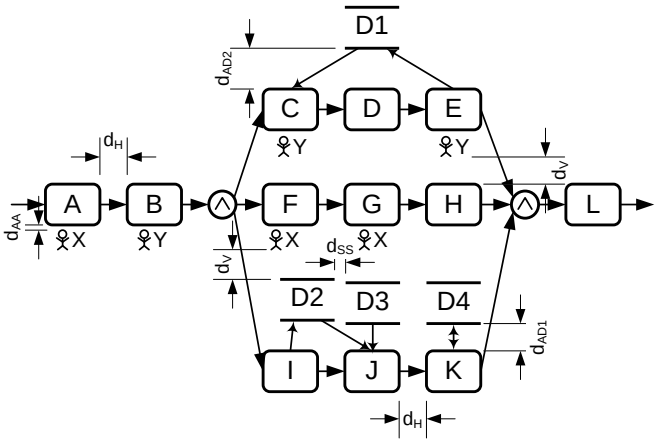


Abbildung 7.11: Beispiele für konfigurierbare Abstände zwischen Prozesselementen

Des Weiteren geben wir die relative Position der Satellitenobjekte bezogen auf die adjazente Aktivität an. Hierfür verwenden wir Himmelsrichtungen wie in Abbildung 7.12 dargestellt.

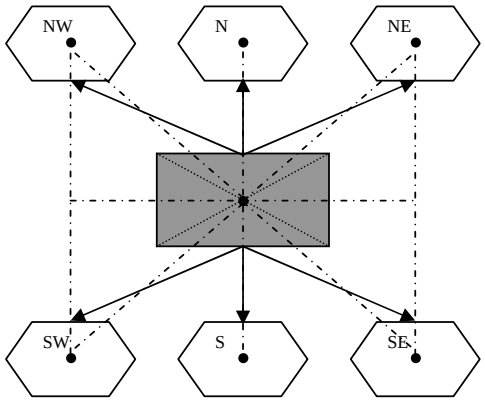


Abbildung 7.12: Positionierung von Satellitenobjekten nach Himmelsrichtungen [Zha07]

Unter den Ästhetikkriterien beschreiben Constraints spezielle Anforderungen an das Layout. Als optionale Kriterien wird versucht, diese Bedingungen bei der Berechnung des Layouts zu erfüllen. Folgende Constraints können für den Algorithmus momentan definiert werden:

Abstände zwischen Objekten Bei bestimmten Prozessvisualisierungen kann zum Beispiel erwünscht sein, zwei bestimmte Aktivitäten in einem anderen als dem vorgegebenen Mindestabstand zu zeichnen. Im Beispielprozess aus Abbildung 7.13 ist ein spezieller Abstand zwischen Aktivitäten D und E definiert.

Ausrichtung der Objekte Mit Hilfe dieses Constraints kann eine relative Ausrichtung von Aktivitäten vorgenommen werden (horizontal wie vertikal). Im Beispiel aus Abbildung 7.13 ist dies für Aktivitäten D, I und E, G dargestellt, die dadurch vertikal bündig ausgerichtet werden.

Reihenfolge von Verzweigungsästen Dieses Constraint erlaubt eine bestimmte Reichenfolge von Verzweigungsästen in der Prozessvisualisierung festzulegen. Dies ist in Abbildung 7.13 für die mit den Aktivitäten C und F beginnenden Äste illustriert.

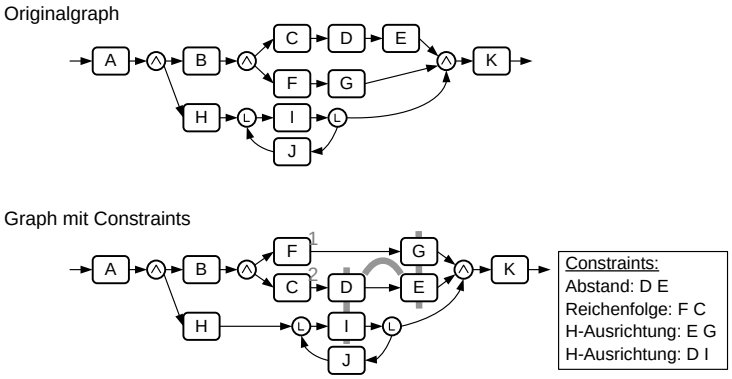


Abbildung 7.13: Beispiel für die Auswirkungen der Constraints

Für den Einsatz in der Praxis sind derartige Ausrichtungs-Constraints sehr wichtig. Im Produktionsplanungsprozess (siehe Fallstudie III, Abschnitt 2.2.3 auf Seite 17) etwa müssen die Aktivitäten horizontal an den Meilensteinen ausgerichtet werden. Ohne diese Ausrichtung wäre für den Betrachter die zeitliche Einordnung der Aktivitäten erheblich erschwert.

Im Vergleich zu den theoretischen Ansätzen aus der Literatur, die stets mit einheitlichen Knotengrößen rechnen, ergeben sich bei der Entwicklung eines realen Layout-Algorithmus zusätzliche Schwierigkeiten durch potentiell unterschiedlich große und geformte Symbole für die Knoten. Neben der Positionierung der Knoten hat dieser Umstand auch Auswirkungen auf die Zeichnung der Kanten, die möglichst überschneidungsfrei geführt werden sollen.

7.3.2 Schiebealgorithmus für das Layout von Prozessgraphen

Der Einsatz des zuvor beschriebenen Layout-Algorithmus wirft verschiedene Probleme auf. Zum einen ist es ineffizient, selbst bei kleinen Änderungen am Prozess das komplette Layout erneut zu berechen. Zum anderen kann das Neuberechnen eines Layouts dazu führen, dass sich die Positionierung der Knoten grundlegend ändert. Das bedeutet für den Betrachter, dass er den neu gezeichneten Prozess auf den ersten Blick gegebenenfalls nicht wiedererkennt. Man bezeichnet dieses Bild, das sich beim Betrachten einen Graphen im Kopf bildet auch als *Mental Map*. Für die Wiedererkennung eines Graphen ist somit die Bewahrung der Mental Map sehr wichtig und daher ein Hauptziel aller Layout-Algorithmen [PHG06].

In der Literatur existieren Ansätze für ein inkrementelles Layout von Graphen [Nor96, Ham03]. Dabei wird bei Änderungen des Graphen versucht, dessen Zeichnung so wenig wie möglich zu anzupassen.

Der im vorangehenden Abschnitt vorgestellte Algorithmus berechnet ein vollständig neues Layout für einen Prozessgraphen. Demgegenüber verfolgt der zweite Algorithmus, den wir in [Zha07] entwickelt haben, eine spezielle Art des inkrementellen Layouts. Er verwendet dabei

Informationen aus vorhandenen Prozessgraph-Layouts in Kombination mit Informationen zu Änderungen am Prozessgraphen. Das Prinzip des Algorithmus beruht auf dem Verschieben existierender Teil-Layouts. Nur für veränderte Teile des Graphen wird ein neues Layout berechet, die unveränderten Teile werden in Blöcken verschoben. Der Vorteil dabei ist, dass sich das Verschieben der bestehenden Positionen sehr effizient realisieren lässt.

Eine Herausforderung dieses Ansatzes ist das Erkennen von veränderten Bereichen. Dazu muss der Prozessgraph sowohl in der alten, unveränderten Version wie auch in der neuen, veränderten Version bekannt sein. Zusätzlich benötigen wir die Positionsdaten des alten Prozesses. Aus einem Vergleich der beiden Prozesse unter Nutzung der alten Positionsdaten kann man die Veränderungen ableiten und anschließend ein neues Layout berechnen. Der Vorgang lässt sich noch effizienter gestalten, wenn explizite Informationen über die Veränderungen durch die View-Bildung vorliegen (z.B. welche Knoten wurden gelöscht, welche eingefügt, welche Knoten sind deren Nachbarn).

Abbildung 7.14 illustriert die Idee des Algorithmus anhand eines Beispiels für die Reduktion. Aus der Information, welche Knoten entfernt wurden, berechnen wir zunächst die unveränderten Bereiche und fassen die Knoten, die gemeinsam verschoben werden können, in Boxen zusammen. Der nächste Schritt verschiebt dann die Boxen derart, dass die geforderten Zeichenkonventionen (z.B. die Minimalabstände) eingehalten werden.

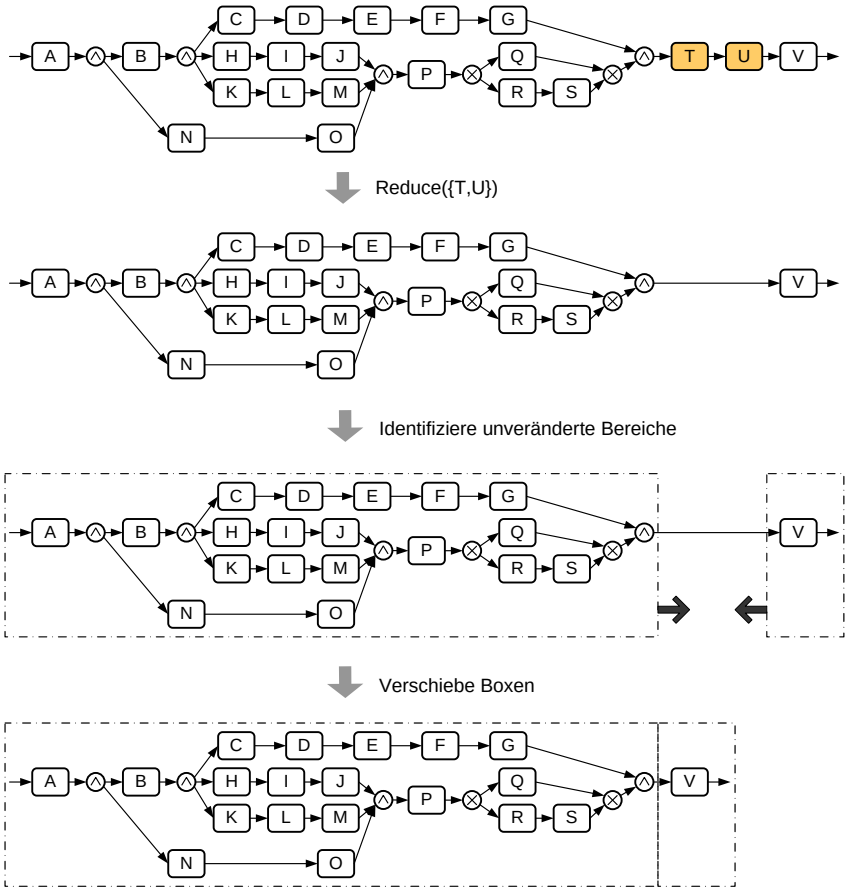


Abbildung 7.14: Beispiel für Layout-Anpassung durch Schieben

Die zusätzlichen Darstellungsobjekte, wie Überschriften oder Applikationsdaten, die für den Betrachter von großem Nutzen sind, können nicht vollautomatisch positioniert werden. Eine manuelle Anpassung bzw. Spezifikation der Positionierung dieser Objekte wird für ein optimales Layout immer erforderlich bleiben.

7.4 Zusammenfassung

In diesem Kapitel haben wir verschiedene Aspekte behandelt, die für eine praktische Einsetzbarkeit einer Prozessvisualisierungskomponente wichtig sind. Ein Aspekt hierbei ist die Anbindung der prozessausführenden Systeme an die Visualisierungskomponente. Da die zu visualisierenden Prozesse auf verschiedene Systeme fragmentiert sein können, müssen zunächst die Modelle integriert werden. Proviado stellt hierfür ein generisches Prozessmetamodell zur Verfügung. Bei der Integration müssen nun die durch die unterschiedlichen Quellmetamodelle auftretenden Heterogenitäten bereinigt werden. Für die Visualisierung von Prozessinstanzen muss darüber hinaus eine Anbindung an Laufzeit- und Applikationsdaten realisiert werden. Ein nicht-triviales Problem dabei ist die Korrelation der Daten, d.h. die richtige Zuordnung von Instanzdaten zu den jeweiligen Instanzen. Durch die Fragmentierung der Prozessausführung wird dieses Problem noch verschärft. In diesem Kapitel haben wir daher eine Architektur für die Anbindung der Laufzeitdaten verschiedener Systemtypen vorgeschlagen.

Ein weiterer relevanter Aspekt ist die Layout-Berechnung von Prozessgraphen. Gerade durch die in vorangehenden Kapiteln vorgestellten Mechanismen zur strukturellen und graphischen Anpassung von Prozessen ergibt sich eine hohe Dynamik. Eine manuelle Anpassung des Layouts ist daher in der Regel zu aufwendig und Algorithmen zur automatischen Berechnung des Layouts sind erforderlich. Wir haben in diesem Kapitel zwei Layout-Algorithmen vorgestellt. Um möglichst ansprechende Layouts zu erhalten, erlauben die Algorithmen mittels Constraints bestimmte Vorgaben zu definieren (z.B. relative Ausrichtung von Aktivitäten zueinander). Der erste Algorithmus berechnet ausgehend vom Prozessmodell und von den Constraints ein neues Layout. Bei Änderungen am Prozess und anschließender Neuberechnung des Layouts, kann sich dieses jedoch vollständig ändern. Dies erschwert den Benutzern die Wiedererkennung des Prozesses. Der zweite entwickelte Algorithmus behebt dieses Problem, in dem er ausgehend von einem existierenden Layout durch geschicktes Verschieben der Prozesselemente ein neues Layout berechnet.

Im folgenden Kapitel werden wir sehen, wie alle bislang gezeigten Mechanismen in Proviado zusammenspielen, um hochgradig konfigurierbare Prozessvisualisierungen zu ermöglichen.

Teil III

Praktische Realisierung

8

Gesamtablauf der Visualisierung und prototypische Umsetzung

8.1 Motivation

Proviado bietet ein Rahmenwerk für die benutzerspezifische Prozessvisualisierung an. In den Kapiteln 4 bis 6 haben wir die grundlegenden Mechanismen vorgestellt, mit deren Hilfe Prozessdarstellungen an die Bedürfnisse der verschiedenen Benutzergruppen anpassbar sind. In Kapitel 7 wurden weitere Aspekte behandelt, die für eine konfigurierbare Visualisierungskomponente erforderlich sind. Darüber hinaus existieren weitere wichtige Anforderungen, um die praktische Anwendbarkeit einer solchen Komponente zu gewährleisten. Beispielsweise muss die Komponente auf Prozessdaten aus verschiedenen Systemen zugreifen können (vgl. Anforderung 2-5). Zudem kann die Ausführung der zu visualisierenden Prozesse auf mehrere Systeme verteilt sein. Auch hier muss eine integrierte Visualisierung möglich sein (vgl. Anforderung 2-3). Generell gilt es bei der Entwicklung der Visualisierungskomponente zu berücksichtigen, dass diese einfach in die Systemlandschaft eines Unternehmens integrierbar ist und keine aufwendigen Installationen bzw. Konfigurationen am Benutzerarbeitsplatz erfordert. Idealerweise kann auf die Prozessdarstellungen via Internet zugegriffen werden (vgl. Anforderung 2-12). In diesem Kapitel stellen wir dar, wie die in den vorangegangenen Kapiteln entwickelten Konzepte zusammenwirken, um auch die hier oben diskutierten Anforderungen aus den Fallstudien (vgl. Kapitel 2) zu erfüllen.

Abbildung 8.1 zeigt den bereits in Kapitel 3 eingeführten, allgemeinen Visualisierungsprozess [SM00, Grö02], an dem sich das Vorgehen von Proviado bei der Prozessvisualisierung orientiert.

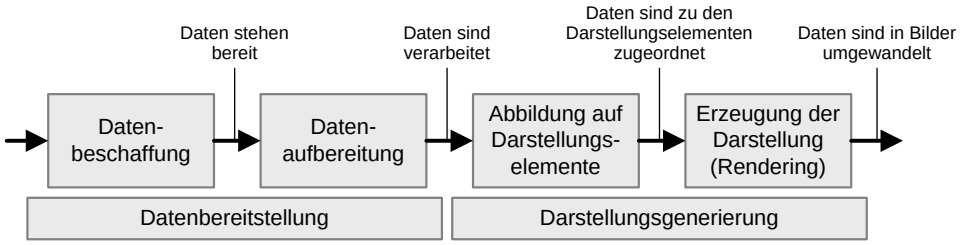


Abbildung 8.1: Abstrakter Visualisierungsprozess

Der Gesamtablauf der Generierung einer Prozessvisualisierung in Proviado wird in Abschnitt 8.2 vorgestellt. Zur Anpassung einer bestimmten Visualisierung an den jeweiligen Betrachter können sowohl Prozess-Views als auch der vorgestellte Template-Mechanismus eingesetzt werden. Abschnitt 8.3 diskutiert, welcher dieser Mechanismen abhängig von der jeweiligen Situation besser geeignet ist. Optimierungen, die unter bestimmten Voraussetzungen die Effizienz des Visualisierungsablaufs erhöhen, erörtert Abschnitt 8.4. Den Gesamtansatz von Proviado illustriert Abschnitt 8.5 anhand eines Beispiels. Abschnitt 8.6 gibt den Realisierungsstand der Proviado-Konzepte wieder, bevor Abschnitt 8.7 die Ergebnisse zusammenfasst.

8.2 Visualisierungsprozess in Proviado

Wir haben in Kapitel 3 bereits einen groben Überblick über den Ablauf einer Prozessvisualisierung in Proviado gegeben. Nachdem nunmehr die hierzu benötigten Konzepte im Detail bekannt sind, beschreiben wir nun den Gesamtablauf vom Prozessmodell zur entsprechenden Prozessvisualisierung im Ganzen.

8.2.1 Ablauf der Visualisierung vom Prozess bis zur Graphik

Die notwendigen Schritte, um vom Prozess in den Quellsystemen bis zur fertigen Prozessvisualisierung zu gelangen, können in Build- und Run-Time-Schritte untergliedert werden (vgl. Tabelle 8.1). Zu den Build-Time-Schritten zählen alle vorbereitenden Tätigkeiten, die typischerweise nur einmalig durchgeführt werden. Sobald der Betrachter einer Prozessvisualisierung ebendiese anfordert, werden die Run-Time-Schritte abgearbeitet, die aus den bzw. mit Hilfe der vorbereiteten Daten die gewünschte Darstellung generieren.

Abbildung 8.2 zeigt das Zusammenspiel der Einzelkomponenten innerhalb der Proviado-Visualisierung.

Schritte zur Build-Time

Schritt 1 (Prozesstransformation und -integration): Die Prozessschemata der verschiedenen Quellsysteme werden zunächst in das durch das Proviado-Metamodell gegebene Format transformiert und anschließend zu einem Gesamtprozess integriert. Es liegt nun ein vollständiges Prozessschema in einheitlichem Format vor, das als Grundlage für die Visualisierung dient.

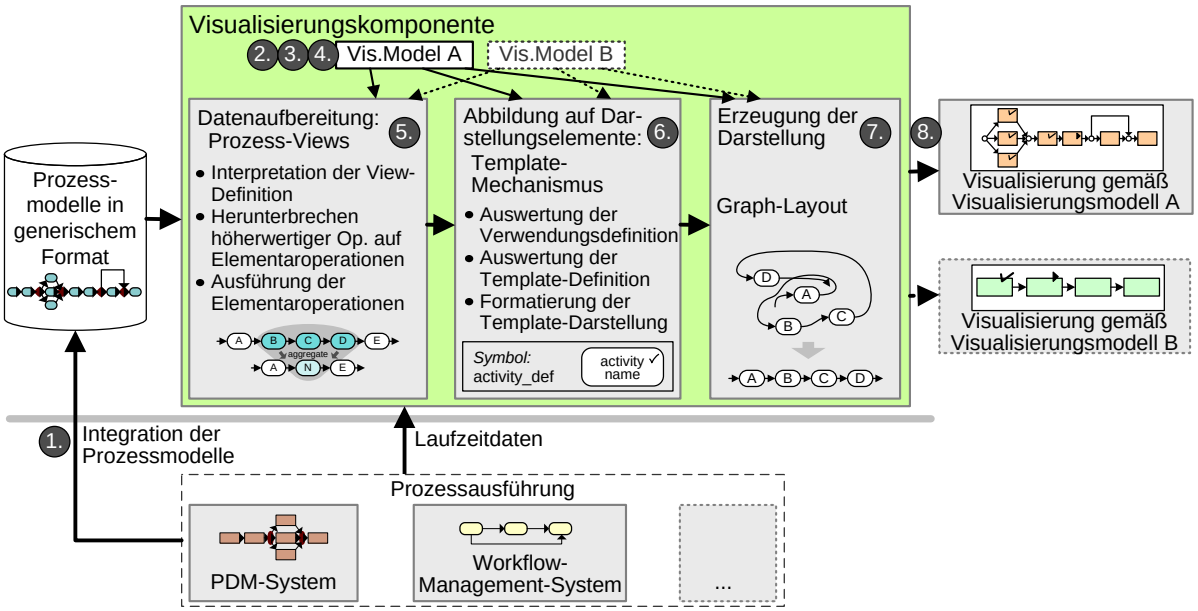


Abbildung 8.2: Architekturübersicht

Schritte zur Build-Time
1. Integration und Transformation des Prozessmodells
2. Definition der View
3. Definition der Notation
4. Definition des Visualisierungsmodells
Schritte zur Run-Time
5. Berechnung der View
6. Auswertung der Notationsdefinition und Befüllung der Templates mit Daten
7. Berechnung des Layouts
8. Rendering

Tabelle 8.1: Schritte zur Erzeugung einer Prozessvisualisierung

Schritt 2 (Definition der View): Basierend auf dem Prozessschema wird die gewünschte View definiert. Dies erfolgt mittels der in Abschnitt 5.5 vorgestellten View-Definitionssprache. Es stehen dabei alle in Kapitel 4 und 5 definierten View-Bildungsoperationen zur Verfügung.

Schritt 3 (Definition der Notation): Die Symbole, die für die Visualisierung der verschiedenen Prozessobjekte verwendet werden sollen, müssen entworfen werden. Die Symboldefinition zusammen mit den Parameterdefinitionen (vgl. Kapitel 6) ergeben jeweils ein Template für ein bestimmtes Prozessobjekt. Schließlich muss die Template-Verwendungsdefinition angelegt werden. Durch sie wird spezifiziert, welches Template für welches Prozessobjekt verwendet werden soll.

Schritt 4 (Definition des Visualisierungsmodells): Im Visualisierungsmodell werden alle Parameter der Visualisierung zusammengefasst. Es umfasst neben Angaben zum darzustellenden Prozess (inkl. View-Definition) und der zu verwendenden Notation weitere, für die Darstellung

wichtige Parameter (z.B. Parameter für das Layout oder zusätzliche Darstellungsobjekte; vgl. Tabelle 6.2 auf Seite 157).

Mit Abschluss von Schritt 4 sind alle Daten, die für eine Visualisierung benötigt werden, vollständig definiert. Sie stehen der Visualisierungskomponente zur Verfügung, die sie auf Anfrage interpretiert und die Visualisierung generiert.

Schritte zur Run-Time

Schritt 5 (Berechnung der View): Nachdem eine Visualisierung angefordert worden ist, wird die im Visualisierungsmodell hinterlegte View-Definition interpretiert und die View-Berechnung auf dem Basisprozess durchgeführt. Ist die View-Berechnung zusätzlich von Laufzeitdaten einer Prozessinstanz abhängig, werden diese aus den Quellsystemen über die entsprechende Laufzeitdatenschnittstelle abgefragt.

Schritt 6 (Auswertung der Notationsdefinition): Die Auswertung der Template-Verwendungsdefinition ordnet jedem Prozesselement ein Template und damit ein für die Visualisierung zu verwendendes Symbol zu. Die Template-Definition beschreibt insbesondere, welche Prozessdaten an welcher Stelle des Symbols dargestellt werden sollen. Eventuell benötigte Laufzeitdaten (z.B. zum Status einer Aktivität) werden wieder über die Laufzeitdatenschnittstelle abgefragt.

Schritt 7 (Berechnung des Layouts): Auf Grundlage der zuvor berechneten Prozess-View sowie der den Prozesselementen zugewiesenen Templates wird das Layout des darzustellenden Prozessgraphen berechnet. Dabei werden die Konfigurationsdaten aus dem Visualisierungsmodell (z.B. Minimalabstände zwischen Prozesselementen) berücksichtigt.

Schritt 8 (Rendering): Die letzte Phase erzeugt aus den logischen Daten einer Prozessvisualisierung ein konkretes Graphikformat. Dabei werden die den Prozesselementen zuvor zugeordneten Symbole ausgelesen. Die Symbole enthalten nach den vorangegangenen Schritten alle darzustellenden Prozessinformationen und die Layout-Daten, d.h. die Koordinaten in der Zeichenebene. Die im Visualisierungsmodell enthaltenen allgemeinen Darstellungsoptionen (z.B. Farbschemata) werden zusammen mit zusätzlichen Darstellungsobjekten (z.B. Überschriften) ebenfalls in die erzeugte Graphik integriert. In Proviado verwenden wir als Graphikformat SVG (Scalable Vector Graphics [JN05]). Die Umwandlung des SVG in ein Pixel-Bild erfolgt durch die Client-Programme (z.B. Browser-Plug-In).

Die Schritte 5 bis 8 werden jedes Mal, wenn eine Visualisierung angefordert wird, wiederholt. In Abschnitt 8.4 werden wir Varianten dieses Ablaufs beschreiben, die in bestimmten Szenarien eine besonders effiziente Realisierung ermöglichen.

8.2.2 Realisierungsvarianten der Clients

Für die Realisierung der Anzeige der Prozessgraphik beim Betrachter existieren verschiedene Möglichkeiten. Abhängig von den Rahmenbedingungen des jeweiligen Projekts hat jede ihre Vor- und Nachteile. Zunächst gilt es abzuwägen, ob der Client als Thin- oder als Rich-Client umgesetzt wird. Thin-Clients haben den Vorteil, dass sie keine aufwendige Installation erfordern,

sondern zumeist im Browser direkt ausgeführt werden können. Sie eignen sich jedoch nur bedingt für komplexere Aufgaben. Demgegenüber ist bei Rich-Clients eine Installation auf jedem Arbeitsplatz erforderlich. Dadurch sind jedoch auf Rich-Clients auch komplexe Aufgaben realisierbar. In jüngster Zeit verschwimmen die Grenzen zwischen Thin- und Rich-Client durch das Aufkommen von AJAX [Gar05], RAP [RAP07] und verwandten Technologien zusehends.

Weitere Realisierungsentscheidungen betreffen das für die Darstellung verwendete Graphikformat. Wir schränken die Diskussion hier auf Graphikformate, die sich für Browser-basierte Thin-Clients eignen, ein.

1. **Rastergraphikformate** wie zum Beispiel PNG oder GIF werden von allen Browsern standardmäßig unterstützt. Sie haben aber den Nachteil, dass die erzeugten Graphiken nicht skalierbar, d.h. Zoom-bar, und relativ groß sind, d.h. viel Speicherplatz benötigen. In Proviado wäre eine Konvertierung in ein Rastergraphikformat durch ein spezielles Programm, das die SVG-Daten vor Auslieferung an den Client in PNG transformiert, einfach realisierbar.

2. **Vektorgraphikformate** wie beispielsweise SVG benötigen üblicherweise noch ein Plug-in für den jeweiligen Browser (Ausnahme im Fall von SVG: Firefox). Demgegenüber stehen Vorteile bezüglich Größe und Skalierbarkeit. Ein weiterer Vorteil von SVG ist, dass sich die Graphiken mittels beliebiger XML-Bearbeitungsmethoden verarbeiten lassen. Auch das finale Erzeugen der SVG-Datei ist einfach und effizient realisierbar und verursacht bedeutend weniger Speicher- und Rechenlast auf dem Server als die vergleichbare Generierung einer Rastergraphik.

3. **Objektgraphikformate** beschreiben Graphiken mittels eines meist proprietären Objektmodells (z.B. Visio). Für die Darstellung im Client benötigt man ein passendes Plug-In. Alternativ können die Graphiken auch in ein Raster- oder Vektorformat transformiert werden. Durch Umgestaltung der Symboldefinition und der Parameterauflösung des Template-Mechanismus, wäre auch diese Variante mit Proviado relativ einfach realisierbar.

Die Proviado-Visualisierungskomponente ist als Thin-Client auf Basis von SVG ausgelegt. Zum einen sind für die gewählte Form der Visualisierung keine komplexen Benutzerinteraktionen erforderlich. Zum anderen ermöglicht die Wahl von SVG eine einfache Integration der realisierten Prozessvisualisierungskomponente in bestehende Unternehmensportale oder andere Inter- bzw. Intranetanwendungen, da praktisch kein Installationsaufwand auf Seiten des Client anfällt. Die Integration in bestehende Portale bietet zudem den Vorteil, dass im Portal existierende Applikationen über entsprechende Verlinkung angebunden werden können. Dadurch wird zum Beispiel die Navigation zwischen Prozessmodellen und Organisationsmodellen möglich.

8.3 Abgrenzung der Mechanismen

Für eine benutzergerechte Visualisierung von komplexen Prozessmodellen sollten Objekte, die für den jeweiligen Betrachter irrelevant sind, nicht dargestellt werden. Zu erwähnen sind in diesem Zusammenhang auch Vertraulichkeitsanforderungen. Sie erfordern unter Umständen, dass gewisse Prozessinformationen dem aktuellen Betrachter nicht zugänglich sind. In den Kapiteln 4, 5 und 6 haben wir die Proviado-View- und Template-Mechanismen vorgestellt. Wir diskutieren nun, wie entsprechende Anforderungen auf Grundlage der Mechanismen umsetzbar sind.

Vorbehandlung des View-Mechanismus Die in Kapitel 5 vorgestellte Vorbehandlung des View-Mechanismus (siehe Seite 121) eignet sich vor allem, um Prozessmodelle anzupassen, die strukturelle Abnormalitäten aufweisen und daher durch den View-Mechanismus verarbeitet werden können. Ein Beispiel ist die Umwandlung von impliziten in explizite Verzweigungsknoten. Eine weitere Anwendung der View-Vorbehandlung ist die Rekombination fragmentierter Prozesse, wie sie beispielsweise bei der hierarchischen Modellierung entstehen.

View-Mechanismus Der Proviado-View-Mechanismus dient der strukturellen Anpassung des Prozessmodells durch Reduktion und Aggregation. Dadurch können Aktivitäten und/oder Satellitenobjekte (z.B. Bearbeiterknoten) aus dem Prozessmodell entfernt werden. Die Daten der entfernten Objekte stehen folglich für eine Visualisierung nicht mehr zur Verfügung. Sollen beispielsweise Attributwerte der entfernten Objekte dennoch in der Visualisierung enthalten sein, können diese durch eine Attributtransformationsoperation (vgl. Abschnitt 4.4.3) explizit einem anderen Objekt hinzugefügt werden. Für Satellitenobjekte kann das Ausblenden alternativ durch den Template-Mechanismus vorgenommen werden.

Nachbehandlung des View-Mechanismus Nicht-View-konforme Transformationen können im Anschluss an die View-Bildung durch die Nachbehandlung der erzeugten View vollzogen werden. Zum Beispiel können nicht erwünschte Kanten und Knoten entfernt werden oder zuvor durch die Vorbehandlung durchgeführte Veränderungen wieder rückgängig gemacht werden (z.B. Rücktransformation von expliziten in implizite Verzweigungsknoten).

Template-Mechanismus Der Template-Mechanismus definiert, welche Prozesselemente in welcher Notation dargestellt werden sollen. Durch Weglassen von Verwendungsregeln (z.B. für Satellitenobjekte) werden die betroffenen Prozesselemente in der Visualisierung nicht angezeigt. Der erreichte Effekt ist auf den ersten Blick ähnlich der Reduktion. Jedoch sind die Daten der Satellitenobjekte nach wie vor im Prozessmodell vorhanden und werden lediglich nicht angezeigt. Die Daten können aber weiterhin mittels Definition einer anderen Verwendungsregel (an anderer Stelle) dargestellt werden. Nach einer Reduktion der Prozesselemente ist dagegen eine Anzeige nicht mehr möglich, da die Daten in der Prozess-View nicht mehr vorhanden sind.

Welcher Mechanismus letztlich für das Entfernen von Prozessobjekten aus der Visualisierung herangezogen wird, sollte letztlich von dem konkreten Szenario abhängig gemacht werden. Beispielsweise könnten auf Grundlage einer View, die die Satellitenobjekte nicht entfernt, mit Hilfe des Template-Mechanismus zwei verschiedene Visualisierungen definiert werden, von denen eine die Satellitenobjekte enthält und die andere sie ausblendet.

Bei Realisierung der Visualisierung als Rich-Client-Architektur spielt zudem der Sicherheitsaspekt eine Rolle. Vertrauliche Daten, die schon auf dem Server durch eine View-Bildung aus dem Prozessmodell entfernt wurden, können auf dem Weg zum Client, der die Visualisierung erstellt, nicht abgefangen werden.

Schon bei der Modelldatentransformation und -integration sind prinzipiell Anpassungen des Prozessmodells möglich. Diese sind dann aber für alle Visualisierungen gültig.

8.4 Varianten des Visualisierungsprozesses

Bei der Konzeptionierung des Gesamtablaufs der Visualisierung sind wir von gewissen Voraussetzungen ausgegangen. Dazu zählt auch die Annahme, dass es sich um langlaufende Prozesse handelt, die ihre Zustände im Abstand von Stunden oder Tagen ändern. Außerdem wird in Proviado davon ausgegangen, dass sich die Darstellungen in Abhängigkeit von Instanz- bzw. Applikationsdaten ändern können. In anderen Szenarien mit anderen Voraussetzungen kann der Ablauf der Erzeugung einer Visualisierung verändert und dadurch optimiert werden.

Optimierungspotentiale des Gesamtablaufs liegen größtenteils in einer Verschiebung von Aufgaben zwischen Build- und Run-Time. Im Folgenden diskutieren wir einige Szenarien beispielhaft. In konkreten Anwendungsfällen muss dann geprüft werden, welche Anforderungen bestehen und wie der Visualisierungsprozess dahingehend optimiert werden kann.

Wichtigster Faktor für mögliche Optimierungen ist die Abhängigkeit von Laufzeitdaten. Abbildung 8.3 zeigt die verschiedenen Ausprägungen des Visualisierungsprozesses in Abhängigkeit von der Dynamik der View-Bildung und der Template-Zuweisung. Gemäß den in Abschnitt 4.2 eingeführten Arten von Views, bezeichnen wir eine View-Bildung bzw. Template-Zuordnung als dynamisch, falls das Ergebnis von Laufzeitdaten abhängt. Andernfalls gehen wir von einer statischen View-Bildung bzw. Template-Zuordnung aus. Fall 1 entspricht der dynamischen Visualisierung in Proviado, bei der sowohl die View als auch die Verwendung der Notation von Laufzeitdaten abhängen kann. Für den Fall einer dynamischen View-Generierung, birgt eine statische Notation kein weiteres Optimierungspotential, da die Auswertung der Notationsdefinition erst nach der View-Bildung erfolgt (vgl. Fall 2). Gehen wir jedoch von statischen Views aus, so kann, wie in Fall 3 illustriert, die View-Bildung zur Build-Time erfolgen. Lediglich die Zuweisung der Templates erfolgt dann abhängig von den Laufzeitdaten zur Run-Time. Ausnahmsweise kann auch die Layout-Berechnung vor der Template-Zuweisung erfolgen, falls die Templates keine Auswirkung auf das Layout haben, d.h. die Templates in ihrer Größe unverändert bleiben. Im Fall 4, einer rein statischen Visualisierung, können alle Berechnungen zur Build-Time erfolgen. Zur Run-Time werden bei einer Instanzvisualisierung eventuell Laufzeitdaten in die Visualisierung eingebettet.

Für die Visualisierung kurzlebiger Prozesse, deren Status sich sekündlich ändert, kann eine dynamische Visualisierung, im Speziellen die ständige Neuberechnung von View und Template-Zuweisung, zu Laufzeitproblemen führen. Führt die Dynamik der View-Bildung zu einer überschaubaren Anzahl unterschiedlicher Prozessmodelle (z.B. Views abhängig vom Betrachter), können die Views vorab berechnet werden und dadurch der Gesamtablauf erheblich beschleunigt werden.

8.5 Beispiel für eine Prozessvisualisierung mit Proviado

Anhand des in Kapitel 2 eingeführten Änderungsmanagement-Prozesses zeigen wir, wie der View-Mechanismus in Kombination mit dem Template-Mechanismus verwendet werden kann, um eine personalisierte Prozessvisualisierung zu erzeugen. Das Basismodell des Änderungsmanagement-Prozesses ist in Abbildung 8.4 dargestellt (vgl. Abbildung 2.2). Es zeigt neben dem Kontrollflussaspekt auch Datenfluss und Bearbeiterzuordnungen.

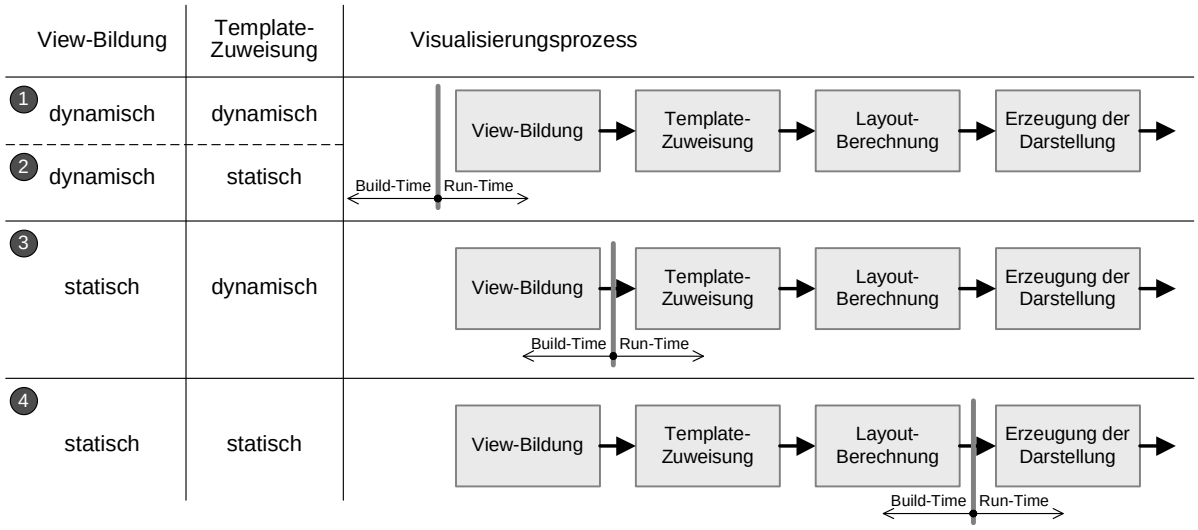


Abbildung 8.3: Optimierung des Visualisierungsprozesses

Dieser Prozess soll nun für einen prozessbeteiligten Ingenieur visualisiert werden. Zu diesem Zweck müssen mittels einer Prozess-View nicht-relevante Prozesselemente entfernt werden, etwa automatische Aktivitäten die ausschließlich dem Austausch oder der Transformation von Daten dienen (Schritte 4, 5, 10 und 17). Dasselbe gilt für bestimmte interaktive Schritte, die für den Betrachter irrelevant sind (Schritte 3, 18, 20 und 21). Andere Aktivitäten, die im Detail für den Ingenieur nicht interessant sind, die jedoch nicht vollständig entfernt werden sollen, werden zusammengefasst (Schritte 1, 2 und Schritte 11, 12, 13, 14 und 15). Schließlich sollen die Vorwärts- und Rückwärtssprungkanten im Kontrollfluss entfernt werden (Kanten 22, 23, 24, 25 und 26). Insgesamt kann die View durch die Anwendung der folgenden View-Operationen realisiert werden:

```
Reduce(Select(activity.type=automatic))
Reduce({3, 18, 20, 21})
Aggregate({1, 2}, Request Creation)
Aggregate({11, 12, 13, 14, 15}, CR Evaluation)
DeleteEdge({22, 23, 24, 25, 26})
```

Das sich ergebende View-Modell enthält immer noch eine große Zahl von Satellitenobjekten (z.B. Datenelemente und Bearbeiterzuordnungen), die im vorliegenden Fall nicht dargestellt werden sollen. Wie vorangehend diskutiert existieren prinzipiell zwei Möglichkeiten, um diese Prozesselemente „auszublenden“. Zum einen können sie mittels View-Operationen aus dem Modell vollständig entfernt werden. Damit geht die in den Elementen enthaltene Information für die Darstellung verloren. Zum anderen erlaubt unser Template-Mechanismus, beliebige Elemente nicht darzustellen, in dem ihnen keine Symbole zugeordnet werden (siehe Abschnitt 6.4.3 auf Seite 159). Dadurch ist die Information der Elemente weiterhin in dem der Darstellung zugrunde liegenden Modell (Basisprozess oder View) enthalten und kann in der Visualisierung an anderer Stelle genutzt werden. Abbildung 8.5 zeigt eine Darstellung einer entsprechenden Prozessinstanz-View, in der die Bearbeiterinformation in die Symbole der Aktivitäten integriert wurde.

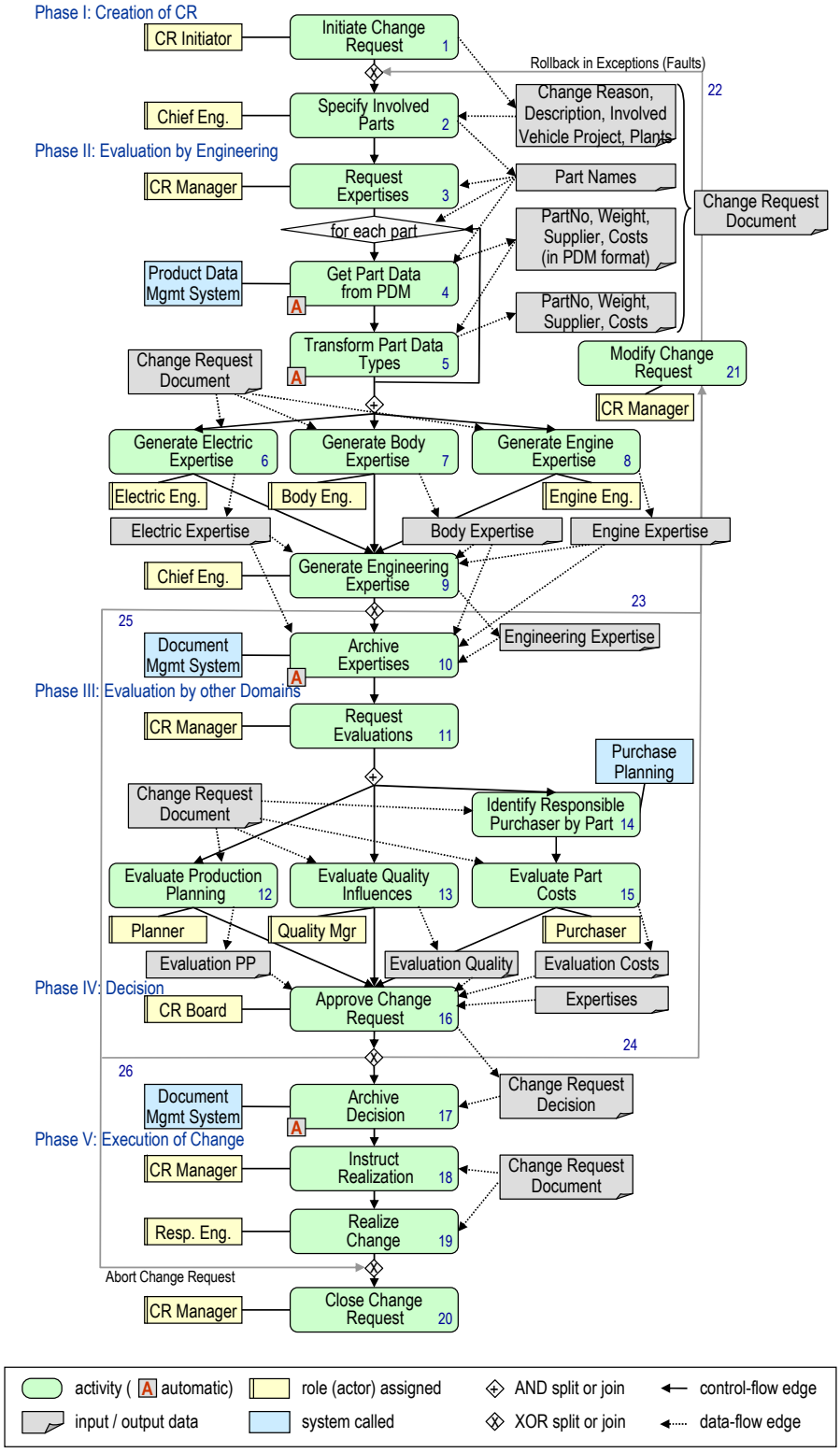


Abbildung 8.4: Ausgangsmodell für die Visualisierung

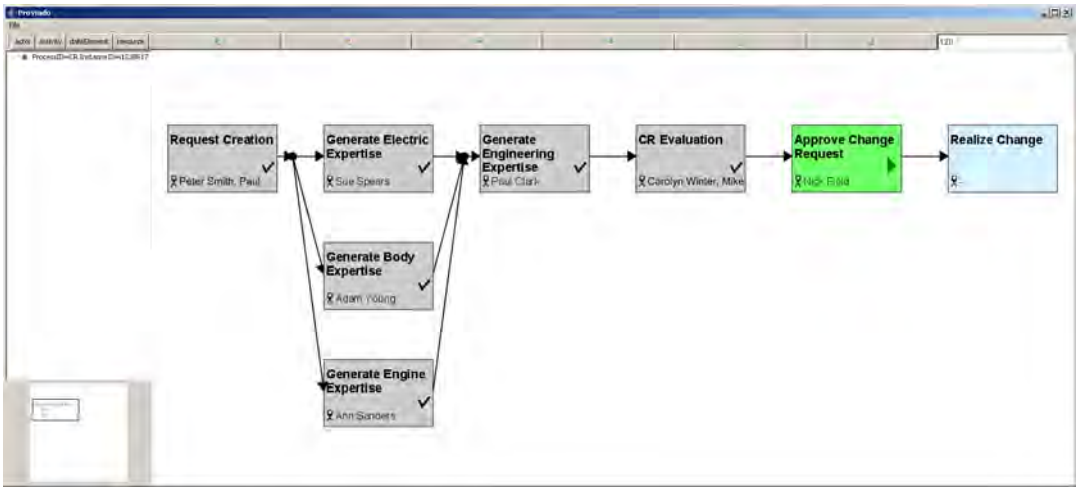


Abbildung 8.5: Visualisierung einer Prozessinstanz

Eine weitere Optimierung der Visualisierung erreichen wir durch die Darstellung von Zusatzobjekten, wie etwa Überschriften oder instanzspezifischen Anwendungsdaten (vgl. Abschnitt 6.4). Mittels unseres Visualisierungsmodells lassen sich beliebige Daten in die Visualisierung des Prozesses integrieren. In Abbildung 8.6 wurde die Darstellung derselben View durch die zusätzliche Anzeige von detaillierten Informationen der vom Änderungsvorhaben betroffenen Teile aufgewertet. Des Weiteren ermöglicht die Darstellung der Nummer des Änderungsvorhabens in der Überschrift eine schnelle Zuordnung. Weitere Details, etwa zum Änderungsgrund, werden durch Tooltipps eingeblendet.

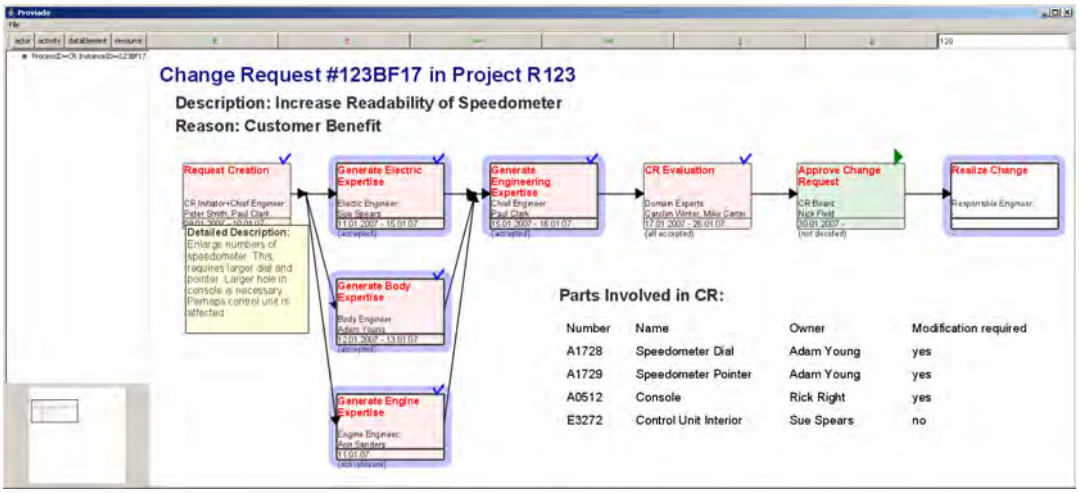


Abbildung 8.6: Optimierte Visualisierung durch zusätzliche Applikationsdaten

In ähnlicher Art und Weise können dank des Proviado-View-Mechanismus sehr schnell beliebige weitere Visualisierungen erzeugt werden. Abbildung 8.7 zeigt eine stark vereinfachte View des Änderungsmanagement-Prozesses, in der jede der in Abbildung 8.4 ersichtlichen Phasen zu einer abstrakten Aktivität aggregiert wurde. Solch eine Darstellung eignet sich in erster Linie, um

einen schnellen Überblick über den Status des Gesamtprozesses zu erhalten. Basierend auf dieser View kann beispielsweise auch die in Abbildung 2.4 auf Seite 16 gezeigte Statusübersicht aller laufender Prozessinstanzen generiert werden.

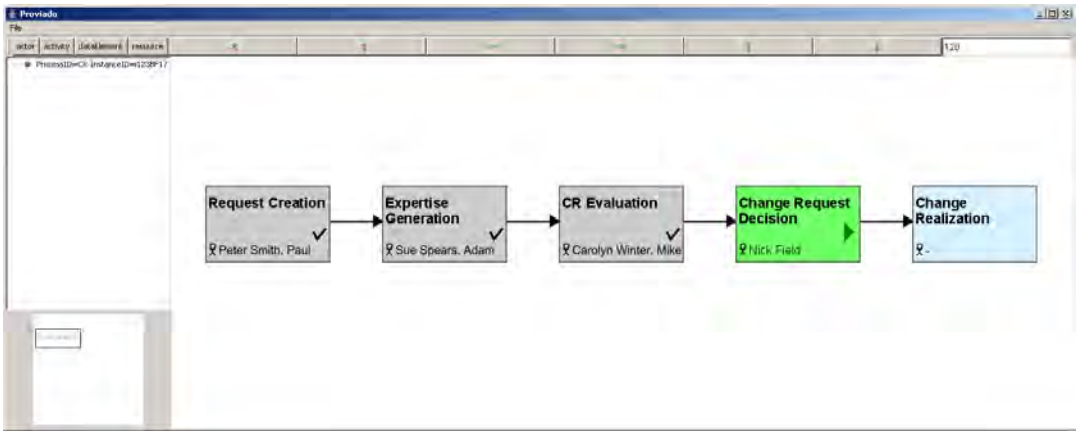


Abbildung 8.7: Visualisierung der Prozessphasen einer Instanz

8.6 Stand der Realisierung

Die Kernkonzepte dieser Arbeit wurden prototypisch auf Grundlage von Java [Rei06] realisiert [BB07b]. Es existieren drei Prototypen: einer für die View-Bildung, ein zweiter für den Template-Mechanismus und ein dritter für die Layout-Algorithmen.

Der View-Prototyp transformiert ein als XML gegebenes Prozessmodell in eine XML-Beschreibung der View. Zum Zeitpunkt des Schreibens ist der Prototyp in der Lage Kontrollflussgraphen ohne Schleifen zu verarbeiten. Die implementierten Operationen analysieren den Graphen und führen anschließend die geeignete View-Operation aus.

Der Prototyp zur Demonstration des Template-Mechanismus generiert auf Grundlage eines Prozessmodells, einer Notationsdefinition, Layout-Daten und Applikationsdaten eine entsprechende Visualisierung des Prozesses in SVG. Der Template-Mechanismus wurde dabei in seiner vollen Mächtigkeit realisiert. Zusätzlich umfasst dieser Prototyp eine GUI-Komponente, die in der Lage ist, die Visualisierung des Prozesses darzustellen (inkl. Zoom). Abbildungen 8.5 bis 8.7 zeigen die Anwendung.

Der Prototyp für die Layout-Algorithmen ist im Rahmen einer Diplomarbeit entstanden [Zha07]. Für ein gegebenes Prozessmodell berechnet der Prototyp unter Berücksichtigung optionaler Constraints (z.B. Mindestabstände zwischen verschiedenen Prozesselementen) eine neue Layout-Information. Diese kann anschließend vom vorhergehend beschriebenen Prototyp zur Darstellung des Prozesses verwendet werden.

8.7 Zusammenfassung

Dieses Kapitel hat gezeigt, wie die in den vorangegangenen Kapiteln vorgestellten Konzepte zu einem Gesamtsystem für die Visualisierung von Prozessen zusammengefügt werden können. Des Weiteren wurden verschiedene Varianten des Visualisierungsprozesses diskutiert. Diese können abhängig von den Anforderungen des jeweiligen Einsatzszenarios die Generierungsgeschwindigkeit der Prozessvisualisierung erhöhen. Anhand der realisierten Prototypen wurde gezeigt, dass die entwickelten Konzepte umsetzbar sind. Die durch Proviado erzielbaren Visualisierungen sind flexibel konfigurierbar und an die Bedürfnisse der Benutzer anpassbar.

Teil IV

Abschluss

9 Verwandte Arbeiten

Dieses Kapitel diskutiert die existierenden Ansätze und Werkzeuge zur Prozessvisualisierung. Es ergänzt die in den vorangehenden Kapiteln für bestimmte Teilgebiete bereits vorgestellten verwandten Arbeiten:

- **Grundlagen von Prozess-Views** (siehe Abschnitt 4.7): Es existieren verschiedenste Ansätze für Prozesse zugehörige Views zu definieren und zu verwenden. Für die Zwecke der Visualisierung sind diese jedoch nicht ausreichend. Es wird zumeist eine strikte Korrektheit der resultierenden Prozessmodelle gefordert. Für viele relevante Fälle, in denen die zu aggregierenden Aktivitäten beispielsweise nicht zusammenhängend sind, ist eine View-Bildung mit diesen Ansätzen nicht möglich.
- **Praktische Aspekte von Prozess-Views** (siehe Abschnitt 5.7): In den meisten existierenden Ansätzen für Prozess-Views wird davon ausgegangen, dass der Modellierer die View-Modelle händisch modelliert. Es gibt jedoch einige wenige, die ein operationales Vorgehen für die Definition von Views beschreiben. Mit dem in dieser Arbeit beschriebenen Verfahren wird dagegen ein mehrschichtiger, operationaler Ansatz verfolgt, der die Möglichkeiten existierender Ansätze bei weitem übersteigt und der somit eine solide Grundlage für jede praktische Anwendung bietet.
- **Konfiguration der graphischen Darstellung** (siehe Abschnitt 6.5): Die existierenden Ansätze für die Generierung einer graphischen Darstellung aus einem abstrakten Prozessmodell unterscheiden sich vorrangig hinsichtlich ihrer Konfigurierbarkeit. Während einige Ansätze die Prozessmodelle direkt in das Zielformat transformieren, erlauben andere die Konfiguration dieser Transformation mittels einer Darstellungsbeschreibung. Unterschiede gibt es auch in Bezug auf das verwendete Zielformat. Durch die Verwendung von SVG als Graphikformat, in Verbindung mit einer XML-basierten Darstellungsbeschreibung, ermöglicht Proviado eine flexible Anpassbarkeit der graphischen Darstellung, die in keinem der anderen Ansätze zu finden ist.

Dieses Kapitel fokussiert nun auf die direkt mit der Visualisierung von Prozessen verwandten Arbeiten. Zunächst diskutieren wir in Abschnitt 9.1 wissenschaftliche Ansätze im engeren Sinne. Da diese Ansätze jedoch nicht sehr weit reichen oder nur sehr spezielle Aspekte abdecken und einige der am Markt verfügbaren Werkzeuge viel versprechend sind, gehen wir in Abschnitt 9.2 auch auf diese kommerziellen Prozessvisualisierungswerkzeuge ein.

9.1 Konzeptionelle Ansätze für die Prozessvisualisierung

Mehrere Quellen betonen die Bedeutung der Prozessvisualisierung für das Geschäftsprozessmanagement im Allgemeinen [Lof02] und für das Management von Projekten im Speziellen [YNY07]. So sieht [Rem02] die Prozessvisualisierung als wichtigen Teil des Knowledge-Managements in Unternehmen. Dagegen existiert nur eine überschaubare Anzahl an Arbeiten, die sich explizit mit Ansätzen bzw. Themen der Visualisierung von Prozessen befassen. Keine dieser Ansätze sind aber derart umfassend und generisch wie die in Proviado. Oftmals handelt es sich um Arbeiten, die eine einfache, nicht konfigurierbare Visualisierung für eine spezielle Prozessmodellierungssprache entwickeln. Darunter fallen zum Beispiel visualBPEL, eine graphische Notation für BPEL[1] [Wei03, MSW+04], Ansätze für die Visualisierung von EPKs[2] als SVG-Graphiken [MBN04] oder von BPEL in EPK-Notation [MZ05a]. Des Weiteren gibt es Arbeiten, die einen Visualisierungsansatz für die Darstellung von Prozessen in größerem Kontext (z.B. Enterprise-Architecture-Management [Lan05]) bieten [IL04, SADL04, BEL+07]. Die in diesem Zusammenhang gebotenen Ansätze und Konzepte stehen zumindest teilweise in Relation zu Proviado.

Ein sehr früher Ansatz für die Visualisierung von Prozessen war Improvise [BKC95]. Er unterstützt die Darstellung von Prozessmodellen in einer eigenen Notation. Die Besonderheit von Improvise liegt in der Möglichkeit, Multimedia-Inhalte (z.B. Videos) mit Aktivitäten zu verknüpfen.

ShowBiz [WW96a, WW96b] ist ebenfalls ein relativ alter Ansatz zur Prozessvisualisierung. Zur Vereinfachung des Prozessmodells wird eine Aggregation zusammenhängender Blöcke unterstützt, die interaktiv vom Benutzer in der Oberfläche definiert und dann automatisch zu SESE-Blöcken (vgl. Kapitel 4) erweitert werden. Neben der Zeichnung eines Prozesses als Graph bietet ShowBiz auch die Möglichkeit einer Swimlane-Darstellung. Dabei können beliebige Prozessattribute zur Partitionierung der Prozessaktivitäten verwendet werden. Der Fokus von ShowBiz liegt in der Layout-Berechnung für Prozesse. Im Vergleich zu Proviado ist die Aggregation hier auf zusammenhängende Blöcke von Kontrollflusselementen beschränkt. Andere Prozessaspekte (Datenfluss, Attribute, Bearbeiter) werden nicht betrachtet. Der in ShowBiz realisierte Layout-Algorithmus berechnet das Layout des Prozesses jedes Mal vollständig neu. Die damit verbundenen Probleme (vgl. „Mental Map“ in Abschnitt 7.3.2) werden zwar angesprochen, aber im Wesentlichen als zukünftiges Forschungsthema gesehen.

[Her99] präsentiert Anforderungen für eine möglichst einfache, d.h. für Laien verständliche, Prozessdarstellung. Dabei wird auf die eigens entwickelte Modellierungsmethode und Prozessnotation SeeMe zurückgegriffen. Aktivitäten sollen in mehreren Stufen zu abstrakten Aktivitäten

[1] Business Process Execution Language [OAS07]
[2] Ereignis-gesteuerte Prozessketten [KNS92, SN00]

zusammengefasst und dann schrittweise wieder eingeblendet werden können. Aktivitäten werden dabei immer innerhalb ihrer abstrahierten Aktivität gezeichnet (Abbildung 9.1 zeigt ein Beispiel für eine Aktivität „begutachten“ und deren (Teil-)Aktivitäten). Eine weitere Vereinfachung soll durch das Ausblenden von Prozesselementen erzielt werden. Obwohl die formulierten Anforderungen dieses Ansatzes den Ideen von Reduktion und Aggregation in Proviado ähneln, ist ein Vergleich ob der verwendeten Modellierungsmethodik im Detail nicht möglich. Proviado legt sich im Gegensatz zu diesem Ansatz nicht auf eine Prozessnotation fest, sondern erlaubt die Verwendung beliebiger Notationen.

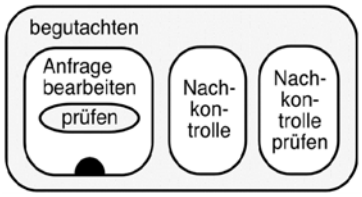


Abbildung 9.1: Beispiel für die Darstellung einer Aktivität in [Her99]

[SBE00] zeigt, wie Prozesse im dreidimensionalen Raum dargestellt werden können. Abbildung 9.2 zeigt ein Beispiel, in dem zusätzliche Kennzahlendiagramme in die Prozessdarstellung integriert sind. Die Interaktion mit einer solchen Prozessdarstellung dürfte für die meisten Betrachter allerdings zu komplex sein. Problematisch ist vor allem die Navigation im dreidimensionalen Raum mit zweidimensionalen Steuergeräten (Maus).

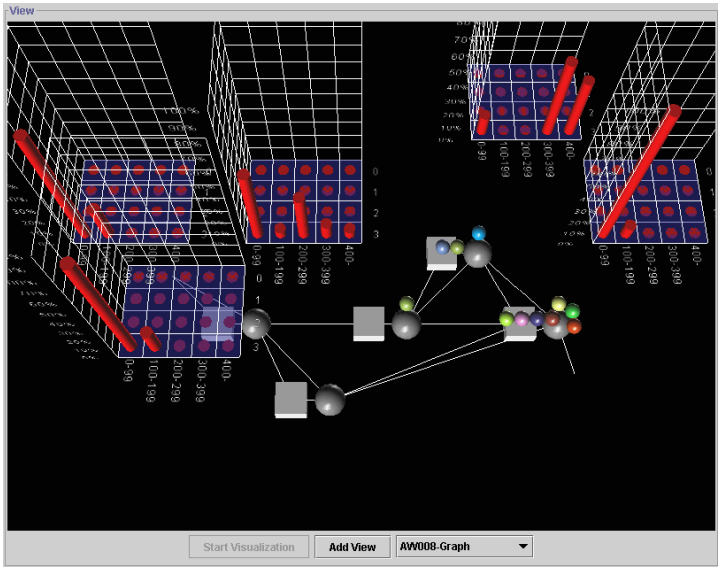


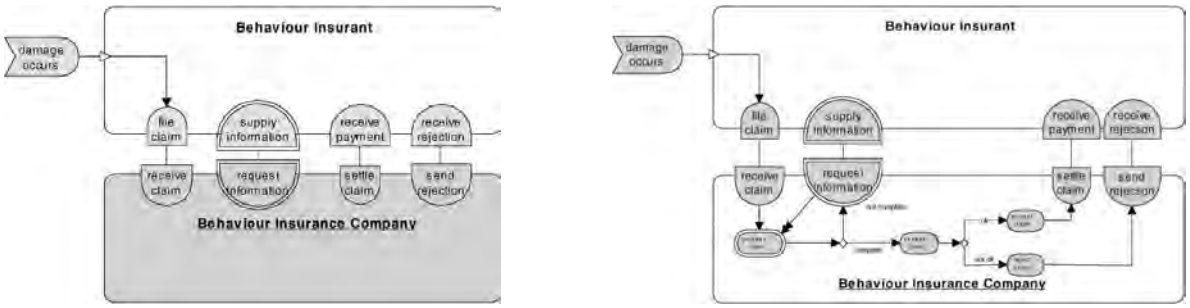
Abbildung 9.2: Dreidimensionale Darstellung eines Prozesses [SBE00]

Perspektiven von Prozessmodellen

Ähnlich wie Proviado motivieren verschiedene Ansätze, dass für unterschiedliche Betrachter bzw. Akteure mit jeweils unterschiedlichen Anforderungen an eine Visualisierung verschiedene

Perspektiven auf ein und dasselbe Prozessmodell benötigt werden. In Proviado ermöglicht uns der View-Mechanismus, in Kombination mit der flexiblen Gestaltung und Auswahl der Notation, eine umfassende Adaptierung und damit Personalisierung des Prozessmodells. Ferner ermöglichen es die Proviado-Darstellungsformen (vgl. Abschnitt 2.2.4.2), die Prozessinformationen in unterschiedlicher, auf den jeweiligen Verwendungszweck zugeschnittener Art und Weise darzustellen.

[LLW+01] beschreibt eine Visualisierungslösung für die Prozessmodellierungssprache AMBER. In AMBER lässt sich, neben der üblichen Kontrollflussdarstellung, auch eine Interaktion zwischen mehreren Prozessen darstellen. Für eine Vereinfachung der Prozessdarstellung können Blöcke eines blockstrukturierten Kontrollflussgraphen ein- bzw. ausgeklappt werden. Da neben dem Kontroll- und Datenflussaspekt auch Bearbeiter abgebildet werden können, werden entsprechende Perspektiven angeboten, um beispielsweise die Organisationsstruktur darzustellen. Dieser Visualisierungsansatz ist auf die Modellierungssprache AMBER fokussiert. Während eine Swimlane-Darstellung unterstützt wird, ist eine Instanzvisualisierung ist nicht vorgesehen. Der Ansatz zum Ein- und Ausblenden von Prozessblöcken kann mit der Aggregation in Proviado verglichen werden. Die Mächtigkeit bleibt jedoch weit hinter den von Proviado bereitgestellten Mechanismen zur Aggregation zurück. Beispielsweise können in AMBER nur abgeschlossene Blöcke aggregiert werden. Wichtige Details, die für die Realisierung wichtig sind (z.B. die Abbildung der Prozesselementattribute), werden nicht behandelt.



(a) (b)

Abbildung 9.3: Ein- und Ausblendung von Prozessbereichen in [LLW+01]

Der in [JG07] beschriebene Ansatz basiert auf der Perspektiven-orientierten Modellierung (POPM, Perspective Oriented Process Modeling). Diese Modellierungssprache bietet fünf fundamentale Perspektiven zur Modellierung von Prozessen. Sie beschreiben, wie ein bestimmter Prozessaspekt (Datenfluss, Bearbeiter, Kontrollfluss, Systeme, etc.) in den Prozess involviert ist. In [JG07] wird ein Basisdatenmodell für die Visualisierung von POPM-Prozessen vorgestellt. Davon werden für jede Perspektive spezielle Datenmodelle abgeleitet, welche die benötigten Daten für eine Visualisierung der jeweiligen Instanz repräsentieren. Was in Proviado mittels Konfiguration von Templates in SVG realisiert wird, wird hier durch fixe Datenstrukturen abgebildet und kann daher nicht so einfach und flexibel an die Bedürfnisse angepasst werden.

Enterprise-Architecture-Management

Weitere Ansätze für die Prozessvisualisierung stammen aus dem Bereich des Enterprise-Architecture-Management (EAM) [JBA+03, DIL+04, IL04, LDL04, SADL04, JLB+04, Lan05, Fil06, FH06, FK06, BEL+07]. Hierbei geht es in erster Linie um die Verwaltung der IT-Systeme in einem Unternehmen, von der Kontrolle der eingesetzten Software-Komponenten und ihrer Versionen bis hin zur Planung von System- bzw. Versionsmigrationen. Ein nicht unbedeutender Aspekt dabei sind die Geschäftsprozesse und die zu ihrer Unterstützung eingesetzten IT-Systeme. Aus diesem Grund bieten viele Lösungen für das Enterprise-Architecture-Management auch Komponenten zur Abbildung von Prozessen bzw. deren Visualisierung an.

Wie bereits in Abschnitt 6.5 beschrieben, verwendet [Fil06, FH06, FK06] eine Ontologie für die Zuordnung von Visualisierungselementen (Symbolen) zu Prozesselementen. Das hierdurch erzielte Ergebnis ist vergleichbar mit dem in Kapitel 6 vorgestellten Template-Mechanismus, bleibt jedoch hinsichtlich Flexibilität hinter dem von Proviado unterstützten Template-Mechanismus zurück, da die Symbole direkt bestimmten Metamodellelementen zugeordnet werden. Eine flexible oder dynamische Zuordnung, etwa abhängig von Applikationsdaten, ist nicht möglich.

Ein sehr umfangreiches Konzept für das Enterprise-Architecture-Management bietet ArchiMate [JBA+03, DIL+04, IL04, LDL04, SADL04, JLB+04, Lan05]. ArchiMate erlaubt es, die diversen Aspekte des Enterprise-Architecture-Management (Organisationsmodell, Datenmodell, Systemlandkarte, Prozessmodelle) abzubilden. Dabei wird aber nicht auf ein einziges Werkzeug zurückgegriffen, sondern es werden die in den Unternehmen etablierten Werkzeuge für die Abbildung der einzelnen Teilaspekte genutzt und miteinander verknüpft. Anstelle einer Anbindung der jeweiligen Systeme über unzählige, systemspezifische Schnittstellen, setzt ArchiMate eine einheitliche, gemeinsame Datenhaltung voraus [SADL04]. Während die einzelnen Aspekte dadurch immer noch mittels ihrer originären Werkzeuge bearbeitet werden können, verwaltet ArchiMate als integrative Komponente die Zusammenhänge zwischen den Aspekten. Demnach können auch Prozesse mit der ArchiMate eigenen Notation, in Verbindung mit involvierten IT-Systemen, visualisiert werden. Für die Visualisierung wird dabei, ebenso wie in Proviado, der Inhalt der Visualisierung (Daten) getrennt von der Darstellungsbeschreibung verwaltet.

Ein interessanter Ansatz ist in [SL06] beschrieben. Basierend auf UML und BPMN werden Unternehmensarchitekturen modelliert. Zur Vereinfachung der Darstellung werden mittels einer Metamodellprojektion (vgl. Abschnitt 4.7) ausgewählte Elemente des Metamodells (z.B. Metaobjekt Bearbeiter) entfernt. Folglich sind die entsprechenden Objekte auf Modellebene (z.B. Bearbeiterobjekte) in den Prozessmodellen ebenfalls nicht mehr vorhanden. Dadurch lassen sich Details verbergen, um beispielsweise eine Management-taugliche Darstellung einer IT-Architektur zu erstellen.

Den vorgestellten Ansätzen zum Enterprise-Architecture-Management ist gemein, dass diese zwar prinzipiell die Visualisierung von Prozessen unterstützen. Da deren Fokus jedoch auf anderen Themen liegt, sind die Konzepte für die Prozessvisualisierung hinsichtlich Konfigurierbarkeit und Personalisierung nicht mit den in Proviado angebotenen vergleichbar.

Monitoring von Prozessen

Für das Monitoring laufender Prozesse werden häufig graphische Darstellungen um entsprechende Prozessdaten bzw. -kennzahlen angereichert. Wie in Kapitel 3 diskutiert, liegt der Fokus

von Proviado auf der Visualisierung einzelner Prozessmodelle bzw. -instanzen. Zwei Varianten für die Visualisierung von Prozessinstanzen beschreibt [Bau04b]. Die erste davon geniert aus einer Darstellungsbeschreibung und einem Prozessmodell die Prozessdarstellung. Dies entspricht einem vereinfachten Vorgehensmodell von Proviado, wobei Views und Templates nicht betrachtet werden. Die zweite Variante beschreibt einen pragmatischen Ansatz, bei dem die Prozessvisualisierung von Hand erstellt (d.h. gemalt) wird. Die endgültige Darstellung wird dann zur Laufzeit erzeugt, indem aktuelle Laufzeitdaten in die Visualisierung eingebettet werden.

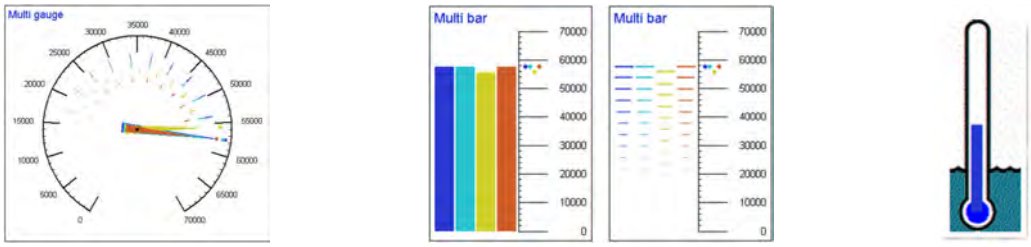
[KFL03] bietet eine technische Beschreibung eines vom Prinzip her einfachen Ansatzes, der Prozesse als SVG-Graphiken darstellt. Den Fokus dieses Ansatzes bildet die Anpassung der Visualisierung infolge einer Aktualisierung der Prozessdaten zur Laufzeit. Dazu werden die geänderten Daten in Form eines Events an den Visualisierungs-Client übermittelt, der die Darstellung dann über JavaScript anpasst. Die Zuordnung, welche Darstellungsänderung auf ein bestimmtes Event zu erfolgen hat, muss vorab definiert werden. Eine weitergehende Adaption der Visualisierung an die Bedürfnisse des Betrachters bzw. eine personalisierte Prozessvisualisierung, wie von Proviado unterstützt, ist mit diesem Ansatz nicht möglich. Dafür ermöglicht er eine effiziente Anpassung von Instanzdarstellungen, während in Proviado für diesen Anwendungsfall eine neue Darstellung erzeugt werden muss. Dies ist eine Folge des flexiblen Visualisierungsansatzes von Proviado, bei dem sich mit Änderung der Ausführungsdaten sowohl die View-Berechnung als auch die Notation ändern können.

[MN03b] beschreibt eine Erweiterung der Prozessnotation BPMN um graphische Elemente zur Darstellung von Aktivitätenzuständen. Es werden Darstellungselemente vorgeschlagen, um beispielsweise verschiedene Eskalationsstufen bei Verspätung einer Aktivität zu visualisieren. Detailinformationen zur Ausführung werden, wie es in Proviado auch möglich ist, als Tooltipps dargestellt. Insgesamt beginnt [MN03b] mit einer sehr viel versprechenden Anforderungsbeschreibung für eine Prozessvisualisierung, von der sich viele Anforderungen in Proviado wieder finden. Letztlich bietet die vorgeschlagene Lösung jedoch nur eine Erweiterung von BPMN an, d.h. allgemeine Konzepte für die personalisierte Prozessvisualisierung werden nicht geboten.

Neben dem Monitoring einzelner Prozessinstanzen gibt es auch Ansätze für das Monitoring und die Darstellung von Instanzkollektionen. Ein Ziel dabei ist, aus den Ausführungsdaten der Instanzen Kennzahlen abzuleiten. Unter den Stichworten „Business Intelligence“ oder „Process Warehouses“ finden sich hierzu einige Arbeiten, von denen ein paar Vertreter im Folgenden besprochen werden. Viele der Arbeiten fokussieren auf die Gewinnung der notwendigen Daten und der Ableitung entsprechender Kennzahlen (Key Performance Indicator). Einen guten Überblick über zum Einsatz kommende Techniken und verwandte Arbeiten gibt [Mue04].

Ein Beispiel für ein solches Monitoring-Werkzeug ist das Business Process Cockpit (inzwischen iBOM), ein Prototyp aus der Forschung von Hewlett-Packard [SCDS02, CCS04, GCC+04, CCSD05]. Hier werden die Daten verschiedener prozessausführender Systeme in einem Process Warehouse abgelegt. Aus ihnen wird dann eine Vielzahl aggregierter Kennzahlen berechnet, die zum Monitoring sowie zur Leistungsmessung der Prozesse dienen. Ein weiteres Beispiel für die Darstellung von aus der Prozessausführung abgeleiteten Kennzahlen findet sich in [HKD04, HKDS06]. Bei all diesen Ansätzen geht es nicht, wie im Fall von Proviado, um die Visualisierung einzelner Prozesse, sondern es wird immer eine Kollektion von Prozessen betrachtet bzw. analysiert.

Eine ähnliche Form der Darstellung von Prozesslaufzeitdaten findet man beim Monitoring von technischen Prozessen, beispielsweise eines chemischen Umwandlungsprozesses. Oft ist in diesem Zusammenhang auch von Leitständen die Rede. Dem Begriff des Prozesses kommt in diesem Kontext eine andere Bedeutung zu. Es handelt sich meist nicht um einzeln ablaufende Prozessinstanzen, sondern um kontinuierliche Prozesse, die z.B. zu einer chemischen Veränderung eines Stoffes führen. Ein anderes Beispiel ist die Darstellung von Messwerten in Absicherungs- oder Testprozessen. In einer Prozessvisualisierung werden hierbei bestimmte Werte aus dem Prozess in Form von Tachodiagrammen oder Thermometern dargestellt. Beispiele für die verwendeten Visualisierungselemente finden sich in [MHSG02a] (vgl. Abbildung 9.4).



(a) (b) (c)

Abbildung 9.4: Darstellungselemente für das Monitoring von Kennzahlen [MHSG02b]

9.2 Werkzeuge für die Prozessvisualisierung

In der Praxis existiert eine Reihe verschiedener Systeme, die zur Visualisierung von Prozessen verwendet werden können. Eine Kategorisierung zeigt Abbildung 9.5, eine ausführliche Diskussion findet sich in [RR03]. Generell kann man zwischen prozessneutralen Visualisierungswerkzeugen und prozessorientierten Werkzeugen unterscheiden.

9.2.1 Prozessneutrale Werkzeuge

Die prozessneutralen Visualisierungswerkzeuge charakterisiert, dass sie nicht originär für die Darstellung von Prozessen bzw. Prozessinformationen ausgelegt sind. Vielmehr handelt es sich um allgemeine Visualisierungswerkzeuge, die auch für die Darstellung von Prozessen verwendet werden können und in der Praxis auch werden. Im Einzelnen:

Graphikeditoren Jeder Graphikeditor kann zum Malen von Prozessen verwendet werden. Es existieren jedoch Graphikprogramme, die komplexe Zeichnungsobjekte (z.B. Symbole für die verschiedenen Prozesselemente) bereitstellen und verwalten. Ein Vertreter dieser Gruppe ist Visio [Mic07b]. Hier können Prozessnotationen als „Stencils“ definiert werden, die dann alle Objekte einer Prozessnotation zusammenfassen. Zur Erstellung einer Prozessdarstellung können einzelne Objekte der Notation wie Schablonen wieder verwendet werden. Durch umfangreiche Anpassungen lässt sich Visio auch zu einem Prozessmodellierungswerkzeug ausbauen [ITP06].

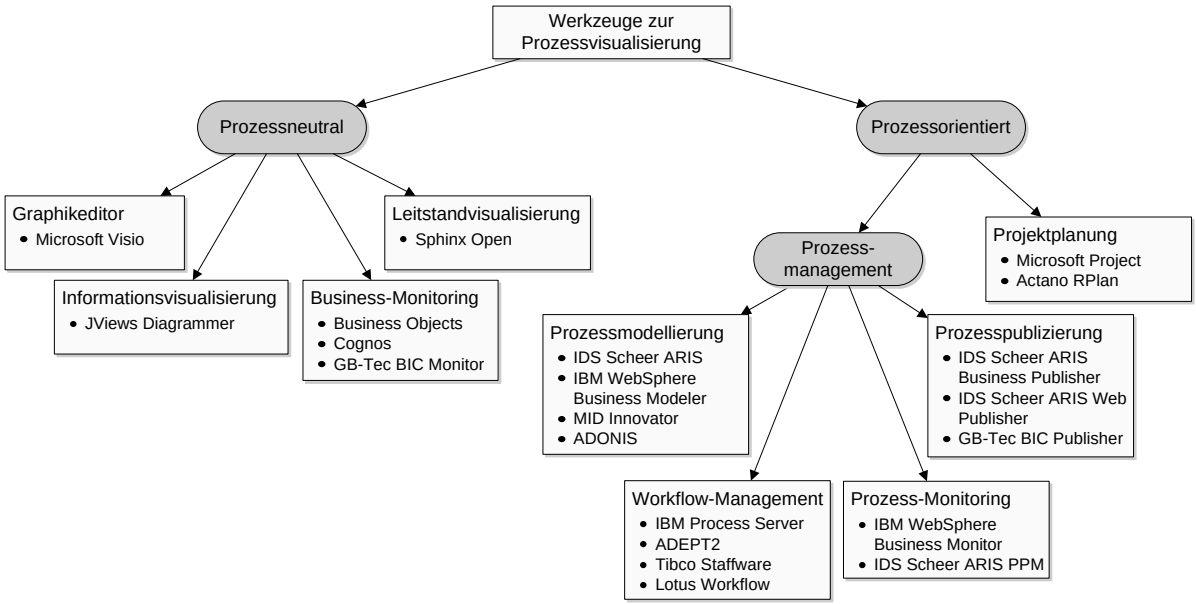


Abbildung 9.5: Überblick über ausgewählte Systeme mit Bezug zur Prozessvisualisierung

Informationsvisualisierungswerkzeuge Mit dem JViews Diagrammer existiert ein Werkzeug für die Informationsvisualisierung, mit dem auch Prozessmodelle dargestellt werden können [ILO04]. Wie in Abschnitt 6.5 bereits angesprochen, steht der Flexibilität, die ein generisches Werkzeug grundsätzlich bietet, ein erhöhter Aufwand für dessen Konfiguration gegenüber. Im Kontext JViews wurden insbesondere auch Layout-Algorithmen für Prozessgraphen betrachtet [SV01, DDK+02].

Business-Monitoring-Werkzeuge Für die Visualisierung von Prozesskennzahlen, ohne explizite Darstellung des Prozesses (z.B. in Form eines Prozessgraphen), existieren zahlreiche Produkte aus den Bereichen Business Intelligence und Corporate-Performance-Management [IBM07, SAP07, GBT07].

Leitstandvisualisierung Für das Monitoring technischer Produktionsabläufe kommen in der Praxis so genannte Prozessleitstandsysteme zum Einsatz. Bei derartigen Ansätzen werden bestimmte Messdaten eines kontinuierlichen Produktionsablaufs überwacht und graphisch dargestellt. Ein Beispiel für ein solches System ist Sphinx Open [IN07]. Dieses Werkzeug erlaubt die Darstellung beliebiger Abläufe auf Basis einer Graphikdatei, in welche die jeweiligen Messwerte eingebettet werden.

9.2.2 Prozessorientierte Werkzeuge

Im Gegensatz zu den vorangehend vorgestellten, prozessneutralen Werkzeugen besitzen prozessorientierte Werkzeuge ein Metamodell, auf dem alle Prozesse basieren. Gleichzeitig beschränkt dies natürlich die Menge der abbildbaren Prozesse. Im Einzelnen:

Projektplanungswerkzeuge

Für eine Visualisierung von Prozessen, bei denen zeitliche Aspekte (z.B. Dauern und Fristen) wichtig sind, eignen sich Projektplanungs- bzw. Projektmanagementwerkzeuge. Bereits in Abschnitt 2.2.4.2 haben wir motiviert, dass eine Darstellung eines Prozesses als Gantt-Diagramm für spezielle Anwendungen wünschenswert ist. Gantt-Diagramme sind auch die primäre Darstellungsform von Projektplanungssoftware. Während die Reihenfolge und Dauern der Aktivitäten aus dieser Art der Darstellung gut abzulesen sind, können andere Aspekte, wie Datenfluss oder Bearbeiterzuordnungen, teilweise gar nicht abgebildet werden. Eine Option zur Verwaltung der Prozessmodelle besteht darin, diese mit Prozessmodellierungswerkzeugen zu definieren und bei Bedarf in das Projektplanungswerkzeug zu exportieren.

Bekannte Werkzeuge dieser Kategorie sind MS Project [Mic07a] und RPlan [Act07]. [Bau04a] nennt weitere Systembeispiele. Dort wird detailliert beschreibt, wie eine Kopplung von Projektplanungswerkzeugen und Workflow-Management-Systemen realisiert werden kann.

Prozessmanagementwerkzeuge

Prozessmanagementwerkzeuge besitzen alle ein zugrunde liegendes Metamodell, das dem in Proviado definierten Modell mehr oder weniger ähnelt. D.h. es können Kontrollfluss, Datenfluss, Bearbeiterzuordnungen und Systemverwendungen abgebildet werden. Jedes dieser Werkzeuge verfügt über eine Komponente zur Modellierung und Pflege der Prozessmodelle. Manche unterstützen zudem eine Ausführung der modellierten Prozesse (Workflow-Management-Systeme). Keines dieser Werkzeuge bietet die von Proviado unterstützte Konfigurierbarkeit der Darstellung. Auf die Unterschiede gehen wir im Folgenden ein.

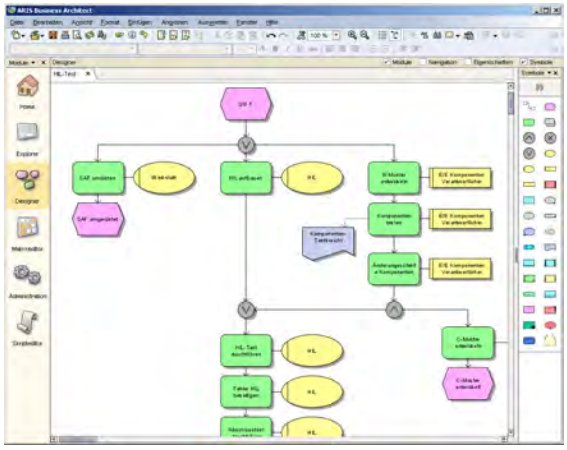
Prozessmodellierung Prozessmodellierungswerkzeuge werden verwendet, um umfangreiche Prozessdokumentationen anzulegen und zu pflegen. Ihnen zugrunde liegt meist eine Datenbank, in der die Modelldaten verwaltet werden. Im Vergleich zu prozessneutralen bzw. nicht datenbankbasierten Systemen haben sie den Vorteil, dass Änderungen, die an einer Stelle eines Prozesses gemacht werden, sofort in der gesamten Prozessdokumentation sichtbar sind (z.B. Änderung eines Namens einer Aktivität). Dies vereinfacht die Pflege der Modelle signifikant. Eine Darstellung der Modelle unterstützt jedes dieser Werkzeuge mittels einer entsprechenden Benutzeroberfläche. Dabei lässt sich teilweise auch die Prozessnotation anpassen. Änderungen an der Prozessstruktur unterstützen die Produkte dagegen nicht. Zielgruppe für diese Art von Werkzeugen sind Prozessmodellierer. Entsprechend muss eine für andere Betrachter angepasste Darstellung von Hand angelegt werden. D.h. auf Basis eines bestehenden Modells werden mehrere Darstellungen desselben Prozesses erstellt, die jeweils eine andere Perspektive auf den Prozess bieten (z.B. ohne Datenfluss und Bearbeiter, ohne technische Aktivitäten wie Datentransformationsschritte, etc.).

Kommerzielle Modellierungswerkzeuge unterscheiden sich vor allem bezüglich der von ihnen verwendeten Modellierungssprache und damit der verwendeten Prozessnotation. Da beide Aspekte direkt mit dem zugrunde liegenden Metamodell korreliert sind, kann die Modellierungssprache nicht und die Notation nur in engen Grenzen angepasst werden. Tabelle 9.1 zeigt bekannte Modellierungswerkzeuge und die von ihnen verwendete Modellierungssprache bzw. -notation. Insbesondere die ARIS Plattform hat sich vom ursprünglich reinen Prozessmodellierungswerkzeug [Sch96a] mittlerweile zur universellen Plattform für die Abbildung von Prozessen,

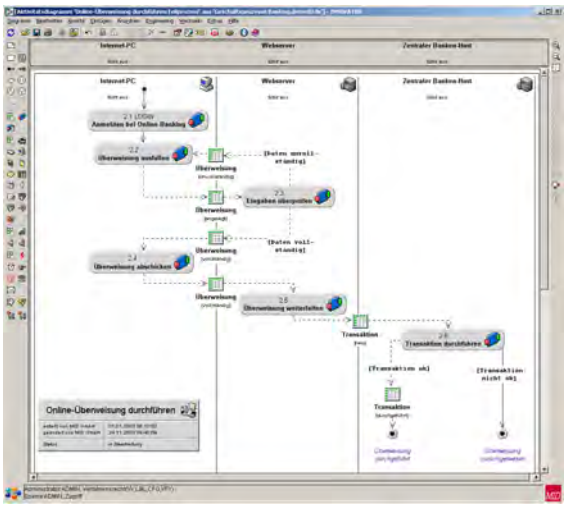
IT-Landschaften, SOA-Architekturen, Organisationsstrukturen und vielem mehr entwickelt [DB07].

Werkzeug	Modellierungssprache	Abb.
IDS Scheer ARIS [DB07]	EPK (Ereignis-gesteuerte Prozessketten)	9.6a
MID Innovator [MID06]	UML (Unified Modeling Language) Activity Charts	9.6b
IBM WebSphere Business Modeler [IBM06, WAM+07]	proprietär	9.6c
BOC Information Systems ADONIS [BOC07]	BPMN (Business Process Modeling Notation), EPKs, UML, u.a.	9.6d

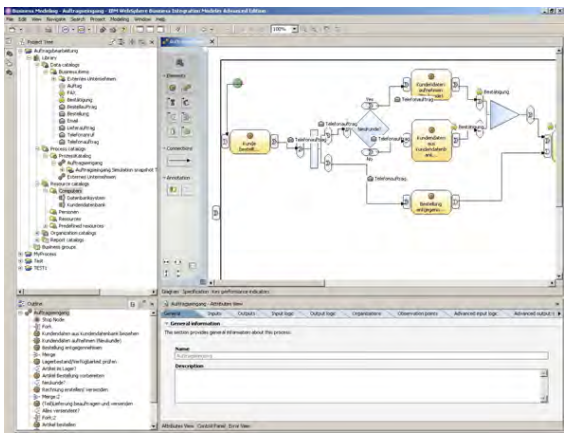
Tabelle 9.1: Bekannte Prozessmodellierungswerkzeuge



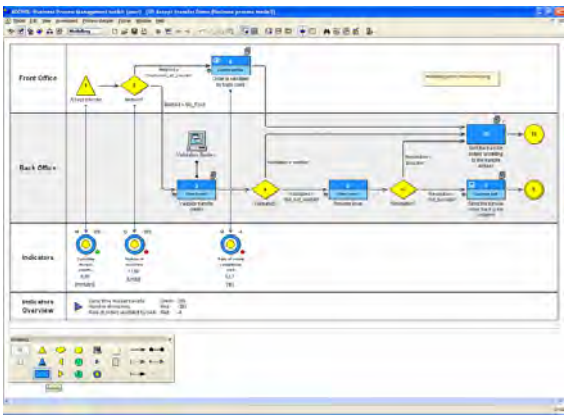
(a) ARIS Business Designer



(b) Innovator



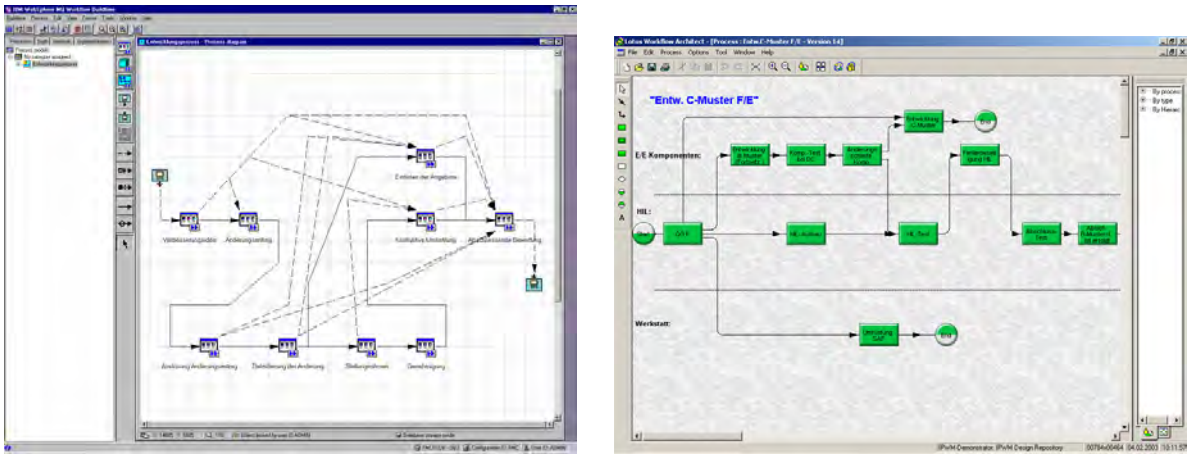
(c) WBI Modeller



(d) ADONIS [BPT05]

Abbildung 9.6: Beispiele für Prozessmodellierungswerkzeuge

Workflow-Management-Systeme Workflow-Management-Systeme (WfMS) steuern die Ausführung von Prozessen. Dabei fallen neben Modelldaten (d.h. Daten zum Prozessschema) auch Laufzeitdaten der Prozessinstanzen, wie Status, tatsächliche Bearbeiter und Applikationsdaten, an. Diese sollten auch in einer Instanzvisualisierung darstellbar sein. WfMS werden daher in der Regel mit einer Komponente zur Anzeige bzw. zum Monitoring einzelner laufender Prozessinstanzen ausgeliefert (vgl. Einzelinstanzdarstellung in Abschnitt 3.2). Zielgruppe dieser Visualisierung sind der Prozessadministrator, -entwickler oder -tester, so dass entsprechende Darstellungen meist sehr technisch sind und nicht oder nur geringfügig an die Bedürfnisse des Betrachters anpassbar sind. Abbildung 9.7 zeigt Beispiele für eine Prozessinstanzvisualisierung in MQ Workflow (Abbildung 9.7a) und Lotus Workflow (Abbildung 9.7b)



(a) IBM MQ Workflow (b) Lotus Workflow

Abbildung 9.7: Beispiele für eine Prozessinstanzvisualisierung in WfMS

Im Editor von ADEPT2, einem WfMS, das Änderungen der Prozessinstanzen zur Laufzeit gestattet [RD98], können zur Vereinfachung der Prozessdarstellung zum Beispiel Datenkanten ausgeblendet werden, um so die Übersichtlichkeit der Darstellungen zu verbessern (vgl. Abbildung 9.8). Gerade ein System wie ADEPT2, bei dem Endanwender zur Laufzeit Modifikationen von sich in Ausführung befindlichen Prozessinstanzen vornehmen können, wird von der Möglichkeit einer personalisierbaren Prozessvisualisierung enorm profitieren.

Ein anderes WfMS ist der WebSphere Process Server, der aus dem Process Choreographer hervorgegangen ist [KKL+04b, KKL+04a, PG05, WAM+07]. Er unterstützt die Ausführung von in BPEL spezifizierten Prozessen. Ein Beispiel für die Darstellung eines Prozesses in der Modellierungskomponente des Process Servers, genannt WebSphere Integration Developer (WID), zeigt Abbildung 9.9.

Prozess-Monitoring Werkzeuge für das Monitoring von Prozessinstanzkollektionen zeigen die Ausführungsdaten dieser Instanzen in aggregierter Form an (vgl. aggregierte Multi-Instanzdarstellung in Abschnitt 3.2). Es existieren Werkzeuge wie der IBM WebSphere Business Monitor, die das Prozessmodell anzeigen und in dieses die aggregierten Daten bzw. Kennzahlen einblenden [IBM06, WAM+07]. Die Kennzahlen können auch graphisch aufbereitet und in Form

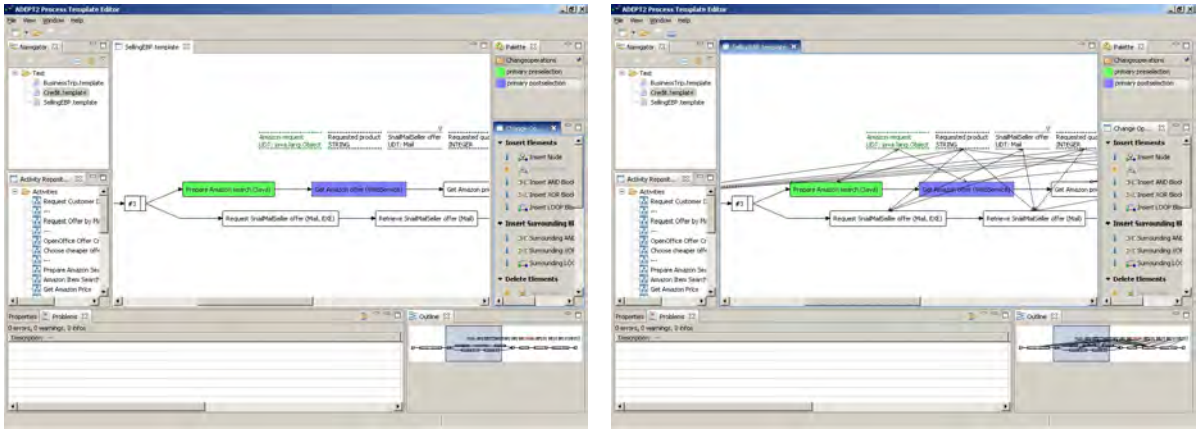


Abbildung 9.8: Ein- und Ausblenden von Datenkanten im ADEPT2-Editor

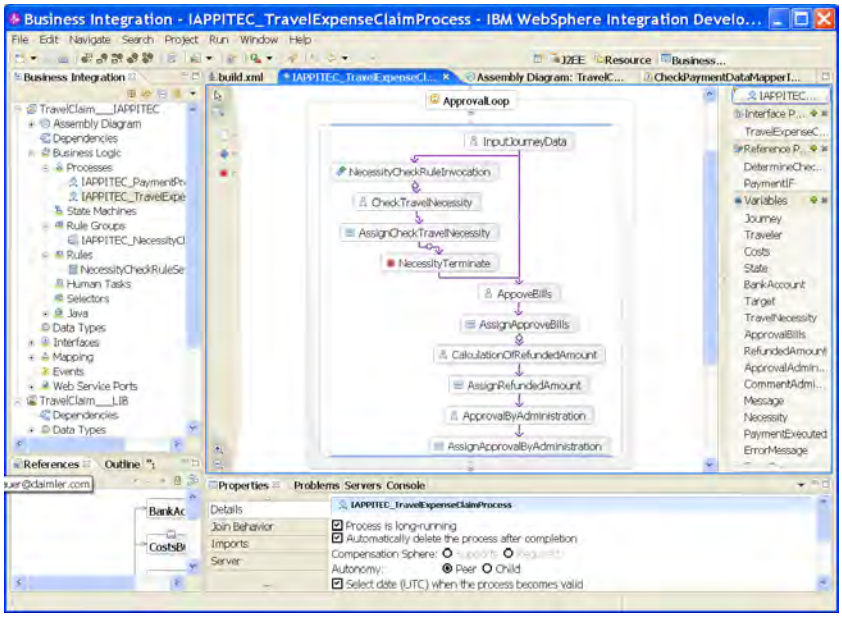


Abbildung 9.9: BPEL-Prozess im IBM WebSphere Integration Developer (WID)

von Balken-, Tachodiagrammen, Fieberthermometern oder ähnlichen graphischen Elementen dargestellt werden (vgl. Abbildung 9.10).

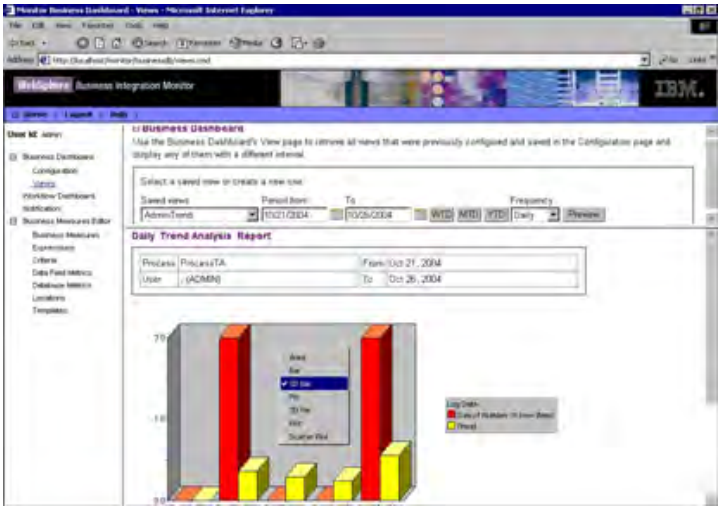
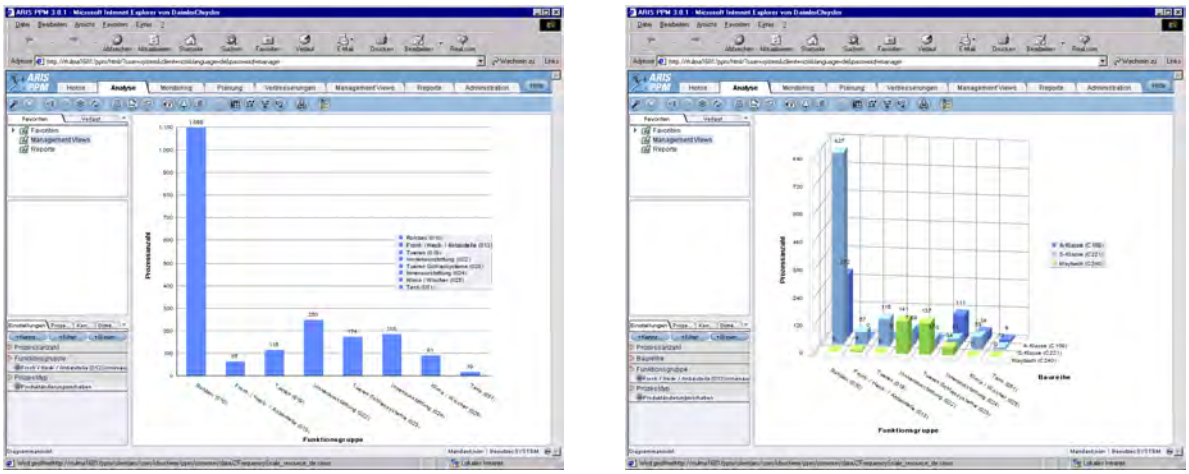


Abbildung 9.10: IBM WBI Monitor

Andere Werkzeuge für die Visualisierung von Instanzkollektionen verzichten auf eine explizite Darstellung des Prozessmodells und legen den Fokus auf die Visualisierung von aus der Prozessausführung gewonnenen Kennzahlen. Ähnlich Datawarehouse- oder Business-Intelligence-Anwendungen werden Daten gesammelt und zur Auswertung bereitgestellt bzw. in aggregierter Form dargestellt. Der ARIS Process Performance Monitor [Heß96, IDS04] etwa bietet ausgefeilte Möglichkeiten zur explorierenden Analyse von Ausführungsdaten bis hinab auf die Ebene einer einzelnen Instanz (vgl. Abbildung 9.11). Als Besonderheit ist der Process Performance Monitor explizit darauf ausgelegt, die Ausführungsdaten systemübergreifender Prozesse zu erfassen und zu analysieren. Verschiedene Varianten zur Anbindung an bestehende Systeme haben wir in [BB07a] diskutiert.



(a) vorher (b) nachher

Abbildung 9.11: Explorierende Analyse von Prozesskennzahlen

Generell kann man für alle vorangehend vorgestellten Werkzeuge sagen, dass die Visualisierung von einzelnen Prozessinstanzen nicht anpassbar ist. Selbst graphische Anpassungen, wie in Kapitel 6 beschrieben, werden nicht in dem von Proviado gebotenen Maße unterstützt, d.h. man muss üblicherweise mit der Notation auskommen, die das Werkzeug zur Verfügung stellt.

Prozesspublizierung Ein weiterer Anwendungsfall ist es, die dokumentierten Modelle öffentlich verfügbar zu machen. Analog soll bei Prozessausführungssystemen der Zustand der Prozessinstanzen dargestellt werden können. Dementsprechend bieten die meisten Werkzeuge mehr oder weniger flexible Visualisierungskomponenten an, mit deren Hilfe die Prozessmodelle im Intranet des Unternehmens einem breiten Nutzerkreis zur Verfügung gestellt werden können.

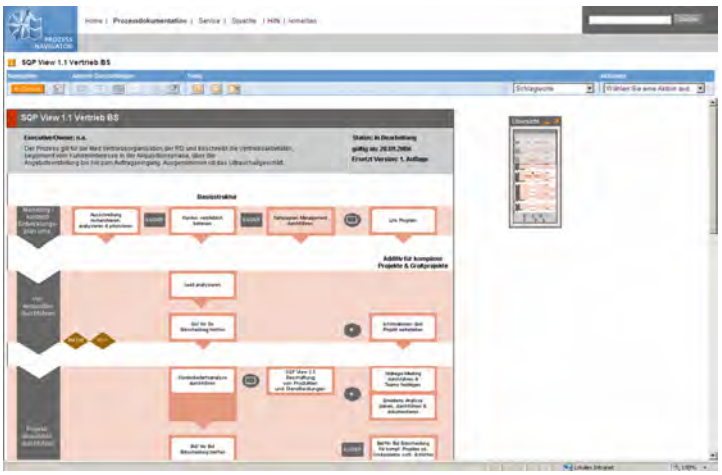
Für die ARIS Plattform bietet IDS Scheer zwei Publizierungswerkzeuge an:

1. Der ARIS Web Publisher [IDS07b] generiert aus der ARIS Datenbank einen statischen HTML-Export. Beim Export werden für alle Prozessmodelle entsprechende Graphiken erzeugt. Im Gegensatz zu Proviado ist bei dieser Vorgehensweise eine Anpassung der Modelle, sei es graphisch oder strukturell, nach dem Export nicht mehr möglich.

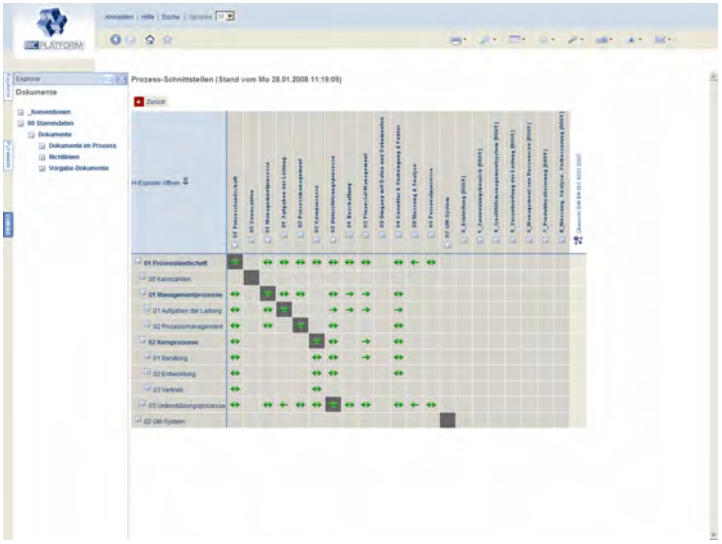
2. Deutlich flexibler ist dagegen der ARIS Business Publisher [IDS07a]. Er verwendet eine eigene Datenbank, aus der die Prozessdarstellungen auf Anfrage generiert werden. Prinzipiell erfolgt die Darstellung ähnlich wie in ARIS üblich, wobei gegebenenfalls die Symbole ausgetauscht werden können. Mit so genannten HTML-Knoten unterstützt der Business Publisher auch eine Anpassung der Notation, wie sie unser Template-Mechanismus aus Kapitel 6 erlaubt. Dennoch bleibt die Lösung in Sachen Konfigurierbarkeit hinter dem hier vorgestellten Mechanismus zurück, da beispielsweise keine Berechnungen (z.B. Hinterlegung einer Funktion, die einem konkreten Zahlenwerte eine Kategorie hoch, mittel, tief zuordnet; vgl. Abschnitt 6.3.1.1) innerhalb der Symbole möglich sind.

Da ARIS als reines Modellierungswerkzeug keine Ausführungskomponente besitzt, wird die Visualisierung von Prozessinstanzen auch von den Publizierungswerkzeugen nicht unterstützt.

Das momentan vielseitigste Werkzeug für die Prozessvisualisierung ist BIC Publish [GBT06]. Ursprünglich entwickelt als Publikationswerkzeug für ARIS, unterstützt es heute auch andere Prozessmodellierungswerkzeuge. Ausgehend von deren Datenbasis und beliebig definierbaren Prozessnotationen werden die Darstellungen der Prozesse erzeugt. Standardmäßig verfügt BIC Publish über Mechanismen, um die Ereignisse der ARIS EPKs aus den Modellen zu entfernen. Weitergehende strukturelle Anpassungen, wie sie unser View-Mechanismus beschreibt, sind hingegen nicht möglich. Die Notation kann flexibel angepasst werden, um beispielsweise Bearbeiter und Dokumente innerhalb des Aktivitätensymbols darzustellen (vgl. Abbildung 9.12a). Zusätzlich kann ein Prozess auch in Tabellenform oder in Swimlane-Darstellung angezeigt werden. Ebenso unterstützt wird eine Matrix-Darstellung (vgl. Abbildung 9.12b). Im Gegensatz zu den ARIS-Werkzeugen können im BIC Publisher auch Laufzeitdaten in die Prozessvisualisierung integriert werden.



(a) Prozessdarstellung



(b) Matrixdarstellung

Abbildung 9.12: Prozessvisualisierung mit BIC Publish

10 Zusammenfassung und Ausblick

Heutige Unternehmen sehen sich mit einer immer komplexer werdenden Prozesslandschaft konfrontiert. Um die steigenden Anforderungen an eine effiziente und rasche Abwicklung ihrer Geschäftsprozesse zu erfüllen, zielen viele Unternehmen auf eine Erhöhung des Automatisierungsgrades durch den Einsatz von Technologien zur Prozessunterstützung. Die mitunter komplexen Abläufe sowie die wachsende Vernetzung dieser Prozesse unter Geschäftspartnern führen zu immer komplexer werdenden Prozesslandschaften. Oftmals existieren keine zentralen Steuersysteme mehr, sondern die Abwicklung der Geschäftsvorfälle läuft verteilt auf heterogene „Subsysteme“ ab. D.h. das Wissen und die Informationen zu den Prozessen sowie operationale Prozessdaten liegen fragmentiert über mehrere Systeme vor. In einer solchen Welt den Überblick über den aktuellen Ausführungszustand eines oder mehrerer laufender Prozesse zu behalten, ist nicht einfach. Hier mangelt es an Werkzeugen zur integrierten Darstellung des Gesamtprozesses, aus der neben dem aktuellen Zustand auch weitere Prozessdaten (z.B. Informationen zum aktuell bearbeiteten Fall) entnommen werden können.

An der Ausführung eines Prozesses sind oftmals viele verschiedene Personengruppen (z.B. Manager, IT-Mitarbeiter, Prozessbeteiligte, Prozessverantwortliche) beteiligt. Jede dieser Gruppen erfüllt eine spezielle Rolle bei der Ausführung des Prozesses. Folglich existieren unterschiedliche Anforderungen an die Visualisierung der bearbeiteten Prozesse. Heutige Prozesswerkzeuge sind nicht in der Lage, auf die individuellen Bedürfnisse der Betrachter zugeschnittene Darstellungen automatisch zu erzeugen. Vielmehr muss jede vom Standardmodell abweichende Visualisierung von Hand definiert (d.h. gezeichnet) werden.

Im Rahmen der Dokumentation komplexer Prozesse entstehen häufig sehr große Modelle, die als mehrere Quadratmeter große Plots an die Prozessbeteiligten verteilt werden. Für die Betrachter derartiger „Wandtapeten“ ist es mithin unmöglich, die eigenen Tätigkeiten im Prozess zu identifizieren. Abhilfe schafft in diesen Fällen eine aus dem Gesamtmodell abgeleitete Darstellung, die jeweils nur die Aktivitäten einer bestimmten Zielgruppe enthält. Aktuelle Werkzeuge unterstützen diesen Anwendungsfall leider nicht.

In dieser Arbeit wurde mit Proviado ein umfassendes Konzept für die konfigurierbare Visualisierung komplexer Prozessmodelle entwickelt. Motiviert wurde diese Arbeit durch die Resultate von Fallstudien im Automobilbereich, jedoch sind die Ergebnisse auf beliebige Domänen übertragbar. Die wichtigsten Beiträge der Arbeit lassen sich wie folgt zusammenfassen:

Konfigurierbare Prozess-Views

Der in dieser Arbeit vorgestellte Proviado-View-Mechanismus ermöglicht es, die Komplexität von Prozessmodellen durch strukturelle Modifikationen zu reduzieren. Zum einen können Modellelemente entfernt und somit die Größe des Prozesses verringert werden (Reduktion). Zum anderen kann durch Zusammenfassen von Modellelementen zu neuen, abstrakten Elementen eine abstrahierte Sicht auf einen Prozess generiert werden (Aggregation).

Im Zentrum des Proviado-View-Mechanismus steht dabei ein konfigurierbares, mehrschichtiges Konzept aus View-Bildungsoperationen. Grundlage bildet eine Reihe von elementaren View-Bildungsoperationen, die alle Prozessaspekte (Kontrollfluss, Datenfluss, Bearbeiterzuordnungen) abdecken. Jede dieser Elementaroperationen kann jeweils auf eine bestimmte Struktur von Prozesselementen (z.B. Sequenzen von Aktivitäten oder abgeschlossene Kontrollflussblöcke, d.h. „SESE“) angewendet werden und diese Elemente reduzieren bzw. aggregieren. Des Weiteren wurde für jede dieser Elementaroperationen analysiert, welche Eigenschaften für das resultierende Modell noch zugesichert werden können. Können nun bei Generierung einer View mehr als eine Elementaroperation angewendet werden, so lässt sich mittels der identifizierten Eigenschaften parametrisieren, welche Operation letztlich ausgeführt werden soll.

Proviado bietet darüber hinaus höherwertige View-Bildungsoperationen an, die die Operationen der darunter liegenden Schichten kombinieren und so auch komplexe View-Bildungen einfach beschreibbar machen. Eine View-Bildung ist zudem dynamisch, d.h. abhängig von Laufzeitdaten der Prozesse möglich. Mittels Graphmanipulationsoperationen kann darüber hinaus in einer Vor- bzw. Nachbehandlung das Prozessmodell weiter angepasst werden, um ein möglichst optimales Ergebnis als Grundlage für die anschließende Visualisierung des Prozesses zu erhalten.

Das View-Konzept wird durch eine vorgeschlagene View-Definitionssprache und Betrachtungen zu optimierten Implementierungsstrategien abgerundet.

Flexible Adaption der Prozessnotation

Um eine an die Bedürfnisse des Betrachters angepasste Visualisierung des Prozesses zu ermöglichen, muss das Prozessmodell nicht nur strukturell sondern auch graphisch anpassbar sein. Hierzu wurde in dieser Arbeit der Proviado-Template-Mechanismus vorgestellt. Er erlaubt es zum einen, die Prozessnotation, d.h. die zur Darstellung verwendete Menge von Symbolen, einfach zu definieren. Zum anderen kann flexibel festgelegt werden, welche Symbole in Abhängigkeit von den Prozesselementen oder -daten für die Visualisierung verwendet werden sollen.

Die Templates für die Prozesssymbole werden generisch unter Verwendung von SVG und diversen anderen XML-Standards beschrieben. Dabei können beliebig komplexe Graphiksymbole definiert werden. Komplexe Symbole, die abhängig von Laufzeitdaten ihre Form ändern (z.B. Tacho- oder Balkendiagramme), können durch die Einbindung von ausführbaren Programmbausteinen ebenso realisiert werden wie komplex ineinander verschachtelte Symbole.

Die Verwendung der Templates, d.h. die Zuordnung von Symbolen zu Prozesselementen, wird mittels eines eigen entwickelten, flexiblen Beschreibungsverfahrens definiert. Großen Wert wurde darauf gelegt, dass die Auswertung dieser Verwendungsdefinition immer eindeutige Symbolzuweisungen ergibt. Die Verwendung der Templates kann sowohl statisch, z.B. abhängig vom Typ des Prozesselements erfolgen, wie auch dynamisch, d.h. abhängig von beliebigen Laufzeit- oder Zustandsdaten des Prozesses.

Das endgültige Aussehen der Prozessvisualisierung (z.B. Farben, Schriftarten, Schriftgrößen) lässt sich weiter durch die Angabe von Stylesheets konfigurieren. Dadurch kann beispielsweise die Farbgebung einer Prozessdarstellung einfach an die Vorgaben des Bereichs- oder Unternehmensstandards angepasst werden. Schließlich fasst das Proviado-Visualisierungsmodell alle für die Generierung einer Prozessvisualisierung erforderlichen Parameter in einer zentralen Konfiguration zusammen. Das Visualisierungsmodell, das die Darstellung eines Prozesses in genau einer Art und Weise (Detaillierung, d.h. View, und graphische Aufbereitung) beschreibt, kann mittels Parametern konfigurierbar gestaltet werden, so dass es für die Darstellung anderer Prozessschemata und Prozessinstanzen wieder verwendet werden kann.

Weitere Aspekte des Proviado-Visualisierungskonzepts

Neben den beiden beschriebenen Kernkonzepten wurden im Rahmen von Proviado weitere Themen bearbeitet, die für ein umfassendes Konzept für die Prozessvisualisierung wichtig sind.

Prozessdatenintegration Große, komplexe Prozesse erstrecken sich meist über mehrere Systeme, die zudem häufig auf unterschiedlichen Prozessmetamodellen basieren. Daher ist es für eine integrierte Visualisierung erforderlich, diese Heterogenität durch Bereitstellung eines generischen Prozessmetamodells zu beseitigen. Bei der anschließenden Integration bzw. Transformation der Prozessmodelle müssen die aus der Heterogenität resultierenden Probleme (z.B. Abbildung unterschiedlicher Metamodellelemente) behoben werden.

Laufzeitdatenanbindung Um in der Prozessvisualisierung neben den reinen Modelldaten auch Applikationsdaten der prozessausführenden Systeme und sonstiger Datenquellen anzeigen zu können, muss ein Weg geschaffen werden, auf diese Laufzeitdaten zuzugreifen. Dazu wurde ein Konzept für die generische Anbindung verschiedener Systemarten entwickelt.

Prozessgraph-Layout Durch die beschriebenen Mechanismen zur strukturellen und graphischen Adaption von Prozessmodellen unterliegen die Positionsdaten der Prozesselemente ständigen Änderungen. Diese manuell zu definieren, ist in solch einer dynamischen Umgebung zu aufwendig. In dieser Arbeit wurden zu dieser Thematik zwei Algorithmen entwickelt. Der erste eignet sich, um initial ein Layout für einen Prozessgraphen unter Berücksichtigung diverser Constraints (z.B. spezielle Abstandsvorgaben zwischen Prozesselementen) zu berechnen. Der zweite Algorithmus erzeugt aus einem bestehenden Layout durch geschicktes Verschieben der Prozesselemente ein neues. Der Vorteil liegt dabei in der Konstanz des Layouts, d.h. es treten zwischen zwei Layout-Berechnungen keine gravierenden Änderungen auf.

Die zentralen Konzepte von Proviado wurden prototypisch umgesetzt und vermitteln einen Eindruck der Mächtigkeit der Gesamtlösung.

Ausblick

Wie so oft, wirft die Bearbeitung eines Problems neue bzw. weiterführende Fragestellungen auf, die Gegenstand von Folgearbeiten sein können. Aufbauend auf der vorliegenden Arbeit bieten sich unter anderem die folgenden Themen für weitergehende Untersuchungen an:

- In Abschnitt 5.6.2 wurde kurz erläutert, wie eine Prozess-View bei Änderung des Basisprozessmodells aktualisiert werden kann. Umgekehrt kann man untersuchen, in welchen Fällen Änderungen sowohl an den in der View dargestellten Prozessdaten als auch am Prozessgraphen der View selbst eindeutig auf das Basisprozessmodell zurückpropagiert werden können. Eine ähnliche Problematik wie die Änderung von Prozessdaten ist seit langem in relationalen Datenbanken bekannt [CP84].

- Prozess-Views können neben der Vereinfachung von Prozessmodellen zur Visualisierung auch für andere Zwecke eingesetzt werden. Ein Beispiel ist die Verwendung von Views, um bei Adhoc-Modifikationen von Prozessinstanzen (vgl. ADEPT [RD98]) dem Benutzer ein auf die relevanten Teile reduziertes Prozessmodell zur Verfügung zu stellen, anstatt ihn mit der Komplexität des Gesamtmodells zu verwirren. Dies setzt zum einen voraus, dass die zuvor beschriebene View-Update-Problematik für strukturelle Änderungen an der View gelöst ist. Zum anderen müssen in diesem Anwendungsfall die Parameter der View-Bildung strikt in dem Sinne gesetzt werden, als dass die View-Eigenschaften wie Zustandskonsistenz und Abhängigkeitserhaltung erfüllt werden (vgl. Kapitel 4).

- Für den Einsatz des Proviado-Visualisierungsrahmenwerks muss im Einzelfall eine Anbindung an Quellsysteme realisiert werden. Dabei muss zum einen die Abbildung des Prozessmetamodells des Quellsystems auf das Proviado-Metamodell realisiert werden. Zum anderen muss für die Visualisierung von Applikationsdaten die Schnittstelle zum Zugriff auf Laufzeitdaten der unterschiedlichen Systeme (z.B. Workflow-Management-System, Anwendungssystem, Datenbanksystem) umgesetzt werden. Die in Abschnitt 7.2 beschriebenen Konzepte sind dabei generisch einsetzbar, müssen jedoch an die konkreten Systeme und ihre Metamodelle angepasst werden.

- In Abschnitt 7.3 wurden zwei Algorithmen für das Layout von Prozessgraphen vorgestellt. In dieser Thematik gibt es jedoch grundsätzlichen Forschungsbedarf, da die Problematik an sich hochkomplex ist. Die vorgeschlagenen Algorithmen bieten einen Ausgangspunkt, von dem aus weitere Lösungsansätze getestet werden können. Leider ist das Problem von derart hoher Komplexität, dass die eine ideale Lösung nicht erwartet werden kann.

- Prozessmodelle enthalten oftmals sensible Informationen, die nicht jedem Benutzer einer Visualisierungskomponente frei zur Verfügung stehen sollten. In [BRBB07] haben wir einen ersten Versuch unternommen, ein entsprechendes Zugriffskonzept zu entwickeln. In diesem Bereich ist jedoch weiterer Forschungsbedarf vorhanden.

- Neben der Verwendung von Proviado für die Visualisierung von Prozessmodellen sollten sich die entwickelten Konzepte, insbesondere der Template-Mechanismus aus Kapitel 6, auch für die Visualisierung anderer strukturierter Daten verwenden lassen. Ein Beispiel wären die komplexen Produktstrukturen und die damit verbundenen Object-Live-Cycles aus COREPRO [MHHR06, MRH06, MRH07, MRHP07, MRH08], aus denen komplexe Prozessstrukturen abgeleitet und gesteuert werden.

Es bleibt zu hoffen, dass die Konzepte dieser Arbeit allmählich ihren Weg in die kommerziellen Prozessmanagementwerkzeuge finden und dadurch dem Einen oder Anderen mit komplexen Prozessmodellen konfrontierten Anwender die tägliche Arbeit erleichtern.

Literaturverzeichnis

[Aal98] W. M. P. v. d. Aalst: *The Application of Petri Nets to Workflow Management.* Journal on Circuits, Systems and Computers, 8(1):21–66, 1998.

[Aal02] W. M. P. v. d. Aalst: *Inheritance of Interorganizational Workflows: How to agree to disagree without loosing control?* Information Technology and Management Journal, 2(3):195–231, 2002.

[AC95] D. Avrilionis und P. Y. Cunin: *Using views to maintain Petri-net-based process models.* In: *Proceedings of the International Conference on Software Maintenance (ICSM'95)*, Seiten 318–326, Opio (Nice), France, Oktober 1995. IEEE Computer Society.

[AC96] D. Avrilionis und P. Y. Cunin: *View-based Mechanisms for Structured and Distributed Enactment.* In: *Proceedings of 2nd International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints'96) on SIGSOFT'96 Workshops*, Seiten 259–262, San Francisco, California, USA, Oktober 1996.

[ACF96] D. Avrilionis, P. Y. Cunin und C. Fernstrom: *OPSIS: A View Mechanism for Software Processes which Supports their Evolution and Reuse.* In: *Proceedings 18th International Conference on Software Engineering (ICSE'96)*, Seiten 38–47. IEEE Computer Society, 1996.

[Act07] Actano: *RPlan.* Website: http://www.actano.de/, 2007. zuletzt besucht: 9.12.2007.

[Ada04] M. Adam: *Ansätze für Prozessvisualisierung im Workflow-Management.* Diplomarbeit, Fachhochschule Stuttgart - Hochschule der Medien, 2004.

[AH05] W. M. P. v. d. Aalst und A. H. M. t. Hofstede: *YAWL: Yet Another Workflow Language.* Information Systems, 30(4):245–275, 2005.

[Ark01] A. Arkin: *Business Process Modeling Language (BPML).* Proposed Draft, Version 0.4, BPMI.org, August 2001.

[AW01] W. M. P. v. d. Aalst und M. Weske: *The P2P Approach to Interorganizational Workflows.* In: *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, Band 2068 der Reihe *LNCS*, Seiten 140–156, Interlaken, Schweiz, 2001.

[AWM04] W. M. P. v. d. Aalst, T. Weijters und L. Maruster: *Workflow Mining: Discovering Process Models from Event Logs*. IEEE Transactions on Knowledge and Data Engineering (TKDE), 16(9):1128–1142, September 2004.

[Bat08] *Batik SVG Toolkit 1.7*, Januar 2008. http://xmlgraphics.apache.org/batik/.

[Bau04a] T. Bauer: *Kooperation von Projekt- und Workflow-Management-Systemen*. Informatik - Forschung und Entwicklung, 19:74–86, November 2004.

[Bau04b] T. Bauer: *Visualisierung laufender Prozesse*. In: *Proceedings Informatik 2004, Workshop Geschäftsprozessorientierte Architekturen*, LNI, Seiten 543–548, Ulm, September 2004.

[Bau05] T. Bauer: *Integration von prozessorientierten Anwendungen*. In: *Proceedings Workshop on Enterprise Application Integration*, Seiten 66–73, Marburg, Juni 2005.

[BB07a] T. Bauer und R. Bobrik: *Applikationsübergreifendes Monitoring von Geschäftsprozessen*. EMISA Forum, 27(1):14–19, Januar 2007.

[BB07b] R. Bobrik und T. Bauer: *Towards Configurable Process Visualizations with Proviado*. In: *Proceedings WETICE IEEE-Workshop on Agile Cooperative Process-Aware Information Systems (ProGility'07)*, Seiten 367–369, Paris, France, Juni 2007.

[BBC+07] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie und J. Siméon: *XML Path Language (XPath) 2.0*. W3C Recommendation, World Wide Web Consortium (W3C), Januar 2007.

[BBR06] R. Bobrik, T. Bauer und M. Reichert: *Proviado - Personalized and Configurable Visualizations of Business Processes*. In: *Proceedings of 7th International Conference on Electronic Commerce and Web Technologies (EC-Web'06)*, Band 4082 der Reihe *LNCS*, Seiten 61–71, Krakow, Poland, September 2006.

[BCBD07] O. Biton, S. Cohen-Boulakia und S. B. Davidson: *Zoom*UserViews: Querying Relevant Provenance in Workflow Systems*. In: *Proceedings 33rd International Conference on Very Large Data Bases (VLDB'07)*, Vienna, Austria, September 2007.

[BDK07] J. Becker, P. Delfmann und R. Knackstedt: *Konfigurierbare Handelsinformationssysteme. Referenzmodelle als Beitrag zur Sicherung des Softwarestandorts Deutschland?* Wirtschaftsinformatik, 49:17–27, 2007.

[BEL+07] S. Buckl, A. M. Ernst, J. Lankes, C. M. Schweda und A. Wittenburg: *Generating Visualizations of Enterprise Architectures using Model Transformations*. In: *Proceedings Enterprise Modelling and Information Systems Architectures (EMISA'07)*, 2007.

[BETT94] G. d. Battista, P. Eades, R. Tamassia und I. G. Tollis: *Algorithms for Drawing Graphs: An Annotated Bibliography*. Computational Geometry: Theory and Applications, 4(5):235–282, 1994.

[Beu03] T. Beuter: *Workflow-Management für Produktentwicklungsprozesse.* Der Andere Verlag, Februar 2003.

[BJM97] F. J. Brandenburg, M. Jünger und P. Mutzel: *Algorithmen zum automatischen Zeichnen von Graphen.* Informatik Spektrum, 20(4):199–207, 1997.

[BK01] U. Brandes und B. Köpf: *Fast and Simple Horizontal Coordinate Assignment.* In: *Proceedings 9th International Symposium on Graph Drawing (GD'01)*, Band 2265 der Reihe *LNCS*, Seiten 31–44, Vienna, Austria, 2001.

[BK02] J. Becker und R. Knackstedt: *Referenzmodellierung 2002. Methoden - Modelle - Erfahrungen.* Arbeitsbericht 90, Institut für Wirtschaftsinformatik, Westfälische Wilhelms-Universität Münster, August 2002.

[BK06] C. Bauer und G. King: *Java Persistence with Hibernate.* Manning Publications, Überarbeitete Auflage, November 2006.

[BKC95] N. S. Barghouti, E. Koutsofios und E. Cohen: *Improvise: Interactive Multimedia Process Visualization Environment.* In: *Proceedings 5th European Software Engineering Conference (ESEC'95)*, Band 989 der Reihe *LNCS*, Seiten 28–43, Sitges, Spain, September 1995.

[BLM04] M. Bin Lai und M. z. Muehlen: *Information Requirements of Process Stakeholders: A Framework to Evaluate Process Monitoring and Controlling Applications.* In: *Proceedings of the Pre-ICIS workshop on Process Management*, Washington, D.C., Dezember 2004.

[BM01] O. Bastert und C. Matuszewski: *Layered Drawings of Digraphs.* In: M. Kaufmann und D. Wagner (Herausgeber): *Drawing Graphs: Methods and Models*, Nummer 2025 in *LNCS*, Kapitel 5, Seiten 87–120. Springer, 2001.

[BM04] R. A. Burkhard und M. Meier: *Tube Map: Evaluation of a Visual Metaphor for Interfunctional Communication of Complex Projects.* In: *Proceedings of the 4th International Conference on Knowledge Management (I-KNOW'04)*, Graz, Austria, Juni 2004.

[BM05] R. A. Burkhard und M. Meier: *Tube Map Visualization: Evaluation of a Novel Knowledge Visualization Application for the Transfer of Knowledge in Long-Term Projects.* Journal of Universal Computer Science, 11(4):473–494, 2005.

[BMG05] D. Benslimane, Z. Maamar und C. Ghedira: *A View-based Approach for Tracking Composite Web Services.* In: *Proceedings of the Third European Conference on Web Services (ECOWS '05)*, Seite 170, Washington, DC, USA, 2005.

[BMG06] D. Benslimane, Z. Maamar und C. Ghedira: *How to Track Composite Web Services? A Solution Based on the Concept of View.* Journal of Electronic Commerce Research (JECR), Special Issue on Trust and Process Management in E-Commerce, 7(3):123–137, 2006.

[BMR+05] R. Burkhard, M. Meier, P. Rodgers, M. Smis und J. Stott: *Knowledge Visualization: A Comparative Study between Project Tube Maps and Gantt Charts.* In: *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW'05)*, Seiten 388–395, Juni 2005.

[Bob04a] R. Bobrik: *Anforderungen an eine Prozessvisualisierung.* Interne Präsentation DaimlerChrysler AG, November 2004.

[Bob04b] R. Bobrik: *Sprachen und Notationen für die Prozessvisualisierung.* Interner Bericht DaimlerChrysler AG, 2004.

[Bob05] R. Bobrik: *Metamodell für die Prozessvisualisierung.* Interner Bericht DaimlerChrysler AG, 2005.

[BOC07] BOC: *Das Geschäftsprozessmanagement-Werkzeug ADONIS - Einsatzszenarien und Vorteile.* Whitepaper http://www.boc-eu.com/documents/products/adonis_whitepaper_de.pdf, 2007. zuletzt besucht: 9.12.2007.

[BPT05] BPTrends: *ADONIS.* The 2005 EA, Process Modeling BPTrends & Simulation Tools Report, 2005.

[BRB05] R. Bobrik, M. Reichert und T. Bauer: *Requirements for the Visualization of System-Spanning Business Processes.* In: *Proceedings 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, Seiten 948–954, Copenhagen, Denmark, August 2005. IEEE Computer Society.

[BRB07] R. Bobrik, M. Reichert und T. Bauer: *Parameterizable Views for Process Visualization.* Technical Report TR-CTIT-07-37, Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands, April 2007.

[BRBB07] S. Bassil, M. Reichert, R. Bobrik und T. Bauer: *Access Control for Monitoring System-Spanning Business Processes.* Technical Report TR-CTIT-07-20, Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands, März 2007.

[Bro03] J. v. Brocke: *Referenzmodellierung - Gestaltung und Verteilung von Konstruktionsprozessen.* Dissertation, Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik, 2003.

[Buc05] S. Buchwald: *Konzeption und Realisierung einer Infrastruktur zur prozessorientierten Integration von Applikationen in Workflows.* Diplomarbeit, Fachhochschule Ulm, Juni 2005.

[Buc07] S. Buchwald: *Modellierung von Abhängigkeiten zwischen prozessorientierten Applikationen.* Masterarbeit, Hochschule Ulm, 2007.

[Bur05] R. A. Burkhard: *Impulse: Using Knowledge Visualization in Business Process Oriented Knowledge Infrastructures.* Journal of Universal Knowledge Management, 0(2):170–188, 2005.

[Bus06] Business Process Management Initiative (BPMI): *Business Process Modeling Notation (BPMN) 1.0: OMG Final Adopted Specification*, Februar, 6 2006. Version 1.0.

[Cas05] F. Casati: *Industry Trends in Business Process Management: Getting Ready for Prime Time.* In: *Proceedings 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, Seiten 903–907, Copenhagen, Denmark, August 2005. IEEE Computer Society.

[CBB+00] R. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda und F. Velez: *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann, 2000.

[CBBCD08] S. Cohen-Boulakia, O. Biton, S. Cohen und S. Davidson: *Addressing the Provenance Challenge using ZOOM.* Concurrency and Computation: Practice and Experience, 20(5):497–506, 2008.

[CBD06] S. Cohen, S. C. Boulakia und S. B. Davidson: *Towards a Model of Provenance and User Views in Scientific Workflows.* In: *Proceedings Third International Workshop Data Integration in the Life Sciences (DILS'06)*, Band 4075 der Reihe *LNCS*, Seiten 264–279, Hinxton, UK, Juli 2006.

[CCK02a] D. K. W. Chiu, S. C. Cheung und E. Kafeza: *Three-tier View-based Support for Mobile Workflow.* In: *Proceedings First International Conference on Mobile Business*, Athens, Greece, Juli 2002.

[CCK+02b] D. K. W. Chiu, S. C. Cheung, K. Karlapalem, Q. Li und S. Till: *Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment.* In: *Proceedings of Workshop on Web Services, e-Business, and the Semantic Web in conjunction with CAiSE'02*, Band 2512 der Reihe *LNCS*, Seiten 41–56, Ontario, Canada, Mai 2002.

[CCKL03] D. K. W. Chiu, S. C. Cheung, E. Kafeza und H. f. Leung: *A Three-Tier View-Based Methodology for M-Services Adaptation.* IEEE Transactions on Systems, Man, and Cybernetics, Part A, 33(6):725–741, 2003.

[CCL03] D. K. W. Chiu, S. C. Cheung und H. f. Leung: *A Three-Tier View-Based Methodology for Adapting Human-Agent Collaboration Systems.* In: *Proceedings 15th International Conference on Advanced Information System Engineering (CAiSE'03)*, Band 2681 der Reihe *LNCS*, Seiten 226–241, Velden, Austria, Juni 2003.

[CCS04] F. Casati, M. Castellanos und M. C. Shan: *Enterprise Cockpit for Business Operation Management.* In: *Proceedings of the International Conference on Conceptual Modeling (ER'04)*, Band 3288 der Reihe *LNCS*, Seiten 825–827, 2004.

[CCSD05] M. Castellanos, F. Casati, M. C. Shan und U. Dayal: *iBOM: A Platform for Intelligent Business Operation Management.* In: *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, Seiten 1084–1095, 2005.

[CCT+04] D. K. W. Chiu, S. C. Cheung, S. Till, K. Karlapalem, Q. Li und E. Kafeza: *Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment.* Information Technology and Management, 5(3-4):221–250, Juli 2004.

[CDT06] I. Chebbi, S. Dustdar und S. Tataa: *The View-based Approach to Dynamic Inter-Organizational Workflow Cooperation.* Data Knowledge Engineering, 56(2):139–173, Februar 2006.

[CHCK06] D. K. Chiu, D. Hong, S. C. Cheung und E. Kafeza: *Adapting Ubiquitous Enterprise Services with Context and Views.* In: *Proceedings 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, Seiten 391–394, 2006.

[CHCK07] D. K. W. Chiu, D. Hong, S. C. Cheung und E. Kafeza: *Towards Ubiquitous Government Services through Adaptations with Context and Views in a Three-Tier Architecture.* In: *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07).* IEEE Computer Society, 2007.

[CKL01] D. K. W. Chiu, K. Karlapalem und Q. Li: *Views for Inter-organization Workflow in an E-commerce Environment.* In: *Proceedings of the IFIP TC2/WG2.6 Ninth Working Conference on Database Semantics*, Seiten 137–151. Kluwer, 2001.

[CKLK02] D. K. W. Chiu, K. Karlapalem, Q. Li und E. Kafeza: *Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment.* Distributed and Parallel Databases, 12(2-3):193–216, 2002.

[CP84] S. S. Cosmadakis und C. H. Papadimitriou: *Updates of Relational Views.* Journal of the ACM (JACM), 31(4):742–760, 1984.

[CR03] A. Cooper und R. Reimann: *About Face 2.0. The Essentials of Interaction Design.* John Wiley & Sons, 2003.

[CSH+05] D. K. W. Chiu, Z. Shan, P. C. K. Hung, S. C. Cheung und Q. Li: *Process Views with Flows for Heterogeneous and Complex System Integration: A Service Requirement Approach.* http://teaching.ust.hk/~csit600c/flowview.doc, 2005. zuletzt besucht: 20.01.2008.

[CSHL04] D. K. W. Chiu, Z. Shan, P. C. K. Hung und Q. Li: *Designing Workflow Views with Flows for Large-Scale Business-to-Business Information Systems.* In: M. C. Shan, U. Dayal und M. Hsu (Herausgeber): *Proceedings Technologies for E-Services: 5th International Workshop(TES'04)*, Band 3324 der Reihe *LNCS*, Seiten 107–121, Toronto, Canada, August 2004.

[Dad96] P. Dadam: *Verteilte Datenbanken und Client/Server-Systeme.* Springer, 1996.

[DAH05] M. Dumas, W. M. v. d. Aalst und A. H. t. Hofstede (Herausgeber): *Process Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley-Interscience, 2005.

[DAV05] B. F. v. Dongen, W. M. P. v. d. Aalst und H. M. W. Verbeek: *Verification of EPCs: Using Reduction Rules and Petri Nets.* In: *Proceedings 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, Band 3520 der Reihe *LNCS*, Seiten 372–386, Porto, Portugal, Juni 2005.

[DB07] R. Davis und E. Brabänder: *ARIS Design Platform Getting Started with BPM.* Springer, 2007.

[DDK+02] G. Diguglielmo, E. Durocher, P. Kaplan, G. Sander und A. Vasiliu: *Graph Layout for Workflow Applications with ILOG JViews.* In: *Proceedings Graph Drawing (GD'02)*, Band 2528 der Reihe *LNCS*, Seiten 362–363, 2002.

[Die06] R. Diestel: *Graphentheorie.* Springer, September 2006.

[DIL+04] H. t. Doest, M. E. Iacob, M. Lankhorst, D. v. Leeuwen und R. Slagter: *Viewpoints Functionality and Examples.* Technischer Bericht TI/RS/2003/091, Telematica Instituut, 2004.

[DNB+06] E. Dahlström, C. Northway, R. Berjon, C. Lilley, D. Schepers, S. Hayman, J. Ferraiolo, A. Neumann, V. Hardy, N. Ramani, O. Andersson, A. Shellshear, A. Quint, D. Jackson und A. Emmons: *Scalable Vector Graphics (SVG) Tiny 1.2 Specification.* W3C Candidate Recommendation, World Wide Web Consortium (W3C), August 2006.

[DOM08] *Dom4J.* Website: http://www.dom4j.org, 2008. zuletzt besucht: 17.02.2008.

[DPS02] J. Díaz, J. Petit und M. J. Serna: *A Survey of Graph Layout Problems.* ACM Computing Surveys, 34(3):313–356, 2002.

[DV02] P. Domokos und D. Varró: *An Open Visualization Framework for Metamodel-Based Modeling Languages.* In: *Proceedings of International Workshop on Graph-Based Tools (GraBaTs'02)*, Band 72 (2) der Reihe *ENTCS*, Seiten 78–87, Barcelona, Spain, Oktober 2002.

[ECM99] ECMA International: *ECMAScript Language Specification*, Dezember 1999. Standard ECMA-262.

[EG07] R. Eshuis und P. Grefen: *Constructing Customized Process Views.* Working Paper 197, Beta Research School for Operations Management and Logistics, The Netherlands, Februar 2007.

[EG08] R. Eshuis und P. Grefen: *Constructing Customized Process Views.* Data & Knowledge Engineering, 64:419–438, 2008.

[EGP00] J. Eder, W. Gruber und E. Panagos: *Temporal Modeling of Workflows with Conditional Execution Paths.* In: *Proceedings of the 11th International Conference on Database and Expert Systems Applications, (DEXA'00)*, Band 1873 der Reihe *LNCS*, Seiten 243–253, London, UK, 2000.

[EH07] M. A. H. Elias und S. Hristova: *Proviado Prozessvisualisierung - Implementierung von Prozess Views.* Praktikumsdokumentation, 2007.

[Ehr06] K. Ehrlenspiel: *Integrierte Produktentwicklung.* Carl Hanser Verlag, 3 Auflage, 2006.

[EN02] R. Elmasri und S. B. Navathe: *Fundamentals of Database Systems, Fourth Edition.* Addison Wesley, 2002.

[EP02] J. Eder und H. Pichler: *Duration Histograms for Workflow Systems.* In: *Proceedings IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, Band 231 der Reihe *IFIP Conference Proceedings*, Seiten 239–253, Kanazawa, Japan, September 2002. Kluwer.

[EPGN03] J. Eder, H. Pichler, W. Gruber und M. Ninaus: *Personal Schedules for Workflow Systems.* In: *Proceedings of the International Conference on Business Process Management (BPM'03)*, Band 2678 der Reihe *LNCS*, Seiten 216–231, Eindhoven, The Netherlands, Juni 2003.

[FB95] G. W. Furnas und B. B. Bederson: *Space-Scale Diagrams: Understanding Multiscale Interfaces.* In: *Proceedings of ACM SIG Computer Human Interaction (CHI'95)*, 1995.

[FFJ03] J. Ferraiolo, J. Fujisawa und D. Jackson: *Scalable Vector Graphics (SVG) 1.1 Specification.* W3C Recommendation, World Wide Web Consortium (W3C), Januar 2003.

[FH06] H. G. Fill und P. Höfferer: *Visual Enhancements of Enterprise Models.* In: *Proceedings Minitrack on Visualization Methods for KM Applications (MKWI'06)*, Seiten 541–550, Passau, Germany, 2006.

[Fil06] H. G. Fill: *Semantic Visualization of Heterogenous Knowledge Sources.* In: *Modellierung für Wissensmanagement - Workshop im Rahmen der Modellierung'06*, 2006.

[FK92] D. Ferraiolo und R. Kuhn: *Role-Based Access Control.* In: *Proceedings 15th NIST-NSA National Computer Security Conference*, Seiten 554–563, 1992.

[FK06] H. G. Fill und D. Karagiannis: *Semantic Visualization for Business Process Models.* In: *Proceedings of International Workshop on Visual Languages and Computing*, Seiten 168–173, Grand Canyon, USA, 2006.

[Fur86] G. W. Furnas: *Generalized Fisheye Views.* In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'86)*, Seiten 16–23. ACM Press, 1986.

[Gar05] J. J. Garrett: *Ajax: A New Approach to Web Applications.* http://www.adaptivepath.com/ideas/essays/archives/000385.php, Februar 2005. zuletzt besucht: 29.10.2007.

[GBT06] GBTec Software + Consulting AG: *BIC Publish.* Whitepaper, 2006.

[GBT07] GBTec Software + Consulting AG: *BIC Monitor.* Website: http://www.gbtec.de/index.php/categories/show/55, 2007. zuletzt besucht: 9.12.2007.

[GCC+04] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal und M. C. Shan: *Business Process Intelligence.* Computers in Industry, 53(3):321–343, 2004.

[GCG04] D. Grigori, F. Charoy und C. Godart: *Coo-Flow: A Process Technology To Support Cooperative Processes.* International Journal of Software Engineering and Knowledge Engineering, 14(1):61–78, 2004.

[GCG06] D. Grigori, F. Charoy und C. Godart: *Enhancing the Flexibility of Workflow Execution by Activity Anticipation.* International Journal of Business Process Integration and Management (IJBPIM), 1(3):143–155, 2006.

[GHJV95] E. Gamma, R. Helm, R. Johnson und J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[GHK06] J. Gausemeier, A. Hahn und H. D. Kespohl: *Vernetzte Produktentwicklung. Der erfolgreiche Weg zum Global Engineering Networking.* Hanser Fachbuchverlag, 2006.

[GLK+06a] U. Greiner, S. Lippe, T. Kahl, J. Ziemann und F. W. Jäkel: *Designing and Implementing Cross-organizational Business Processes - Description and Appliction of a Modelling Framework.* In: *Proceedings Interoperability for Enterprise Software and Applications Conference (I-ESA)*, Bordeaux, France, März 2006.

[GLK+06b] U. Greiner, S. Lippe, T. Kahl, J. Ziemann und F. W. Jäkel: *A Multi-level Modeling Framework for Designing and Implementing Cross-Organizational Business Processes.* In: *Proceedings of Workshop on Technologies for Collaborative Business Process Management (TCoB'06)*, Seiten 13–23, 2006.

[Grö02] E. Gröller: *Insight into Data through Visualization.* In: *Proceedings 9th International Symposium on Graph Drawing (GD'01)*, Band 2265 der Reihe *LNCS*, Seiten 352–366, Vienna, Austria, 2002.

[Gär04] T. Gärtner: *Wissenserwerb mit Prozessmodellen als Grundlage für eine anwenderorientierte Visualisierung von Prozessen bei DaimlerChrylser.* Magisterarbeit, Philosophisch-Sozialwissenschaftliche Fakultät der Universität Augsburg, Oktober 2004.

[Ham03] M. R. Hammori: *Interactive Workflow Mining.* Diplomarbeit, University of Ulm, 2003.

[Har02] E. R. Harold: *Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX.* Addison-Wesley Professional, 2002.

[Hav05] M. Havey: *Essential Business Process Modeling.* O'Reilly Media, 2005.

[HBR07] A. Hallerbach, T. Bauer und M. Reichert: *Managing Process Variants in the Process Life Cycle.* Technischer Bericht TR-CTIT-07-87, Centre for Telematics and Information Technology, University of Twente, 2007.

[HBR08a] A. Hallerbach, T. Bauer und M. Reichert: *Anforderungen an die Modellierung und Darstellung von Prozessvarianten.* Datenbank Spektrum, 8(24):48–58, 2008.

[HBR08b] A. Hallerbach, T. Bauer und M. Reichert: *Managing Process Variants in the Process Life Cycle.* In: *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS'08)*, Barcelona, Spain, Juni 2008. accepted for publication.

[HBR08c] A. Hallerbach, T. Bauer und M. Reichert: *Modellierung und Darstellung von Prozessvarianten in Provop.* In: *Proceedings of Modellierung 2008*, 2008.

[Heß96] H. Hess: *Monitoring von Geschäftsprozessen.* ZWF, 91:76–79, 1996.

[Her99] T. Herrmann: *Flexible Präsentation von Prozeßmodellen.* In: U. Arend, E. Eberleh und K. Pitschke (Herausgeber): *Proceedings Software-Ergonomie*, Band 53 der Reihe *Berichte des German Chapter of the ACM*, Seiten 123–136, Walldorf, Germany, März 1999. Teubner.

[Her03] J. Herbst: *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen.* Dissertation, Universität Ulm, 2003.

[Hic07] I. Hickson: *XML Binding Language (XBL) 2.0.* W3C Working Draft, World Wide Web Consortium (W3C), Januar 2007.

[HK01] V. J. Hardy und T. Kormann: *Leveraging SVG on the Java platform with Batik.* In: *Proceedings XML Europe 2001*, Berlin, Germany, Mai 2001.

[HKD04] M. C. Hao, D. A. Keim und U. Dayal: *VisBiz: A Simplified Visualization of Business Operation.* In: *Proceedings IEEE Visualization, Poster Session.* IEEE Computer Society, 2004.

[HKDS06] M. C. Hao, D. A. Keim, U. Dayal und J. Schneidewind: *Business process impact visualization and anomaly detection.* Information Visualization, 5:15–27, 2006.

[Hol95] D. Hollingsworth: *The Workflow Reference Model.* Technischer Bericht WFMC-TC-1003, Workflow Management Coalition, Januar 1995.

[HSS05] S. Hinz, K. Schmidt und C. Stahl: *Transforming BPEL to Petri Nets.* In: W. M. P. v. d. Aalst, B. Benatallah, F. Casati und F. Curbera (Herausgeber): *Proceedings of the Third International Conference on Business Process Management (BPM'05)*, Band 3649 der Reihe *LNCS*, Seiten 220–235, Nancy, France, September 2005.

[HW06a] M. Heller und R. Wörzberger: *Management Support of Interorganizational Cooperative Software Development Processes based on Dynamic Process Views.* In: *Proceedings 15th International Conference on Software Engineering and Data Engineering (SEDE'06)*, Seiten 15–28, Los Angeles, California, USA, Juli 2006.

[HW06b] M. Heller und R. Wörzberger: *A Management System Supporting Interorganizational Cooperative Development Processes in Chemical Engineering.* In: *Proceedings of International Conference on Integrated Design and Process Technology (IDPT-2006)*, Seiten 639–650, San Diego, California, USA, Juni 2006.

[HW06c] M. Heller und R. Wörzberger: *A Management System Supporting Interorganizational Cooperative Development Processes in Chemical Engineering.* Journal of Integrated Design and Process Science: Transactions of the SDPS, 10(2):57–78, 2006.

[IBM06] *Best Practices for Using WebSphere Business Modeler and Monitor.* IBM Redbook Paper, 2006.

[IBM07] IBM: *Cognos 8 Business Intelligence.* Website: http://www.cognos.com/products/cognos8businessintelligence/index.html, 2007. zuletzt besucht: 9.12.2007.

[IDS04] IDS Scheer AG: *ARIS Process Performance Manager (PPM).* White Paper, September 2004.

[IDS07a] IDS Scheer AG: *ARIS Business Publisher 7.0.2.* Fact Sheet, Februar 2007.

[IDS07b] IDS Scheer AG: *ARIS Web Publisher 7.0.2.* Fact Sheet, Februar 2007.

[IL04] M. E. Iacob und D. v. Leeuwen: *Visualisation for Enterprise Architecture - from Conceptual Framework to Prototype.* In: *Proceedings 6th International Conference on Enterprise Information Systems (ICEIS)*, Porto, Portugal, 2004.

[ILO04] ILOG Inc.: *Business Process Management Visualization with ILOG JViews.* Technical White Paper, 2004.

[IN07] IN - Integrierte Informationssysteme GmbH: *Sphinx Open*, 2007. Website: http://www.in-gmbh.de/en/Produkte/Visualisierung/sphinxopen/SphinxOpen.html, zuletzt besucht: 9.12.2007.

[ISO95] *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 10: Dialogue principles.* European Standard EN ISO 9241-10, 1995.

[ISO98] *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11 : Guidance on usability.* European Standard EN ISO 9241-11, 1998.

[ISO01] *Software Engineering – Product Quality – Part 1: Quality Model.* ISO-Standard ISO/IEC 9126-1, 2001.

[ITP06] ITP Commerce: *Process Modeller for Mircorsoft Visio - Management Overview.* Whitepaper, 2006.

[Jan05] T. Jania: *Änderungsmanagement auf Basis eines integrierten Prozess- und Produktdatenmodells mit dem Ziel einer durchgängigen Komplexitätsbewertung.* Dissertation, Fachbereich 10 Maschinentechnik der Universität Paderborn, März 2005.

[JB96] S. Jablonski und C. Bussler: *Workflow Management: Modeling Concepts, Architecture and Implementation.* International Thomson Computer Press, 1996.

[JBA+03] H. Jonkers, R. v. Buuren, F. Arbab, F. S. d. Boer, M. M. Bonsangue, H. Bosma, H. W. L. t. Doest, L. Groenewegen, J. G. Scholten, S. Hoppenbrouwers, M. E. Iacob, W. Janssen, M. M. Lankhorst, D. v. Leeuwen, E. Proper, A. Stam, L. W. N. v. d. Torre und G. V. v. Zanten: *Towards a Language for Coherent Enterprise Architecture Descriptions.* In: *Proceedings 7th International Enterprise Distributed Object Computing Conference (EDOC'03)*, Seiten 28–39. IEEE Computer Society, 2003.

[JBS97] S. Jablonski, M. Böhm und W. Schulze: *Workflow-Management: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie.* dpunkt-Verlag, 1997.

[JFr08] *JFreeChart.* Website: http://www.jfree.org/jfreechart/, 2008. zuletzt besucht: 17.02.2008.

[JG07] S. Jablonski und M. Götz: *Perspective Oriented Business Process Visualization.* In: *Proceedings 3rd International Workshop on Business Process Design (BPD'07)*, Brisbane, Australia, September 2007.

[JLB+04] H. Jonkers, M. Lankhorst, R. v. Buuren, S. Hoppenbrouwers und M. Bonsangue: *Concepts for Modelling Enterprise Architectures.* International Journal of Cooperative Information Systems (IJCIS), 13(3):257–287, September 2004.

[JN05] D. Jackson und C. Northway: *Scalable Vector Graphics (SVG) Full 1.2 Specification.* W3C Working Draft, World Wide Web Consortium (W3C), April 2005.

[Jol05] C. Jolif: *Comparison between XML to SVG Transformation Mechanisms.* In: *Proceedings 4th Annual Conference on Scalable Vector Graphics (SVG Open'05)*, Enschede, Netherlands, August 2005.

[JPP93] R. C. Johnson, D. Pearson und K. Pingali: *Finding Regions Fast: Single Entry Single Exit and Control Regions in Linear Time.* Technischer Bericht TR93-1365, Cornell University, Ithaca, New York, USA, Juli 1993.

[Kay07] M. Kay: *XSL Transformations (XSLT) Version 2.0.* W3C Recommendation, World Wide Web Consortium (W3C), Januar 2007.

[KBR+07] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon und C. Barreto: *Web Services Choreography Description Language Version 1.0 (WS-CDL).* W3C Candidate Recommendation, World Wide Web Consortium (W3C), November 2007.

[KCK01] E. Kafeza, D. K. W. Chiu und I. Kafeza: *View-based Contracts in an E-service Cross-Organizational Workflow Environment.* In: *Proceedings of the Second International Workshop on Technologies for E-Services (TES'01)*, Band 2193 der Reihe *LNCS*, Seiten 74–88, Rom, 2001.

[KFL03] J. W. Koolwaaij, P. Fennema und D. v. Leeuwen: *SVG for Process Visualization.* In: *Proceedings 2nd Annual Conference on Scalable Vector Graphics (SVG Open'03)*, Vancouver, Canada, Juli 2003.

[KH06] R. Klein-Hessling: *Template-Mechanismus.* Praktikumsdokumentation, 2006.

[KHB00] B. Kiepuszewski, A. H. M. t. Hofstede und C. Bussler: *On Structured Workflow Modelling.* In: *Proceedings 12th International Conference on Advanced Information Systems Engineering (CAiSE'00)*, Band 1789 der Reihe *LNCS*, Seiten 431–445, Stockholm, Sweden, Juni 2000.

[KKL+04a] M. Kloppmann, D. König, F. Leymann, G. Pfau und D. Roller: *Business process choreography in WebSphere: Combining the power of BPEL and J2EE.* IBM Systems Journal, 43(2):270–296, 2004.

[KKL+04b] M. Kloppmann, D. König, F. Leymann, G. Pfau und D. Roller: *Enabling Technology: Ein J2EE-basiertes Business Process Management System zur Ausführung von BPEL- und Web Service-basierten Geschäftsprozessen.* it - Information Technology, 46(4):184–192, 2004.

[Klo04] A. Klotz: *View-Unterstützung in Prozess-Management-Systemen.* Diplomarbeit, Universität Ulm, Abteilung Datenbanken und Informationssysteme, Oktober 2004.

[KLRZ94] D. Kimelman, B. Leban, T. Roth und D. Zernik: *Reduction of Visual Complexity in Dynamic Graphs.* In: *Proceedings of the DIMACS International Workshop Graph Drawing (GD'94)*, Band 894 der Reihe *LNCS*, Seiten 218–225, Princeton, New Jersey, USA, Oktober 1994.

[KNS92] G. Keller, M. Nüttgens und A. W. Scheer: *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK).* Technischer Bericht 89, Universität des Saarlandes, Institut für Wirtschaftsinformatik, 1992.

[Kon96] I. Konyen: *Organisations- und Ablaufstrukturen im Krankenhaus – Anforderungen, Werkzeuge und deren Anwendung.* Diplomarbeit, Universität Ulm, Abteilung Datenbanken und Informationssysteme, 1996.

[KP88] G. E. Krasner und S. T. Pope: *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80.* Journal on Object Oriented Programming, 1(3):26–49, 1988.

[KP96] Y. Kambayashi und Z. Peng: *An Object Deputy Model for Realization of Flexible and Powerful Objectbases.* Journal of Systems Integration, 6(4):329–362, 1996.

[KV06] A. Kalnins und V. Vitolins: *Use of UML and Model Transformations for Workflow Process Definitions.* In: *Proceedings Baltic Databases and Information Systems (BalticDB&IS'06)*, Seiten 3–15, Vilnius, Lithuania, Juli 2006.

[KW01] M. Kaufmann und D. Wagner (Herausgeber): *Drawing Graphs: Methods and Models*, Band 2025 der Reihe *LNCS.* Springer, 2001.

[Lan05] M. Lankhorst: *Enterprise Architecture at Work: Modelling, Communication and Analysis.* Springer, 2005.

[LDL04] D. v. Leeuwen, H. t. Doest und M. M. Lankhorst: *A Tool Integration Workbench for Enterprise Architecture - Integrating Heterogeneous Models and Tools.* In: *Proceedings 6th International Conference on Enterprise Information Systems (ICEIS)*, Seiten 470–478, Porto, Portugal, 2004.

[LGB05] S. Lippe, U. Greiner und A. Barros: *A Survey on State of the Art to Facilitate Modelling of Cross-Organisational Business Processes.* In: *Proceedings of the 2nd GI-Workshop XML4BPM 2005*, Seiten 7–22, Karlsruhe, Germany, März 2005.

[LJBL98] C. Lilley, I. Jacobs, B. Bos und H. W. Lie: *Cascading Style Sheets, level 2 (CSS2) Specification.* W3C Recommendation, World Wide Web Consortium (W3C), Mai 1998.

[LK01] A. M. Latva-Koivisto: *User Interface Design for Business Process Modelling and Visualisation.* Diplomarbeit, Helsinki University of Technology, Januar 2001.

[LLW+01] P. O. Luttighuis, M. Lankhorst, R. v. d. Wetering, R. Bal und H. v. d. Berg: *Visualising business processes.* Computer Languages, 27(1–3):39–59, 2001.

[Lof02] N. Lofts: *Process Visualization: An Executive Guide to Business Process Design.* Wiley, August 2002.

[LR00] F. Leymann und D. Roller: *Production Workflow.* Prentice Hall, 2000.

[LR07] R. Lenz und M. Reichert: *IT Support for Healthcare Processes - Premises, Challenges, Perspectives.* Data and Knowledge Engineering, 61(1):39–58, 2007.

[LRZ06] U. Lindemann, R. Reichwald und M. F. Zäh (Herausgeber): *Individualisierte Produkte – Komplexität beherrschen in Entwicklung und Produktion.* Springer, 2006.

[LS01] D. R. Liu und M. Shen: *Modeling Workflows with a Process-View Approach.* In: *Proceedings of the 7th International Conference on Database Systems for Advanced Applications (DASFAA'01)*, Seiten 260–267, Hong Kong, China, 2001. IEEE Computer Society.

[LS03] D. R. Liu und M. Shen: *Workflow Modeling for Virtual Processes: an Order-Preserving Process-View Approach.* Information Systems, 28(6):505–532, 2003.

[LS04] D. R. Liu und M. Shen: *Business-to-Business Workflow Interoperation Based on Process-Views.* Journal of Decision Support Systems, 38(3):399–419, 2004.

[LSH+06] Q. Li, Z. Shan, P. C. K. Hung, D. K. W. Chiu und S. C. Cheung: *Flows and Views for Scalable Scientific Process Integration.* In: *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale'06)*, 2006.

[LZLC02] H. Lin, Z. Zhao, H. Li und Z. Chen: *A Novel Graph Reduction Algorithm to Identify Structural Conflicts.* In: *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS02)*, Band 9, Big Island, Hawaii, 2002.

[Mac04] J. Maciuga: *Anforderungen und Lösungsansätze an eine zielgruppengerechte Visualisierung von Prozessmodellen.* Diplomarbeit, Fachhochschule Esslingen, Hochschule für Technik, August 2004.

[May99] D. J. Mayhew: *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design.* Morgan Kaufmann, 1999.

[MBGM05a] Z. Maamar, D. Benslimane, C. Ghedira und M. Mrissa: *On Tracking Personalized Web Services Using Views.* In: *Proceedings of International Conference on e-Technology, e-Commerce, and e-Services (EEE'05)*, Seiten 432–437, Hong Kong, China, März 2005. IEEE Computer Society.

[MBGM05b] Z. Maamar, D. Benslimane, C. Ghedira und M. Mrissa: *Views in Composite Web Services.* IEEE Internet Computing, 9(4):52–57, 2005.

[MBN04] J. Mendling, A. Brabenetz und G. Neumann: *EPML2SVG - Generating Websites from EPML Processes.* In: *Proceedings of the 3rd GI Workshop on Event-Driven Process Chains (EPK'04)*, Luxembourg, Oktober 2004.

[Men04] J. Mendling: *A Survey on Design Criteria for Interchange Formats.* Technischer Bericht JM-2004-06-02, Vienna University of Economics and Business Administration, 2004.

[Mey96] J. Meyer: *Anforderungen an zukünftige Workflow-Management-Systeme: Flexibilisierung, Ausnahmebehandlung und Dynamisierung – Erörterung am Beispiel medinzinisch-organisatorischer Abläufe.* Diplomarbeit, Universität Ulm, Abteilung Datenbanken und Informationssysteme, 1996.

[MHHR06] D. Müller, J. Herbst, M. Hammori und M. Reichert: *IT Support for Release Management Processes in the Automotive Industry.* In: *Proceedings International Conference Business Process Management (BPM'06)*, Band 4102 der Reihe *LNCS*, Seiten 368–377, 2006.

[MHSG02a] K. Matković, H. Hauser, R. Sainitzer und M. E. Gröller: *Process Visualization with Levels of Detail.* In: *Proceedings IEEE Symposium on Information Visualization*, Seiten 67–70. IEEE Computer Society Press, 2002.

[MHSG02b] K. Matković, H. Hauser, R. Sainitzer und M. E. Gröller: *Process Visualization with Levels of Detail.* Technischer Bericht, VRVis Research Center, Vienna, 2002.

[Mic07a] Microsoft: *Project.* Website: http://office.microsoft.com/project, 2007. zuletzt besucht: 9.12.2007.

[Mic07b] Microsoft: *Visio.* Website: http://office.microsoft.com/visio, 2007. zuletzt besucht: 9.12.2007.

[MID06] MID Enterprise Software Solutions GmbH: *Innovator 2007 Leitfaden.* Tutorial, Oktober 2006.

[Mih05] T. Mihalca: *Prozessdatenintegration und -transformation für die systemübergreifende Visualisierung von Arbeitsabläufen.* Diplomarbeit, Universität Ulm, Abteilung Datenbanken und Informationssysteme, November 2005.

[MK07] M. Murzek und G. Kramler: *Business Process Model Transformation Issues.* In: *Proceedings 9th International Conference on Enterprise Information Systems (ICEIS'07)*, Funchal, Madeira - Portugal, Juni 2007.

[MMN06] J. Mendling, M. Moser und G. Neumann: *Transformation of yEPC business process models to YAWL.* In: *Proceedings of the 2006 ACM symposium on Applied Computing (SAC'06)*, Seiten 1262–1266, Dijon, France, 2006. ACM Press.

[MMT02] K. Marriott, B. Meyer und L. Tardif: *Fast and Efficient Client-Side Adaptivity for SVG.* In: *Proceedings of the 11th International World Wide Web Conference (WWW '02)*, Honolulu, Hawaii, USA, Mai 2002.

[MN03a] J. Mendling und M. Nüttgens: *Konzeption eines XML-basierten Austauschformates für Ereignisgesteuerte Prozessketten (EPK).* Informationssystem Architekturen, Wirtschaftsinformatik Rundbrief der GI Fachgruppe WI-MobIS, 10(2):89–103, 2003.

[MN03b] M. Momotko und B. Nowicki: *Visualisation of (Distributed) Process Execution based on Extended BPMN.* In: *Proceedings 14th International Workshop on Database and Expert Systems Applications (DEXA Workshops'03)*, Seiten 280–284, Prague, Czech Republic, 2003.

[MNN04] J. Mendling, G. Neumann und M. Nüttgens: *A Comparison of XML Interchange Formats for Business Process Modelling.* In: *Proceedings EMISA 2004 - Informationssysteme im E-Business und E-Government*, Lecture Notes in Informatics (LNI), Seiten 129–140, Luxemburg, Oktober 2004.

[Mol06] M. Moldmann: *Visualisierungskonzepte für Prozessinformationen.* Diplomarbeit, Universität Ulm, Abteilung Datenbanken und Informationssysteme, Februar 2006.

[MRH06] D. Müller, M. Reichert und J. Herbst: *Flexibility of Data-Driven Process Structures.* In: *Proceedings BPM'06 Workshops*, Band 4103 der Reihe *LNCS*, Seiten 181–192, 2006.

[MRH07] D. Müller, M. Reichert und J. Herbst: *Data-Driven Modeling and Coordination of Large Process Structures.* In: *Proceedings International Conference on Cooperative Information Systems (CoopIS'07)*, Band 4803 der Reihe *LNCS*, Seiten 131–149, 2007.

[MRH08] D. Müller, M. Reichert und J. Herbst: *A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures.* In: *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, Montpellier, France, Juni 2008. (accepted for publication).

[MRHP07] D. Müller, M. Reichert, J. Herbst und F. Poppa: *Data-Driven Design of Engineering Processes with COREPRO$_{Modeler}$.* In: *Proceedings WETICE IEEE-Workshop on Agile Cooperative Process-Aware Information Systems (ProGility'07)*, Seiten 376–378, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[MSW+04] A. Martens, C. Stahl, D. Weinberg, D. Fahland und T. Heidinger: *Business Process Execution Language for Web Services – Semantik, Analyse und Visualisierung.* Technischer Bericht, Humboldt-Universität zu Berlin, Institut für Informatik, 2004.

[Mue04] M. z. Muehlen: *Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems.*, Band 6 der Reihe *Advances in Information Systems and Management Science*. Logos, Berlin, 2004.

[MZ05a] J. Mendling und J. Ziemann: *EPK-Visualisierung von BPEL4WS Prozessdefinitionen*. In: *Proceedings 7th Workshop Software-Reengineering (WSR'05)*, Bad Honnef, Germany, 2005.

[MZ05b] J. Mendling und J. Ziemann: *Transformation of BPEL Processes to EPCs*. In: *Proceedings of the 4th GI Workshop on Event-Driven Process Chains (EPK'05)*, Band 167 der Reihe *CEUR Workshop Proceedings*, Seiten 41–53, Hamburg, Germany, Dezember 2005.

[NH98] N. H. Narayanan und R. Hübscher: *Visual Language Theory: Towards a Human-Computer Interaction Perspective*. In: *Visual Language Theory*, Seiten 85–127. Springer, 1998.

[NL06] F. Naumann und U. Leser: *Informationsintegration*. dpunkt-Verlag, 2006.

[NM04] M. Nüttgens und J. Mendling (Herausgeber): *XML4BPM XML Interchange Formats for Business Process Management*, Marburg, März 2004. Gesellschaft für Informatik (GI) e.V. in conjunction with the 7th GI Conference Modellierung 2004.

[NM05] M. Nüttgens und J. Mendling (Herausgeber): *XML4BPM 2005, Proceedings of the 2nd GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 11th GI Conference BTW 2005*, Karlsruhe, Germany, März 2005.

[Nor96] S. C. North: *Incremental Layout in DynaDAG*. In: *Proceedings of the Symposium on Graph Drawing (GD'95)*, Band 1027 der Reihe *LNCS*, Seiten 409–418, Passau, Germany, September 1996.

[OAS07] OASIS: *Web Services Business Process Execution Language (WS-BPEL)*, 2007. Version 2.0.

[ODHA06] C. Ouyang, M. Dumas, A. H. M. t. Hofstede und W. M. P. v. d. Aalst: *From BPMN Process Models to BPEL Web Services*. In: *Proceedings IEEE International Conference on Web Services (ICWS'06)*, Seiten 285–292, Chicago, USA, 2006.

[OMG03] Object Management Group (OMG): *Business Process Definition Metamodel*, Januar 2003. Request for Proposal.

[ÖV99] M. T. Özsu und P. Valduriez: *Principles of Distributed Data Base Systems*. Prentice Hall, 1999.

[PAC02] H. C. Purchase, J. A. Allder und D. A. Carrington: *Graph Layout Aesthetics in UML Diagrams: User Preferences*. Journal of Graph Algorithms and Applications, 6(3):255–279, 2002.

[PCJ96] H. C. Purchase, R. F. Cohen und M. James: *Validating Graph Drawing Aesthetics.* In: *Proceedings of the Symposium on Graph Drawing (GD'95)*, Nummer 1027 in *LNCS*, Seiten 435–446, London, UK, 1996.

[PF93] K. Perlin und D. Fox: *Pad: An Alternative Approach to the Computer Interface.* In: *Proceedings of the 20st Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'93)*, Seiten 57–64, 1993.

[PG05] G. v. d. Putte und L. Gavin: *Technical Overview of WebSphere Process Server and WebSphere Integration Developer.* IBM Redbook, Dezember 2005.

[PHG06] H. C. Purchase, E. E. Hoggan und C. Görg: *How Important Is the Mental Map? - An Empirical Investigation of a Dynamic Graph Layout Algorithm.* In: *Proceedings 14th International Symposium on Graph Drawing*, Band 4372 der Reihe *LNCS*, Seiten 184–195, Karlsruhe, Germany, 2006.

[PMN03] T. Poranen, E. Mäkinen und J. Nummenmaa: *How to Draw a Sequence Diagram.* In: *Proceedings 8th Symposium on Programming Languages and Software Tools (SPLST'03)*, Seiten 91–102, Kuopio, Finland, Juni 2003.

[PS05] V. Pankratius und W. Stucky: *A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets.* In: S. Hartmann und M. Stumptner (Herausgeber): *Proceedings Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*, Band 43 der Reihe *CRPIT*, Seiten 79–88, Newcastle, Australia, 2005.

[Pur00] H. C. Purchase: *Effective Information Visualisation: A Study of Graph Drawing Aesthetics and Algorithms.* Interacting with Computers, 13(2):147–162, 2000.

[Pur02] H. C. Purchase: *Metrics for Graph Drawing Aesthetics.* Journal Visual Language Computing, 13(5):501–516, 2002.

[RAP07] *Rich Ajax Platform (RAP) 1.0.* Website: http://www.eclipse.org/rap/, 2007. zuletzt besucht: 29.10.2007.

[RBRB06] S. Rinderle, R. Bobrik, M. Reichert und T. Bauer: *Business Process Visualization - Use Cases, Challenges, Solutions.* In: *Proceedings 8th International Conference on Enterprise Information Systems (ICEIS'06), Track on Information Systems Analysis and Specification*, Seiten 204–211, Paphos, Cyprus, Mai 2006.

[RC02] M. B. Rosson und J. M. Carroll: *Usability Engineering: Scenario-Based Development of Human Computer Interaction.* Morgan Kaufmann, 2002.

[RD98] M. Reichert und P. Dadam: *ADEPT flex - Supporting Dynamic Changes of Workflows Without Losing Control.* Journal of Intelligent Information Systems, 10(2):93–129, 1998.

[Ree79] T. Reenskaug: *Thing-Model-View-Editor – an Example from a planningsystem.* Xerox PARC technical note, Website: http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf, Mai 1979. zuletzt besucht: 29.10.2007.

[Rei00] M. Reichert: *Dynamische Ablaufänderungen in Workflow-Management-Systemen.* Dissertation, Universität Ulm, Mai 2000.

[Rei06] M. Reinhold: *Java SE 6 Release Contents*. Java Specification Request JSR-000270, Java Community Process, 2006.

[Rem02] U. Remus: *Prozessorientiertes Wissensmanagement. Konzepte und Modellierung*. Dissertation, Universität Regensburg, Wirtschaftswissenschaftliche Fakultät, 2002.

[Rhi06] *Rhino: JavaScript for Java 1.6R5*. http://www.mozilla.org/rhino/, November 2006.

[RR03] M. Reichert und S. Rinderle: *System- und bereichsübergreifende Visualisierung von Entwicklungsprozessen – Anforderungen, Werkzeuge und Methoden*. interner Bericht DaimlerChrysler AG, 2003.

[SADL04] M. Steen, D. Akehurst, H. t. Doest und M. Lankhorst: *Supporting Viewpoint-Oriented Enterprise Architecture*. In: *Proceedings 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC'04)*, Seiten 201–211, Monterey, California, September 2004.

[SAP07] SAP: *BusinessObjects XI*. Website: http://www.businessobjects.com/products/businessobjectsxi/, 2007. zuletzt besucht: 9.12.2007.

[SBE00] B. Schönhage, A. v. Ballegooij und A. Elliëns: *3D Gadgets for Business Process Visualization: a Case Study*. In: *Proceedings of the fifth Symposium on Virtual Reality Modeling Language (Web3D-VRML)*, Seiten 131–138, Monterey, California, 2000. ACM Press.

[SBH+05] H. Schulze, H. Brau, S. Haasis, M. Weyrich und T. Rhatje: *Human-Centered design of engineering applications: Success factors from a case study in the automotive industry*. Human Factors in Ergonomics & Manufacturing, 15(4):421–443, 2005.

[SCDS02] M. Sayal, F. Casati, U. Dayal und M. C. Shan: *Business Process Cockpit*. In: *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'02)*, Seiten 880–883, 2002.

[Sch96a] A. W. Scheer: *ARIS-House of Business Engineering*. Heft 133, Institut für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Saarbrücken, Germany, September 1996.

[Sch96b] B. Schultheiss: *Prozeßreengineering in klinischen Anwendungsumgebungen – Beispiele, Vorgehensmodelle, Werkzeuge*. Diplomarbeit, Universität Ulm, Abteilung Datenbanken und Informationssysteme, 1996.

[Sch00] A. W. Scheer: *ARIS - Business Process Modeling*. Springer, 2000.

[Sch01] A. W. Scheer: *ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen*. Springer, 2001.

[SCL05] Z. Shan, D. K. Chiu und Q. Li: *Systematic Interaction Management in a Workflow View Based Business-to-business Process Engine*. In: *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS05)*, Big Island, Hawaii, 2005.

[Sei06] H. Seidlmeier: *Prozessmodellierung mit ARIS. Eine beispielorientierte Einführung für Studium und Praxis.* Vieweg, 2006.

[Shn94] B. Shneiderman: *Designing the User Interface.* Addison Wesley, 1994.

[SJHK06] A. W. Scheer, W. Jost, H. Hess und A. Kronz: *Corporate Performance Management: ARIS in practice.* Springer, 2006.

[SL01] M. Shen und D. R. Liu: *Coordinating Interorganizational Workflows based on Process-Views.* In: *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA'01)*, Band 2113 der Reihe *LNCS*, Seiten 274–283, München, September 2001.

[SL03] M. Shen und D. R. Liu: *Discovering Role-Relevant Process-Views for Recommending Workflow Information.* In: *Proceedings 14th International Conference on Database and Expert Systems Applications (DEXA'03)*, Band 2736 der Reihe *LNCS*, Seiten 836–845, Prag, 2003.

[SL04] M. Shen und D. R. Liu: *Discovering Role-Relevant Process-Views for Disseminating Process Knowledge.* Expert Systems with Applications, 26(3):301–310, April 2004.

[SL06] T. Schreiter und G. Laures: *A Business Process-centered Approach for Modeling Enterprise Architectures.* In: *Proceedings Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen (EMISA'06)*, Band 95 der Reihe *LNI*, Seiten 147–162, Hamburg, Germany, Oktober 2006.

[SLLP04] Z. Shan, Z. Long, Y. Luo und Z. Peng: *Object-Oriented Realization of Workflow Views for Web Services - An Object Deputy Model Based Approach.* In: *Proceedings 5th International Conference Advances in Web-Age Information Management (WAIM'04)*, Band 3129 der Reihe *LNCS*, Seiten 468–477, Dalian, China, Juli 2004.

[SLLP05] Z. Shan, Q. Li, Y. Luo und Z. Peng: *Deputy Mechanism for Workflow Views.* In: *Proceedings 10th International Conference Database Systems for Advanced Applications (DASFAA'05)*, Band 3453 der Reihe *LNCS*, Seiten 816–827, Beijing, China, 2005.

[SM00] H. Schumann und W. Müller: *Visualisierung. Grundlagen und allgemeine Methoden.* Springer, Berlin, 2000.

[SM04] J. Schiefer und C. McGregor: *Correlating Events for Monitoring Business Processes.* In: *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS'04)*, Seiten 320–327, 2004.

[SMW07] M. Strommer, M. Murzek und M. Wimmer: *Applying Model Transformation By-Example on Business Process Modeling Languages.* In: *Proceedings Advances in Conceptual Modeling - Foundations and Applications (ER'07 Workshops)*, Band 4802 der Reihe *LNCS*, Seiten 116–125, Auckland, New Zealand, November 2007.

[SN00] A. W. Scheer und M. Nüttgens: *ARIS Architecture and Reference Models for Business Process Management.* In: *Proceedings Business Process Management: Models, Techniques, and Empirical Studies*, Band 1806 der Reihe *LNCS*, Seiten 376–389, 2000.

[SO99] W. Sadiq und M. E. Orlowska: *Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models.* In: *Proceedings 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*, Nummer 1626 in *LNCS*, Seiten 195–209, Heidelberg, Germany, Juni 1999.

[SO00] W. Sadiq und M. E. Orlowska: *Analyzing Process Models Using Graph Reduction Techniques.* Information Systems, 25(2):117–134, 2000.

[SO04] K. A. Schulz und M. E. Orlowska: *Facilitating Cross-Organisational Workflows with a Workflow View Approach.* Data Knowledge Engineering, 51(1):109–147, 2004.

[Som03] M. Sommer: *Konzeption und Realisierung externer Sichten auf interne Workflows.* Diplomarbeit, Universität Stuttgart, Institut für parallele und verteilte Systeme, Februar 2003.

[SPB05] A. Streit, B. Pham und R. Brown: *Visualization Support for Managing Large Business Process Specifications.* In: *Proceedings 3rd International Conference Business Process Management (BPM)*, Band 3649 der Reihe *LNCS*, Seiten 205–219, Nancy, France, September 2005.

[ST01] J. M. Six und I. G. Tollis: *Automated Visualization of Process Diagrams.* In: P. Mutzel, M. Jünger und S. Leipert (Herausgeber): *Proceeding of 9th International Symposium on Graph Drawing (GD'01)*, Band 2265 der Reihe *LNCS*, Seiten 45–59, Vienna, Austria, September 2001.

[STT81] K. Sugiyama, S. Tagawa und M. Toda: *Methods for Visual Understanding of Hierarchical Systems.* IEEE Transactions on Systems, Man and Cybernetics, 11(2):109–125, 1981.

[SV01] G. Sander und A. Vasiliu: *The ILOG JViews Graph Layout Module.* In: *Proceedings 9th International Symposium on Graph Drawing (GD'01)*, Band 2265 der Reihe *LNCS*, Seiten 469–475, Vienna, Austria, 2001.

[SYL+06] Z. Shan, Y. Yang, Q. Li, Y. Luo und Z. Peng: *A Light-Weighted Approach to Workflow View Implementation.* In: *Proceedings of 8th Asia-Pacific Web Conference (APWeb'06)*, Band 3841 der Reihe *LNCS*, Seiten 1059–1070, Harbin, China, Januar 2006.

[Tan05] D. Tan: *Supporting Collaboration with User Oriented Process Visualisation.* In: *Proceedings Interdisciplinary Information Management Talks (IDIMT'05)*, Budweis, Tschechien, September 2005.

[Tan08] D. Tan: *Prozessorientierte Nutzerunterstützung in Workflows und deren Auswirkung auf Process Awareness und Leistung.* Dissertation, Berlin, 2008.

[TBET98] I. G. Tollis, G. D. Battista, P. Eades und R. Tamassia: *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1998.

[TF05] D. Tan und L. Fritzsche: *Kultur- und kontextunabhängige Visualisierung von Geschäftsprozessen.* Interner Bericht DaimlerChrysler AG, 2005.

[Tie03] O. Tietze: *Strategische Positionierung in der Automobilbranche.* Deutscher Universitätsverlag, 2003.

[Tuf90] E. R. Tufte: *Envisioning Information.* Graphics Press, 1990.

[TW07] D. Tan und H. Wandke: *Process-Oriented User Support for Workflow Applications.* In: *Proceedings 12th International Conference on Human-Computer Interaction*, Band 4553 der Reihe *LNCS*, Seiten 752–761, Beijing, China, Juli 2007.

[UB08] T. Unger und T. Bauer: *Towards a Standardized Task Management.* In: *Proceedings 4th GI-Workshop XML Integration and Transformation for Business Process Management*, München, Februar 2008. Extended Abstract in: *Proc. Multikonferenz Wirtschaftsinformatik*, München, Februar 2008.

[UML04] *Unified Modeling Language (UML 2.0) Superstructure Specification*, 2004. Final Adopted Specification.

[VVL07] J. Vanhatalo, H. Völzer und F. Leymann: *Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition.* In: *Proceedings 5th International Conference on Service-Oriented Computing (ICSOC 2007)*, Band 4749 der Reihe *LNCS*, Seiten 43–55, September 2007.

[WAM+07] U. Wahli, V. Avula, H. Macleod, M. Saeed und A. Vinther: *Business Process Management: Modeling through Monitoring Using WebSphere V6.0.2 Products.* IBM Redbook, August 2007.

[War04] C. Ware: *Information Visualization, Second Edition: Perception for Design.* Morgan Kaufmann, 2004.

[Wei03] D. Weinberg: *Grafische Repräsentation von BPEL4WS.* Studienarbeit, Humboldt-Universität Berlin, August 2003.

[WfM99] WfMC: *Terminology and Glossary.* Technischer Bericht WFMC-TC-1011, Workflow Management Coalition, 1999.

[Wic06] A. Wickenhäuser: *Business Process Management und Workflow-Technologie.* IBM, 2006.

[WLS98] A. Woodruff, J. Landay und M. Stonebraker: *Goal-Directed Zoom.* In: *Proceedings of Conference on Human Factors in Computing Systems (CHI'98)*, Seiten 305–306, 1998.

[WVA+06a] M. T. Wynn, H. M. W. Verbeek, W. M. P. v. d. Aalst, A. t. Hofstede und D. Edmond: *Reduction Rules for Reset Workflow Nets.* Technischer Bericht, BPMCenter.org, Eindhoven, 2006.

[WVA+06b] M. T. Wynn, H. M. W. Verbeek, W. M. P. v. d. Aalst, A. t. Hofstede und D. Edmond: *Reduction Rules for YAWL Workflow Nets with Cancellation Regions and OR-joins.* Technischer Bericht, BPMCenter.org, Eindhoven, 2006.

[WW96a] K. Wittenburg und L. Weitzman: *Process Visualization in ShowBiz.* In: *Proceedings of the Symposium on Graph Drawing (GD'96)*, Nummer 1190 in *LNCS*, London, UK, 1996.

[WW96b] K. Wittenburg und L. Weitzman: *Qualitative Visualization of Processes: Attributed Graph Layout and Focusing Techniques.* In: *Proceedings of the Symposium on Graph Drawing (GD'96)*, Band 1190 der Reihe *LNCS*, Seiten 401–408, London, UK, 1996.

[YCP+06] F. Yergeau, J. Cowan, J. Paoli, T. Bray, E. Maler und C. M. Sperberg-McQueen: *Extensible Markup Language (XML) 1.1.* W3C Recommendation, World Wide Web Consortium (W3C), September 2006.

[YLS+04] Y. Yang, W. Lai, J. Shen, X. Huang, J. Yan und L. Setiawan: *Effective Visualisation of Workflow Enactment.* In: *Proceedings Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference (APWeb'04)*, Band 3007 der Reihe *LNCS*, Seiten 794–803, Hangzhou, China, April 2004.

[YNY07] A. Yuichi, N. Noriyasu und A. Yutaka: *Development Process Visualization and Project Management.* Fujitsu Scientific and Technical Journal, 43(1):97–104, 2007.

[ZD04] F. Zhang und E. H. D'Hollander: *Using Hammock Graphs to Structure Programs.* IEEE Transactions on Software Engineering, 30(4):231–245, 2004.

[Zha07] X. Zhao: *Automatisches Layout von Prozessgraphen.* Masterarbeit, Universität Ulm, Institut für Datenbanken und Informationssysteme, Juni 2007.

[ZLY05] X. Zhao, C. Liu und Y. Yang: *An Organisational Perspective on Collaborative Business Processes.* In: *Proceedings of International Conference on Business Process Management (BPM'05)*, Band 3649 der Reihe *LNCS*, Seiten 17–31, Nancy, France, September 2005.

Teil V

Anhang

A

Zustandsabbildungsfunktion

Für die View-Bildung auf Prozessinstanzen brauchen wir eine Abbildungsvorschrift, die bei der Aggregation einer Menge von Aktivitäten den Ausführungszustand der abstrakten Aktivität berechnet. In Abschnitt 4.3.3.1 auf Seite 73 haben wir dazu die Funktion *VNS* eingeführt. Tabelle A.1 zeigt, wie die Funktion aus den Zuständen, die in der zu aggregierenden Aktivitätenmenge existieren, abgeleitet werden kann. Ein + in der Tabelle steht dabei für die Existenz mindestens einer Aktivität mit dem jeweiligen Zustand.

Vorkommen der Zustände der Basisaktivitäten in X					$VNS(X)$
NotActivated	*Activated*	*Running*	*Completed*	*Skipped*	
					nicht erlaubt
				+	*Skipped*
			+		*Completed*
			+	+	*Completed*
		+			*Running*
		+		+	*Running*
		+	+		*Running*
		+	+	+	*Running*
	+				*Activated*
	+			+	*Activated*
	+		+		*Running*
	+		+	+	*Running*
	+	+			*Running*
	+	+		+	*Running*
	+	+	+		*Running*
	+	+	+	+	*Running*
+					*NotActivated*
+				+	*NotActivated*
+			+		*Running*

...

NotActivated	*Activated*	*Running*	*Completed*	*Skipped*	*VNS*(X)
+			+	+	*Running*
+		+			*Running*
+		+		+	*Running*
+		+	+		*Running*
+		+	+	+	*Running*
+	+				*Activated*
+	+			+	*Activated*
+	+		+		*Running*
+	+		+	+	*Running*
+	+	+			*Running*
+	+	+		+	*Running*
+	+	+	+		*Running*
+	+	+	+	+	*Running*

Tabelle A.1: Herleitung der Zustandsabbildungsfunktion *VNS*

Kompakt lässt sich die Abbildung *VNS* wie folgt schreiben:

$$VNS(X) = \begin{cases} NotActivated & \forall x \in X : NS(x) \notin \{Activated, Running, \\ & Completed\} \ \wedge \ \exists x \in X : NS(x) = NotActivated \\ Activated & \exists x \in X : NS(x) = Activated \ \wedge \\ & \forall x \in X : NS(x) \notin \{Running, Completed\} \\ Running & \exists x \in X : NS(x) = Running \ \vee \\ & \exists x_1, x_2 \in X : NS(x_1) = Completed \ \wedge \\ & (NS(x_2) = NotActivated \vee NS(x_2) = Activated) \\ Completed & \forall x \in X : NS(x) \notin \{NotActivated, Running, \\ & Activated\} \wedge \exists x \in X : NS(x) = Completed \\ Skipped & \forall x \in X : NS(x) = Skipped \end{cases}$$

B

Metamodell

In Proviado sollen Prozesse aus verschiedensten Quellsystemen integriert visualisiert werden. Um eine einheitliche Basis für die Visualisierung zu schaffen, wurde das Proviado-Metamodell entwickelt. Es hat zum Ziel, die Modellierungselemente möglichst vieler unterschiedlicher Prozessmetamodelle abbilden zu können. Im Gegenzug wird auf eine streng formale Ausführungssemantik verzichtet, da dies für die reine Visualisierung nicht erforderlich ist und die Ausdrucksmächtigkeit stark einschränken würde.

Das Proviado-Metamodell ist an ein allgemeines Graphenmodell mit Knoten und Kanten angelehnt (siehe UML-Klassendiagramm in Abbildung B.1). Die einzelnen Prozesselemente sind daher als Spezialisierungen von Knoten bzw. Kanten realisiert. Zusätzlich existieren Objekte für die Repräsentation des Gesamtprozessmodells (vgl. *Prozess*). Zur Abbildung von Prozessinstanzen sind in Abbildung B.1 weitere Attribute angegeben, welche die Modellobjekte verfeinern.

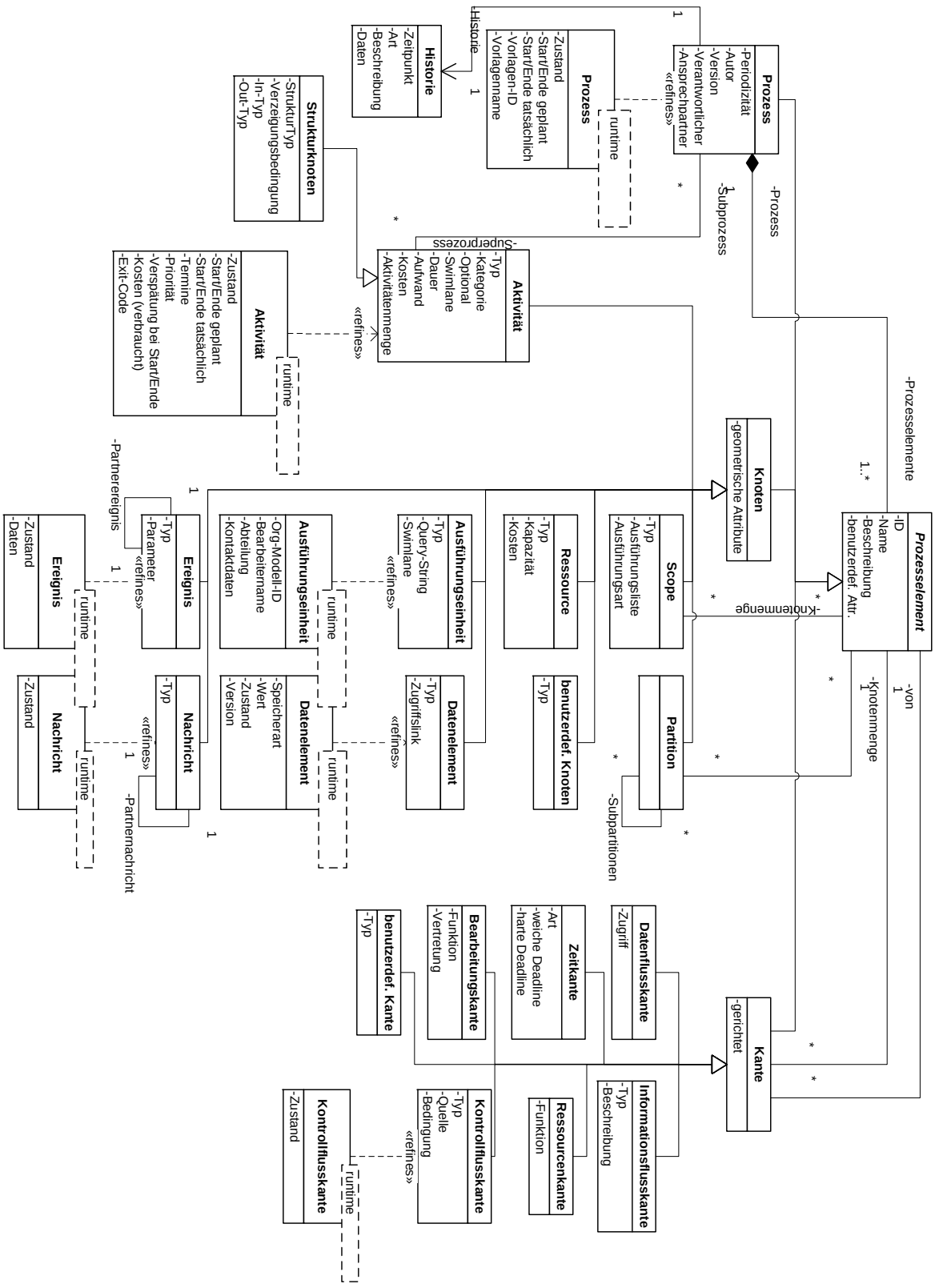


Abbildung B.1: Kanonisches Metamodell im Detail