M. Sebastian E. K Görg

Entwicklung einer Komponente zur automatisierten Adaption von Workflows

Diplomarbeit

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de/ abrufbar.

Impressum:

Copyright © 2010 GRIN Verlag
ISBN: 9783640892358

M. Sebastian E. K Görg

Entwicklung einer Komponente zur automatisierten Adaption von Workflows

GRIN Verlag

Diplomarbeit

Entwicklung einer Komponente zur automatisierten Adaption von Workflows

M. Sebastian E. K. Görg

Universität Trier
Lehrstuhl für Wirschaftsinformatik II

"An dieser Stelle möchte ich allen aufrichtig danken, die mich während der Schaffung meiner Diplomarbeit unterstützt haben. Damit meine ich nicht nur den unschätzbaren fachlichen Rat meiner Arbeitskollegen und Freunde, sondern auch die moralische Unterstützung meiner Eltern, meiner Geschwister, meiner ganzen Familie und meiner Laura."

Inhaltsverzeichnis

Kapitel 1 - Einleitung

1.1 Motivation

Die Menschheit hat vor 200 Jahren einen Prozess begonnen, der unsere Gesellschaft bis heute prägt und sich ständig zu beschleunigen scheint. Die Rede ist von Arbeitsteilung und Arbeitseffizienz durch Technik mit Beginn der industriellen Revolution. Arbeitsteilung und Arbeitseffizienz sind zwei der wesentlichsten Faktoren, die darüber entscheiden, ob ein Unternehmen am Markt überlebensfähig ist. Ein wachsender Wettbewerb innerhalb einer Branche erfordert von Unternehmen eine genauere Betrachtung ihrer Organisationsstruktur und der darin enthaltenen Arbeitsabläufe. In klassischen Branchen, wie der Autobranche oder Stahlindustrie herrscht für Arbeitsabläufe schon lange ein hoher Grad der Formalisierung. Ganz anders sieht es dagegen bei Dienstleistern aus. Kosten und Arbeitseffizienz lassen sich bei Dienstleistungen oft nur schwer kalkulieren. Dieses Problem gewinnt immer mehr an Bedeutung, da der Großteil der Gründungen der letzten Jahre im Dienstleistungssektor stattfand[1]. Darüber hinaus ist es für Dienstleistungsunternehmen schwieriger, eine gleichbleibende Qualität an ihre Kunden zu liefern, wenn es keine Standardisierung für die erbrachte Dienstleistung gibt. Ein Unternehmen hat einen Vorteil gegenüber seinen Mitbewerbern, wenn es in der Lage ist, das Wissen über seine Arbeitsabläufe zu formalisieren, sodass es in der Organisation verbleibt und auch an neue Mitarbeiter weitergegeben werden kann. Neben Produktionsabläufen existiert in einer Organisation eine Vielzahl verborgener Prozesse, die die interne Kommunikation steuern und aufrechterhalten. Diese Abläufe in einer Organisation sind häufig unstrukturiert, da sie nicht explizit bekannt sind und somit nicht von allen Mitarbeitern auf Anhieb verstanden werden können. Um nun wettbewerbsfähiger zu werden, haben viele Unternehmen begonnen, ihre eigenen Prozesse festzuhalten, zu formalisieren und zu optimieren. Nicht selten wird dabei versucht „Best Practice“ Abläufe der Branche im eigenen Unternehmen zu verankern, um wettbewerbsfähig zu bleiben. Für viele kleine und mittelständische Unternehmen bedeutet das Überschreiten einer gewissen Größe einen Wandel des Aufgabenbereichs der Führung; weg von ausführenden hin zu planenden Aufgaben. Solche Unternehmen im Mittelstand werden bspw. durch die UPTODATE-Offensive [2] unterstützt. Die UPTODATE-Offensive hilft Handwerksunternehmen ihr

[1] Laut KfW Gründungsmonitor machte allein 2009 der Anteil der Gründungen im Dienstleistungssektor 86% aus.
[2] Für nähere Informationen: http://www.profi-im-handwerk.de

Wachstum zu bewältigen, indem standardisierte Prozesse im Unternehmen etabliert werden. Größere Unternehmen nutzen zur Betreuung ihres Kundenstamms häufig die „Best Practice“ Vorschläge aus der „Information Technology Infrastructure Library“(ITIL). ITIL stellt eine Sammlung von Leitfäden dar, die bewährte, aus der Praxis gewonnene Erkenntnisse, Modelle und Architekturen beschreibt, die als Richtlinie zum systematischen Aufbau und zum Betrieb einer durchgängig abgestimmten IT-Servicestruktur benutzt werden kann [Ol04].

Die Bedeutung der Prozessorganisation von Unternehmen wird in den kommenden Jahren nicht nur erhalten bleiben, sondern größer werden. Denn die Gründe, die zu einer intensiveren Auseinandersetzung mit den Prozessen geführt haben, werden nicht wegfallen, sondern sich eher noch verstärken [Wi07]. Diese Arbeit beschäftigt sich mit dem Problem, dass solche etablierten und standardisierten Prozesse fehlerhaft sein können oder geändert werden müssen, weil sich Umweltbedingungen geändert haben. Unerwartete Fehler können immer in Prozessen auftreten, unabhängig davon, ob es ein selbst erstellter oder übernommener Prozess ist. Der hier vorgestellte Forschungsansatz unterstützt den Prozessmodellierer, indem Prozesse automatisch von einem System sowohl zur Laufzeit als auch während der Modellierung geändert werden können. Dazu wird eine Methode aus einem Teilgebiet der künstlichen Intelligenz verwendet, die später genauer erklärt wird.

1.2 Grundbegriffe

Um die nächsten Kapitel und insbesondere die technologischen Grundlagen dieser Arbeit nachvollziehen zu können, ist es notwendig einige Begriffe zu verstehen. Die Abarbeitung von Prozessen wird dazu erläutert, beginnend auf einer eher abstrakten Beschreibungsebene bis hin zur technischen Realisierung. Die abstrakte Beschreibungsebene ist für die betriebswirtschaftliche Sicht interessant, da sie der Führung eines Unternehmens hilft, in einem ersten Schritt die Prozesse zu begreifen und darauf aufbauend Entscheidungen zu treffen. Auf der anderen Seite ist das Verständnis für die technische Realisierung und Abarbeitung eines Prozesses unerlässlich, um den Forschungsansatz in dieser Arbeit zu begreifen und analysierte Probleme zu verstehen. Neben der Prozessthematik und ihrer technischen Realisierung wird der Begriff des Erfahrungsmanagements näher beschrieben, um ihn in den darauf folgenden technologischen Grundlagen fassbarer zu machen.

1.2.1 Business-Prozess / Geschäftsprozess

Ein Prozess, der sich auf ein Unternehmen und dessen Geschäftsziele bezieht, wird als ***Business-Prozess*** (Synonym zu ***Geschäftsprozess***) bezeichnet. In der Literatur herrscht keine einheitliche Definition des Begriffes Geschäftsprozess [RvHS04]. So definieren Aalst und Hee Kees einen Geschäftsprozess als:

„Ein Geschäftsprozess ist ein Prozess, der auf die Produktion bestimmter Produkte abzielt. Diese Produkte können physischer Natur oder auch weniger fassbar sein, wie ein Design, ein Rezept oder eine Bewertung. Mit anderen Worten, das Produkt kann ebenso ein Service sein.“ [vdAvHK04, S.346, eigene Übersetzung]

Aalst und Hee Kees erwähnen in ihrer Definition bereits explizit, dass sich ein Prozess auch auf immaterielle Güter und Produkte außerhalb eines betrieblichen Kontextes beziehen kann. So ist der Entwurf eines Designs oder das Kochen eines Rezeptes vielmehr ein kreativer Akt, dessen Rahmenablauf prozedural beschrieben werden kann. Die tatsächliche Ausführung des Prozesses weicht jedoch häufig vom ursprünglichen Prozess ab. Eine etwas andere Sicht auf einen Geschäftsprozess wird von der Workflow Management Coalition[3] (WfMC) getroffen. Sie widerspricht nicht der Definition von Aalst und Hee Kees und kann als eine Erweiterung ihrer Definition angesehen werden. Sie erweitert die Definition von Aalst und Hee Kees um eine Organisationsstruktur, die für eine Arbeitsteilung im Prozess notwendig ist:

Ein Geschäftsprozess ist „eine Menge von einer oder mehreren Prozeduren oder Aktivitäten, die gemeinsam ein Geschäftsziel oder eine Geschäftsstrategie verwirklichen. Normalerweise geschieht dies im Kontext einer Organisationsstruktur, in der funktionale Rollen und Beziehungen definiert werden müssen.“ [Fi07, S.18, eigene Übersetzung]

1.2.2 Workflows

„Bei einem Workflow handelt es sich um die automatische Abarbeitung – im Ganzen oder in Teilen – eines Business-Prozesses. Während dieser Abarbeitung werden Dokumente, Informationen oder Aufgaben von einem Teilnehmer an einen anderen Teilnehmer weitergereicht, um Aktionen auszuführen, die einer Menge prozeduraler Regeln entsprechen.“ [Fi07, S.18, eigene Übersetzung]

Ein ***Workflow*** kann also als Pendant zum Business-Prozess auf der technischen Ebene verstanden werden, in der ein (Business-)Prozess maschinell ausgeführt wird. Die

[3] Die Vereinigung wurde 1993 gegründet und hat sich zum Ziel gesetzt ein Referenzmodell für Workflow Management Systeme zu entwickeln, sodass Module unterschiedlicher Hersteller, wie z.B. Bedienoberflächen für Modellierungswerkzeuge, miteinander verknüpft und betrieben werden können. Zu den bekanntesten Mitgliedern gehören unter anderem Fujitsu und NEC.

Abarbeitung wird ermöglicht, in dem ein vorgegebenes (Teil-)Ziel aus einem Prozess in einzelne, logisch abgeschlossene Arbeitsschritte zerlegt wird. Diese abgeschlossenen Arbeitseinheiten werden als ***Tasks*** bezeichnet. Handelt es sich bei einem Workflow um einen ausgeführten Prozess im Sinne von Aalst und Hee Kees, so lassen sich unterschiedliche Alltagssituationen finden, die Merkmale von Prozessen aufweisen. Kochen ist sicherlich nicht das typischste Beispiel für einen Prozess, trotzdem weißt es alle Merkmale eines Prozesses auf: Unterschiedliche Aufgaben/Tasks müssen in einer bestimmten Reihenfolge abgearbeitet werden, damit ein bestimmtes Ziel, die Speise, erreicht werden kann. Modellierte Beispiele aus der Kochdomäne werden später wegen ihrer eigenen Komplexität genauer betrachtet werden.

1.2.3 Workflow Management System (WfMS)

Ein WfMS ist von der Workflow Management Coalition (WfMC) definiert als „ein System, das die Ausführung von Workflows definiert, erzeugt und kontrolliert. Die dabei benutzte Software läuft auf einer oder mehreren Workflow Engines, die in der Lage sind Prozess-Definitionen zu interpretieren, mit den Benutzern des Workflow-Systems zu interagieren und wo notwendig auch IT-Tools und Anwendungen aufzurufen.“ [WFMC08, S.9, eigene Übersetzung]

Ein ***Workflow Management System*** lässt sich nach obiger Definition in drei funktionale Bestandteile aufteilen [Mü05]:

- Die „build time“ - Definieren des Workflows.
- Die “run time process control” - Erzeugen des Workflows.
- Die “run time activity interaction” - Kontrollieren des Workflows.

Im klassischen Workflow Management gehört die Trennung zwischen „build time“ und „run time“ zu den wesentlichen Bestandteilen [We07]. Während der „***build time***“ eines Workflows wird ein Workflow-Modell vollständig spezifiziert. Normalerweise wird dabei ein grafisches Tool zur Modellierung genutzt. Workflow-Modelle – im Folgenden ***Workflow Definitionen*** genannt – sind die Vorlage für die Ausführung eines Business-Prozesses im WfMS. Workflow Definitionen müssen im Einklang mit dem Business-Prozess sein und erweitern diesen um technische Informationen, die notwendig sind, um den Workflow später auszuführen. Ist die Workflow Definition erstellt und entspricht sie dem Anforderungsprofil des Geschäftsprozesses, so ist die „build time“ eines Workflows abgeschlossen. Die so erstellten Workflow Definitionen können geordnet abgespeichert werden, um später z. B. Projekten zugeordnet zu werden.

Sobald eine Workflow Definition gestartet wird, erzeugt das Workflow Management System von ihr eine Kopie, die als ***Workflow Instanz*** bezeichnet wird. Mit Anlegen der

Workflow Instanz beginnt die „***run time process control***“-Phase (kurz „***run time***“). Die Workflow Definition dient also als Vorlage, von der es mehrere laufende Instanzen geben kann. Die Workflow-Instanz ist eine spezifische interne Repräsentation, die das WfMS nutzt, um den Workflow auszuführen [Mü05].

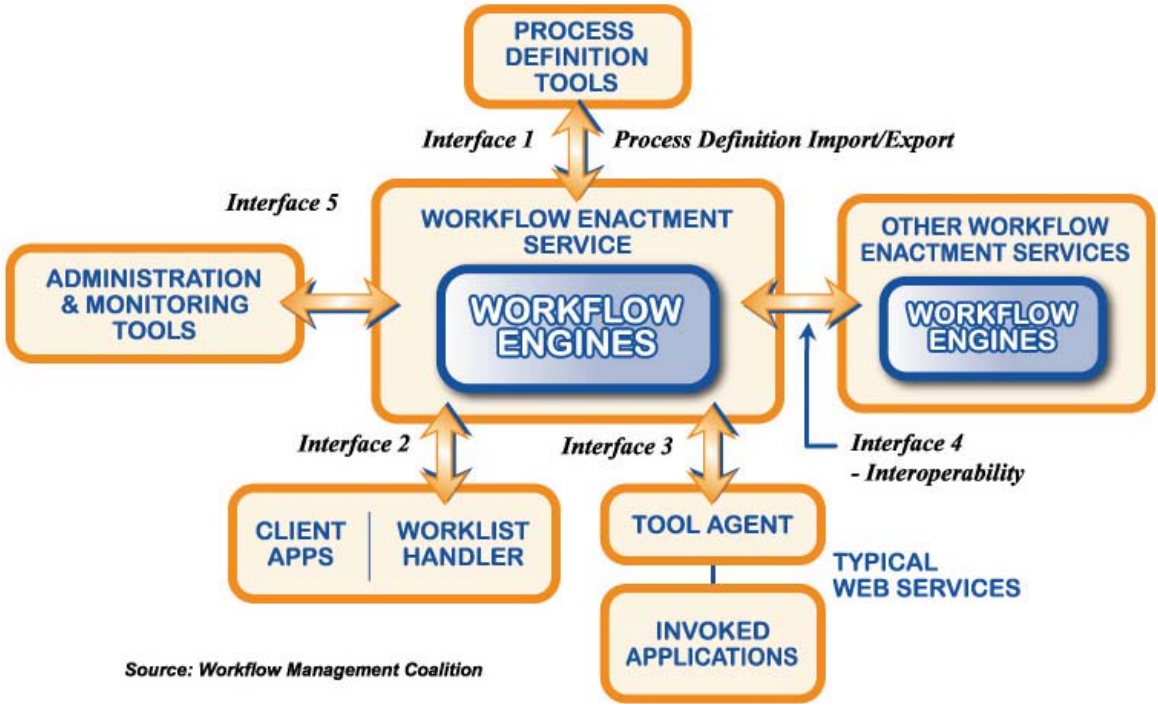


Abbildung 1 - Das WfMC Referenzmodell

Die „***run time activity interaction***“ sorgt dafür, dass Benutzer und System während der Laufzeit von Instanzen miteinander interagieren können [Mü05].

Für die Entwicklung eines Workflow Management Systems existiert ein Referenzmodell (siehe Abbildung 1), das von der WfMC entworfen wurde. Es umfasst dabei fünf Komponenten, von denen vier für diese Arbeit als wesentlich angesehen werden können:

- Tools zur Modellierung von Workflow Definitionen (Interface 1).
- Monitoringtools um aktive Workflows zu untersuchen und ggf. Fehler aufzuspüren (Interface 5).
- Der „Worklist Handler“, der Aufgaben automatisch an Mitarbeiter delegiert und das Fortschreiten des Workflows sicherstellt (Interface 2).
- Die Kernkomponente „Workflow Enactment Service“.

Diese Arbeit beschäftigt sich in erster Linie mit Workflows, in denen Menschen miteinander arbeiten und interagieren. Aus diesem Grunde steht das Aufrufen externer Anwendungen (Interface 3) nicht im Mittelpunkt. Ist in späteren Kapiteln die Rede von Tasks, so kann davon ausgegangen werden, dass es sich um Tasks handelt, die von Menschen ausgeführt werden müssen. Das WfMS interagiert über seine Schnittstellen/Interfaces permanent mit unterschiedlichen Nutzergruppen. Personen, deren Sicht durch das WfMS darauf beschränkt ist, dass sie neue Aufgaben zu verrichten haben, werden als ***Workflowteilnehmer*** bezeichnet. Dabei handelt es sich in der Regel um normale Mitarbeiter innerhalb einer Organisationsstruktur. Um den Mitarbeitern Aufgaben zuzuordnen, wird der sog. „Worklist Handler“ (Interface 2) eingesetzt. Im einfachsten Fall wird ein Mitarbeiter per E-Mail darüber informiert, dass eine neue Aufgabe ansteht. Komplexere „Worklist Handler“ können als Browseranwendung entworfen werden, die benutzerspezifische To-do-Listen generieren (siehe Abschnitt 3.2.1). Neben den Workflowteilnehmern existiert noch eine weitere Benutzergruppe innerhalb des WfMS. Diese Gruppe beschäftigt sich mit der Modellierung von Workflows und wird aus diesem Grunde auch als ***Workflowmodellierer*** bezeichnet. Die Aufgabe des Workflowmodellierers besteht darin Prozesse seiner Umwelt, z. B. in einem Unternehmen, zu erkennen und diese mittels eines Workflows zu formalisieren. Der Workflowmodellierer sieht die Abarbeitung des Workflows aus einer anderen Perspektive (Interface 1) als der Workflowteilnehmer. Während der Workflowteilnehmer nur die Aufgaben sieht, die er durchführen soll, besitzt der Workflowmodellierer das Wissen über den gesamten Workflow, das z. B. den Abarbeitungsstand des Workflows oder aufgetretene Probleme umfasst. Diese abstraktere Sicht auf den ganzen Workflow ermöglicht es dem Workflowmodellierer auch administrative Tätigkeiten auszuüben, die durch die Schnittstelle zu Monitoringtools (Interface 5) unterstützt werden.

Das Herzstück eines WfMS ist der ***„Enactment Service[4]“***. Er kümmert sich um das Erzeugen und Starten von Workflows. Er beinhaltet die Logik, wie der Workflow auszuführen ist (siehe Abschnitt 1.2.4) und sorgt während der Ausführungsphase dafür, dass sowohl die Konsistenz als auch die Persistenz des Workflows gewahrt bleibt. Um die Persistenz zu wahren, muss der „Enactment Service“ garantieren, dass keine Daten während eines Stromausfalls verloren gehen. Ein Mangel an Persistenz würde zu inkonsistenten Workflows führen. Ein Beispiel für Inkonsistenz in Workflows wären strukturelle Mängel, die Teile des Workflows unerreichbar machen.

[4] Synonym zu „Process Execution Environment“ [WFMC08]

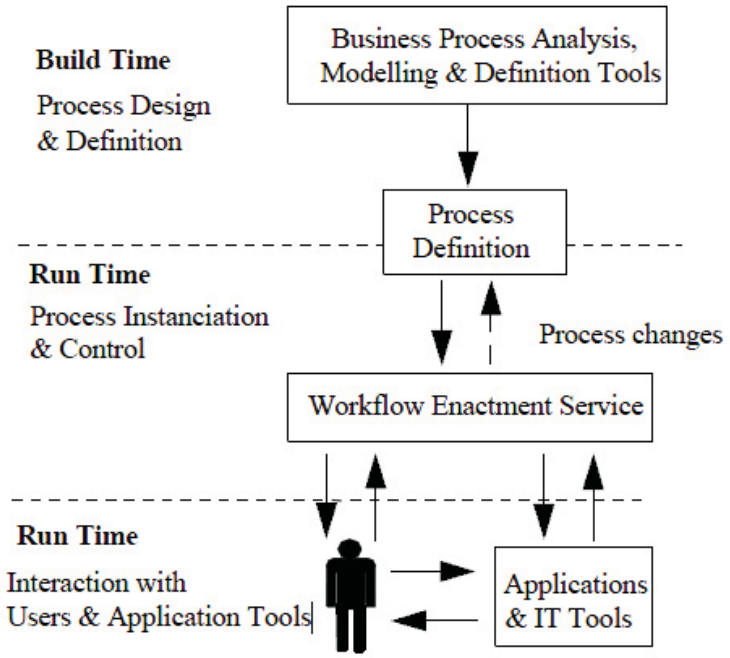


Abbildung 2 - Komponenten eines WfMS und ihre funktionale Bedeutung (In Anlehnung an [WFMC08])

Es ist nun ersichtlich, wie die vier vorgestellten Komponenten des Referenzmodells die drei funktionalen Bestandteile eines WfMS unterstützen (siehe Abbildung 2). Während der „build time“ setzt sich der Workflowmodellierer mit der Analyse eines Prozesses (Business Process Analysis) auseinander und nutzt dafür ein Tool zur Modellierung von Workflow Definitionen (Modelling & Definition Tools). Die erstellte Workflow Definition wird dann während der „run time“ vom „Enactment Service“ gestartet und gesteuert. Treten Probleme während der Ausführung des Workflows auf, so sollte es möglich sein den Workflow zu stoppen und wieder zurück in die „build time“ zu bringen. Dass dies bei heutigen WfMS in der Regel noch nicht möglich ist, ist in Abbildung 2 durch einen gestrichelten Pfeil zwischen „Enactment Service“ und „Process Definition“ symbolisiert. In Abschnitt 1.3.1 werden Techniken vorgestellt, die diese Rückkopplung ermöglichen. Die Interaktion des „Enactment Service“ mit den Workflow-teilnehmern und Workflowmodellierern führt zur „run time activity interaction“, die den „Enactment Service“ kontinuierlich mit Steuerungssignalen wie „Aufgabe beendet“ oder „Aufgabe kann nicht erfüllt werden“ versorgt. Dabei können die Workflowteilnehmer zur Erfüllung ihrer Tätigkeiten auch auf externe Programme (Applications & IT Tools) zugreifen. Falls ein WfMS direkt über einen Workflow mit externen Anwendungen (Interface 3) kommuniziert, wären die externen Programme ebenso direkter Bestandteil der „run time activity interaction“.

1.2.4 Abarbeitung und Steuerung von Workflows

Workflows können auf verschiedenartige Weise dargestellt werden. Die Repräsentation basiert auf der ***Modellierungssprache***, die das Tool zur Workflow-modellierung bereitstellt. Die Formulierung des Workflows muss dabei nicht in graphischer (prozeduraler) Form geschehen, sondern kann auch deklarativ beschrieben werden [vdAP06]. Die deklarative Beschreibung eines Workflows ist allerdings ungeeignet, wenn Workflows kommunizierbar bleiben sollen. Gängige Standards zur Modellierung von Workflows (z. B. BPMN [5]) sind graphorientiert. Daher wird die Funktionsweise des „Enactment Service" in dieser Arbeit auf prozedurale Beschreibungen beschränkt. Um einen Workflow als Graph zu beschreiben, existieren mehrere Möglichkeiten. Eine der ersten Formen, die auch eine maschinelle Verarbeitung ermöglicht, geht auf Petri-Netze zurück [vdA98]. Der Übergang in einem Petri-Netz von einem Zustand in einen anderen wird dabei als „Feuern" (im Englischen als „enable") bezeichnet [WP08]. Für die Konstruktion heutiger Workflows existieren zwei graph-basierte Modellierungsansätze, die sich im Wesentlichen dadurch unterscheiden was ein „Feuern" und somit einen neuen Zustand des Workflows auslöst. Das sind zum einen datengetriebene („data driven") Workflows und zum anderen Kontrollfluss getriebene („control flow driven") Workflows [TDG07].

In einem ***datengetriebenen Workflow*** spezifizieren die Verbindungen zwischen Tasks die Abhängigkeit von Daten. Eine eingehende Verbindung bedeutet, dass der Task Daten benötigt, die von einem anderen Task produziert werden. Diese Abhängigkeit von Daten zur Zustandsänderung eines Workflows erhöht die Nebenläufigkeit in einem Workflow. Die Ausführung des Workflow stoppt erst, wenn es einen Task gibt, der aufgrund einer nicht erfüllten Abhängigkeit blockiert wird und es keinen anderen Task mehr gibt, der ausgeführt werden kann [TDG07]. Datengetriebene Workflows kommen häufig in Anwendungsfeldern zum Einsatz, in denen digitale Eingangssignale verbunden werden, um Geräte zu testen [TDG07]. Ein populäres neues Forschungsfeld, in dem häufig datengetriebene Workflows genutzt werden, ist e-Science [TDG07]. E-Science versucht die Forschungsgemeinde dahingehend zu unterstützen, dass Experimente standardisiert ablaufen und wiederholt werden können. Ein Beispiel für einen e-Science Workflow wäre ein kompliziertes Experiment zur Herstellung eines neuen Medikaments, an dem viele Personen über einen langen Zeitraum beschäftigt sind. Es ist möglich, dass einige chemische (Teil-)Produkte unabhängig voneinander hergestellt werden können, während andere Forscher ihre Arbeit erst erledigen können, wenn bestimmte Zwischenprodukte erzeugt wurden. Eine datengetriebene Workflowmodellierung

[5] BPMN steht für Business Process Modeling Notation und wurde von der Business Process Management Initiative entwickelt. Der Standard ist aktuell implementiert unter anderem in Tools von Borland, Fujitsu, IDS-Scheer und SAP [OMG]

verkürzt die Dauer des Experiments, da viele Forscher Teile des Experiments gleichzeitig ausführen können, ohne dass dabei Abhängigkeiten verletzt werden.

Wird ein Workflow von einem ***Kontrollfluss*** gesteuert, so werden Tasks im einfachsten Fall sequenziell abgearbeitet. Um komplexere Prozesse abbilden zu können, verfügen Modellierungssprachen, die auf einem Kontrollfluss basieren, über Konstrukte, um die Abarbeitungsreihenfolge festzulegen. Aalst untersucht in seiner Arbeit über Workflow Patterns [vdA03] Kontrollstrukturen, die genutzt werden können, um einen Kontrollfluss zu steuern. Drei dieser Patterns/Kontrollstrukturen sind in Abbildung 3 in Form von BPMN dargestellt. Das Loop-Konstrukt erlaubt es einen Subgraph des Workflows iterativ auszuführen, bis ein Abbruchkriterium die Iteration beendet. Das AND-Konstrukt verzweigt den Kontrollfluss und erlaubt damit eine parallele Verarbeitung von Aufgaben. Nach Abarbeitung aller Tasks im AND-Konstrukt wird eine Synchronisation erzwungen, sodass der Kontrollfluss wieder sequenziell weiterlaufen kann. Das XOR-Konstrukt verzweigt den Kontrollfluss genauso wie das AND-Konstrukt mit dem Unterschied, dass nur ein Zweig ausgeführt wird. Die Auswahl des Zweiges geschieht durch eine zuvor definierte Vorbedingung.

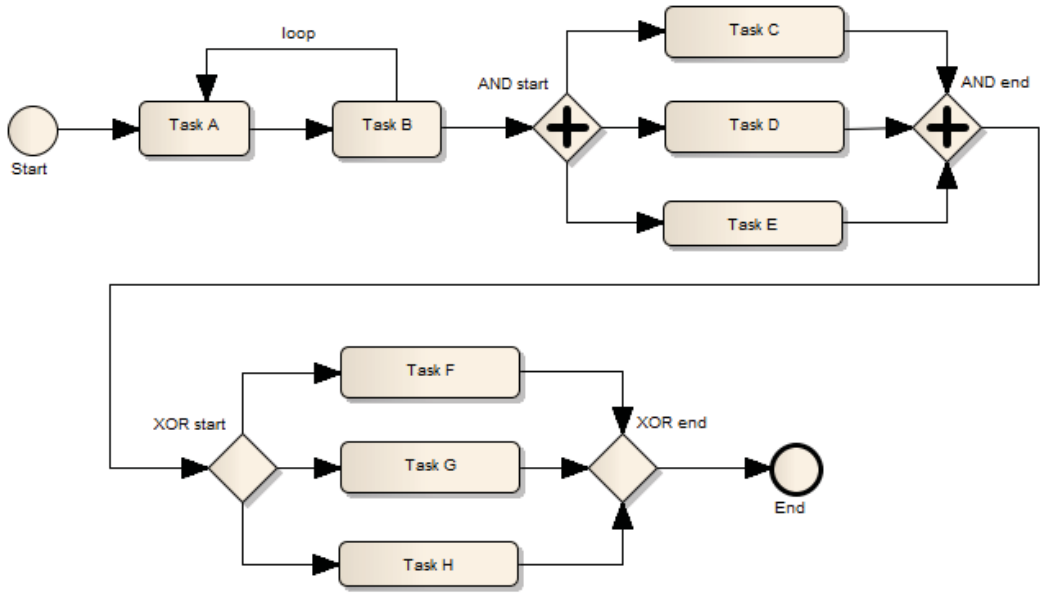


Abbildung 3 - Beispielworkflow in BPMN, Quelle: eigene Erstellung

Der Workflow in Abbildung 3 ermöglicht eine iterative Ausführung von Task A und Task B. Nach Beendigung des Loops werden Task C, D und E parallel ausgeführt und der Workflow verlässt das AND erst, wenn Task C, D und E beendet wurden. Danach wird im XOR gewählt, ob Task F, G oder H ausgeführt werden soll. Obwohl Aalst in [vdA03] noch weitere Kontrollstrukturen nennt, die eine größere Ausdrucksmächtigkeit ermöglichen

würden, lässt sich mit Hilfe dieser drei Kontrollstrukturen normalerweise jeder Geschäftsprozess ausdrücken [HJM04][6]. Dabei handelt es sich jedoch um Konstrukte, die eher in Spezialfällen nützlich wären, als in allgemeinen Anwendungsgebieten. Im Folgenden werden alle Elemente in einem Kontrollfluss als ***Workflowelemente*** bezeichnet. Soll die Modellierungssprache kommunizierbar bleiben, muss ein Kompromiss zwischen Ausdrucksmächtigkeit und Komplexität der Modellierungsprache gefunden werden.

Diese zwei Typen zur Abarbeitung und Steuerung von Workflows sind zueinander ähnlich, da sie eine Interaktion zwischen zwei individuellen Aktivitäten definieren, sie unterscheiden sich aber in der Art und Weise, wie diese Interaktion implementiert ist. In Kontrollfluss getriebenen Workflows bedeutet die Verbindung zwischen zwei Aktivitäten, dass die nachfolgende Aktivität erst ausgeführt werden kann, wenn die vorherige Aktivität beendet ist. Datengetriebene Workflows werden eingesetzt, wenn Daten die Reihenfolge der Abarbeitung bestimmen. Die Abhängigkeiten von Aktivitäten werden hier durch den Datenfluss zwischen Produzent und Verbraucher der Daten beschrieben. Die erhöhte Nebenläufigkeit in datengetriebenen Workflows erfordert jedoch auch ausgeklügelte Transaktionskonzepte, die die Konsistenz des Workflows sicherstellen. Von großer Bedeutung ist hier die Abwägung zwischen der Wahrung der Flexibilität der Ausführung und notwendigem Einschränken durch Transaktionen [SW02].

1.2.5 Korrektheit

Bei der Modellierung kleiner Workflows, die nur aus wenigen Tasks bestehen, ist es recht einfach Fehler aufzuspüren. Mit wachsender Komplexität eines Workflows, steigendem Synchronisationsaufwand und der Anzahl der Bedingungen in einem Workflow wird die Verifikation eines Workflows immer schwieriger [Sa97]. Die Fehlerfreiheit und somit Korrektheit eines Workflows lässt sich dabei in syntaktische und semantische Korrektheit unterteilen. Um die ***syntaktische Korrektheit*** eines Workflows zu überprüfen, bedarf es keiner Metainformationen über die Tasks oder ihrer Kontrollstrukturen. In [Sa97] werden fünf Typen syntaktischer Fehler klassifiziert:

- „Incorrect Usage“, z. B. wenn ein AND nur eine Sequenz beinhaltet, die synchronisiert werden soll.
- „Deadlocks“, z. B. wenn eine Synchronisation unmöglich wird, da die eingehenden Sequenzen sich gegenseitig ausschließen.
- „Livelocks“, z. B. wenn ein Loop keine Abbruchbedingung beinhaltet.

[6] In [HJM04] wird statt von einem XOR-Pattern von einem Preference-OR-Pattern gesprochen, das semantisch jedoch das Gleiche meint.

- „Unintentional Multiple Execution“, z. B. wenn es erlaubt ist, dass ein Task mehrere eingehende Kanten besitzt. Ohne Synchronisation wird dieser Task womöglich unbeabsichtigt mehrmals gestartet.
- „Active Termination“, z. B. wenn mehrere Endpunkte eines Workflows existieren. Das Erreichen eines Endpunktes würde die Beendigung des Workflows bedeuten, obwohl parallele Sequenzen noch aktiv sein könnten.

In Abschnitt 3.2.3 wird beschrieben wie die CAKE-Modellierungssprache mit diesen syntaktischen Fehlern umgeht.

Semantische Fehler entstehen, wenn bereits der Business-Prozess, der die Grundlage für den Workflow darstellt, fehlerhaft ist [Sa97]. Um die semantische Korrektheit eines Workflows zu überprüfen, werden domänenspezifische Metainformationen benötigt, um spezifizierte Tasks und Bedingungen inhaltlich zu verstehen. Eine syntaktische Korrektheit garantiert noch keine semantische Korrektheit. Ein Beispiel: Es existiert ein XOR mit der Auswertungsvariablen *x*. Das XOR besitzt zwei ausgehende Sequenzen. Sequenz A wird ausgeführt, wenn *x* den Wert 1 annimmt, Sequenz B wird ausgeführt, wenn *x* den Wert 2 annimmt. Die Bedingung im XOR mag syntaktisch korrekt sein, aber falls der Modellierer nicht bedacht hat, dass *x* auch den Wert 3 annehmen kann, so führt dieser semantische Fehler ebenfalls zu einem Deadlock und der Workflow wird nie einen Endzustand erreichen.

1.2.6 Erfahrungsmanagement

Der menschliche Problemlösungsprozess basiert im Wesentlichen auf strategischem Denken, dessen Grundlage Erfahrungen sind. Insbesondere erfahrene Personen, zu denen natürlich auch Experten gehören, sind in der Lage Zusammenhänge zwischen Vergangenem und Ereignissen der Gegenwart herzustellen [Gö09]. Ein Experte zeichnet sich durch umfangreiches Wissen, erfolgreiches Vorgehen beim Erkennen und Bearbeiten von Problemen, Effizienz, Fehlerfreiheit und Flexibilität gegenüber neuen Problemsituationen aus [Hu02]. Es ist also die Expertise in einer Organisation, die als eine der wertvollsten Ressourcen genutzt werden kann, damit ein Unternehmen erfolgreich ist. Aus diesem Grunde setzen fast alle großen Unternehmen IT-gestützte Systeme ein, um das erlangte Wissen zu finden, zu erforschen und festzuhalten. Dies wird als Wissensmanagement bezeichnet [Be02]. Erfahrungsmanagement ist eine besondere Form des Wissensmanagements, die sich darauf beschränkt, spezifisches Wissen aus einem einzelnen Problemfall zu verwalten. Erfahrungsmanagement bezieht sich auf das Sammeln, Modellieren, Speichern, Wiederverwenden, Evaluieren und Bewahren von Erfahrungen [Be02].

Die Trennung zwischen allgemeinem Wissen und Erfahrungswissen ist nicht immer klar zu ziehen und hängt wohl auch mit der Frage zusammen, wie stark der Kontextbezug von Erfahrungen als allgemeines Wissen angesehen wird. So schreiben Althoff et al. In [AlthoffEtAl01], dass der Hauptunterschied zwischen der Anwendung von allgemeinem Wissen und Erfahrungswissen darin besteht, dass Erfahrungswissen dort zum Einsatz kommt, wo kontinuierlich Wissen benötigt wird. Der benötigte, kontinuierliche Wissensstrom wäre dann also so groß und mit wechselnden Anforderungen, dass er sich gar nicht als allgemeines Wissen formalisieren ließe. Minor in [Mi06] meint dagegen, dass Erfahrungswissen einen Gegensatz zu allgemeinem, regelbehaftetem Wissen darstellt, da Erfahrungswissen durch seinen Kontextbezug problemspezifischer ist.

Für die Leistungsfähigkeit einer intelligenten Organisation, die einen Bedarf nach einem problemspezifischen und kontinuierlichen Wissensstrom hat, ist es von besonderem Interesse, ob die Strukturiertheit von verfügbarem und gespeichertem organisationalem Wissen die Fähigkeit zum Lösen spezifischer Problemstellungen erhöht und damit zur Steigerung der Leistungsfähigkeit der Organisation beitragen kann [Hu02]. In Abschnitt 1.3.2 wird eine Technik vorgestellt, die sich in der Praxis bewährt hat, um Erfahrungswissen zu verwalten.

1.3 Technologische Grundlagen

1.3.1 Agilität in Workflows

Agilität, d. h. Flexibilität, in Workflows ermöglicht die Anpassung von laufenden Workflows. Der Bedarf nach flexiblen Workflows geht einher mit dem Wandel der Unternehmenslandschaft hin zur Tertiärisierung und dem Bedarf nach prozess-orientierten Organisationen [Wi07]. Es ist immer öfter erforderlich, dass Prozesse auch flexibel sein können. Die Notwendigkeit einen laufenden Prozess zu ändern, tritt vor allem bei Prozessen auf, die über einen langen Zeitraum (Wochen oder Monate) laufen. Diese sog. ***„long-term" Workflows*** unterliegen häufig externen Einflüssen. So erfordert der Rechtsstreit um die Vorratsdatenspeicherung von Internet Service Provider wie der Telekom eine ständige Anpassung ihrer Prozesse, da sie sich an geltendes Recht anpassen müssen. Neben sich ändernden Rahmenbedingungen können ebenfalls unvorhersehbare Störfälle eintreten. In Krankenhäusern werden Patienten nach genormten Workflows, den klinischen Pfaden, behandelt. Diese klinischen Pfade werden für eine Kostenträgerrechnung herangezogen. Bleibt man bei klinischen Aufgaben-stellungen, so stelle man sich vor, ein Antibiotikum wirke nicht aufgrund einer Resistenz

des Bakterienstammes. Dann muss der Workflow angepasst werden und bspw. ein anderes Antibiotikum verabreicht werden. Die Ursache solcher Änderungen, die die Anpassung eines Workflows erforderlich machen, wird im Folgenden als „***Exception***“ bezeichnet. Der Begriff soll dabei nicht als „Exception“ im Sinne eines Fehlers verstanden werden, wie ihn Russell et al. [RvdAH06] verwendet. Hier soll als Exception jeder Umstand gemeint sein, der es erforderlich macht eine laufende Instanz zu ändern.

Der Grad an Agilität eines WfMS kann daran abgeschätzt werden, welche Agilitätstypen unterstützt werden. Für Workflow Instanzen, die von einer Workflow Definition abgeleitet wurden, kann Agilität in drei unterschiedliche Arten klassifiziert werden [MSB08]:

- Ad-hoc Änderungen.
- „Schema Evolution“.
- Eine hierarchische Dekomposition mittels „Late Modelling“ oder „Late Binding“.

Die ***hierarchische Dekomposition*** [MSB08] eines Workflows erlaubt es, dass von einer Workflow Definition bereits Instanzen abgeleitet werden können, ohne dass die Workflow Definition dabei vollständig spezifiziert sein müsste. Dazu werden in der Definition Platzhalter eingefügt, die zu einem späteren Zeitpunkt mit einem Sub-Workflow gefüllt werden können. „***Late Binding***“ bedeutet in diesem Zusammenhang, dass in dem Platzhalter zur „build time“ mehrere Sub-Workflows hinterlegt werden, von denen einer zur Laufzeit ausgewählt werden kann. „***Late Modelling***“ geht noch einen Schritt weiter und erlaubt es den Platzhalter auch während der „build time“ leer zu halten, wodurch eine freie Modellierung zur Laufzeit ermöglicht wird. Erst wenn der Platzhalter ausgeführt werden soll, wird eine situationsgerechte Spezifikation notwendig, ohne dabei aus einer vorgegebenen Menge aus Workflowfragmenten wählen zu müssen. Der Begriff der hierarchischen Dekomposition verdeutlicht, dass die Tasks in einem Workflow für Subgraphen stehen können und somit ein hierarchisches Geflecht entsteht.

Schema Evolution [RRD03] erlaubt die Änderung von Workflow Instanzen über eine Anpassung der Workflow Definition, von der sie abgeleitet wurden. Änderungen an einer Workflow Definition führen dazu, dass die vorgenommene Korrektur an alle abgeleiteten Workflow Instanzen propagiert wird. Dieser Mechanismus muss dafür sorgen, dass keine Instanzen geändert werden, deren Ausführungsstatus eine Modifikation unmöglich macht oder die durch Ad-hoc Änderungen so abgewandelt wurden, dass die Änderung an der Definition die syntaktische Korrektheit der Instanz verletzten würde.

Ad-hoc Änderungen [RRD03] an einem Workflow erfordern ein Höchstmaß an Agilität. Ein typisches Beispiel für einen Businessprozess wäre z. B. das Ändern eines

Zulieferers innerhalb eines Supply-Chain Prozesses, falls der ursprüngliche Zulieferer Lieferengpässe hat. Ad-hoc Änderungen sind in den meisten Anwendungsgebieten nicht planbar und umfassen jede Form der Anpassung. Dazu zählen das Hinzufügen, Ändern und Löschen von Workflow Elementen und auch Änderungen am Datenfluss zwischen Tasks. Für ein WfMS bedeutet dies, dass es Methoden bereitstellen muss, die die Ausführung eines Workflows nicht unterbrechen während ein bestimmter Bereich des Workflows geändert wird. Das WfMS muss dafür Sorge tragen, dass zu jedem Zeitpunkt der Ad-hoc Änderung der Workflow ausführbar und konsistent bleibt. Auf technischer Ebene muss dafür ein Mechanismus existieren, dessen Komplexität vor den Benutzern des Systems versteckt wird.

1.3.2 Fallbasiertes Schließen

Fallbasiertes Schließen [Be02] (engl. ***Case-based reasoning*** (***CBR***)) ist ein Untergebiet der wissensbasierten Systeme[7], das sich darauf konzentriert, Probleme mit bereits gemachten Erfahrungen zu lösen. Zur Umsetzung dieses Ansatzes existieren Methoden und Techniken, die es ermöglichen, Erfahrungen zu erfassen, zu speichern, zu finden und an neue Probleme anzupassen. Fallbasiertes Schließen liefert eine vielfältige Menge an Techniken, die im Erfahrungsmanagement verwendet werden können.

In Alltagssituationen hat jeder Mensch so einen Prozess schon durchlebt: Ein Problem oder eine Aufgabe ähnelt einer Situation in der Vergangenheit. Man erinnert sich an die damalige Lösung des Problems und man erkennt, dass die alte Lösung auch das neue Problem lösen könnte. Das folgende Beispiel sei aus dem Leben eines KFZ-Mechanikers gegriffen: Ein Mechaniker stellt fest, dass ein Fahrzeug (VW Golf TDI) unter einem Leistungsverlust leidet. Der Motor arbeitet über den kompletten Drehzahlbereich, aber der Turbolader setzt erst bei 3500 statt 2300 Umdrehungen/Minute ein. Plötzlich erinnert er sich, dass er vor längerer Zeit ein anderes Fahrzeug (Audi A4 TDI) mit fast gleichen Symptomen in der Werkstatt hatte. Damals führte ein Austausch des Luftmassenmessers zur Beseitigung der Probleme. Aufgrund dieser Erfahrung des Mechanikers wurde am VW Golf der Luftmassenmesser getauscht. Nach ausgiebigen Tests stellte sich heraus, dass das Problem am VW Golf tatsächlich behoben wurde durch den Austausch des Luftmassenmessers. Der Mechaniker konnte also die Lösung eines vergangenen Problems auf ein neues Übertragen.

Im CBR wird versucht durch Formalisierung und Erfassung von gemachten Erfahrungen, neu auftretende Probleme zu lösen. Gemachte Erfahrungen werden als eine Menge von Fällen abgespeichert. Ein ***Fall*** ist ein Paar bestehend aus dem eigentlichen Problem und der dazugehörigen Lösung.

[7] hier sind vor allem Expertensysteme gemeint

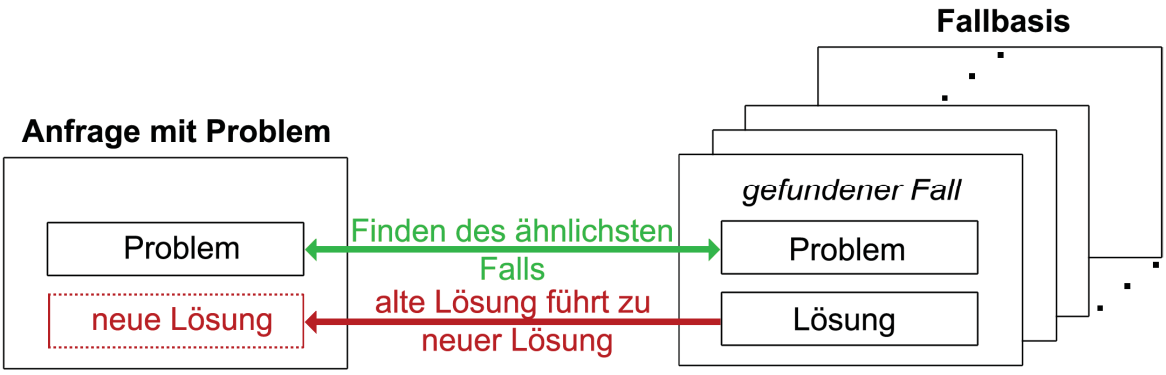


Abbildung 4- Grundprinzip des CBR (eigene Erstellung in Anlehnung an [Be02])

Das CBR-System wird üblicherweise mit einem neuen Problem angefragt. Daraufhin wird die Menge aller Fälle – die ***Fallbasis*** (siehe Abbildung 4) – nach einem Problem durchsucht, das dem aus der ***Anfrage*** ähnelt. Wird ein Fall gefunden, dessen Problembeschreibung eine genügend hohe Ähnlichkeit zu dem angefragten Problem aufweist, so ist die Hypothese, dass die Lösung in dem Fall auch eine hohe Nützlichkeit zum Lösen des aktuellen Problems bietet. Unter ***Ähnlichkeit*** wird dabei ein numerischer Wert verstanden, der zwischen dem Problem aus der Anfrage und dem Problem aus dem Fall berechnet werden kann. Die Grundlage für die Berechnung der Ähnlichkeit ist ein zuvor definiertes ***Ähnlichkeitsmaß***. Für einen Überblick oft genutzter Ähnlichkeitsmaße sei an dieser Stelle auf [Be02] verwiesen.

Der 4R-Zyklus

Der Prozess des fallbasierten Schließens kann als Zyklus beschrieben werden [Be02], dessen Bezeichnung den Anfangsbuchstaben seiner vier Phasen entstammt: In der „Retrieve“-Phase wird die Fallbasis nach einem Fall durchsucht, dessen Problemteil ähnlich ist zu dem aktuellen Problem. Danach wird der Lösungsteil, der aus dem gefundenen Fall stammt, in der „Reuse“-Phase wiederverwendet, um das aktuelle Problem zu lösen. Möglicherweise gibt es noch eine Adaptionsphase im „Reuse“-Schritt, um die gefundene Lösung an das aktuelle Problem anzupassen. Die (adaptierte) Lösung wird dem Benutzer präsentiert, der dann in der „Revise“-Phase die Lösung überprüft. Konnte durch die vorgeschlagene Lösung das aktuelle Problem gelöst werden, so wurde eine neue Erfahrung gemacht, die vom CBR-System während der "Retain"-Phase in Form eines neuen Falls abgespeichert werden kann.

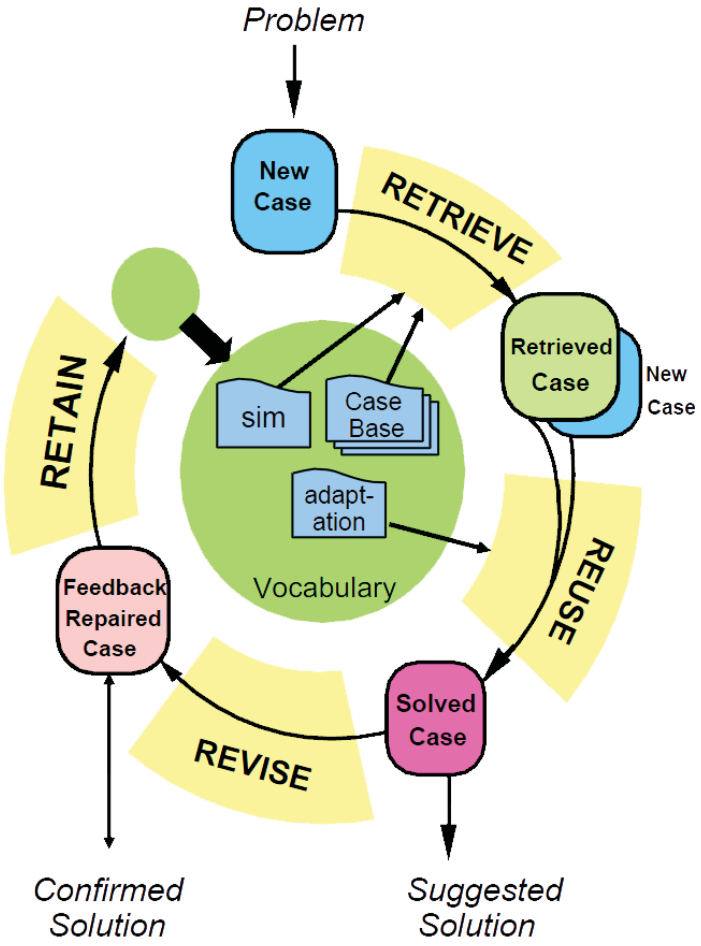


Abbildung 5 - Der 4R-Zyklus, Quelle: [Be02] [AP94][We07]

Retrieve

Der 4R-Zyklus beginnt, wenn der Benutzer eine Anfrage an das System schickt. Als Ergebnis erhält der Benutzer vergangene Fälle geliefert, die eine möglichst hohe Ähnlichkeit zu dem Problem aus der Anfrage aufweisen. Der Begriff der Ähnlichkeit nimmt hier eine zentrale Position ein, da dadurch eine unscharfe Suche ermöglicht wird, d. h. Anfrage und Fall können in einem zuvor definierten Maß voneinander abweichen. Die Annahme hierbei ist, dass ähnliche Probleme auch ähnliche Lösungen haben. Die Wahl eines passenden Ähnlichkeitsmaßes zwischen Anfrage und den Fällen in der Fallbasis ist das entscheidende Kriterium, um eine passende Problemlösung zu finden [Be02].

Reuse

Sobald ein oder mehrere Fälle gefunden wurden, werden die Lösungen aus diesen Fällen genommen, um das aktuelle Problem zu lösen. Typischerweise wird dies durch Adaption der gefundenen Lösung erreicht. Dabei unterscheidet man hauptsächlich zwischen drei Arten der Adaption [Be02]:

- Generative Lösungsanpassung („Generative Adaptation“)
- Kompositionelle Lösungsanpassung („Compositional Adaptation“)
- Transformationsbasierte Lösungsanpassung („Transformational Adaptation“)

Bei der ***generativen Lösungsanpassung*** gilt die Annahme, dass es einen wissensbasierten Problemlöser gibt, der bereits in der Lage ist, alle Probleme auch ohne Verwendung von Fällen zu lösen. Fälle werden dann nur noch genutzt, um Lösungen schneller zu finden oder um Lösungen zu finden, die ähnlich sind zu bekannten Lösungen. Der Problemlöser und die Erfahrung aus dem Fall arbeiten gemeinsam an der Lösung des Problems.

Die ***kompositionelle Lösungsanpassung*** zerlegt das Hauptproblem in Teilprobleme und versucht dabei ein Teilproblem nach dem anderen zu lösen. Die Reihenfolge, in der die Teilprobleme abgearbeitet werden, spielt dabei eine wesentliche Rolle, da das Resultat der Lösungsanpassung davon beeinflusst wird. Das CODAW System, das in Kapitel 2 vorgestellt wird, implementiert eine solche Lösungsanpassung durch eine Erweiterung der Fälle mit Prädikatenlogik.

Bei der ***transformationsbasierten Lösungsanpassung*** gibt es normalerweise eine feste Menge von Adaptionsregeln und Operatoren, nach denen eine Lösung angepasst werden darf [Be02]. Abhängig vom Grad der angewandten Adaption wird von einer Substitutionsadaption oder einer strukturellen Adaption gesprochen. Die Substitutions-adaption wird verwendet, wenn die Anpassung der Lösung recht einfach ist und nur weniger Änderungen bedarf. Der Begriff Substitution deutet bereits an, dass lediglich Werte in der Anfrage ausgetauscht werden. Die Struktur innerhalb der Anfrage wird dabei nicht verändert. Reicht eine bloße Abänderung von Werten nicht aus und muss die Anfrage strukturell verändert werden, so wird von einer strukturellen Adaption gesprochen. Eine strukturelle Änderung kann bspw. das Hinzufügen oder Entfernen eines Attributes sein. Sowohl die Substitutionsadaption als auch die strukturelle Adaption wenden eine vorgegebene Menge von Regeln und/oder Operatoren zur Lösungsanpassung an.

In dieser Arbeit wird eine transformationsbasierte Lösungsanpassung verwendet, die im Gegensatz zu den klassischen Verfahren keine feste Menge an Operatoren oder

Regeln kennt. Die Adaptationsoperatoren sind fallabhängig und somit direkt an Erfahrungswissen gebunden. Das Erfahrungswissen wird mittels eines Rekonstruktionsalgorithmus an die Anfrage angepasst. Aus diesem Grunde wird eine solche Adaption auch als derivative („***derivational analogy***“) Adaption bezeichnet [Be02] [Carbonell83]. Es wird dabei eine Brücke zur generativen Lösungsanpassung geschlagen mit dem Unterschied, dass es keinen konkreten wissensbasierten Problemlöser gibt. Es ist Aufgabe des Rekonstruktionsalgorithmus diese Lücke zur Anpassung des Erfahrungswissens zu schließen und die im Fall spezifizierten Operatoren auf die Anfrage zu übertragen.

Revise

In der „Revise“-Phase wird die Lösung nun vom Benutzer, der die Anfrage gestellt hat, hinsichtlich ihrer Nützlichkeit bewertet. Die Güte des Systems hängt dabei wesentlich davon ab, wie gut das Ähnlichkeitsmaß aus dem „Retrieve“-Schritt die Nützlichkeit approximiert hat. Kann die vorgeschlagene Lösung das Problem noch nicht adäquat lösen, so sollte es für den Benutzer möglich sein, die Lösung noch manuell anzupassen. Das setzt natürlich voraus, dass der Benutzer ein fundiertes Wissen über den Gegenstandsbereich besitzt.

Retain

Die „Retain“-Phase des Zyklus kann als Lernphase verstanden werden. Wenn eine Lösung nach dem „Revise“-Schritt vom Nutzer akzeptiert wird, so kann die angepasste Lösung zusammen mit dem Problem aus der ursprünglichen Anfrage als neuer Fall in der Fallbasis abgespeichert werden. Damit wird die Fallbasis um neues Wissen erweitert, das bei einer erneuten Anfrage zur Verfügung steht.

1.4 Ziel dieser Arbeit

Nach Einführung der Begriffe und der verwendeten Technologien kann das Ziel dieser Arbeit genauer beschrieben werden. Diese Arbeit beschäftigt sich mit der Konzipierung und Realisierung einer Softwarekomponente, die Workflows automatisch anpasst. Das hier vorgestellte System ermöglicht es für eine konkrete Problemstellung des Workflowmodellierers semantisch ähnliche Workflows zu finden, von denen Teile in einer neuen Workflow Definition wiederverwendet werden können. Das beschleunigt zum einen den Vorgang der Prozessmodellierung und zum anderen werden Fehler bei der Konstruktion vermieden, da das Expertenwissen des CBR-Systems genutzt werden

kann. Auf der Ausführungsebene eines Workflows geschehen wiederum oft unvorhergesehene Ereignisse: Bestimmte Bedingungen fallen weg oder kommen hinzu oder es kommt zu sog. Exceptions (vgl. Abschnitt 1.3.1) während der Ausführung des Workflows. In solchen Situationen muss die Workflow Instanz so abgeändert werden, dass sie den neuen Anforderungen genügt und die Konsistenz der Instanz weiter gewährleistet ist.

Zur Realisierung des Konzeptes wird eine Softwarekomponente als Bestandteil der CAKE III Architektur implementiert (siehe Kapitel 3). Der Adaptionsprozess an sich soll mittels CBR geschehen. Im Gegensatz zu klassischen CBR-Systemen wird hier jedoch ein innovativer Ansatz aus [MTSB08][MinorEtAl10a] verwendet, der vorschlägt, dass das Erfahrungswissen für die Adaption in Änderungsepisoden des Workflows erfasst wird und nicht an eine Menge fester Operatoren gebunden wird (vgl. Abschnitt 1.3.2). Als Bestandteil dieser Arbeit muss ein Konzept für eine Fallrepräsentation entwickelt werden, die in der Lage ist, Änderungsepisoden zu formalisieren und sie für eine Wiederverwendung nutzbar zu machen. Da sich die Änderungsepisoden auf Workflows beziehen, muss eine adäquate Repräsentation für Workflows gefunden werden, die nicht nur der Berechnung von Änderungen an Workflows dient, sondern auch ein Retrieval der Fälle ermöglicht. Für das Retrieval muss ein Ähnlichkeitsmaß erdacht werden, das die Ähnlichkeit zwischen Fällen ermittelt, deren Inhalte nicht durch Attribut-Wert-Paare, sondern durch Strukturen von Workflows bestimmt werden. Die Übertragung von Änderungsepisoden von einem Workflow auf einen anderen ist ein neuartiger Vorgang. Um das Konzept von seinem Einsatzgebiet weitgehend unabhängig zu halten, gilt die Vorbedingung, dass es nicht notwendig ist, Metawissen über ein Einsatzgebiet in den Fällen explizit zu erfassen. Dadurch ermöglicht man eine Fallbasis, die in Interaktivität mit den Workflowmodellierern des WfMS wachsen und somit lernen kann, ohne dass Erfahrungen/Fälle von CBR Experten erfasst werden müssen. Die dabei entstehende Abtrennung des Workflowmodellierers von der technischen Ebene, in der Fälle modelliert werden müssen, erhöht die Akzeptanz bei der Einführung eines solchen Systems. Eine stark personengebundene Modellierung von Fällen würde wahrscheinlich auch zu stark unterschiedlichen Lösungen führen, die für unterschiedliche Personen unterschiedliche Nützlichkeiten aufweisen. Der Verzicht auf CBR Experten verursacht jedoch einen Mangel an Metawissen über das Anwendungsgebiet. Daher werden für diese Arbeit unterschiedliche Algorithmen entworfen und implementiert, die die Übertragung der Änderungsepisoden ermöglichen. Auf Basis dieser Algorithmen wird eine erste formative Studie durchgeführt, die die Realisierbarkeit des Konzepts evaluiert. Dazu müssen Fallbasen angelegt werden, deren Fälle empirisch erfasst werden. Vor der Umsetzung des Konzepts werden Anforderungsanalysen durchgeführt, um herauszufinden, ob und wie die bestehenden CAKE Systeme zu modifizieren sind, um die automatische Adaption von Workflows zu realisieren.

1.5 Aufbau dieser Arbeit

Nach diesem einleitenden Kapitel, das für ein Verständnis der Grundbegriffe und der verwendeten Technologien gesorgt hat, werden im nächsten Kapitel einige Forschungsansätze aufgeführt, die ähnliche Ziele wie diese Arbeit verfolgen. Dabei werden die dort vorgestellten Systeme zum Teil dahingehend bewertet, ob sie für die formulierten Ziele dieser Arbeit verwendet werden können. Im dritten Kapitel werden die CAKE Systeme des Lehrstuhls für Wirtschaftsinformatik II vorgestellt. Diese realisieren ein WfMS und ein CBR-System, wie es in diesem Kapitel vorgestellt wurde. Nach der Beschreibung der Systemlandschaft wird in Kapitel vier das Konzept beschrieben, mit dem eine automatische Adaption von Workflows ermöglicht werden soll. Die Konzeptentwicklung läuft dabei von einer Anforderungsanalyse bis zu einer formativen Evaluation des Konzepts, die die grundsätzliche Realisierbarkeit überprüft. Kapitel fünf beschäftigt sich mit der Umsetzung des Konzepts. Es werden notwendige Änderungen an den bestehenden CAKE Systemen beschrieben und die Implementierung der neuen Softwarekomponente, die die automatische Adaption von Workflows ermöglicht. Das letzte Kapitel ist eine Stellungsnahme des Autors zur zukünftigen Nutzung des Konzepts in einem sehr allgemeinen Sinne.

Kapitel 2 - Ähnliche Forschungsansätze

Die Wiederverwendung von Workflow Definitionen wird bereits seit längerer Zeit von gängigen Tools unterstützt, um Prozesse standardisiert wiederholen zu können. Als Beispiel sei hier die ARIS Plattform der IDS-Scheer AG genannt. Für die Wiederverwendung von prozeduralem Wissen, das zur Adaption von Workflows genutzt werden kann, gibt es bisher nur minimale bis keine Unterstützung [MZM04] [MinorEtAl10a].

2.1 AdaptFlow

AdaptFlow wurde auf Grundlage des ADEPT$_{flex}$ Workflow Systems entwickelt [RD98], das bereits die manuelle Adaption von Workflows unterstützt. Das AdaptFlow System [GreinerEtAl04] unterstützt Adaptivität in Workflowinstanzen, indem es die API von ADEPTflex nutzt und erweitert. AdaptFlow wendet dabei zuvor definierte ECA (Event-Condition-Action) Regeln an, um Exceptions zu behandeln. Die Fehlererkennung von AdaptFlow arbeitet sogar proaktiv, erwartet aber vom Workflowmodellierer, dass er jeden Umstand kennt, der während der Laufzeit geschehen kann. Da der Fokus dieser Arbeit auf der Adaption von Workflows mittels CBR liegt, werden in den nächsten Abschnitten auch nur Systeme vorgestellt, die die Adaption von Workflows mittels CBR unterstützen.

2.2 CODAW

Das CODAW System [MZM04] unterstützt ein inkrementelles Verbessern der Workflowmodellierung, indem ein ähnlichkeitsbasiertes Retrieval für Workflows genutzt wird, um ähnliche Workflows zu finden, mit deren Hilfe eine kompositionelle Adaption auf dem aktuellen Workflow ausgeführt werden kann, dessen Ausführung geändert werden muss. Um eine kompositionelle Adaption zu ermöglichen, bedarf es einem Metawissen über den Gegenstandsbereich des Workflows. Aus diesem Grund speichern Fälle in CODAW nicht nur prozedurales sondern auch deklaratives Wissen ab. Das prozedurale Wissen wird durch den Graphen des Workflows beschrieben, der wiederum

ein komplexes Kontextmodell besitzt, während das deklarative Wissen auf Prädikatenlogik basiert. Auf diese Fallrepräsentation aufbauend wurde ein „Hierarchical Task Network“ (HTN) implementiert, welches die deklarativen Beschreibungen nutzt, um neue Workflows zusammenzusetzen, die geänderten oder neuen Anforderungen genügen. CODAW unterstützt hierbei sogar die Synthetisierung komplett neuer Workflows, falls in der Retrieval-Phase kein Fall gefunden wurde, dessen prozedurales Wissen zu der Anfrage passt. In solch einem Fall wird ausschließlich das deklarative Wissen aus dem Fall genutzt. CODAW benötigt ein sehr komplexes Metamodell über die aktuelle Domäne, in der es eingesetzt wird.

2.3 CBRFlow

Die Ausführung von Workflows wird von CBRFlow [WWB04] durch ein dialogorientiertes CBR-System erweitert, um eine von einer Workflow Definition abgeleitete Workflow Instanz an geänderte Bedingungen anpassen zu können. Ein dialogorientiertes CBR-System kann als ein interaktives System beschrieben werden, das den Benutzer durch einen Fragen-Antworten-Katalog führt, um einen passenden Fall zu finden. In einem dialogorientierten CBR-System entfällt somit die Anforderung an den Benutzer, dass er a priori das Problem spezifizieren muss, um ein Retrieval durchzuführen. CBRFlow benutzt für die Steuerung seiner Workflows einen hybriden Ansatz. Es werden neben den dialogorientierten Fällen hauptsächlich zuvor festgelegte Regeln genutzt, um den Kontrollfluss zu steuern. Dabei kommen die gleiche Art von ECA Regeln zum Einsatz, wie man sie auch bei AdaptFlow finden kann. Die Fälle entsprechen spezifischem Wissen von zuvor gemachten Erfahrungen mit konkreten Problemsituationen, während die Regeln das allgemeine Wissen über die Domäne repräsentieren. Besteht ein Änderungswunsch an einem Workflow, so muss das Problem genau spezifiziert werden, sodass er auch später als Fall abgespeichert werden kann. Als Fall wird dann eine Menge von Frage-Antworten-Paaren mit ihren entsprechenden Handlungen angelegt. Kommt es zu einer Situation, in der keine Regel den Kontrollfluss mehr vorantreiben kann, muss der Benutzer den Inferenzprozess zur Lösungsbehebung manuell durchführen. Dies ist in dialogorientierten CBR-Systemen auch ganz bewusst so gehalten, da es Situationen gibt, in denen das Problem a priori nicht artikuliert werden kann. In einem Krankenhaus bspw. kennt ein Arzt nicht sofort den Grund für eine Erkrankung, sondern muss zunächst einige Faktoren ausschließen. Für eine automatische Adaption wäre in CBRFlow sehr viel Adaptationswissen nötig, sodass der Workflow konsistent und semantisch korrekt bleibt.

2.4 Phala

Das Phala CBR-System [LKM08] unterstützt proaktiv die sukzessive Komposition von Workflows, indem Teile früherer Workflows wiederverwendet werden. Zu jedem Zeitpunkt der Workflowkonstruktion wird der aktuelle Zustand des Workflows als Anfrage an das Phala System gesendet. Existiert in der Fallbasis eine passende Erweiterung für den Workflow, so wird diese Erweiterung dem Workflowmodellierer automatisch vorgeschlagen. Ein Fall in der Phala Fallbasis besteht aus sog. „execution traces“. Dabei handelt es sich um den Ablauf ehemaliger Workflowkompositionen. Bei einer Anfrage wird jede Kante mit ihrem Vorgänger- und Nachfolgerknoten des aktuellen Workflowgraphen dahingehend untersucht, ob es eine ähnliche Kante in der Fallbasis gibt. Die Bewertung der Fälle für ihre Anwendbarkeit beruht zum größten Teil darauf welcher Fall mit seinen „execution traces“ die meisten ähnlichen Kanten zum aktuellen Workflow besitzt.

In einer neueren Forschungsarbeit [CELP09] aus 2009 zur Workflowadaption mittels CBR orientiert sich eine Forschungsgruppe um David Leake hauptsächlich am Datenfluss, um eine Adaption von Workflows durchzuführen. Die Formulierung des Problems für das Retrieval wird durch gewünschte Input- und Outputdatenobjekte beschrieben, ohne dabei auf den Kontrollfluss zu achten. Da die Datenobjekte sehr komplex strukturiert sein können, ist es schwieriger für Input- und Outputdatenobjekte Übereinstimmungen zu finden als für Kontrollflusselemente. Aus diesem Grunde schlagen die Autoren vor, Mechanismen zu integrieren, die die Fälle anpassen können, sodass sie besser zur Anfrage passen. Der Ansatz befindet sich in einem frühen Stadium, da keine konkreten Implementierungsdetails erwähnt werden und eine Evaluation noch aussteht.

2.5 Bewertung der Systeme

Ist die Erweiterung einer Fallbasis wichtig für die Nützlichkeit eines Systems, so ist der Ansatz, den das CODAW System verfolgt, schwer in der Praxis umzusetzen. Die Erstellung neuer Fälle kann im CODAW System nur durch Experten geschehen, die neben der Modellierung von Prozessen innerhalb des Fachgebiets auch Kenntnisse über Prädikatenlogik besitzen. Neben der Erweiterbarkeit der Fallbasis ist die Konfiguration des CODAW Systems sehr kompliziert und statisch: Workflows können nur aus einer Menge von Tasks gebildet werden, die zuvor definiert und durch Metainformationen charakterisiert wurden. Die Erfassung von solch explizitem Metawissen, das später zur Adaption benötigt wird, ist sehr schwierig und nimmt sehr viel Zeit in Anspruch. Nachträgliche Änderungen am Metawissen würden eine komplette Überarbeitung der

Fallbasis nach sich ziehen. Aus diesem Grunde sollten heutige CBR-Systeme, die auch praxistauglich sein wollen, zu aufwendige Adaptionsmechanismen vermeiden [BergmannEtAl09]. Die Adaption von Workflows mittels Prädikatenlogik wird aufgrund ihrer Komplexität vom CODAW System auch noch nicht automatisch unterstützt[8]. Der komplette „Reuse"-Schritt als auch der „Retain"-Schritt müssen im CODAW System per Hand durchgeführt werden.

CBRFlow benötigt keine so aufwendige Modellierung des Metawissens wie es im CODAW System nötig ist, aber eine automatische Adaption von Workflows ist auch mit CBRFlow nicht möglich. Die Retrieval-Phase wird in CBRFlow durch ein Dialogsystem ersetzt und verhindert somit, dass dem Benutzer automatisch Vorschläge geliefert werden können. Der Benutzer muss aktiv in den Inferenzprozess mit einbezogen werden. Für jeden Änderungswunsch muss der Benutzer Fragen des Systems beantworten, um einen Lösungsvorschlag zu erhalten. Die Erstellung von Fällen für CBRFlow erfordert kein Expertenwissen, da Änderungen an einem Workflow durch einen neuen Katalog von Frage-Antwort-Paaren abgespeichert werden. Es erfordert aber, dass jede noch so kleine Änderung, z. B. die Verlängerung einer Deadline eines Tasks, aufwendig als Frage-Antwort-Paar beschrieben werden muss, falls die Änderung und ihre Ursache in die Fallbasis als neuer Fall mit einfließen soll.

Das Konzept des Phala Systems unterstützt den Workflowmodellierer bei der Konstruktion von Workflows, in dem proaktiv Änderungsvorschläge angezeigt werden, auf welche Art und Weise der Workflow weitermodelliert werden könnte. Eine Anwendung auf laufende Workflow Instanzen wäre sicherlich möglich, aber dies wurde nicht näher untersucht. Dabei sind es gerade die nötigen Änderungen zur Laufzeit, die ein rasches Handeln des Workflowmodellierers erfordern. Bei der Konstruktion von Workflows wird vom Phala System die Annahme getroffen, dass der gefundene ähnlichste Fall auch wirklich am besten passt. Es gibt keine Möglichkeit für den Workflowmodellierer sich den zweit- oder drittbesten Fall anzeigen zu lassen. Dieser Umstand erschwert die Verbesserung des Ähnlichkeitsmaßes für das Retrieval, da das Feedback des Workflowmodellierers nicht zur Verbesserung genutzt werden kann. Ein weiteres Problem für das Retrieval im Phala System ist durch die Struktur der Fallbasis bedingt, deren Fälle nur aus den sog. „execution traces" bestehen. Der pure Ablauf der Workflowkomposition enthält keine explizite Semantik, die ein Problem beschreibt. Somit hat der Workflowmodellierer auch keine Möglichkeit ein Problem zu artikulieren, um ein spezifisches Problem zu lösen. In einem e-Science Szenario mag diese Art der Problembeschreibung genügen, aber für das in dieser Arbeit formulierte Ziel ist ein Einbezug von Semantik unabdingbar.

[8] Das gilt für den Status der Implementierung zur Zeit der Veröffentlichung des Artikels über das CODAW System.

Kapitel 3 - Die CAKE Systeme

Der Lehrstuhl für Wirtschaftsinformatik II wurde 2004 in Trier errichtet. Seitdem wurden am Lehrstuhl zwei Softwareprototypen entwickelt: CAKE I und II. Um Unklarheiten zu vermeiden, sei an dieser Stelle erwähnt, dass es sich dabei nicht um zwei zusammenhängende Komponenten handelt. CAKE I ist ein Framework [Ma06] zur Entwicklung komplexer CBR Anwendungen. Motiviert wurde die Entwicklung von CAKE I durch das AMIRA Projekt [BergmannEtAl06]. Das Ziel des AMIRA Projektes war es Einsatzhelfer (z. B. Feuerwehrleute), die einzeln oder in Gruppen agieren, bei der Entscheidungsfindung in kritischen Situationen zu unterstützen. Aus diesem Szenario lässt sich auch das Akronym CAKE I ableiten. CAKE I steht für „***C***ollaborative ***A***gent-based ***K***nowledge ***E***ngine“. Trotz der ursprünglichen Entwicklung für dieses Szenario ist die Implementierung von CAKE I sehr generisch gehalten und wurde bereits in anderen Projekten zur Entscheidungsunterstützung verwendet. Der andere Prototyp, CAKE II, ist ein Workflow Management System, das Agilität in Workflows unterstützt. Aus pragmatischen Gründen wurde die Bezeichnung CAKE beibehalten und steht für „***C***ollaborative, ***A***gile ***K***nowledge ***E***ngine“. CAKE II wurde ursprünglich im Rahmen des URANOS Projektes entwickelt, um Workflows zum Entwurf von Mikrochips zu modellieren. Diese Workflowmodelle sind sehr umfangreich und erfordern ein Höchstmaß an Agilität [MTSB08]. Neben dieser Domäne wurden mit CAKE II Prozesse aus der Verwaltungsdomäne und aus dem Bankensektor erfolgreich modelliert [MSK09].

3.1 CAKE I

Die CAKE Architektur liefert Softwareingenieuren eine Plattform zur Entwicklung komplexer, domänspezifischer Anwendungen, in denen Informationen verarbeitet oder weitergeleitet werden müssen. Zur Informationsverwaltung nutzt CAKE Agenten[9] und ein leichtgewichtiges Workflowsystem. Beide Module stützen sich dabei auf CBR Technologie. Für diese Arbeit spielt ausschließlich die CBR-Engine von CAKE eine Rolle. Aus diesem Grund werden die anderen Komponenten von CAKE nicht näher ausgeführt.

Die Erstellung domänspezifischer Applikationen erfordert ein Domänenwissen des Entwicklers über den Anwendungsbereich. Zur Modellierung dieses Metawissens bietet

[9] Der Begriff des Agenten lässt sich nicht so leicht von einem normalen Programm abgrenzen. Wooldridge versucht sich in [Wo02] mit einer Definition: "Ein (Software-)Agent ist ein Computersystem, das in einer bestimmten Umgebung arbeitet und dazu fähig ist in dieser Umgebung autonom zu handeln um seine vorgegebenen Ziele zu erfüllen." (eigene Übersetzung)

CAKE ein universelles Datenmodell an, das an eine Domäne angepasst werden kann, in dem es den Aufbau einer Ontologie[10] ermöglicht. Eine solche Ontologie stellt wiederum die Grundlage dar, um die Repräsentation von Fällen in einer Fallbasis festzulegen.

3.1.1 Architektur

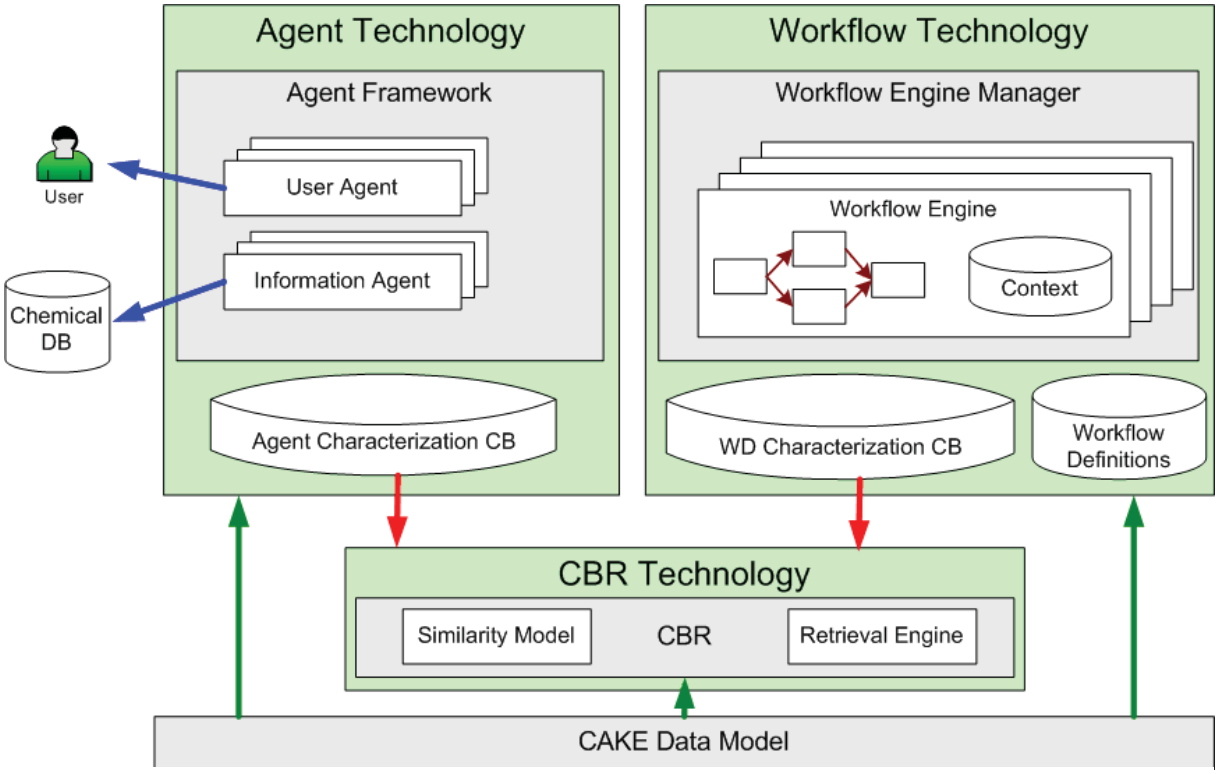


Abbildung 6 - Die CAKE I Architektur, Quelle: [Ma06]

Die CAKE Architektur ist in Abbildung 6 dargestellt. Wie in der Architekturzeichnung ersichtlich, bilden die CBR-Engine und das CAKE Datenmodell die Basistechnologie. Das Datenmodell ermöglicht es eine Ontologie aufzubauen. Ontologien werden dazu verwendet zweckorientierte, homöomorphe Abbildungen der Realität (Modelle) zu schaffen, die dazu genutzt werden, eine Konzeptualisierung genau zu beschreiben [Gr93]. Der Begriff stammt aus der Philosophie, in der eine Ontologie eine systematische Beschreibung des Daseins ist. In wissensbasierten Systemen kann nur Wissen existieren, das auch repräsentiert werden kann. Wenn das Wissen einer Domäne formal repräsentiert wird, so ist die Menge aller darstellbaren Objekte ein durch die Ontologie beschränktes Universum [Gr93]. Diese Menge an Objekten und ihre Beziehung unter-

[10] Das Wort "Ontologie" ist zusammengesetzt aus dem Griechischen "ontos" für Sein und "logos" für Wort.

einander werden von einem wissensbasierten System mittels eines Vokabulars repräsentiert [Gr93]. In Abschnitt 3.1.2 wird beispielhaft ein solches Vokabular modelliert. Die durch das Datenmodell entworfene Ontologie wird von der CBR-Engine genutzt, um festzustellen, ob alle Fälle in der Fallbasis wohlgeformt sind. Denn die Wohlgeformtheit der Fälle ist basal, um eine Suche in der Fallbasis zu ermöglichen. Zur Realisierung dieser Suche ist ein einfacher linearer Suchalgorithmus in der Retrieval-Engine implementiert. Obwohl nur ein lineares Retrieval implementiert ist, ist das System performant genug, um effiziente Suchen in der Fallbasis zu ermöglichen. Dies wurde im Vorfeld dieser Arbeit in Abschnitt 5.1.1 untersucht. Ein zu langsam arbeitendes Retrieval würde es erforderlich machen einen intelligenteren Suchalgorithmus zu implementieren. Um die Ergebnisse einer Suche zu bewerten, wird ein zuvor spezifiziertes Ähnlichkeitsmodell benutzt. Das Ähnlichkeitsmodell bewertet die relative Ähnlichkeit zwischen der Anfrage an das System und den Objekten in der Fallbasis. Das Ähnlichkeitsmodell ist also abhängig von den Objekten in der Fallbasis und der dahinter liegenden Ontologie.

3.1.2 Datenmodell

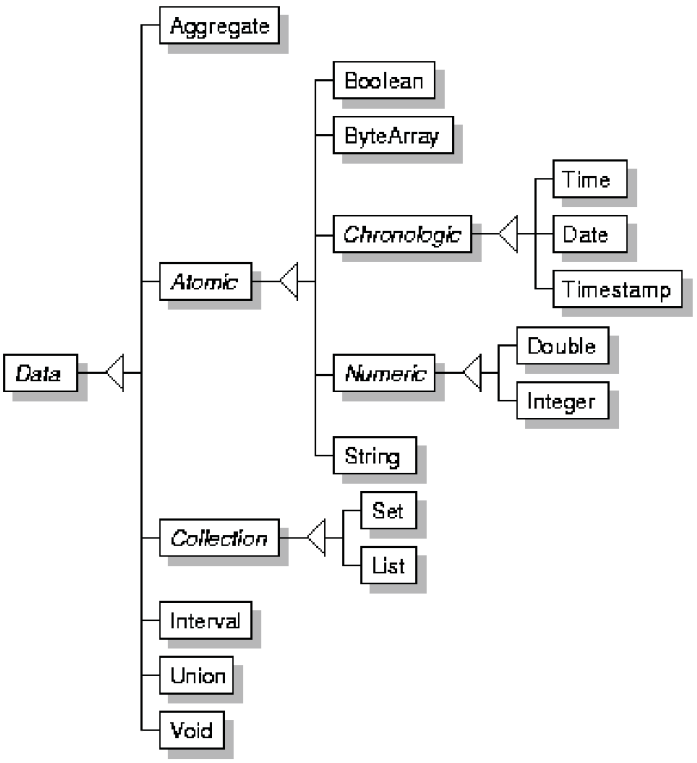


Abbildung 7 - Systemklassen des CAKE Datenmodells, Quelle: [Ma06]

Das Datenmodell beschreibt alle Daten, die vom CAKE System genutzt werden [Ma06]. Es ist ein objektorientiertes Modell, das mittels Vererbung und Aggregation neue und abgeleitete Klassen bilden kann, um komplexere Objekte zu beschreiben. Abbildung 7 zeigt alle Klassen, die innerhalb des Systems vordefiniert sind. Diese sog. ***Systemklassen*** werden beim Aufbau domänenspezifischer Ontologien genutzt. Jede Spezialisierung einer Systemklasse wird als ***Benutzerklasse*** bezeichnet, die wie eine Systemklasse verwendet und weiter spezialisiert werden kann.

Die Aggregate Klasse fasst mehrere Klassen zu einer Klasse zusammen. Die aggregierten Klassen werden dabei als ein Tupel aus Attributname und Datenklasse beschrieben. Ähnlich zur objektorientierten Programmierung wird dabei eine Instanz der Datenklasse innerhalb des Aggregates gebildet. Eine Aggregate Klasse kann aus beliebigen Subklassen von Data bestehen. So sind auch verschachtelte Aggregate denkbar, die eine Hierarchie repräsentieren. Die Gruppe der Atomic Klassen beinhaltet, wie ihr Name schon vermuten lässt, atomare Werte. In einer Programmiersprache würde es sich dabei um Variablen wie Integer, Float, Boolean oder String handeln. Der Wertebereich einer Atomic Klasse lässt sich durch die Verwendung von Intervallen und Enumerationen (Aufzählungen) einschränken. Für eine Atomic Klasse lässt sich nur dann ein Intervall des Wertebereichs definieren, wenn die Ausprägungen der Unterklasse einem kontinuierlichen/stetigen Raum angehören, der eine Ordnung auf den Werten erlaubt. Dazu gehören die Klassen Numeric und Chronologic. Enumerations beschränken den Wertebereich einer Klasse auf eine Menge zuvor definierter, diskreter Werte. Im Gegensatz zu Intervallen können alle Klassen durch eine Enumeration beschränkt werden. Für Enumerationen lassen sich neue Ordnungen und Taxonomien definieren. Bei einer Ordnung werden jedem Element aus der Menge genau ein Vorgänger und ein Nachfolger zugeordnet. Es ist also eine lineare Ordnung. In einer Taxonomie werden die Elemente der Menge in eine Baumstruktur eingeordnet, sodass jedes Element genau einen Vaterknoten besitzt. Dadurch entsteht eine Hierarchie. Die Collection Klassen Set und List können jede andere Klasse beinhalten. Sie unterscheiden sich nur in der Anordnung der Elemente. Während List eine exakte Ordnung der Elemente vorgibt, werden die Elemente in Set beliebig angeordnet. Die Union Klasse dient zur Vereinfachung und Verbesserung eines Datenmodells auf konzeptueller Ebene. Werden bspw. in einer Set Klasse unterschiedliche Klassen verwendet, so muss ihre Elternklasse mit angegeben werden. In vielen Fällen wäre dies die Data Klasse, die Wurzel des Datenmodells, von der alle anderen Klassen erben. Das hat den unerwünschten Nebeneffekt, dass nun alle anderen Klassen des Systems in der Collection erlaubt wären. Die Union Klasse ermöglicht es nun diese Elternklasse sauberer zu definieren, in dem die verwendeten Klassen innerhalb der Set eine Union Klasse zugewiesen bekommen, die die Klassen aus dem Set vereint. Die Union Klasse beschränkt also die möglichen Klassen innerhalb einer Collection. Die Void Klasse findet Verwendung, wenn in Anfragen an das

CBR System Attribute existieren, die noch unbelegt sind. Diese Attribute beinhalten Instanzen der Void Klasse.

```
1. <AtomicClass name="NumType" superClass="Integer">
2.   <ValueInterval lowerBound="0" upperBound="10000" />
3. </AtomicClass>
4. <AggregateClass name="innerAggClass" superClass="Aggregate">
5.   <Attribute name="eins" class="NumType" />
6.   <Attribute name="zwei" class="NumType" />
7. </AggregateClass>
8. <AggregateClass name="AggregatedAggClass" superClass="Aggregate">
9.   <Attribute name="inner" class="innerAggClass" />
10. </AggregateClass>
```

Abbildung 8 - Benutzerklassen, Quelle: eigene Erstellung

Jedes domänenspezifische Datenmodell ist eine Instanz des Datenmodells [Ma06]. Alle Benutzerklassen, die damit spezifiziert werden, lassen sich in XML beschreiben. In Abbildung 8 sind drei Benutzerklassen in XML spezifiziert. In Zeile 1-3 wird die Benutzerklasse „NumType“ erzeugt. „NumType“ erbt die Eigenschaften von der Systemklasse Integer. Der Wertebereich wird dabei beschränkt durch ein Intervall zwischen 0 und 10.000. In Zeile 4-7 wird die Benutzerklasse „innerAggClass“ spezifiziert, die von der Systemklasse Aggregate erbt. Diese Benutzerklasse beinhaltet zwei Attribute, die beide vom Typ „NumType“ sind, der zuvor angelegt wurde. In Zeile 8-10 wird die Benutzerklasse „AggregatedAggClass“ spezifiziert. Sie beinhaltet nur ein Attribut vom Typ „innerAggClass“.

```
1. <Agg c="AggregatedAggClass">
2.       <OA n="inner">
3.             <Agg c="innerAggClass">
4.                   <AA n="eins" v="1" />
5.                   <AA n="zwei" v="1" />
6.             </Agg>
7.       </OA>
8. </Agg>
```

Abbildung 9 - Ein CAKE Datenobjekt in XML, Quelle: eigene Erstellung

Anhand des spezifizierten Datenmodells lassen sich nun Fälle in der Fallbasis verwalten. Ist ein Fall vom Typ „AggregatedAggClass", so besteht er aus zwei ineinander geschachtelten Aggregaten, von denen das innere Aggregat zwei Attribute beinhaltet. Die Attribute „eins“ und „zwei“ sind dabei vom Typ „NumType“ und somit ganze Zahlen im Bereich von 0 bis 10.000. Abbildung 9 zeigt einen Fall aus der Fallbasis, der dem

spezifizierten Datenmodell aus Abbildung 8 entspricht. Der dargestellte Fall ist eine Objektinstanz, der im Datenmodell definierten Klassen. Da alle Benutzerklassen auf den Systemklassen aufbauen, wird auch der Wertebereich aller Aggregationen und Spezialisierungen als Attribut-Wert-Paar angegeben. Im Beispiel aus Abbildung 9 haben die Attribute „eins“ und „zwei“ innerhalb der „innerAggClass“ den Wert 1.

3.1.3 Ähnlichkeitsmodell

Um die Ähnlichkeit zwischen einem Objekt der Fallbasis und einer Anfrage an das System zu berechnen, bietet CAKE die Möglichkeit ein Ähnlichkeitsmodell zu spezifizieren. Zum Aufbau des Ähnlichkeitsmodells stehen 31 Ähnlichkeitsmaße zu Verfügung, die mit erzeugten Objekten aus dem Datenmodell funktionieren. In Abbildung 10 ist ein mögliches Ähnlichkeitsmodell für das Datenmodell aus Abbildung 8 spezifiziert. Für die Benutzerklasse „NumType“ wurden zwei Ähnlichkeitsmaße angegeben, die verwendet werden können: NumericLinear und NumericExponential. Durch die Restriktion von „NumType“ durch ein Intervall lässt sich der Abstand zweier Werte linear oder exponentiell berechnen. Ein exponentielles Ähnlichkeitsmaß sorgt dafür, dass bereits kleine Abstände zu stark unterschiedlichen Ähnlichkeitswerten führen. Das innere Aggregat „innerAggClass“ besitzt zwei Attribute vom Typ „NumType“. Das verwendete Ähnlichkeitsmaß AggregateMinimum sorgt dafür, dass der kleinste Wert

```
1. <NumericLinear class="NumType" name="default" />
2. <NumericExponential alpha="100" class="NumType" name="non-default" />
3. <AggregateMinimum name="default" class="innerAggClass"
   default="true"/>
4. <AggregateAverage name="default" class="AggregatedAggClass"
   default="true">
5.       <AggWeight att="inner" weight="1.0" />
6. </AggregateAverage>
```

Abbildung 10 - Ein benutzerdefiniertes Ähnlichkeitsmodell, Quelle: eigene Erstellung

innerhalb des Aggregates verwendet wird. Das äußere Aggregat nutzt das Ähnlichkeitsmaß AggregateAverage. Mit AggregateAverage lassen sich alle Attribute eines Aggregates gewichten. Die Summe der Gewichte sollte 1 ergeben, um einer normierten Gewichtung zu entsprechen. Da „AggregatedAggClass“ nur ein Attribut besitzt, erhält dieses Attribut die Gewichtung 1.

Für jede Systemklasse und die von ihr abgeleiteten Benutzerklassen existiert eine Menge von Ähnlichkeitsmaßen. Die Ähnlichkeitsmaße beschränken sich darauf die Ähnlichkeit zwischen zwei Werten oder zwischen zwei Mengen von Werten zu berechnen.

3.2 CAKE II

CAKE II wurde für „long-term“ Workflows entwickelt, die während ihrer Ausführung geändert werden müssen [MTSB08]. Die “Collaborative, agile Knowledge Engine” ist somit ein agiles Workflow Management System. CAKE unterstützt sowohl Ad-hoc Änderungen an laufenden Instanzen, als auch „late modelling“. Um laufende Instanzen ändern zu können, beinhaltet die CAKE Modellierungssprache Breakpoint Elemente. Breakpoints sorgen dafür, dass Teile des Workflows vorübergehend angehalten werden können. Die sonstigen Elemente der Modellierungssprache sind an UML Aktivitäts-diagramme angelehnt. Für die Abarbeitung eines Workflows nutzt CAKE eine interne Repräsentation, die auf XML basiert.

3.2.1 Architektur

Die Architektur von CAKE folgt den Empfehlungen der WfMC mit ihrem Referenzmodell (vgl. Abbildung 1). Der Enactment Service von CAKE ist der zentrale Kern, der mit allen anderen Modulen interagiert und kommuniziert. Zur Modellierung von Workflows besitzt CAKE zwei Modellierungstools (Modellierungs-GUIs). Während die eine GUI auf Java basiert und somit eine plattformunabhängige stand-alone Lösung darstellt, nutzt die Zweite GUI das Google Webtool Kit (GWT). Das GWT ermöglicht die Entwicklung komplexer Webanwendungen, indem es Java Code in Javascript übersetzt, das dann von einer Browser-Engine ausgeführt werden kann. Durch Nutzung des GWT ist die Modellierungs-GUI webbasiert und somit nicht nur plattformunabhängig, sondern auch geräteunabhängig[11]. Das erweitert das Einsatzgebiet der GUI auf alle internet-fähigen Endgeräte, wie z. B. Smartphones. Für die Aufgabenverteilung wird ebenfalls eine webbasierte GUI eingesetzt. Jeder Workflowteilnehmer besitzt dafür seine eigenen Log-in-Daten. In einer Tabelle wird dem Workflowteilnehmer eine Liste seiner Aufgaben präsentiert, die als Nächstes erfüllt werden müssen. Dabei werden ihm alle Details zur Verfügung gestellt, die innerhalb des Tasks definiert wurden. Nach Beendigung einer Aufgabe kann diese wie auf einer To-do-Liste als fertig markiert werden. CAKE hat also neben dem Enactment Service die Schnittstellen 1 und 2 des Referenzmodells implementiert (vgl. Abschnitt 1.2.3).

[11] Es wird dabei die Vision verfolgt den Collaboration Gedanken, der Teil des Web 2.0 ist, zu nutzen und zu unterstützen. Innerhalb von Communities, vor allem in sozialen Netzwerken, existieren häufig Gruppen mit gemeinsamen Interessen und Problemen. Eine webbasierte GUI unterstütz dabei den gemeinschaftlichen Entwicklungsprozess neuer Problemlösungen, wenn sich das Problem als Ablauf beschreiben lässt. Diese Vorstellung mag sich etwas abstrakt anhören, aber jeder Mensch, der schon mal eine große Veranstaltung planen musste, bspw. für einen Verein oder eine Hochzeit, wäre froh wenn er zumindestens einen Beispielablauf zur Organisation kennen würde.

3.2.2 Die CAKE Modellierungssprache

Für die Modellierung eines Workflows nutzt CAKE eine graphorientierte Beschreibung, die optisch an UML Aktivitätsdiagramme angelehnt ist und zur Abarbeitung einen Kontrollfluss nutzt. Die dabei zur Verfügung stehenden Kontrollstrukturen entsprechen zum Teil den Vorschlägen von Aalst et al. [vdA03]. Dabei wurden die Basiskontrollstrukturen (Sequence, AND, XOR), das Schleifenkonstrukt (Loop) und Meilensteine (Milestones) in die CAKE Modellierungssprache mit aufgenommen. Die in [vdA03] als „activites“ bezeichneten Elemente wurden in die Modellierungssprache als atomare Elemente eingebaut und entsprechen den in Abschnitt 1.2.2 eingeführten Tasks. Des Weiteren besitzt die Modellierungssprache drei zusätzliche Konstrukte, die zur Unterstützung der Agilität mit aufgenommen wurden [MSKB07]: die Gruppe, die Workflowelemente gruppieren kann; Platzhalter für Subworkflows, durch die „late modelling“ und eine hierarchische Komposition von Workflows möglich wird; Breakpoints, die Teile des Workflows vor der Ausführung schützen (siehe Abschnitt 3.2.6).

Abbildung 11 - Start- und Endelement

Die einfachsten Kontrollstrukturen der Modellierungssprache sind das ***Start-*** und ***Endelement***. Mit diesen zwei Elementen werden der Anfang und das Ende eines Workflows definiert.

Abbildung 12 - Das Sequenceelement

Die ***Sequence*** (Sequenz) ist eine gerichtete Kante zwischen zwei Kontrollstrukturen. Eine Sequence von Task A zu Task B bedeutet, dass Task A vor Task B ausgeführt werden muss.

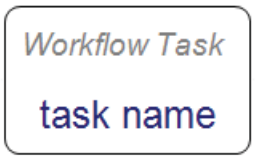


Abbildung 13 - Das Taskelement

Der ***Task*** ist eine atomare Arbeitsanweisung. CAKE bietet bisher keine Unterstützung von maschinell ausführbaren Tasks. Alle Informationen zur Erfüllung einer Aufgabe (Bezeichnung, Beschreibung, etc.) gehören zu den Eigenschaften eines Tasks und werden bei seiner Aktivierung einem Workflowteilnehmer über die Worklist zugewiesen.

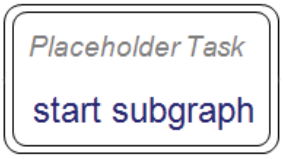


Abbildung 14 - Der Placeholder Task

Der ***Platzhalter für Subworkflows*** erleichtert die Arbeit mit sehr komplexen Workflows. Er erlaubt ein „late modelling“ eines Workflows. Der Platzhalter kann zur „build time“ des Workflows noch leer sein. Erst wenn er in einer abgeleiteten Instanz aktiviert wird, muss der Workflowmodellierer entscheiden, wie weiter zu verfahren ist. Inhaltlich steht der Platzhalter für einen ganzen Workflow, der somit ein Subworkflow ist. Durch das Einbinden von Subworkflows entsteht ein hierarchisches Geflecht. Diese Funktionalität ermöglicht eine effizientere Modellierung, wenn einige Subworkflows in anderen Workflows wiederverwendet werden können. Die Redundanz von Modellierungen wird dadurch verringert.

Abbildung 15 - Der Meilenstein

Die **Meilensteine** werden genutzt, um einfache Monitoringaufgaben zu erfüllen. In der Regel beinhaltet ein Meilenstein ein Datum, das als Deadline gilt. Hat der

Kontrollfluss den Meilenstein vor der Deadline nicht passiert, so wird der Workflow-modellierer darüber informiert.

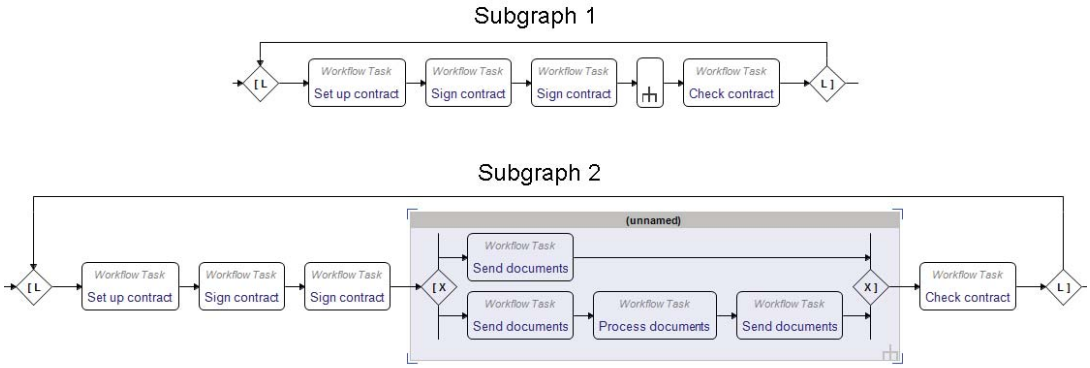


Abbildung 16 - Die Gruppe

Die ***Gruppe*** erhöht die Übersichtlichkeit in sehr großen Workflows, indem logische, zusammenhängende Fragmente des Workflows zusammengefasst werden und bei Bedarf wieder ausgeklappt werden können. Die Subgraphen 1 und 2 in Abbildung 16 sind ein Beispiel für das Gruppieren von Elementen. Während in Subgraph 1 das XOR zu einem Task zusammengefasst wurde, ist es in Subgraph 2 vollständig dargestellt.

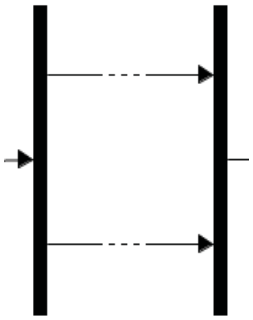


Abbildung 17 - Das AND-Element

Das ***AND-Element*** kann beliebig viele, unabhängig voneinander ablaufende Sequenzen beinhalten. Das AND-Element erlaubt eine parallele Abarbeitung und sorgt am Ende für eine Synchronisation aller innerer Sequenzen.

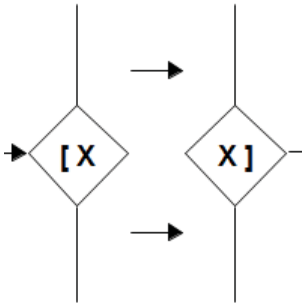


Abbildung 18 - Das XOR-Element

Das ***XOR-Element*** kann beliebig viele, unabhängig voneinander ablaufende Sequenzen beinhalten. Im Unterschied zum AND-Element wird bei seiner Aktivierung jedoch eine vordefinierte Bedingung ausgewertet, die in Javascript formuliert werden kann. Die Bedingung sorgt dafür, dass nur eine innere Sequenz ausgeführt wird, während die restlichen Sequenzen keine Beachtung finden.

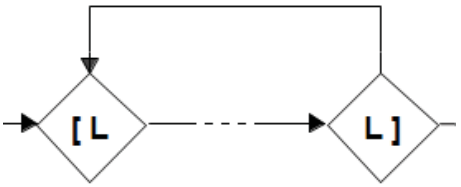


Abbildung 19 - Das Loop-Element

Das ***Loop-Element*** erlaubt Schleifen im Kontrollfluss. Der innere Teil eines Loops kann so lange iteriert werden, bis eine Abbruchbedingung eine weitere Iteration untersagt.

3.2.3 Konsistenz von Workflows

Die syntaktische Korrektheit eines Workflows ist Grundlage für einen konsistenten Workflow. Ist die syntaktische Korrektheit während der Komposition des Workflows nicht gewährleistet, so kann es zu den in Abschnitt 1.2.5 vorgestellten Fehlern kommen. Da der Schwerpunkt von CAKE auf flexiblen und änderbaren Workflows liegt, ist die Modellierungssprache von CAKE durch eine ***Blockorientierung*** beschränkt, um die Adaption von Workflows zu erleichtern. Die Blockorientierung untersagt es, dass sich die ***Blockelemente*** AND, XOR oder Loop überlappen dürfen. Es ist jedoch erlaubt, dass sie beliebig tief ineinander geschachtelt werden können. Diese reduzierte Ausdrucks-

mächtigkeit der Modellierungssprache hat jedoch Vorteile, die im Folgenden kurz erörtert werden.

Die Blockorientierung der Modellierungssprache führt dazu, dass eine Änderung am Workflow zu keiner Inkonsistenz führen kann. Da ein AND nicht überlappend in den Workflow eingebaut werden kann, sondern nur als ganzer Block, kann es zu keinem „Deadlock“ kommen, da die Synchronisation paralleler Sequenzen am Ende des ANDs immer durchgeführt wird. Gleiches gilt für die „Unintentional Multiple Execution“. Da nur das Ende eines ANDs synchronisieren darf, ist es nicht möglich einen Task mit mehreren eingehenden Kanten zu modellieren. Ebenso kann es zu keiner „Active Termination“ kommen, da nur Blockelemente verzweigen dürfen und am Ende immer alle inneren Sequenzen zusammenführen. Somit existiert nur ein Endpunkt im Workflow. Allgemein führt das Hinzufügen oder Löschen von Workflowelementen aufgrund der Blockorientierung immer von einem konsistenten Zustand in einen anderen konsistenten Zustand. Dies wird als ***„correctness-by-construction“*** bezeichnet [LiEtAl08].

3.2.4 Das CAKE Statusmodell

Damit ein WfMS den aktuellen Zustand einer Workflow Instanz kennt, benötigt es ein ***Statusmodell***, das den Zustand für jedes Workflowelement innerhalb der Instanz speichert. Das Statusmodell in CAKE kennt acht Zustände für Workflowelemente, von denen sechs für diese Arbeit wesentlich sind. Wird eine Instanz gestartet, so befinden sich anfangs alle Workflowelemente im Zustand „ready“ und sind somit bereit aktiviert

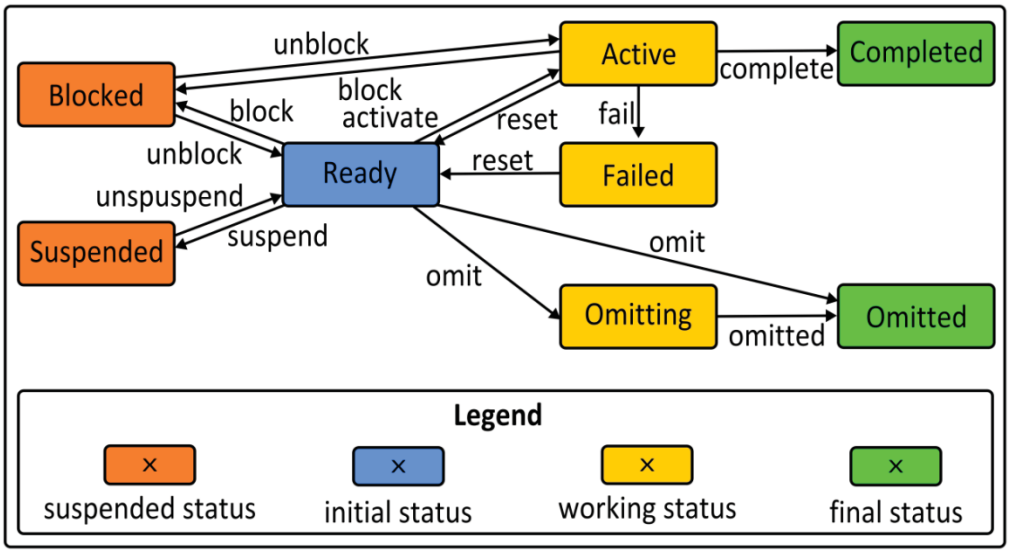


Abbildung 20 - Statusübergangsmodell für Workflowelemente, Quelle: [We08]

zu werden. Der Workflow Enactment Service von CAKE aktiviert nun das erste Workflowelement und es erhält den Status „active“. Kommt es bei der Ausführung eines aktiven Workflowelementes zu einem Fehler, so führt der Abbruch der Ausführung dazu, dass das Element den Status „failed“ erhält. Wird das Workflowelement erfolgreich abgeschlossen, erhält es den Status „completed“. In manchen Fällen, z. B. während einer Änderung an einer laufenden Instanz, ist es notwendig den Workflow zumindest partiell anzuhalten. Jedes angehaltene Workflowelement besitzt dann den Zustand „suspended“. Abschnitt 3.2.6 beschäftigt sich mit der Berechnung, welche Teile eines Workflows für eine Änderung angehalten werden müssen.

Abbildung 20 zeigt das vollständige Statusmodell für ein Workflowelement während seiner Ausführung. Dabei ist ersichtlich, dass auf der Systemebene die Statusübergänge komplexer gestaltet sind. So kann ein Workflowelement nur den Status "suspended" annehmen, wenn es zuvor in den initialen Status "ready" überführt werden konnte.

3.2.5 Abarbeitung von Workflows

Die vorgestellte Modellierungssprache von CAKE könnte bereits genutzt werden, um einen Workflow abzuarbeiten. Dafür müsste eine Repräsentation existieren, die den Workflow als Graph mit all seinen Elementen erfasst. Ein solches proprietäres Datenmodell würde jedoch kompliziert werden, wenn die Persistenz von Workflows sichergestellt werden muss und Workflows zwischen unterschiedlichen Workflowsystemen ausgetauscht werden sollen. Aus diesem Grunde nutzt CAKE eine spezifische, interne Repräsentation, die auf XML basiert (vgl. Abschnitt 1.2.3). Der Einsatz von XML bietet mehrere Vorteile:

- Einer der wesentlichen Vorteile von XML ist, dass es von einem Menschen gelesen und verstanden werden kann. In einem auf XML basierenden Workflow lassen sich Elemente wie ein AND, XOR oder ein Task auch von einem Menschen identifizieren.
- Die Persistenz eines Workflows, die gerade bei langen Laufzeiten eine wichtige Rolle spielt, ist durch die Speicherung in XML sichergestellt.
- XML dient dem Informationsaustausch und ist transformierbar, sodass ein XML Format leicht in ein anderes XML Format überführt werden kann. Das erleichtert die Zusammenarbeit mit anderen WfMS, die ebenfalls auf XML basieren.

Die Blockorientierung der CAKE Modellierungssprache sorgt dafür, dass jeder modellierte Workflow als ein XML Baum dargestellt werden kann. In Abbildung 21 ist dies an einem einfachen Beispiel erklärt. Im oberen Teil befindet sich ein vom Benutzer modellierter Workflow, der aus einem AND und drei Tasks besteht. Der untere Teil der

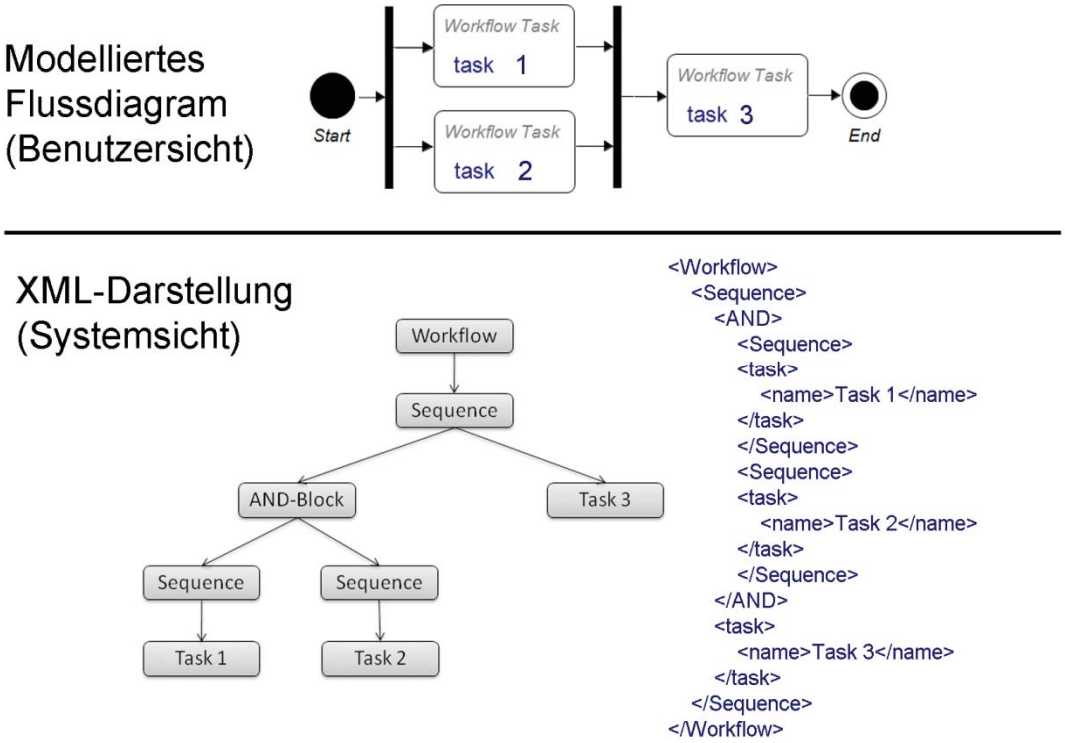


Abbildung 21 - Die interne Workflowrepräsentation in CAKE, Quelle: eigene Erstellung

Abbildung zeigt die interne Umsetzung in CAKE. Der kurze XML-Ausschnitt ist eine vereinfachte Version des dargestellten Baumes. Bei einer Aktivierung des Workflows wird zunächst die oberste Sequenz aktiviert. Die Sequenz aktiviert wiederum das am weitesten links stehende Element. Damit wird das AND aktiv. Wie zuvor bereits beschrieben, erlaubt das AND eine parallele Verarbeitung und aktiviert somit alle seine Sequenzen gleichzeitig. Erst wenn alle Elemente, der inneren Sequenzen (Task 1 und Task 2) abgeschlossen wurden, wird auch das AND auf „completed“ gesetzt. Nun kann das nächste Nachbarelement des ANDs aktiviert werden, indem Task 3 den Status „active" erhält (zuvor stand er auf „ready“). Auf diese Art und Weise wird die blockorientierte Workflowsprache von CAKE maschinell abgearbeitet.

3.2.6 Anhalten von Workflows

Der Enactment Service von CAKE ermöglicht Änderungen an laufenden Workflow Instanzen. Solche Ad-hoc Änderungen an einem Workflow betreffen nur Teile des Workflows und nicht den Workflow im Ganzen. Beispiele dafür wären das Hinzufügen oder Löschen von Workflowelementen. Aber auch Änderungen an einem aktiven Task

Abbildung 22 - Das Breakpoint-Element

gehören dazu und können nicht so einfach durchgeführt werden. Ein aktiver Task muss angehalten werden und seine Arbeitsanweisung muss von der Worklist genommen werden. Ein sehr restriktiver Unterbrechungsmechanismus würde den Workflow im Ganzen anhalten und damit auch Teile des Workflows anhalten, die nicht von den Änderungen betroffen sind. Um eine weniger restriktive Sperrung des Workflows zu gewährleisten, besitzt die CAKE Modellierungssprache ein zusätzliches Element.

Abbildung 22 zeigt die Repräsentation eines Breakpoints wie er in der CAKE Modellierungssprache verwendet wird. Dieses Stoppschild wird vom Workflow-modellierer vor die Stelle im Workflow gesetzt, an der etwas am Workflow geändert werden soll. Dabei kann er beliebig viele Stoppschilder in den Workflow setzen. Je präziser die Breakpoints/Stoppschilder dabei verwendet werden, desto höher bleibt auch die Nebenläufigkeit des Workflows. Ein Workflow, der sehr verschachtelt ist, mit vielen inneren Sequenzen, profitiert davon, da die meisten Workflowteilnehmer nicht in ihrer Arbeit unterbrochen werden. Im Beispiel aus Abbildung 23 wird dies noch einmal verdeutlicht.

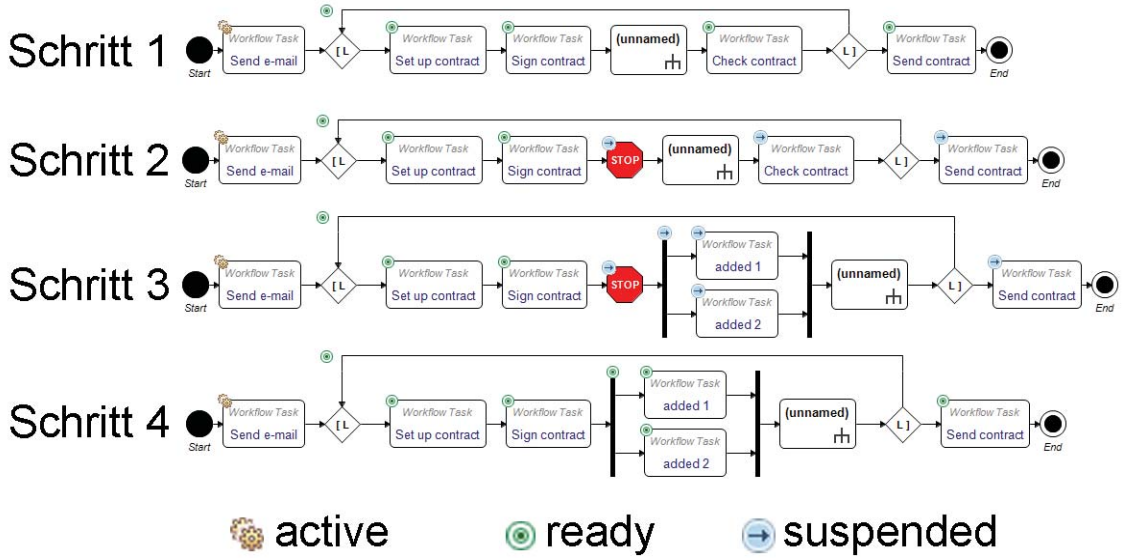


Abbildung 23 - Der Suspendmechanismus, Quelle: eigene Erstellung

In Schritt 1 wird der Workflow zunächst gestartet. Der Task „send e-mail“ erhält den Status „active“ während die anderen Elemente den Status „ready“ innehaben. Der Workflowmodellierer erkennt nun in Schritt 2, dass es nach dem Task „Sign contract“ ein Problem geben könnte. Darum setzt er vor den gruppierenden Platzhalter einen Breakpoint. Darauf hin berechnet der Enactment Service von CAKE den Wirkungsbereich des Breakpoints anhand der internen Baumstruktur und setzt alle Elemente hinter „Sign contract“ auf den Status „suspended“. Der angehaltene Teil des Workflows kann nun wie in der „build time“ ummodelliert werden. Dies geschieht in Schritt 3. Der Workflow-modellierer fügt ein AND mit den Tasks „added 1“ und „added 2“ hinzu und löscht gleichzeitig den Task „Check contract“. Der nicht angehaltene Teil des Workflows kann währenddessen ganz normal weiter abgearbeitet werden. Im letzten Schritt entfernt der Workflowmodellierer den Breakpoint und der Wirkungsbereich des Breakpoints erhält wieder den Status „ready“. Die Workflow Instanz wurde nun während ihrer Laufzeit umgebaut, ohne die Konsistenz des Workflows zu verletzen. Eine Konsistenzverletzung würde bspw. bestehen, wenn ein Task während seiner Änderung von einem Workflowteilnehmer als erfolgreich beendet markiert wird. Der Workflowmodellierer würde somit an einer Stelle arbeiten, die der Kontrollfluss schon abgearbeitet hat. Der Suspendmechanismus von CAKE mit dem Breakpoint-Element verhindert solche Situationen, indem der Workflowmodellierer exklusiv den Zugriff und die Kontrolle über einen Teil des Workflows zurückerhält.

3.3 CAKE III

Die CAKE III Plattform befindet sich zurzeit in Entwicklung und beinhaltet die Vision, dass Workflows mit Erfahrungswissen angepasst werden können. Die in dieser Arbeit entworfene Komponente zur Adaption von Workflows ist ein Bestandteil der CAKE III Plattform. Im weiteren Verlauf wird sie als ***"Workflow Adaptation Manager"*** oder kurz ***"Adaptation Manager"*** bezeichnet. Sie beinhaltet die Logik, um Workflows mittels CBR anzupassen. Der nächste Abschnitt gibt einen ersten Einblick über die Kommunikation der Systeme untereinander, der dann in den späteren Kapiteln detaillierter beschrieben wird.

3.3.1 Integration der CAKE Systeme

Für den Aufbau der CAKE III Plattform wurde eine serviceorientierte Systemlandschaft geschaffen, die die Integration webbasierter Modellierungswerkzeuge ermöglicht (siehe Abbildung 24). Die webbasierte Modellierungs-GUI, mit der die Benutzer interagieren, kann bei ihrer Funktionalität auf zwei Servicekomponenten zurückgreifen: die WfMS API, die ein Datenmodell des CAKE II WfMS darstellt, sorgt dafür, dass das Modellieren, Ausführen und Anhalten von Workflows, wie im Referenzmodell vorgesehen, unterstützt wird. Die Adaptation API wird bei der Kommunikation mit dem Workflow Adaptation Manager Modul genutzt. Dieses Modul ist in der Abbildung grün

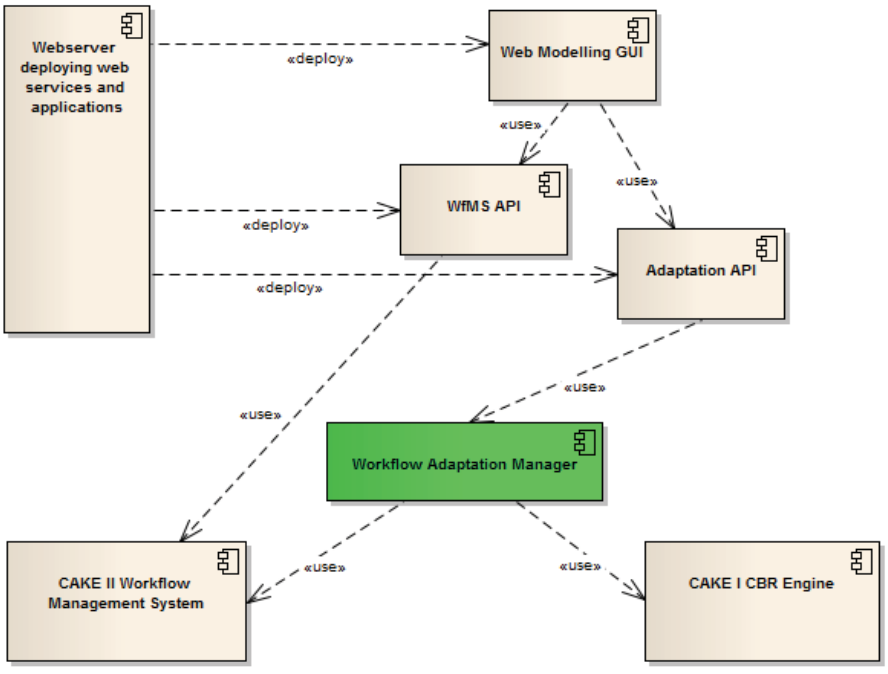


Abbildung 24 - Die CAKE III Architektur, Quelle: eigene Erstellung

markiert ist, um herauszustellen, dass es sich dabei um die Komponente zur Workflowadaption handelt, die in dieser Arbeit entwickelt wurde.

Erfolgt in der Web Modelling GUI ein Wunsch nach einer automatischen Adaption, der durch den Workflowmodellierer initiiert wurde, dann wird über den Service des Adaptation Managers (Adaptation API) der Adaptionsprozess gestartet. Dieser Kommunikationsprozess wird später in Abschnitt 5.4 genauer beschrieben, um die Umsetzung des Konzepts besser verständlich zu machen.

Kapitel 4 - Konzeptentwicklung

Die Entwicklung eines Konzeptes zur automatischen Adaption von Workflows stellt die Grundlage für die spätere technische Realisierung der Softwarekomponente dar. Bevor die Schwerpunkte in der Konzeptentwicklung gesetzt werden konnten, wurde die Adaption aus der Nutzersicht untersucht, um daraus Rückschlüsse an die Anforderungen zu ziehen. Die in diesem Zusammenhang identifizierten Anforderungen versuchen zu klären, wie die automatische Adaption von Workflows ablaufen sollte und auf welche Weise sich dies auf einen CBR Ansatz übertragen lässt. Um diese Fragen zu klären, wird eine Fallrepräsentation beschrieben, die in einem durchdachten Adaptionsprozess verwendet wird, um Workflows automatisch anzupassen. Für die Wiederverwendung von Änderungsepisoden während des Adaptionsprozesses bildete sich bei der Konzeptentwicklung ein Ansatz heraus, der die Änderungsepisoden durch die Benutzung sog. Anker auf neue Workflows überträgt [MinorEtAl10a] [MinorEtAl10b]. Die Überprüfung des Konzepts durch eine erste formative Evaluierung in zwei Anwendungsdomänen und die darauf beruhende Bewertung des Konzepts schließen das Kapitel ab.

4.1 Anforderungsanalyse

Um die Anforderungen an ein Konzept zu entwerfen, ist es wichtig den Ablauf gedanklich zu durchwandern. In dieser Arbeit muss der Ablauf zur automatischen Adaption von Workflows festgelegt werden, um das Szenario zu beschreiben, das durch ein Konzept unterstützt werden soll. Hinsichtlich der Adaption von Workflows existieren drei Szenarien, die zunächst voneinander getrennt werden müssen. Zur Veranschaulichung der Szenarien werden im nächsten Abschnitt UML Use-Case-Diagramme präsentiert und beschrieben. Die darauf folgenden Anforderungen leiten sich zum Teil aus den festgestellten Szenarien ab. Während die Anforderungen an den Adaptionsprozess sich überwiegend an den Szenarien orientieren, erfassen die Anforderungen an eine Fallrepräsentation bereits Gedanken zur Anwendung in unterschiedlichen Anwendungsgebieten, um die Fallrepräsentation generisch zu halten.

4.1.1 Szenarien

Für die Adaption von Workflows existieren drei Szenarien, die zu unterscheiden sind. Das wesentliche Unterscheidungskriterium dieser Szenarien ist der Grad der

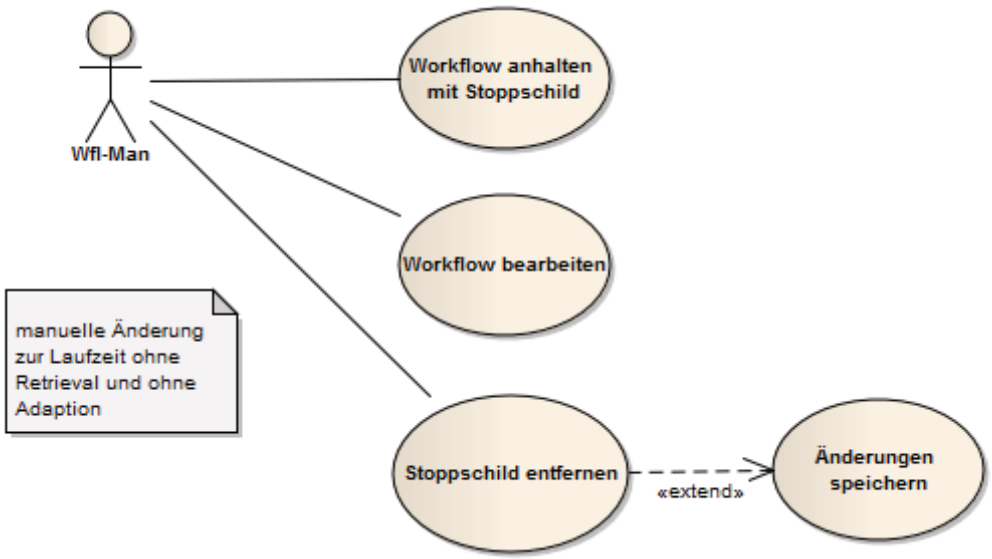


Abbildung 25 - Use Case 1: Die manuelle Adaption, Quelle: eigene Erstellung

Unterstützung für den Workflowmodellierer. Das erste Szenario ist die klassische Anpassung mittels agiler Workflowtechnologie. Die Agilität des Workflow Systems erlaubt es dem Workflowmodellierer den Workflow manuell umzubauen, sobald er ein Problem bemerkt oder antizipiert. Diese Form der Adaption von Workflows war Gegenstand von Abschnitt 1.3.1 und wird in Abbildung 25 nochmals als Use-Case-Diagramm dargestellt[12].

Die Anpassung des Workflows läuft in drei Schritten ab, die alle vom Workflowmodellierer per Hand ausgeführt werden müssen:

1. Breakpoint (Stoppschild) im Workflow setzen.
2. Den angehaltenen Teil des Workflows bearbeiten.
3. Entfernen des Breakpoints.

Das zweite Szenario behandelt eine halbautomatische Adaption. In diesem Szenario kann der Workflowmodellierer einen ***„Change Request“*** (Änderungswunsch) artikulieren, der vom CBR-System interpretiert wird. Dieses erweiterte Szenario findet bspw. dann Verwendung, wenn der Workflowmodellierer unsicher ist, an welcher Stelle Änderungen am Workflow vorzunehmen sind oder um sich mögliche Änderungsschritte anzeigen zu lassen. Die gelieferten Workflows des Systems dienen dem Modellierer als Vorlage, um den aktuellen Workflow manuell umzubauen. Sie entstammen Fällen, die über einen ähnlichen Change Request verfügen. Der Workflowmodellierer setzt dann wie im ersten Szenario einen Breakpoint und baut den Workflow manuell um. Bevor der

[12] Ein UML Use-Case-Diagramm beschreibt einen Ablauf aus der Benutzersicht.

Breakpoint jedoch wieder entfernt wird, sollte es möglich sein, die gemachten Änderungen des Workflowmodellierers als neuen Fall zu erfassen und somit die Fallbasis zu erweitern. Abbildung 26 zeigt die halbautomatische Adaption in einem Use-Case-Diagramm.

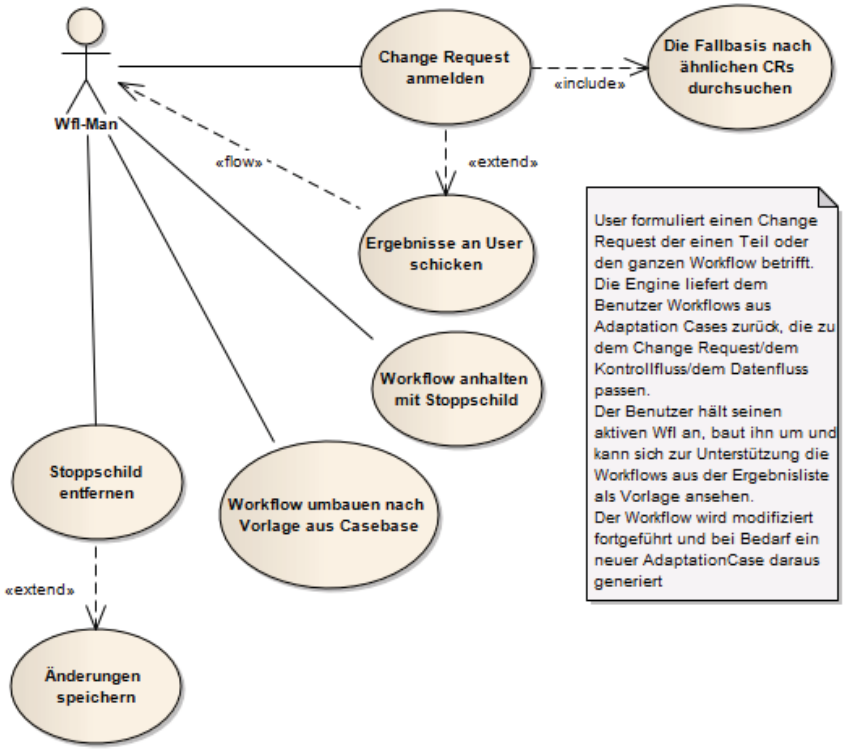


Abbildung 26 - Use Case 2: Die halbautomatische Adaption, Quelle: eigene Erstellung

Zusammenfassend lassen sich sechs Schritte ableiten, die die halbautomatische Adaption charakterisieren:

1. Der Benutzer schickt den Change Request an das CBR-System.
2. Das System schickt die Adaptionsergebnisse aus den Fällen an den Benutzer.
3. Der Benutzer setzt einen Breakpoint in den Workflow.
4. Der Benutzer baut den Workflow anhand der Adaptionsvorschläge um.
5. Der Benutzer entfernt den Breakpoint.
6. Die Änderungen werden als neuer Fall erfasst.

Im Gegensatz zum ersten Szenario existieren in der halbautomatischen Adaption zwei Akteure. Auf der einen Seite der Workflowmodellierer, der immer noch alle Änderungen manuell durchführen muss, auf der anderen Seite das CBR-System, das Vorschläge zur Adaption liefert. Das nun folgende dritte Szenario beinhaltet die

vollautomatische Adaption von Workflows. In Abbildung 27 wird das Szenario mit einem Use-Case-Diagramm veranschaulicht, das fünf relevante Schritte beinhaltet:

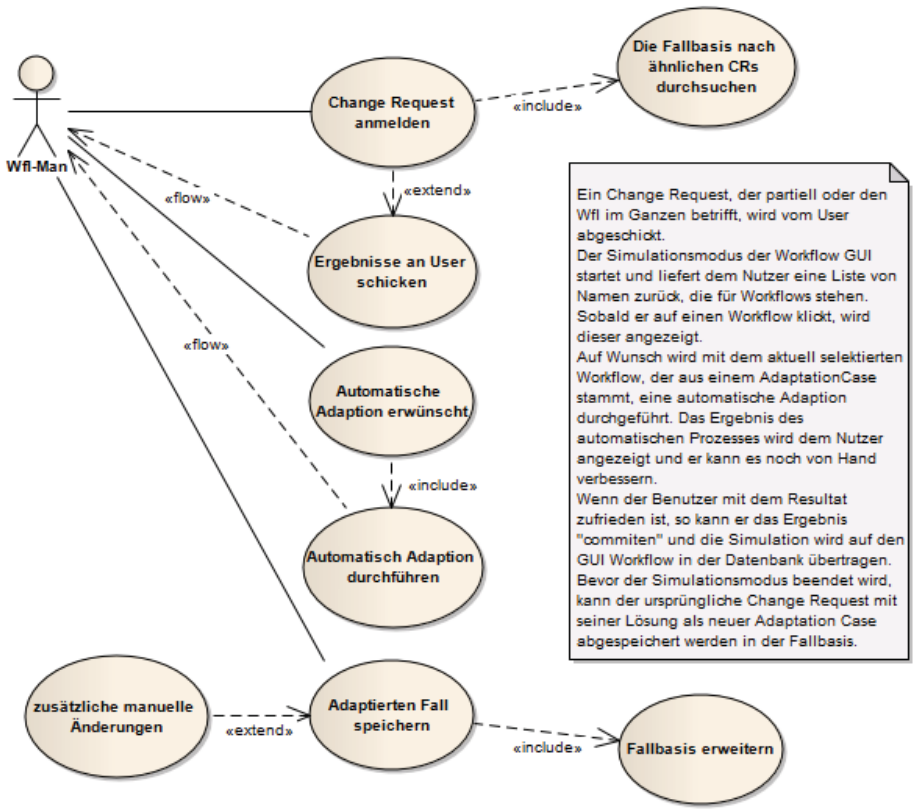


Abbildung 27 - Use Case 3: Die vollautomatische Adaption, Quelle: eigene Erstellung

1. Der Benutzer schickt den Change Request an das CBR-System.
2. Das System schickt ähnliche Fälle an den Nutzer.
3. Nachdem der Benutzer eine automatische Adaption wünscht, wird diese vom System durchgeführt.
4. Der Benutzer führt optional zusätzliche manuelle Änderungen durch.
5. Die Änderungen werden als neuer Fall erfasst.

Anhand des Change Requests bereitet das System nicht nur Vorschläge für den Workflowmodellierer vor, sondern es ist auch imstande, die Workflow Definition oder Instanz selbstständig umzubauen unter Einhaltung aller Konsistenzkriterien. Das dritte Szenario stellt darum die höchsten Anforderungen an das System. Der Prozess der vollautomatischen Adaption sollte aber zunächst nur simuliert werden, um die Kontrolle über die Adaption nicht vollständig an ein Computersystem abzugeben. Das simulierte Adaptionsergebnis wird in letzter Instanz von einem Menschen kontrolliert und

gegebenenfalls nachgebessert, bevor die Änderungen in den echten Workflow übernommen werden.

4.1.2 Anforderungen an den Adaptionsprozess von Workflows

Die Adaption von Workflows mittels CBR soll unabhängig vom Ausführungszustand des Workflows möglich sein. Bei Workflows, die gerade ausgeführt werden, müssen während einer Adaption neben den allgemeinen Anforderungen, die z. B. die Konsistenz des Workflowgraphen betreffen, zusätzliche Anforderungen erfüllt werden, die die fehlerfreie Abarbeitung des Workflows gewährleisten und die Nebenläufigkeit in parallelen Sequenzen nicht unterbindet. Der Adaptionsmechanismus sollte sowohl während der Workflowkonstruktion als auch während der Workflowausführung genutzt werden können. Bei der Komposition eines Workflows ist es oft selbst für den Modellierer schwierig – der normalerweise ein Experte in seinem Fachgebiet ist – alle Eventualitäten und notwendigen Ressourcen eines neuen Prozesses zu erfassen, zu begreifen und dann zu modellieren. Im schlimmsten Fall wird ein Prozess obsolet, da seine Modellierung zu viel Zeit in Anspruch nahm [WWB04]. Aus diesem Grunde ist eine Unterstützung des Workflowmodellierers bereits während der „build time“ des Workflows sinnvoll. Für laufende Workflow Instanzen sollte das System Lösungsvorschläge bei Exceptions liefern, die dem Workflowmodellierer die Korrektur im Ganzen oder in Teilen abnimmt. Um Vorschläge an den Workflowmodellierer liefern zu können, muss für die Retrieve-Phase ein Ähnlichkeitsmaß existieren, das die Ähnlichkeit zwischen der Anfrage des Modellierers und den Fällen in der Fallbasis bewertet. Da nur der Workflowmodellierer mit dem CBR-System interagieren soll, ist es eine spezielle Anforderung an das System, dass neues Adaptionswissen – in Form neuer Fälle – allein durch die Interaktion mit dem System abgeleitet werden kann. Die Erstellung neuer Fälle soll ohne CBR Experten möglich sein.

4.1.3 Anforderungen an die Repräsentation von Fällen

Eine Repräsentationsform für Fälle muss zwei Anforderungen erfüllen. Zum einen muss der Problemteil so beschaffen sein, dass er für ein Retrieval genutzt werden kann, zum anderen muss der Lösungsteil in der Lage sein Änderungsepisoden eines Workflows zu formalisieren. Eine Anfrage, die an das CBR-System gestellt wird, kann aus zwei Informationen konstruiert werden: dem Change Request und dem zu adaptierenden Workflow. Somit muss der Problemteil den Change Request und den Workflow formalisieren. Für den Change Request muss eine Repräsentationsform entworfen werden, die nicht nur eine einfache textuelle Beschreibung wie in den Use-Cases

abbilden kann, sondern universell einsetzbar ist. Für Workflows wurde bereits in Abschnitt 3.2.5 eine auf XML basierende Repräsentationsform vorgestellt. Es ist zu überprüfen, ob diese Repräsentationsform übernommen werden kann, sodass die strukturellen und semantischen Informationen eines Workflows bei einem Retrieval genutzt werden können. Diese Entscheidung ist wichtig, da letztlich die Ähnlichkeit zwischen Workflows berechnet werden muss, um sinnvolle Adaptionsergebnisse zu erhalten. Ein weiterer Anspruch an die Ähnlichkeitsberechnung entsteht durch die Komplexität, die Workflowstrukturen annehmen können. Das neue Workflowformat muss die Berechnung der Ähnlichkeit in einer angemessenen Zeit ermöglichen. Ein Problem, das bei der Übernahme des CAKE II Formats auftreten würde, wäre die fehlende Integration eines Datenflusses. CAKE III soll fähig sein, den Datenfluss zu berücksichtigen, um das Prozesswissen vieler unterschiedlicher Anwendungsdomänen abbilden zu können. Somit muss auch die neue Repräsentationsform für Workflows einen Datenfluss beinhalten. Neben der Integration des Datenflusses wird wie beim Change Request ein generisches Datenmodell benötigt, das die eigentlichen Datenobjekte abbilden kann. Um die spätere Integration in die bestehende Systemlandschaft zu erleichtern[13] und die Übertragbarkeit einer Fallbasis auf andere Systeme zu ermöglichen, würde es sich anbieten bei der Wahl einer neuen Repräsentationsform für Workflows XML weiterhin als Basistechnologie zu nutzen. Im Lösungsteil eines Falls müssen die Änderungsepisoden eines Workflows so repräsentiert werden, dass sie bei der Adaption eines beliebigen anderen Workflows wiederverwendet werden können.

4.2 Fallrepräsentation

Aufbauend auf der Anforderungsanalyse für die Repräsentation von Workflows wurde eine Fallstruktur entwickelt, die das Wissen zur Adaption von Workflows formal abspeichern kann (vgl. [MinorEtAl10a]). Ihrer Aufgabe entsprechend werden sie im Folgenden als ***„Adaptation Cases“*** (Adaptionsfälle) bezeichnet. Die Struktur der Fälle ist traditionell zweigeteilt:

Der Problemteil besteht aus:

1. Einer semantischen Beschreibung, die einen Änderungswunsch beinhaltet, um einen Workflow anzupassen. Dieser sog. ***„Change Request“*** besteht zum Stand dieser Arbeit aus einer einfachen Freitextbeschreibung.

[13] Sowohl die Repräsentation der Workflows in CAKE II, als auch die Fallbasis von CAKE I liegen in einem XML Format vor.

2. Dem ursprünglichen Workflow, der vom Benutzer geändert wurde. Der Workflow liegt dabei in einem neu entwickelten Workflowformat vor, das im nächsten Abschnitt detaillierter beschrieben wird. Dieses Format ermöglicht eine effiziente Berechnung der Ähnlichkeit zwischen Workflows. Aus diesem Grunde wird es als ***„Retrieval Workflow“*** Format bezeichnet.

Der Lösungsteil besteht aus[14]:

3. Dem Adaptionsergebnis des Falls. Dabei handelt es sich um den adaptierten Workflow.
4. Den ***ADD-Listen***, die alle Elemente beinhalten, die in den ursprünglichen Workflow eingefügt werden mussten, um ihn in den adaptierten Workflow zu überführen.
5. Den ***DELETE-Listen***, die alle Elemente beinhalten, die aus dem ursprünglichen Workflow entfernt werden mussten, um ihn in den adaptierten Workflow zu überführen.

Bei genauerer Betrachtung des Lösungsteils fällt auf, dass die ADD- und DELETE-Listen im Prinzip redundant sind, da bereits der adaptierte Workflow Bestandteil des Lösungsteils ist und die ADD- und DELETE-Listen aus der Differenz des adaptierten und des ursprünglichen Workflow berechnet werden können. Eine Onlineberechnung der ADD- und DELETE-Listen würde es aber für einen Menschen schwer machen die Fallbasis zu lesen. Da dies ein erster experimenteller Ansatz ist, der viele Unsicherheiten beinhaltet, werden die ADD- und DELETE-Listen explizit in einem Fall definiert. Umgekehrt könnte argumentiert werden, warum der adaptierte Workflow Bestandteil eines Falles ist. Der Grund liegt in der Unterstützung der halbautomatischen Adaption. In diesem Szenario sollen dem Benutzer Vorschläge unterbreitet werden auf welche Art und Weise Änderungen auf den aktuellen Workflow übertragen werden können. Dies geschieht durch Ansicht vergangener, adaptierter Workflows. Die Darstellung dieser Workflows wird durch das Abspeichern des vollständig adaptierten Workflows vereinfacht.

Unter Bezugnahme auf die Anforderungen ist es für das CAKE III System egal, ob es sich bei einem abgespeicherten Workflow um eine Definition oder eine Instanz handelt. Bei der Repräsentation wird dies nicht unterschieden. Der Workflowmodellierer kann deswegen während der Workflowkomposition als auch bei Ad-hoc Änderungen laufender Instanzen durch Fallwissen unterstützt werden. Es spielt dabei auch keine Rolle, wie weit eine Workflowausführung fortgeschritten ist.

[14] Der Vollständigkeit wegen sei an dieser Stelle auf Implementierungsaspekte aus Kapitel 5 verwiesen. Zusätzlich zum Retrieval Workflow wird der ursprüngliche Workflow im CAKE II Format ebenfalls in einem Fall mit aufgenommen. In Abschnitt 4.2.1 wird der Grund hierfür dargelegt.

4.2.1 Das Retrievalformat für Workflows

Den Anforderungen entsprechend muss das Retrievalformat für Workflows nicht nur einen Kontrollfluss abbilden können, sondern auch einen Datenfluss. Da die Adaption von Workflows auch ein agiles WfMS benötigt, welches Ad-hoc Änderungen erlaubt, war zunächst das CAKE II Workflowschema Ausgangspunkt für den Entwurf eines neuen Retrievalformats für Workflows. Das Workflowschema aus CAKE II liefert ein sehr umfangreiches Format zur Beschreibung eines Kontrollflusses, aber es beinhaltet auch sehr viele Informationen, die nur spezifisch für den Enactment Service sind. Solche systemspezifischen Daten sind für ein Retrieval eher ungeeignet, da der Informationsgehalt systemspezifischer Daten gering ist. Folgendes Beispiel verdeutlicht das Problem: Das System erhält einen Änderungswunsch für eine laufende Instanz. Der beste Adaptionsvorschlag befindet sich in einem Fall, der eine Workflow Instanz beinhaltet, die von der gleichen Definition abgeleitet wurde, wie die im Moment zu ändernde Instanz. Der Einbezug der systemspezifischen Daten in ein Ähnlichkeitsmaß zwischen Workflows könnte zu dem Ergebnis führen, dass die Workflows sehr unterschiedlich sind, obwohl sie strukturell und semantisch sogar gleich sind. Eine um die systemspezifischen Daten bereinigte Form des CAKE II Workflowschemas kann demnach genutzt werden, um die strukturellen und semantischen Informationen eines Workflows abzubilden. Dies erhöht auch die Übertragbarkeit der Fallbasis, falls sie zu einem späteren Zeitpunkt mit einem anderen WfMS zusammenarbeiten soll. Ausgehend von dieser bereinigten Form wurde der Retrieval Workflow für die ersten Experimente (siehe Abschnitt 4.5) weiter eingeschränkt, um nur die Attribute mit aufzunehmen, die bei einer Ähnlichkeitsberechnung verwendet werden. In Abschnitt 5.2.2, der die Implementierung des Retrieval Workflows beschreibt, wird auf die Attributauswahl näher eingegangen. Dieser bewusst in Kauf genommene Informationsverlust bedeutet für das Retrieval einen Performanzgewinn um einen konstanten Faktor[15]. Um diesen Informationsverlust gegebenenfalls kompensieren zu können, wird in die Fallstruktur als zusätzliche Komponente der unveränderte Workflow mit aufgenommen. Anhand von eindeutigen IDs aller Workflowelemente lässt sich der Informationsverlust beseitigen, da anhand der ID die vollständige Beschreibung eines Workflowelements ausgelesen werden kann. Insbesondere für die ersten Tests mit den Adaptionsalgorithmen (siehe Abschnitt 4.4.2) erleichterte dieses Attribut die Untersuchungen. Eigenschaften, die häufig ausgelesen werden mussten, wurden anschließend in das Retrieval Workflowformat fest übernommen. Das Format des Retrieval Workflows ist also nicht unumstößlich, sondern modifizierbar. Es ist eine abstrahierte Workflowrepräsentation, die während der Retrieve- und Reuse-Phase einer automatischen Workflowadaption benutzt wird.

[15] Die aktuelle Form des Retrieval Workflows ist in etwa um den Faktor sechs kleiner als das ursprüngliche CAKE II Workflowformat.

Um ein generisches Datenmodell zu erhalten, sollten sowohl sehr detaillierte als auch abstrahierte Modellierungen durch das Modell unterstützt werden. Für den Entwurf eines Datenflussmodells wurde sowohl über alte als auch über eine neue Anwendungsdomäne nachgedacht. Der Leser sei an dieser Stelle an e-Science Workflows erinnert, die die Produktion eines Medikamentes unterstützen könnten (vgl. Abschnitt 1.2.4) indem chemische Teilprodukte sukzessive zusammengesetzt werden. Eine weitere bekannte Anwendungsdomäne, die ähnliche Anforderungen stellt, sind innerbetriebliche oder öffentliche Verwaltungsapparate [Ke09]. Ein Antrag in der Verwaltung kann dazu führen, dass mehrere Dokumente, als benötigte Daten, verschiedene Instanzen durchlaufen müssen und dass Dokumente zu diesem Zwecke auch gruppiert werden können, um wie eine Art Dokumentenmappe behandelt zu werden. Als neue Anwendungsdomäne wurden Kochprozesse untersucht. Die Verwendung von Zutaten, die Datenobjekten entsprechen, ist in Kochprozessen zum Teil sehr komplex gestaltet. In Abschnitt 5.1.2 werden Kochprozesse untersucht, um eine adäquate Modellierung zu finden, die auch für die ersten Experimente genutzt werden konnte. Bei Modellierungsversuchen, die über die textuelle Beschreibung von Kochrezepten hinausgingen, wurde festgestellt, dass es im Datenfluss implizit zu Aggregationen kam. Bei der Bildung von ***Aggregaten*** werden Datenobjekte zusammengefasst und agieren fortan als neues, eigenständiges Datenobjekt, das sich oft nicht mehr zerlegen lässt. Eine Soße oder ein Teig kann als Aggregat seiner Zutaten angesehen werden, das zu einem späteren Zeitpunkt des Workflows als eigenständiges Datenflussobjekt für eine Aufgabe benötigt wird. In allen drei Anwendungsdomänen scheint die Verwendung von Aggregaten sinnvoll.

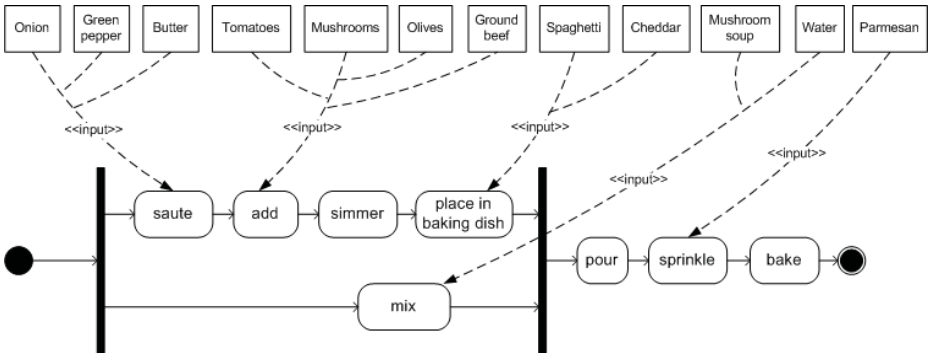


Abbildung 28 - Beispiel für die Integration des Datenflusses, Quelle: [MinorEtAl10b]

Abbildung 28 zeigt einen Kochworkflow mit Zutaten. Um den Datenfluss mit dem Kontrollfluss zu verbinden, erhält jeder Task einen Input- und Outputcontainer (vgl. [MinorEtAl10a]). Der ***Inputcontainer*** beinhaltet Referenzen auf Datenobjekte, die für die Ausführung des Tasks notwendig sind. Der ***Outputcontainer*** beinhaltet Referenzen auf

aggregierte oder transformierte Datenobjekte, die durch die Ausführung des Tasks entstanden sind. Um die Datenobjekte nicht gesondert zum Kontrollfluss erfassen zu müssen, verfügt das neue Retrieval Format über einen dynamischen[16] ***Datenkontext***, der sowohl atomare als auch aggregierte Datenobjekte verwalten kann. Bezogen auf die Kochdomäne handelt es sich bei den Datenobjekten um Zutaten. Zur beispielhaften Visualisierung der Verbindung von Datenfluss und Kontrollfluss werden in Abbildung 28 gestrichelte Kanten zwischen den Tasks und den Datenobjekten verwendet. Gestrichelte Kanten, die in einen Task hineinführen, kommen von Datenobjekten, die als Input dienen, während ausgehende, gestrichelte Kanten zu Datenobjekten führen, die als Output erzeugt wurden. Abbildung 28 zeigt ein vereinfachtes Beispiel, das ausschließlich Inputdatenobjekte verwendet.

Wollte man den Ablauf vollständig korrekt modellieren, so müsste sehr viel mehr in die Modellierung von Workflows mit aufgenommen werden, was nicht unbedingt einen Mehrwert für das Verständnis des Workflows bedeutet. Letzten Endes soll ein Task so beschrieben werden können, dass er von einem Menschen ausgeführt werden kann. Aus diesem Grund wurden die Kochworkflows zur Evaluierung des Systems nur mit Inputdatenobjekten modelliert, was selbstverständlich auch eine Vereinfachung des Adaptionsszenarios ist. In späteren Experimenten könnte die Modellierung auf die Nutzung von Aggregaten ausgeweitet werden. Für die erste formative Untersuchung wurde allerdings darauf verzichtet, da in Abschnitt 5.1.2 festgestellt wird, dass auch die textuellen Beschreibungen von Kochrezepten sehr selten Aggregate erwähnen [MinorEtAl10b].

4.2.2 Die ADD- und DELETE-Listen

Wie bereits im vorherigen Abschnitt erwähnt, könnten die ADD- und DELETE-Listen auch automatisch aus dem ursprünglichen und dem adaptierten Workflow berechnet werden. Eine andere Möglichkeit wäre es die manuellen Änderungen des Workflowmodellierers aufzuzeichnen und aus diesen Schritten ADD- und DELETE-Listen zu erzeugen. Ein solches Monitoringtool existiert zurzeit jedoch nicht und es ist unwahrscheinlich, dass auf diese Weise erzeugte Listen zu besseren Resultaten führen würden, als die Berechnung der Listen als Differenz zwischen dem ursprünglichen und dem adaptierten Workflow. Manuell ausgeführte Adaptionen sind nicht immer von vornherein zielgerichtet und würden zu „verschmutzten“ ADD- und DELETE-Listen führen.

[16] "dynamisch" ist hier im Sinne von "nicht-statisch" und somit veränderbar gemeint.

Abbildung 29 zeigt einen Fall einer einfachen Workflow Adaption. Der hier betrachtete Workflow beschreibt einen Geschäftsprozess, in dem einem Kunden oder

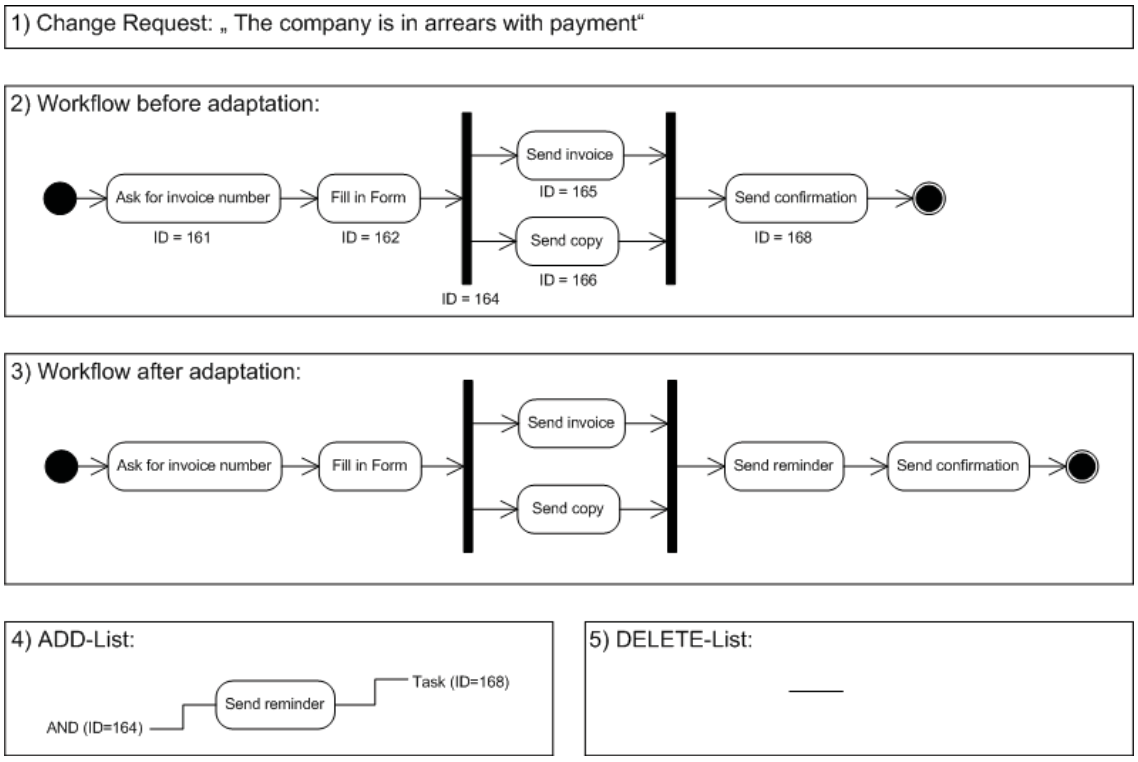


Abbildung 29 - Beispielfall "Schicken einer Rechnung" (send an invoice), Quelle: [MinorEtAl10a]

Geschäftspartner etwas in Rechnung gestellt wird. Der dazugehörige ursprüngliche Workflow ist in Box (2) als UML Aktivitätsdiagramm dargestellt. Kurz vor Ende der Workflowausführung ereignete es sich, dass der Empfänger der Rechnung in Zahlungsverzug kam. Dies wird in Box (1) mit dem Change Request („the company is in arrears with payment“ / „Die Firma ist mit der Zahlung im Verzug“) in textueller Form verdeutlicht. Der ursprüngliche Workflow wurde daraufhin angepasst und dem Empfänger der Rechnung wurde eine Mahnung geschickt („send reminder“). In Box (3) ist der adaptierte Workflow zu sehen, in den der Task „send reminder“ (Mahnung schicken) eingefügt wurde. In Box (4) und (5) sind die entsprechenden ADD- und DELETE-Listen dargestellt, die auf dem ursprünglichen Workflow angewendet wurden. In der Menge der ADD-Listen existiert eine Liste, die aus dem Task „send reminder“ und zwei Ankern besteht. Der „pre anchor“ ist das AND-Element aus dem ursprünglichen Workflow mit der ID 164. Der „post anchor“ ist der Task „send confirmation“ mit der ID 168. Die Menge der DELETE-Listen ist in diesem Beispiel leer.

Jede ADD- und DELETE-Liste besteht aus einer geordneten Menge aus verketteten Workflowelementen [MinorEtAl10a]. Eine solche ***Kette*** beinhaltet die maximale Menge verbundener Workflowelemente mit einem Eingangspunkt und einem Ausgangspunkt. Darüber hinaus besitzt jede Kette ein ***Paar aus Ankern***. Der ***„pre anchor“*** (Vorgängeranker) ist das vorangehende Workflowelement aus dem ursprünglichen Workflow (der in einem Adaptation Case als Retrieval Workflow repräsentiert wird), hinter dem die Workflowelemente, die sich in der Kette befinden, hinzugefügt oder gelöscht werden. Der ***„post anchor“*** (Nachfolgeranker) ist das Workflowelement aus dem ursprünglichen Workflow, das hinter dem letzten Element der Kette steht. Somit beschreibt der pre anchor die Position im Workflow, an der die Adaption beginnt und der post anchor bestimmt die erste Position im ursprünglichen Workflow, die nicht mehr von der Adaption betroffen ist. Diese Anker werden später in der Reuse-Phase der Workflow Adaption genutzt, um in einem neuen Workflow die Positionen zu identifizieren, an denen die Änderungen angewendet werden sollen. Dazu wird zwischen den Ankern und den Elementen des neuen Workflows die Ähnlichkeit berechnet.

4.3 Der Adaptionsprozess

In diesem Abschnitt wird der Adaptionsprozess nicht aus der Sicht eines Benutzers beschrieben, sondern aus der Sicht der CAKE III Plattform, wobei insbesondere die Aufgaben des Adaptation Managers herausgestellt werden. Dabei wird auch darauf eingegangen, wie der Ad-hoc Mechanismus, der Teile eines Workflows suspendieren kann, genutzt werden kann, um die Ketten aus den ADD- und DELETE-Listen auf eine laufende Instanz anzuwenden. Für die Retrieve-Phase von Adaptation Cases wird am Ende ein erster Algorithmus präsentiert.

4.3.1 Der Adaptionszyklus

Eine automatische Adaption kann beliebig oft auf einen Workflow angewendet werden, bis iterativ der gewünschte Sollzustand erreicht ist. Aus diesem Grund kann die Workflowadaption als ein zyklischer Prozess beschrieben werden. In Minor et al. [MinorEtAl10a] wird dieser Zyklus generisch beschrieben in Anlehnung an den bereits vorgestellten CBR-Zyklus (vgl. Abschnitt 1.3.2). Der hier beschriebene Zyklus orientiert sich an der Struktur der CAKE Systeme und lässt sich in neun Schritte unterteilen. Der Zyklus war der Ausgangspunkt um die Interaktion der CAKE Systeme für die CAKE III Plattform zu entwickeln, die in Abschnitt 5.4 erläutert wird. Abbildung 30 illustriert den Zyklus, dessen Schritte nun erläutert werden:

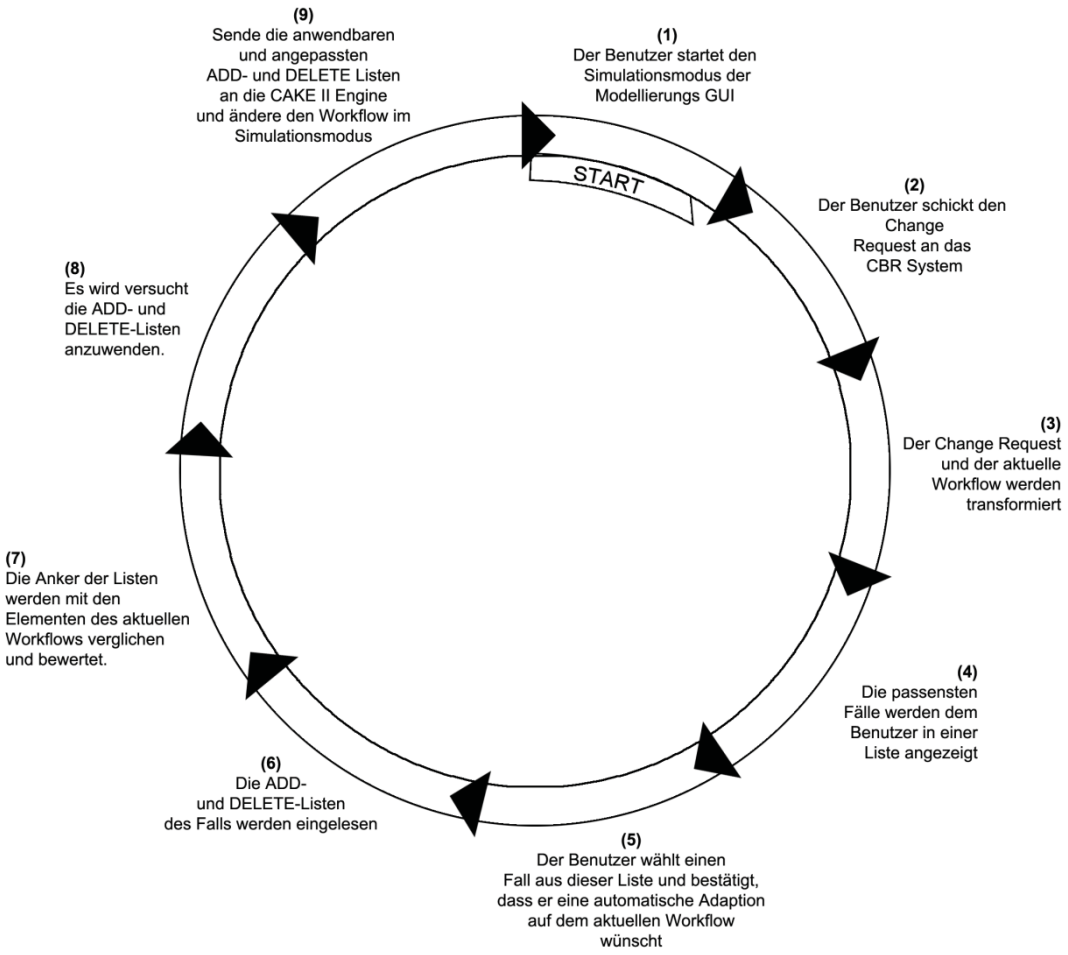


Abbildung 30 - Der Zyklus einer vollautomatischen Adaption, Quelle: eigene Erstellung

(1) An dieser Stelle startet der Adaptionszyklus. In Abschnitt 3.2.1 wurde bereits erwähnt, dass eine webbasierte Modellierungs-GUI entwickelt wird. Es ist angedacht, dass diese GUI über einen Simulationsmodus verfügt. In diesem Modus werden Änderungen an der aktuell geöffneten Instanz oder Definition nicht direkt übertragen, sonder nur simuliert. Wenn der Benutzer eine halb- oder vollautomatische Adaption wünscht, startet er zunächst den Simulationsmodus der GUI, da die vom System vorgeschlagenen Änderungen immer einer Endkontrolle durch den Benutzer unterliegen sollten, bevor sie auf den richtigen Workflow übertragen werden.

(2) Der Benutzer sendet seinen Change Request an das System. Dazu formuliert er textuell einen Satz, der das Problem beschreibt, welches eine Exception (siehe Abschnitt 1.3.1) verursacht hat. Das System erhält dann als Input den Change Request und den Workflow aus dem Simulationsmodus.

(3) Der Change Request und der übermittelte Workflow müssen vom System transformiert werden, damit ein Retrieval ermöglicht wird. Der Change Request sollte in ein zuvor spezifiziertes Datenmodell umgewandelt werden, das im einfachsten Fall aus einem Attribut vom Typ String besteht. Der Workflow aus dem Simulationsmodus wird in diesem Schritt in das Retrieval Format umgewandelt. Aus diesen zwei Attributen wird ein Adaptation Case generiert, der als Anfrage dient.

(4) Das System führt jetzt ein Retrieval auf der Fallbasis aus. Dazu wird jeder Fall aus der Fallbasis hinsichtlich seiner Ähnlichkeit zu der Anfrage bewertet. Die daraus resultierende Ordnung kann als Liste vollständig oder teilweise an die Modellierungs-GUI geschickt werden[17].

(5) Der Benutzer wählt aus dieser Liste einen Adaptation Case aus und wählt zwischen einer halbautomatischen Adaption, die ihm vergangene Workflows präsentieren würde, und einer vollautomatischen Adaption. Wählt der Benutzer hier die halbautomatische Adaption, so ist der Zyklus mit der Präsentation der adaptierten Workflows aus den Fällen beendet. Die Schritte (6) bis (9) spielen nur für die vollautomatische Adaption eine Rolle.

(6) Der für die vollautomatische Adaption ausgewählte Adaptation Case wird vom System eingelesen. Dazu gehören insbesondere die ADD- und DELETE-Listen. Das System wählt einen Adaptionsalgorithmus (siehe Abschnitt 4.4.2) und startet den Adaptionsprozess.

(7) Die Ketten in den Listen werden einzeln betrachtet und die pre und post anchors der Ketten werden Elementen aus dem neuen Workflow zugeordnet. Dieser Schritt wird als ***„Mapping“*** bezeichnet und stellt den wesentlichsten Schritt im Adaptionsprozess dar. Abschnitt 4.4 beschäftigt sich ausführlich mit dem Mapping von Ankern.

(8) Wenn die Anker erfolgreich zugeordnet wurden und damit eine Position im neuen Workflow gefunden wurde, die adaptiert werden kann, muss entschieden werden, ob eine Kette ganz oder nur partiell angewendet wird. Zurzeit werden Ketten nur ganz oder gar nicht angewendet.

(9) Schlussendlich bleibt eine Menge von ADD- und DELETE-Listen übrig, deren Ketten gemappte Anker enthalten. Die Workflowfragmente innerhalb der Ketten werden an ein WfMS gesendet, das die Logik beinhalten muss, um die Modifikationen an einem simulierten Workflow vorzunehmen. Die Hauptaufgabe des WfMS ist es hier die Einfüge- und Löschoperationen so durchzuführen, dass der Workflow seine Konsistenz am Ende nicht verloren hat.

[17] Eine mögliche Beschränkung wäre bspw. das Festsetzen einer Ähnlichkeitsschwelle, die ein Fall überschreiten muss.

4.3.2 Setzen von Breakpoints

Jede Ad-hoc Änderung an einem laufenden Workflow setzt voraus, dass ein Breakpoint vor den betroffenen Teil des Workflows gesetzt wird. Nur in suspendierten Teilgraphen des Workflows darf während der Ausführung einer Instanz etwas verändert werden. Das Einfügen, Löschen oder Ändern von Elementen durch eine automatische Adaption unterliegt den gleichen Einschränkungen. Folgerichtig muss an einer Stelle des Adaptionsprozesses mittels Breakpoints dafür gesorgt werden, dass die Konsistenz des Workflows gewahrt bleibt. Während des Entwurfs des Adaptionszyklus wurden drei unterschiedliche Lösungsansätze identifiziert.

- Die erste Möglichkeit würde alle aktiven Sequenzen suspendieren, sobald der Simulationsmodus gestartet wird. Dadurch wäre sichergestellt, dass sich der Ausführungsstatus des Workflows nicht ändert, während der Modellierer mit Änderungen am Workflow experimentiert. Dies ist ein sehr restriktiver Unterbrechungsmechanismus.
- Die zweite Möglichkeit setzt einen Breakpoint vor jede Kette, die vor ihrer Anwendung steht. Falls ein Breakpoint nicht an die Stelle gesetzt werden kann, die durch die Anker der Kette festgelegt ist, so wird die automatische Adaption abgebrochen und bereits durchgeführte Änderungen werden wie im Sinne einer Transaktion rückgängig gemacht.
- Eine dritte Möglichkeit wäre es, dem Workflowmodellierer das Setzen der Breakpoints zu überlassen. Der Modellierer würde vom System die Stellen gekennzeichnet bekommen, die durch einen Breakpoint blockiert werden müssen.

Die erste Möglichkeit einer restriktiven Unterbrechung steht den Anforderungen einer automatischen Adaption entgegen. Die Nebenläufigkeit eines Workflows würde unterbunden werden und Workflowteilnehmer, die auf Daten des Workflows zurückgreifen, wären in ihrer Arbeit blockiert. Die dritte Möglichkeit ist ein Spezialfall der zweiten Möglichkeit. In beiden Fällen werden die Punkte, an denen die Breakpoints gesetzt werden müssen, vom Adaptation Manager berechnet. Der Unterschied besteht nur darin, wer die Breakpoints setzt. Der Vorteil der dritten Möglichkeit besteht darin, dass der Benutzer sich den Workflow aus dem Fall noch ansehen könnte, um daraus weitere Rückschlüsse auf die Positionierung der Breakpoints zu ziehen. Da aber eine vollwertige automatische Adaption angestrebt wird, wird im System die zweite Möglichkeit verwendet. Dabei wird in Kauf genommen, dass eine Adaption nicht anwendbar sein könnte. Beim Entfernen der automatisch eingefügten Breakpoints muss darauf geachtet werden, dass nur die vom System generierten Breakpoints entfernt werden und Breakpoints, die eventuell schon vorher gesetzt wurden, erhalten bleiben.

4.3.3 Das Retrieval von Adaptation Cases

Das Retrieval von Adaptation Cases benötigt unterschiedliche Ähnlichkeitsmaße für die Komponenten des Problemteils [MinorEtAl10a]. Obwohl das Retrieval nicht im Vordergrund dieser Arbeit steht, werden die dafür entwickelten Ähnlichkeitsmaße auch

$$sim_{wfl\_el}(x,y) = \begin{cases} sim_{task}(x,y) & , \quad x,y \text{ are tasks} \\ sim_{ctrl\_flow\_el}(x,y) & , \quad x,y \text{ are control flow elements} \\ 0 & , \quad else \end{cases}$$

Abbildung 31 - Das Ähnlichkeitsmaß für Workflows, Quelle: [MinorEtAl10b]

beim Mapping der Anker verwendet. Aus diesem Grunde wird im Folgenden das Ähnlichkeitsmaß beschrieben, das auf Workflows aus der Kochdomäne angewendet wurde (vgl. [MinorEtAl10b]). Während für die Ähnlichkeitsberechnung des Change Requests unterschiedliche Ähnlichkeitsmaße verwendet werden müssten, da dies vom verwendeten, domänenspezifischen Datenmodell abhängt, sollte für die Repräsentationsform des Retrieval Workflows ein neues Ähnlichkeitsmaß entwickelt werden, das auch auf andere Domänen übertragbar ist. Das neue Ähnlichkeitsmaß nutzt dazu die Struktur des Workflows um Workflowelemente hinsichtlich ihrer Ähnlichkeit zu bewerten. Das Ähnlichkeitsmaß für Workflowelemente vergleicht Tasks anhand ihrer Eigenschaften. Zu den Eigenschaften eines Tasks gehören: der Name, die Beschreibung, die Input- und Outputdatenobjekte. Ausgangspunkt für den Vergleich der Eigenschaften ist die Levenshtein-Distanz. Während für die Beschreibung und den Namen eines Tasks die Levenshtein-Distanz direkt angewendet werden kann, wird für die Menge der Input- und Outputdatenobjekte ein einfacher Hill-Climbing Algorithmus verwendet. Dazu werden von zwei Tasks jeweils für die Menge der Input- und Outputdatenobjekte Paare gebildet, sodass jedes Datenobjekt in genau einem Paar vorkommen darf. Die zwei so konstruierten Mengen von Paaren aus Input- und Outputdatenobjekten sollen in der Summe ihrer lokalen Ähnlichkeiten maximal sein. Dabei gilt, dass ein Paar entweder nur Input- oder nur Outputdatenobjekte beinhaltet. Es handelt sich hierbei um ein klassisches Optimierungsproblem, in der die optimale Paarkombination gesucht wird, sodass die Gesamtähnlichkeit maximal ist. Die lokale Ähnlichkeit der Datenobjekte wird anhand ihres Namens mit der Levenshtein-Distanz verglichen. Die übrigen Blockelemente (AND, XOR, etc.) berechnen ihre Ähnlichkeit zueinander ebenfalls mit einem Hill-Climbing Algorithmus, der statt Datenobjekten die Menge der Tasks in einem Block vergleicht. Bei einem Retrieval wird durch die Unterscheidung von Task- und Blockelementen die Ähnlichkeit zweier Workflowelemente rekursiv verglichen (vgl. Abbildung 31). Die Ähnlichkeit zwischen zwei Workflows ergibt sich aus der aggregierten Ähnlichkeit ihrer Workflowelemente.

Das aktuelle Ähnlichkeitsmaß betrachtet somit nur die syntaktische Ähnlichkeit von Workflowelementen und lässt die Semantik vorerst außer Acht. Beim Mapping von Ankern wird dasselbe Ähnlichkeitsmaß verwendet mit dem Unterschied, dass nicht Workflows, sondern nur die Anker mit den Workflowelementen miteinander verglichen werden. Für die Zukunft wird es notwendig sein bessere Ähnlichkeitsmaße zu entwickeln, die auch Metawissen durch Semantik, z. B. in Form einer Ontologie, mit in die Ähnlichkeitsberechnung mit einfließen lassen. Eine andere Möglichkeit für die Berechnung der Ähnlichkeit beim Retrieval könnte sich auf die Struktur der ADD- und DELETE-Listen stützen. Statt die Ähnlichkeit zwischen den zwei Workflows direkt zu berechnen, könnte die Übertragbarkeit der Anker auf den neuen Workflow als Ähnlichkeitskriterium verwendet werden.

4.4 Das Ankerprinzip

Die automatische Bestimmung der Stellen, die in einem Workflow zu ändern sind, kann beliebig komplex gestaltet werden. Die Struktur des Workflows bei der Positionsbestimmung zu nutzen, ist ein naheliegender Ansatz. In diesem Abschnitt wird das Ankerprinzip, das bereits in Abschnitt 4.2.2 vorgestellt wurde, näher beschrieben. Dazu wird das Lokalisierungsproblem für einen Anker bzw. für ein Ankerpaar näher beschrieben und das Konzept auf verschiedene Typen von Ketten erweitert. Anschließend werden Algorithmen für den Adaptionsprozess erörtert, die im Zuge dieser Arbeit entwickelt wurden. Die entwickelten Algorithmen werden in zwei Anwendungsdomänen getestet. Zum Schluss werden die daraus gewonnen Ergebnisse in einer formativen Evaluation dargelegt.

4.4.1 Das Lokalisierungsproblem

In einem manuellen Adaptionsszenario weiß der Workflowmodellierer an welcher Stelle eine Exception im Workflow auftritt. Die Hauptaufgabe des Workflowmodellierers besteht dann darin, herauszufinden, welche Änderungen durchgeführt werden müssen. Bei einer automatischen Adaption verhält es sich genau anders herum. Durch den Inhalt der Ketten ist bereits festgelegt, welche Elemente hinzugefügt werden sollen und welche zu löschen sind. Es ist die Hauptaufgabe des Algorithmus, die passenden Stellen im Workflow zu finden, an denen es sinnvoll ist, die Ketten einzufügen oder zu löschen. Dies ist das ***Lokalisierungsproblem***, das durch die Verwendung von ADD- und DELETE-Listen entsteht. In Abschnitt 4.2.2 wurden die pre anchor und post anchor vorgestellt, die auf

Workflowelemente gemappt werden, um das Lokalisierungsproblem zu lösen. Der Prozess des Mappings besteht seinerseits aus drei Schritten [MinorEtAl10a]:

1. Zunächst werden in dem zu adaptierenden Workflow die Workflowelemente gewählt, die für einen Anker als Kandidat in Frage kommen. Als Kandidat kommen nur Workflowelemente in Frage, die gerade aktiv (Status = „active“) sind oder für die Ausführung bereit (Status = „ready“) stehen. Es würde keinen Sinn machen Workflowelemente in Betracht zu ziehen, die in der Vergangenheit liegen und somit nicht mehr änderbar sind. Deswegen werden diese Elemente vorab herausgefiltert. Dieser erste Schritt kann übersprungen werden, wenn es sich bei dem zu adaptierenden Workflow um eine Definition[18] handelt oder um eine Instanz, die noch nicht gestartet wurde.
2. Im zweiten Schritt werden die so herausgefilterten Kandidaten bewertet. Für jeden einzelnen Anker einer Kette wird eine Menge ***valider Kandidaten*** bestimmt. Diese Bewertung geschieht mit einem Ähnlichkeitsmaß, wie es z. B. in Abschnitt 4.3.3 beschrieben ist, und spezifiziert mittels einer Gültigkeitsschwelle die Menge aller validen Kandidaten für einen Anker. Die Kandidaten für einen spezifischen Anker, die unter der Gültigkeitsschwelle liegen, werden verworfen.
3. Im dritten Schritt wird aus den zwei Mengen valider Kandidaten für ein Ankerpaar ein konsistentes Tupel gebildet. Dieses Tupel bestimmt die Position im Workflow, an der die Kette angewendet wird.

In Bezug auf den Kontrollfluss muss sichergestellt sein, dass das Tupel, welches die gemappten und validen Ankerelemente beinhaltet, konsistent ist. Ein Tupel ist ***konsistent***, wenn die gemappten Ankerelemente die topologische Ordnung des Workflows nicht verletzten, d. h. dass sich die gemappte Position des pre anchors im adaptierten Workflow vor der gemappten Position des post anchors befinden muss (vgl. [MinorEtAl10a]). Eine logische, daraus abgeleitete Konsequenz ist, dass die Kandidaten in einem Tupel sich in derselben Sequenz befinden müssen. Es ist nicht möglich, dass sich ein pre anchor in einer anderen Sequenz als der post anchor befindet. In Abschnitt 3.2.2 wurde die blockorientierte Modellierungssprache von CAKE II vorgestellt. Aufgrund der Blockorientierung dieses Ansatzes werden nur Tasks und die Blockelemente XOR, AND, etc. als Kandidaten für das Mapping beachtet. Der rekursive Algorithmus aus Abschnitt 4.3.3 hängt ebenfalls von diesem Umstand ab. Im Prinzip lässt sich das Ankerprinzip aber auch auf die Elemente übertragen, die einen Block begrenzen. Diese Elemente werden nach Aalst in [vdA03] als “Join”- und “Split”-Elemente bezeichnet. So wird das Element, das eine Sequenz in parallele Sequenzen aufteilt, als AND-Split und

[18] In einer Workflow Definition besitzen alle Workflowelemente den Status "ready".

das Element, das parallele Sequenzen synchronisiert, als AND-Join bezeichnet. Ebenso könnten der Startpunkt und der Endpunkt eines Workflows als Kandidat für ein Mapping dienen. Auf der Implementierungsebene wurde auf die Modellierung dieser Elemente jedoch verzichtet (ebenso wie im Retrieval Format für Workflows), da durch die Blockorientierung auf eine Baumstruktur zurückgegriffen werden kann. Die Baumstruktur erlaubt es für jedes Workflowelement das Elternelement und somit die Tiefe im Baum zu bestimmen. Diese Modellierungsentscheidung wird bedeutsam, sobald,

a. eine Kette als neue Sequenz in einen Block eingefügt oder eine ganze Sequenz in einem Block gelöscht werden soll. In einem solchen Fall wäre bei einem AND Block der pre anchor ein AND-Split und der post anchor ein AND-Join.
b. eine Kette an den Anfang einer Sequenz eingefügt oder am Anfang gelöscht werden soll. Dabei könnte der pre anchor der Startpunkt des Workflows, aber auch ein “Split”-Element eines Blocks sein.
c. eine Kette an das Ende einer Sequenz angefügt oder am Ende gelöscht werden soll. Hier könnte der post anchor das Endelement des Workflows sein oder ein “Join”-Element eines Blocks.

Auf der technischen Ebene werden bei diesen Sonderfällen die betroffenen Anker auf “null” gesetzt. Die Algorithmen nutzen die Baumstruktur, um zu bestimmen, welche Sequenz genutzt werden darf, um ein konsistentes Tupel zu erzeugen. Durch die Untersuchungen von Kochrezepten wurde der Datenfluss in die Überlegungen zur Adaption von Workflows mit einbezogen. Es stellte sich heraus, dass das Ankerprinzip und das damit verbundene Lokalisierungsproblem auf einen Datenfluss übertragbar ist (vgl. [MinorEtAl10b]). Neben den Workflowelementen werden die Ketten auf Datenobjekte und ***“Datalinks”*** erweitert. Das erweiterte Konzept kennt nun drei unterschiedliche Formen von Ketten:

(Typ 1) Die bisher vorgestellten Ketten, die den Kontrollfluss des Workflows verändern.

(Typ 2) Ketten, die Datenobjekte im Datenkontext des Workflows ändern und

(Typ 3) Datalinks die Datenobjekte mit Workflowelementen verbinden.

Die nächsten Unterabschnitte beschäftigen sich mit spezifischen Eigenschaften dieser Ketten in ADD- und DELETE-Listen. Des Weiteren wird erörtert, warum die ADD- und DELETE-Listen genügen, um jede Änderungsoperation am Workflow zu beschreiben und auf einen Modifizierungsoperator verzichtet wurde.

Hinzufügen von Workflowelementen und Datenobjekten

Werden Elemente in einen Workflow eingefügt, so darf dies nicht in beliebiger Reihenfolge geschehen. Datalinks müssen immer als Letztes eingefügt werden, da es möglich ist, dass sie sich auf Datenobjekte oder Workflowelemente beziehen, die erst später in den Workflow eingefügt werden. Würden Datalinks als Erstes in einen Workflow eingefügt, so wäre das Mapping entweder suboptimal oder nicht anwendbar: Suboptimal genau dann, wenn nicht der optimale Kandidat gefunden werden konnte, da er noch nicht in den Workflow eingefügt wurde und nicht anwendbar, falls es keinen Kandidaten gibt, der die Gültigkeitsschwelle überschreitet. In einer Datalink-Kette, die eine Inputverbindung zwischen Datenobjekt und Task darstellt, besteht der pre anchor aus einem Datenobjekt und der post anchor aus einem Workflowelement. Bei einer Datalink-Kette, die eine Outputverbindung darstellt, verhält es sich bei den Ankern umgekehrt. Die eigentliche Kette eines Datalinks ist immer leer. Ob am Anfang Ketten mit Workflowelementen oder Datenobjekten zuerst eingefügt werden, ist beliebig. Das Einfügen von Datenobjekten gestaltet sich recht einfach. Für jedes einzufügende einfache oder aggregierte Datenobjekt existiert genau eine Kette, deren pre und post anchor auf „null“ gesetzt werden, da es einfach in den Datenobjektkontext eines Workflows eingefügt werden kann.

Löschen von Workflowelementen und Datenobjekten

Das Löschen von Workflowelementen und Datenobjekten muss in einer anderen Reihenfolge ablaufen als das Hinzufügen. Es würde zu Inkonsistenzen führen, wenn ein Datenobjekt oder ein Workflowelement gelöscht wird, das noch Teil eines Datalinks ist. Neben der Reihenfolge der Löschoperation spielt das Mapping der Anker bei Ketten, die den Kontrollfluss verändern, eine andere Rolle. Während in Ketten aus ADD-Listen die Anker genutzt werden, um die Position zu bestimmen, an der die Kette eingefügt werden soll, müssen die Elemente in einer Kette aus einer DELETE-Liste gelöscht werden. Das Mapping für solche Ketten konzentriert sich auf die Elemente in der Kette und nicht auf die Anker. Die Position der Elemente, die gelöscht werden sollen, wird über das Mapping der Elemente innerhalb der Kette bestimmt. Die Anker solcher Ketten sind nur ein zusätzliches Kriterium, um vorschnelle Löschoperationen zu verhindern. Generell ist das Löschen von Elementen immer als kritischer anzusehen, als das Hinzufügen. Daher sind die späteren Gültigkeitsschwellen in den Algorithmen bei Löschoperationen restriktiver als bei Einfügeoperationen.

Modifikation von Workflowelementen und Datenobjekten

Änderungen an einem Workflowelement oder einem Datenobjekt können sich ebenso an den Eigenschaften der Objekte vollziehen. Bei einem Task kann bspw. die Beschreibung oder die Deadline verändert werden, bei einem Datenobjekt kann eine Maßeinheit verändert werden. Diese Änderungen müssen genauso wie strukturelle Änderungen erfasst werden. Eine Möglichkeit besteht darin neben den ADD- und DELETE-Listen eine neue Liste einzuführen, die während der Differenzberechnung des ursprünglichen und des adaptierten Workflows als Modifikationsoperator explizit erfasst wird. So würde jede Änderung an den Eigenschaften eines Workflowelements oder eines Datenobjekts in einer MODIFY-Liste erfasst. Diese Offlineberechnung einer Modifikation kann jedoch auch online mit Hilfe der ADD- und DELETE-Listen geschehen. Jede Änderung der Eigenschaften an einem Workflowelement oder Datenobjekt sorgt dafür, dass das ganze Objekt zunächst gelöscht und anschließend neu eingefügt wird. Bei einer solchen Substitution existiert in der ADD- und DELETE-Liste jeweils eine Kette, die über dasselbe Ankerpaar verfügt. Anhand der IDs, die für jedes Ankerelement gespeichert werden, kann für eine Kette überprüft werden, ob es sich um ein identisches Ankerpaar handelt. Aktuell ist eine solche Substitutionserkennung bei Datenobjekten nicht möglich, da sie ohne Anker in Fällen gespeichert werden. Die ersten Experimente konnten jedoch auch ohne diese spezielle Substitutionserkennung bei Datenobjekten durchgeführt werden. Wurden in einem Kochworkflow Zutaten, die als Datenobjekte repräsentiert werden, substituiert, so wurde bei der Differenzberechnung erkannt, dass das Datenobjekt und seine dazu gehörigen Datalinks entfernt wurden, während auf der anderen Seite ein neues Datenobjekt mit Datalinks eingefügt wurde. In zukünftigen Modellierungen könnte der gesamte Datenkontext eines Workflows als Anker für Datenobjekte genutzt werden. Wird ein Datenobjekt bearbeitet oder substituiert, würde es einen Anker auf der ADD- und DELETE-Liste geben, der den Datenkontext des Workflows ohne das betroffene Datenobjekt beinhaltet. Für einzufügende oder zu löschende Datenobjekte würde der Datenkontext im Anker nicht vom Datenkontext im Workflow abweichen.

4.4.2 Adaptionsalgorithmen

Die Anker der ADD- und DELETE-Listen können auf unterschiedliche Art und Weise genutzt werden, um das Lokalisierungsproblem zu lösen. Die für diese Arbeit entwickelten Algorithmen, die in [MinorEtAl10a] und [MinorEtAl10b] miteinander verglichen wurden, unterscheiden sich in vier Varianten. Das Bestimmen der Kandidaten und die Überprüfung ihrer Gültigkeit läuft in allen Algorithmen gleich ab. Die Algorithmen unterscheiden sich lediglich im dritten Schritt der Adaption. Das Bilden konsistenter Tupel für das Ankermapping geschieht auf unterschiedliche Weise:

- ***PRe Anchor Mapping (PRAM)***: Der post anchor einer Kette wird komplett ignoriert, während für jeden pre anchor aus der Menge der Kandidaten der am besten passende ausgewählt wird. Falls ein Kandidat für mehrere pre anchors als bester Kandidat gewählt wurde, so wird einem beliebigen pre anchor dieser Kandidat zugeteilt, während die anderen pre anchors auf die nächstbesten Kandidaten gemappt werden.
- ***POst Anchor Mapping (POAM)***: Der pre anchor einer Kette wird ignoriert, während die post anchor analog zur PRAM Methode behandelt werden.
- ***Composite Anchor Mapping (CAM)***: Das Mapping wählt sowohl Kandidaten aus der Menge konsistenter Paare als auch einzelne Kandidaten, wenn dies sinnvoll erscheint. Dazu wird die Menge konsistenter Paare in zwei Mengen aufgeteilt. Während die eine Menge die „tight pairs“ (dt. „enge Paare“) bilden, wird der Rest als „non-tight pairs“ (dt. „nicht-enge Paare“) bezeichnet. Ein tight pair zeichnet aus, dass der Kandidat des pre anchors der direkte Vorgänger des post anchor Kandidaten im zu adaptierenden Workflow ist. Die tight pairs dienen als Kandidatenmenge für Anker auf der ADD-Liste, während die non-tight pairs normalerweise für die Anker auf der DELETE-Liste als Kandidaten zur Verfügung stehen. Dies liegt daran, dass die Elemente einer zu löschenden Kette sich zwischen den Ankern befinden und auch gemappt werden müssen. Der Abstand ist der Grund, warum Anker einer DELETE-Liste in der Regel keine direkten Nachbarn sind. Danach werden die beiden Kandidatenmengen nach der kumulierten Ähnlichkeit jedes Tupels sortiert. Das Paar mit der höchsten Ähnlichkeit wird nun verglichen mit den Ähnlichkeitswerten einzelner Kandidaten, die nach der PRAM und POAM Methode berechnet wurden. Schlussendlich wird das Paar oder der einzelne Kandidat mit der höchsten Ähnlichkeit für das Mapping genommen. Falls ein Kandidat mehrmals ausgewählt wurde, wird dieser Kandidat einem beliebigen Ankerpaar zugewiesen und die übrigen Ankerpaare werden auf das nächstbeste Paar oder einzelnen Kandidaten gemappt.
- ***Maximum Anchor Mapping (MAM)***: Diese Methode unterscheidet sich nur wenig von der CAM Methode. Die Berechnung der Ähnlichkeit der Kandidatenpaare erfolgt hier nicht kumulativ, sondern ist durch eine Maximumsfunktion ersetzt.

Ohne das Ergebnis der Evaluation vorwegzunehmen, sei an dieser Stelle schon erwähnt, dass die CAM Methode zum besten Adaptionsergebnis in beiden Testdomänen führte. Da der Algorithmus komplex ist und er zumal für einen Überblick zu groß ist (933 Zeilen), wird er später in Abschnitt 5.5 mittels Pseudocode erklärt.

4.4.3 Beispiel einer Workflow Adaption

Um das Ankerprinzip und seine Ketten besser zu verstehen, folgt nun ein Beispiel aus der Kochdomäne. Das Beispiel stammt aus [MinorEtAl10b] und liegt hier in einer leicht

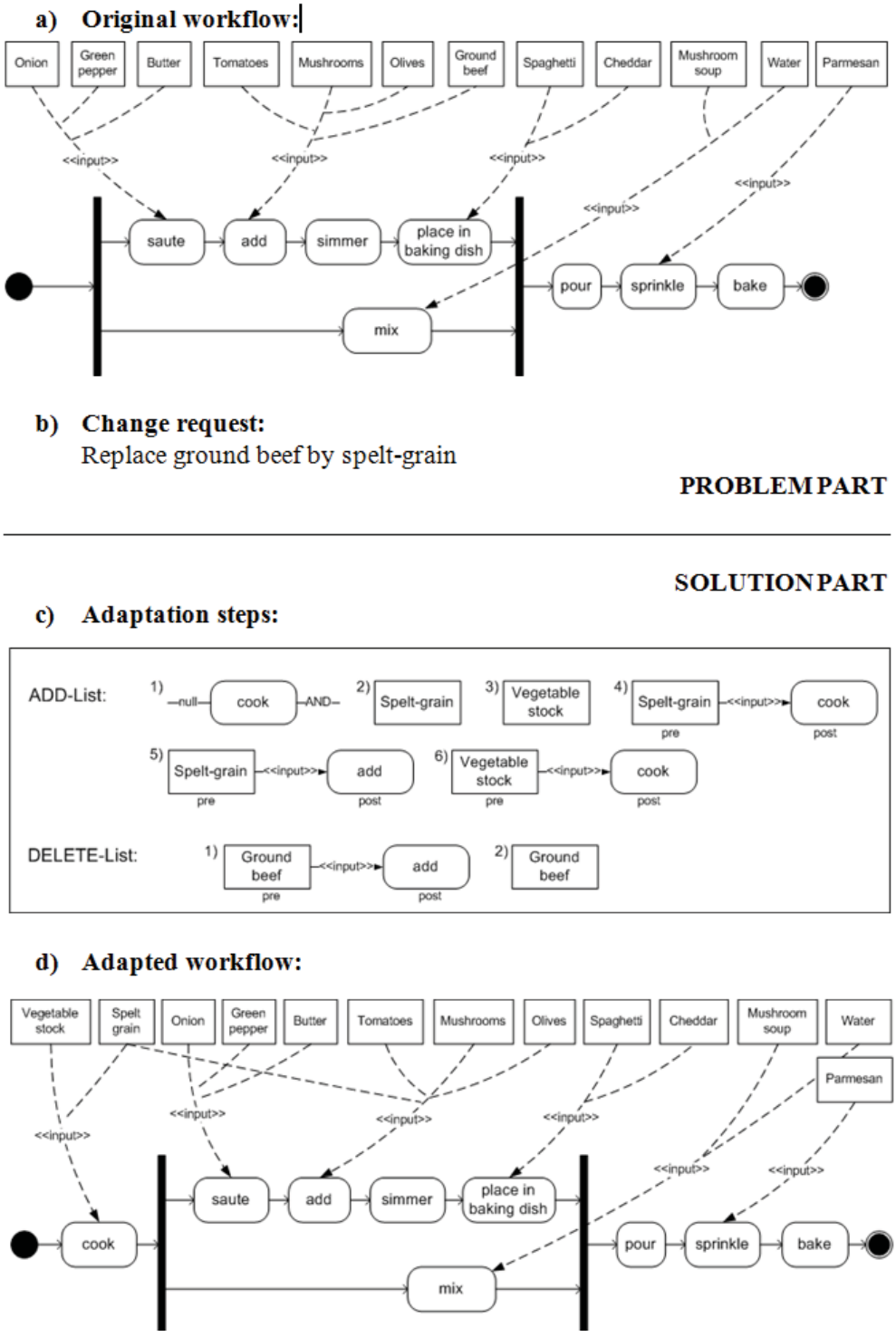


Abbildung 32 - Beispielfall "überbackene Spaghetti", Quelle: [MinorEtAl10b]

modifizierten Version vor, da bereits Änderungen am Adaptionsalgorithmus berücksichtigt wurden, die zu einem besseren Adaptionsresultat führen.

Abbildung 32 zeigt einen beispielhaften Adaptation Case, in dem überbackene Spaghetti zubereiten werden. Der Change Request in diesem Fall besteht darin, dass das Hackfleisch durch Grünkern ersetzt werden soll, um ein vegetarisches Gericht zu erhalten (engl. „Replace ground beef by spelt-grain“). Im Lösungsteil (engl. „solution part“) befinden sich die ADD- und DELETE-Listen, die den ursprünglichen Workflow (a) in den adaptierten Workflow (d) überführen.

Wie bereits in Abschnitt 4.2.1 beschrieben, ist die Workflowmodellierung eine individuelle Entscheidung des Modellierers. Es ist jedoch wichtig, dass der Workflow von seinen Teilnehmern verstanden und umgesetzt werden kann. Kochworkflows werden aktuell in einer vereinfachten aber verständlichen Form modelliert, die auf Aggregate, Output-Datalinks und Input-Output-Verbindungen zwischen Tasks verzichtet. Die Datenobjekte in Abbildung 32 sind in rechteckigen Kästchen abgebildet und repräsentieren die Zutaten, die im Workflow gebraucht werden. Die mit „<<input>>“ gekennzeichneten Verbindungen der Datenobjekte zu Tasks entsprechen den Input-Datalinks.

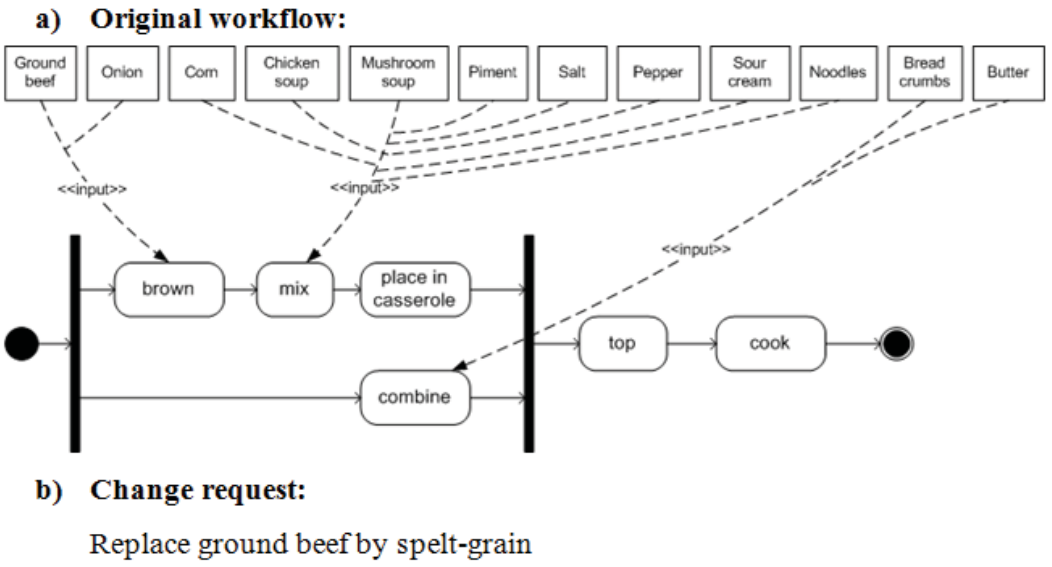


Abbildung 33 - Beispielanfrage "Nudelauflauf mit Fleisch", Quelle: [MinorEtAl10b]

Um die Adaption durchzuführen, genügt es nicht, dass das Datenobjekt Hackfleisch (engl. „ground beef“) durch ein Datenobjekt Grünkern (engl. „spelt-grain“) ersetzt wird. Der Grünkern muss zunächst in einer Gemüsebrühe (engl. „vegetable stock“) gekocht werden[19] bevor er weiterverarbeitet werden kann. Der Fall aus Abbildung 32 wurde ausgewählt, da er die höchste Ähnlichkeit zu der Anfrage aus Abbildung 33 aufweist. Bei der Anfrage handelt es sich um einen Nudelauflauf mit Fleisch (engl. „beef noodle casserole“). Genauso wie bei den überbackenen Spaghetti soll beim Auflauf das Fleisch durch Grünkern ersetzt werden[20].

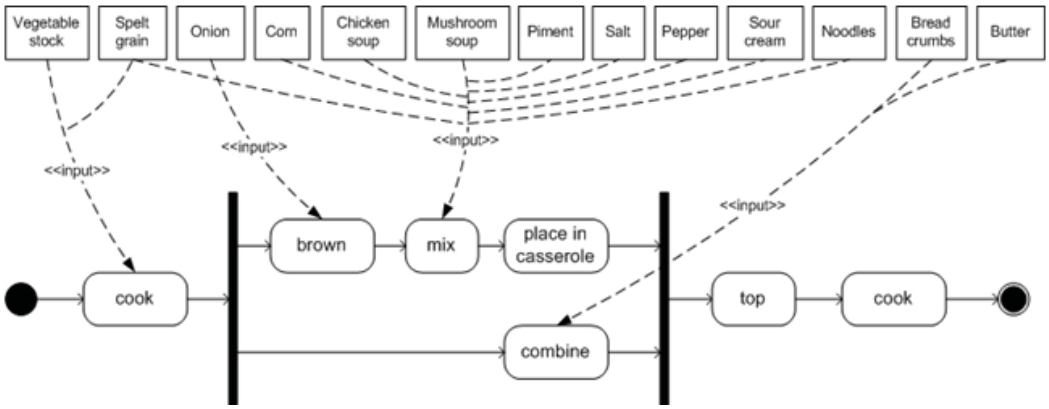


Abbildung 34 - Adaptionsergebnis "vegetarischer Nudelauflauf", Quelle: [MinorEtAl10b]

Dazu werden die ADD- und DELETE-Listen des Falls aus Abbildung 32 auf die Anfrage angewendet. Das Ergebnis der Adaption ist in Abbildung 34 zu sehen. Alle Ketten konnten erfolgreich auf die Anfrage übertragen werden. Dieses Ergebnis, welches auch Bestandteil der Evaluation war, konnte in einer erneuten Überprüfung der Adaption nicht reproduziert werden. Stattdessen wurde ein Ergebnis geliefert, in dem der Datalink der Gemüsebrühe oder des Grünkerns mit dem hinteren „cook“ verbunden wurde. Dies lässt sich zum Teil auf das einfache Levenshtein basierte Ähnlichkeitsmaß zurückführen, das nur die Syntax berücksichtigt, aber auch auf die Textquelle die Grundlage für die Modellierung des Workflows war. In dieser war das Kochrezept unterspezifiziert.

[19] Eine vollständige Modellierung müsste an dieser Stelle ein Aggregat erzeugen, das aus der Gemüsebrühe und dem Grünkern besteht und als "gekochter Grünkern" weiterverwendet werden kann. Anschließend müsste eine Verbindung geschaffen werden für das neu erzeugte Aggregat zwischen den Tasks "kochen" (engl. "cook") und "hinzufügen" (engl. "add").

[20] Für dieses Beispiel wäre auch ein abstrakterer Change Request denkbar. Der Wunsch nach einem vegetarischen Gericht oder einer Reduzierung der Kalorien würde zu ähnlichen Adaptionsketten führen.

4.5 Evaluation der Adaptionsalgorithmen

Das CAKE III System wurde für eine erste empirische Evaluation der vorgeschlagenen Algorithmen genutzt. Ziel der Untersuchung war es, herauszufinden, inwiefern Adaptation Cases einen neuen Workflow richtig anpassen können. Da nur die Adaptionsvarianten untersucht wurden, ist ausschließlich die Reuse-Phase Gegenstand der Evaluation. Fälle wurden manuell selektiert, um dann auf einem Workflow angewendet zu werden. Die Evaluation bezieht sich auf Kochprozesse [MinorEtAl10b] und Verwaltungsprozesse [MinorEtAl10a]. Die Auswahl der Verwaltungsprozesse geschah an der Universität Trier. Sie wurden in mehreren strukturierten Interviews mit der Unterstützung von Bürosekretärinnen erstellt [Ke09]. So konnten 19 typische Verwaltungsworkflows erfasst werden. Anschließend wurden die Interviewpartner zu jedem Workflow nach einem Change Request befragt, wie er üblicherweise auftreten kann und wie dieser zu lösen ist. Anhand der 19 Verwaltungsworkflows, der Change Requests und der Lösungsvorschläge wurde die Fallbasis für die Evaluation manuell erstellt. Die Kochprozesse basieren auf 39 Nudelgerichten aus der Computer Cooking Contest Datenbank von 2008[21]. Von diesen Nudelgerichten wurden sieben Rezepte als Basis für 30 Change Request genommen. Die Change Requests beruhen auf eigenen Erfahrungen. Die meisten Änderungen bei Kochrezepten beziehen sich dabei auf die Zutatenliste. Somit steht für die Experimente der Evaluation eine weitere Fallbasis mit 30 Kochrezepten zur Verfügung. Es wurden jeweils zwei Experimente für beide Fallbasen durchgeführt.

Das Ziel des ersten Experiments war festzustellen, ob die vorgeschlagenen Varianten zur Adaption in der Lage sind die adaptierten Workflows mit Hilfe der ADD- und DELETE-Listen nachzubilden. Für diese Rekonstruktion wurden die ursprünglichen Workflows aus den Fällen auf ihren eigenen Fall angewendet. Dies verlangt von den Algorithmen, dass sie das Lokalisierungsproblem lösen können und dass sie korrekt arbeiten.

Das zweite Experiment nutzt neu erstellte Anfragen, um die Algorithmen zu überprüfen. Für die Kochprozesse wurden 14 Change Requests zufällig aus der Fallbasis ausgewählt. Diese 14 Change Request wurden mit neuen Kochworkflows kombiniert, um neue Anfragen zu erstellen. Bei den Verwaltungsprozessen wurden die Anfragen in einer erneuten Interviewrunde mit Sekretärinnen erstellt. Es wurden so neun Anfragen erstellt, deren Workflows sich von denen aus der Fallbasis unterscheiden. Als Qualitäts-kriterium beider Experimente wurde die Anzahl der korrekt angewendeten, der falsch angewendeten und der gar nicht angewendeten Ketten sowohl in der ADD- als auch der DELETE-Liste gezählt.

[21] Ein Download der Kochrezepte ist unter http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc&act=5 verfügbar.

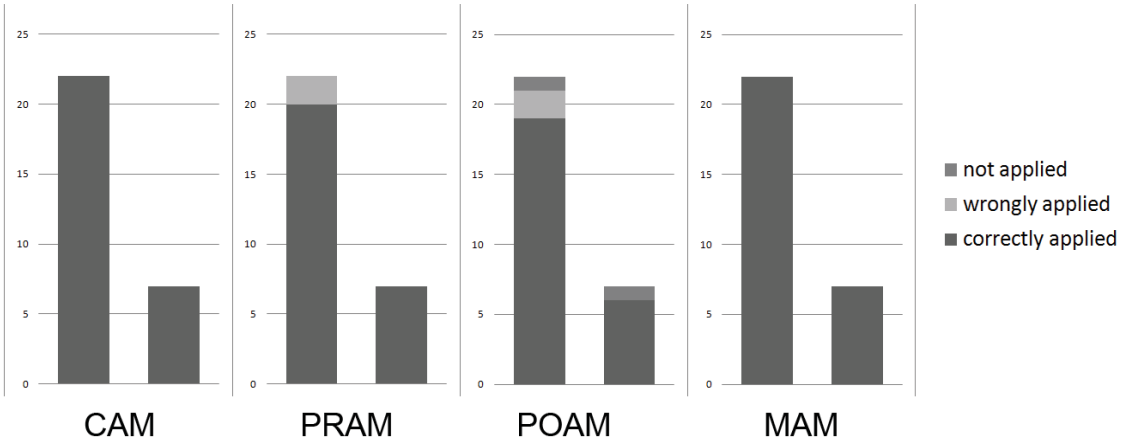


Abbildung 35 - Rekonstruktion der Verwaltungsprozesse, Quelle: [MinorEtAl10a]

Abbildung 35 zeigt das Resultat des ersten Experiments für die Verwaltungsprozesse. Für jede Variante des Algorithmus gibt es zwei Säulen. Die linke Säule zeigt das Resultat der insgesamt 22 Ketten aller ADD-Listen, die rechte Säule zeigt das Resultat der insgesamt sieben Ketten aller DELETE-Listen, die sich in den 19 Verwaltungsfällen befinden. Nur die CAM und MAM Methode waren in der Lage alle Adaptation Cases korrekt nachzubilden. Da der Datenfluss in dieser Evaluation noch nicht Bestandteil der Verwaltungsdomäne war, wurden die Ketten noch nicht differenziert betrachtet. Trotzdem kann bereits hier gezeigt werden, dass bei der Benutzung von nur einem Anker nicht alle Ketten richtig angewendet werden. Für die Kochdomäne zeigte sich kein differenziertes Bild bei der Rekonstruktion von Fällen, da alle Algorithmen vollständig richtig arbeiteten. Das liegt daran, dass die Ketten bei Kochprozessen hauptsächlich Datenobjekte und Datalinks bei Workflows ändern, aber keine Workflowelemente. Das Hinzufügen von Datenobjekten ist bspw. trivial gehalten, da es noch keine Bedingung gibt, die das Hinzufügen unterbindet.

Im zweiten Experiment wurde die Fähigkeit der vier Varianten dahingehend untersucht, wie gut neue Anfragen adaptiert werden können. Für die Verwaltungsprozesse, deren Ketten aktuell nur den Kontrollfluss ändern, wurde das beste Resultat erzielt, wenn die Gültigkeitsschwelle für Ketten der ADD-Liste auf 0.7 und für Ketten der DELETE-Liste auf 0.9 gesetzt wurde. Das Ergebnis für Verwaltungsprozesse ist in Abbildung 36 dargestellt. Das beste Resultat wird von der CAM Variante erzielt, welche 7 von 9 Ketten der ADD-Listen richtig adaptierte. Die beiden Ketten der DELETE-Liste wurden von allen Varianten richtig adaptiert. Für die Kochprozesse wurden mehrere Schwellen für die unterschiedlichen Kettentypen gesetzt. Da sich bei den Verwaltungsprozessen gezeigt hat, dass die CAM Variante am erfolgversprechendsten ist, wurde nur noch sie bei den Kochprozessen für das zweite Experiment verwendet.

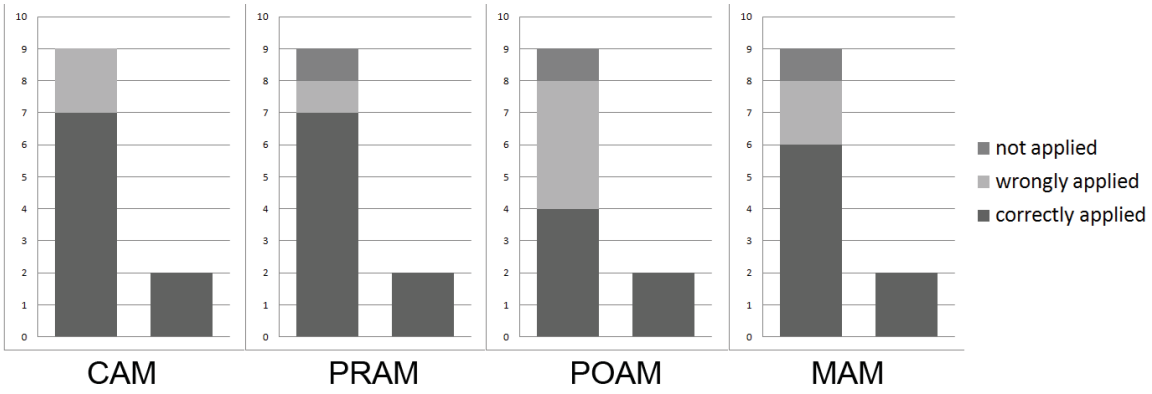


Abbildung 36 - Ergebnis der Verwaltungsworkflows, Quelle: [MinorEtAl10a]

Das beste Resultat mit der CAM Variante bei Kochprozessen wurde erzielt, wenn von der ADD-Liste Workflowelemente bei 0.6, Datenobjekte bei 0.1 und Datalinks bei 0.7 eingefügt wurden und von der DELETE-Liste Workflowelemente bei 0.9, Datenobjekte bei 0.8 und Datalinks bei 0.5 gelöscht wurden. Bei der Anwendung der 14 neuen Anfragen wurde keine Kette falsch adaptiert. Das Ergebnis ist in Abbildung 37 dargestellt. Zusammenfassend lässt sich sagen, dass für die Kochprozesse etwa die Hälfte der Anfragen richtig adaptiert werden konnte. Bei den übrigen Anfragen gab es Probleme bei der Anwendung von Listen mit Datalinks, die sich auf das verbesserungswürdige syntaktische Ähnlichkeitsmaß der Kandidatenbestimmung zurückführen lassen.

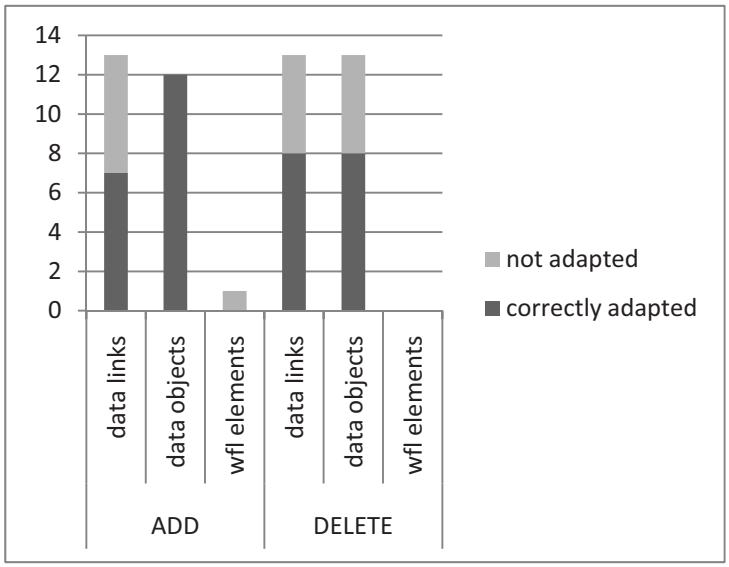


Abbildung 37 - Ergebnis der Kochworkflows, Quelle: [MinorEtAl10b]

4.6 Beurteilung und Zusammenfassung

Dieses Kapitel hat mit ersten Ergebnissen einer formativen Evaluation gezeigt, dass die automatische Adaption von Workflows möglich ist. Adaptation Cases beinhalten ein Workflowpaar und einen Change Request, der den Grund beschreibt, warum ein Workflow in den anderen transformiert werden musste. Die für die Transformation notwendigen Operationen werden in ADD- und DELETE-Listen erfasst, die aus Ketten und Ankern bestehen. Durch das Mapping von Ankern lassen sich die Ketten auf ähnliche Workflows mit ähnlichen Change Requests übertragen. Die ersten Experimente mit Adaptionsvarianten zeigten, dass die Verwendung von zwei Ankern zu besseren Resultaten führt, als die Verwendung von nur einem Anker. Die Experimente zeigten aber auch Schwächen bei den Ähnlichkeitsmaßen auf, die darauf zurückzuführen sind, dass sie nur die syntaktischen Gegebenheiten berücksichtigen. Um Synonyme oder zusammenhängende Begriffe zu erkennen, müsste ein Ähnlichkeitsmaß auf einen Thesaurus oder eine Ontologie zurückgreifen können. Die Entwicklung solcher Ähnlichkeitsmaße ist jedoch sehr aufwendig und von der Anwendungsdomäne abhängig. Es ist äußerst schwierig eine angemessene Anzahl von Workflows für unterschiedliche Anwendungsdomänen zu erfassen, um damit Experimente durchzuführen, da Experten zu Rate gezogen werden müssen, die die Ergebnisse validieren. Aus diesem Grunde sind Untersuchungen an Kochprozessen vorteilhaft. Neue Adaption Cases können auch von Laien erstellt und bewertet werden. Mit der Fertigstellung der CAKE III Plattform und der damit verbundenen Reife des Systems können dann in zukünftigen Untersuchungen andere Anwendungsdomänen analysiert werden, die schwieriger zu evaluieren sind. Zurzeit müssen alle Adaptation Cases manuell erzeugt werden. Für die zukünftige Weiterentwicklung des Konzepts kommen auch komplexere Anker während der Reuse-Phase in Betracht, die bspw. mehrere Workflowelemente oder den Datenfluss nutzen. Hinsichtlich des Kontroll- und Datenflusses sind Graphenmodelle denkbar, die nicht durch die Blockorientierung beschränkt sind. Bei einer Erweiterung auf solche Workflowrepräsentationen, muss sichergestellt werden, dass die angewendeten Ketten den Workflow immer in einen korrekten Zustand überführen.

Kapitel 5 - Umsetzung des Konzeptes

Dieses Kapitel beschreibt wie das in Kapitel 4 beschriebene Konzept technisch umgesetzt wurde. Vor der Implementierung neuer Softwarekomponenten, für die CAKE III Plattform, wurden zwei Untersuchungen durchgeführt. Während die eine die Retrievalgeschwindigkeit des CAKE I Systems abschätzen sollte, war die andere Grundlage für die Erstellung einer Fallbasis mit Kochworkflows und erste Untersuchungen am Datenfluss. Abgesichert durch diese Untersuchungen wurde das CAKE I System erweitert, um Retrieval und Reuse von Adaptation Cases zu ermöglichen. Das modifizierte CAKE I System wird während des Adaptionsprozesses vom Adaptation Manager genutzt. Dieser realisiert die im vorherigen Kapitel vorgestellten Schritte des Adaptionszyklus. Um die CAKE III Plattform ganzheitlich zu beschreiben, wird sowohl der Adaptation Manager mit seinen Komponenten als auch die Kommunikationswege der CAKE III Plattform vorgestellt. Zum Abschluss des Kapitels wird der CAM Algorithmus mit einem systemnahen Pseudocode nochmals ausführlich durchgesprochen, bevor im Anschluss eine Zusammenfassung mit persönlicher Beurteilung folgt.

5.1 Voruntersuchungen

Vor Beginn der Implementierungsphase wurden zwei Untersuchungen durchgeführt. Das Ziel der ersten Untersuchung ist es sicherzustellen, dass das CAKE I System performant genug ist, um auch ein Retrieval auf komplexen Strukturen wie Workflows durchzuführen. Diese Frage musste geklärt werden, da in einem Adaptation Case der unveränderte Workflow zum Problemteil gehört und somit beim Retrieval verwendet werden muss. Das Ziel der zweiten Untersuchung ist es herauszufinden, wie der Kontrollfluss mit einem Datenfluss verbunden werden kann. Das war erforderlich, da das CAKE II System bisher keinen Datenfluss in sein Workflowmodell integriert hat. Die CAKE III Plattform soll jedoch mit einem Datenfluss umgehen können. Die zu diesem Zweck herangezogene Anwendungsdomäne bietet die Möglichkeit auf verhältnismäßig einfache Art und Weise eine Fallbasis zu entwerfen, die in den Experimenten zur Evaluation aus Abschnitt 4.5 genutzt werden konnte.

5.1.1 Performanztest von CAKE I

In Abschnitt 3.1.1 wurde festgehalten, dass CAKE I im Moment nur ein lineares Retrieval implementiert hat. Da es sich bei Workflows um komplexe und große

```
1. <Agg c="PizzaClass">
2.       <AA n="Pizza Name" v="Pizza La Rustica" />
3.       <OA n="VegetableTopping">
4.             <C>
5.                   <A v="rocket"/>
6.             </C>
7.       </OA>
8.       <OA n="MeatTopping">
9.             <C>
10.                  <A v="parma ham"/>
11.            </C>
12.      </OA>
13.      <OA n="CheeseTopping">
14.            <C>
15.                  <A v="mozzarella"/>
16.                  <A v="parmesan"/>
17.            </C>
18.      </OA>
19. </Agg>
```

Abbildung 38 - Beispielfall aus der Pizza-Ontologie, Quelle: eigene Erstellung

Strukturen handelt, sollte das Retrieval daraufhin untersucht werden, ob es performant genug ist, um effizientes Arbeiten und Experimentieren zu ermöglichen. Aus diesem Grunde wurde im Rahmen dieser Arbeit ein Test entwickelt, mit dem sich die Geschwindigkeit des Retrievals abschätzen lässt. Da keine Anwendungsdomäne mit einer ausreichend großen Fallbasis zur Verfügung stand, wurde eine neue Ontologie mit Fallbasis entwickelt. Die Ontologie beschäftigt sich dabei mit dem Aufbau von Pizzen. Eine Pizza ist dabei ein Aggregat, das aus vier Attributen besteht: der Name der Pizza, der Gemüsebelag, der Fleischbelag und der Käsebelag.

Der Name der Pizza besteht aus einem gewöhnlichen String. Die Beläge bestehen aus einer beliebigen Anzahl von Gemüse-, Fleisch- oder Käseobjekten. Aus der Menge von Gemüse (z. B. Zwiebeln, Mais oder Oliven) kann zwischen 23 Gemüsesorten gewählt werden, die einer taxonomischen Ordnung unterliegen. Als Fleischbelag (z. B. Salami, Schinken oder Thunfisch) kann aus einer Menge von 14 Fleischsorten gewählt werden, die auch in einer Taxonomie sortiert sind. Um den Käse (z. B. Mozzarella, Feta oder Parmesan) auszusuchen, der auf die Pizza soll, kann aus einer Taxonomie mit 6 Käsesorten gewählt werden. In Abbildung 38 ist die Pizza „Pizza La Rustica“ dargestellt. Auf der Pizza befindet sich als Gemüse Rucolasalat, als Fleisch Parmaschicken und als Käse zusätzlich Mozzarella und Parmesan. Der Test wurde auf drei Testplattformen durchgeführt:

- Einem Intel DualCore T2300@1,66Ghz mit 1GB RAM
- Einem Intel Pentium 4@2,4Ghz mit 2GB RAM
- Einem Intel Quadcore Q6600@2,4Ghz mit 8GB RAM

Die Größe der Fallbasis wurde dabei iterativ immer verdoppelt. Dazu wurden per Zufall Pizzen generiert und zur Fallbasis hinzugefügt. Die Messungen für das Retrieval liefen damit auf einer Fallbasis mit 1.000 bis 512.000 Pizzen. Das Ergebnis der Messung ist in Abbildung 39 dargestellt.

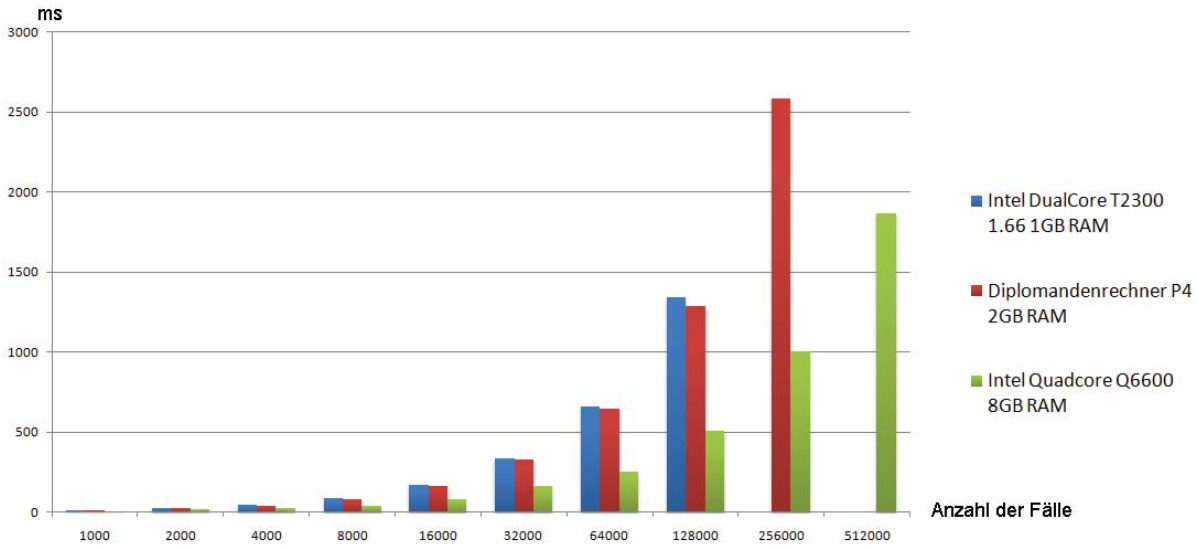


Abbildung 39 – Ergebnis des Retrievaltests, Quelle: eigene Erstellung

Auf einer modernen CPU benötigt der lineare Algorithmus bei einer Fallbasis mit 512.000 Pizzen 1870ms. Der Abbruch des DualCore bei 256.000 Pizzen und des Pentium 4 bei 512.000 Pizzen ist nur darauf zurückzuführen, dass diese Systeme zu wenig Arbeitsspeicher haben, um die Fallbasis anzulegen.

Das Ergebnis dieses Tests lässt sich natürlich nicht 1:1 auf das Retrieval von Workflows übertragen. Workflows besitzen eine komplexere Struktur als die Belagskombination auf einer Pizza und das verwendete Ähnlichkeitsmaß zwischen Workflows wird aufwendiger sein, als der Vergleich zweier Strings. Da der Ansatz dieser Arbeit sich noch in der Forschungsphase befindet und eine kollaborative Erstellung der Fallbasis noch nicht möglich ist, sollte die Geschwindigkeit des Algorithmus für Experimente ausreichend sein. Zum Stand dieser Arbeit wird mit Fallbasen gearbeitet, die weniger als 100 Objekte/Workflows beinhalten.

5.1.2 Untersuchung von Kochprozessen

Für die automatische Adaption von Workflows stand zu Beginn dieser Arbeit nur die Verwaltungsdomäne [Ke09] für eine empirische Untersuchung zur Verfügung. Die damalige Untersuchung ignorierte allerdings den Datenfluss in einem Workflow. CAKE III

und die Adaption von Workflows sollte allerdings auch den Datenfluss beachten. Aus diesem Grunde musste eine Anwendungsdomäne herangezogen werden, die stark datengetrieben ist. Dieser Abschnitt wird Kochprozesse untersuchen, um daraus eine Empfehlung für den Entwurf eines Datenflussmodells und die Modellierung von Kochprozessen abzuleiten.

Die Kochdomäne mag etwas untypisch für eine Prozessanalyse erscheinen. Sie hat jedoch zwei wesentliche Vorteile. Zum einen sind Kochprozesse in ihrer Komplexität gut überschaubar, zum anderen lässt sich leicht für einen Menschen überprüfen, ob der als Workflow beschriebene Kochvorgang sinnvoll ist. Der Evaluationsteil dieser Arbeit nutzt darum auch Kochworkflows, da kein „Expertenwissen“ für die Bewertung des Adaptionsergebnisses notwendig ist. Bevor Kochrezepte als Workflow modelliert wurden, wurde die Kochdomäne dahingehend untersucht, ob der Datenfluss in dieser Domäne wirklich

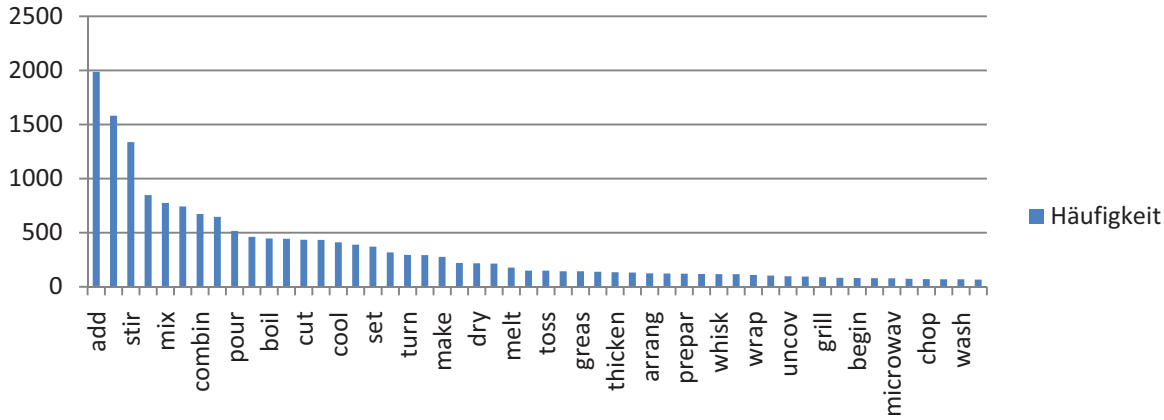


Abbildung 40 – Untersuchung von Kochbeschreibungen, Quelle: eigene Erstellung

eine so wichtige Rolle spielt, um ihn als Grundlage für ein Datenflussmodell zu nehmen. Dazu wurde eine XML Datenbank mit 1489 englischsprachigen Rezepten untersucht[22]. Die Datenbank besitzt bereits eine Vorstrukturierung, die die Zutaten von der Kochbeschreibung trennt. Daher konnten die Beschreibungen leicht untersucht werden. Ziel der Untersuchung war es, das Kochvokabular zu erfassen und daraus die passende Granularität für die Modellierung des Kontroll- und Datenflusses abzuleiten. Der Grad der Granularität gibt an, wie sehr eine Aufgabe en détail beschrieben werden muss. Eine sehr feine Granularität bei der Beschreibung der Tasks würde das Gewicht der Modellierung auf den Kontrollfluss legen, da die Taskbeschreibungen dann wichtiger wären als der Datenfluss. Umgekehrt fällt das Gewicht auf die Modellierung des Datenflusses, wenn die Datenobjekte umfangreich beschrieben werden. Es muss dabei

[22] Die Fallbasis stammt aus dem Computer Cooking Contest 2008. Download: http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc&act=5

ein Verhältnis zwischen der Ausmodellierung des Kontroll- und des Datenflusses geschaffen werden, das im besten Fall auf alle Prozesse der Anwendungsdomäne übertragbar ist.

Für die Untersuchung der Kochbeschreibungen wurde ein Wortindex aufgebaut, der nach den Häufigkeiten der Wörter sortiert ist. Dazu wurden die Beschreibungen durch eine Stopp-Wort-Liste gefiltert. Anschließend wurde auf den verbleibenden Wörtern mit einem Porterstemmer eine Stammformreduktion (sog. „Stemming“) durchgeführt. Das Ergebnis der Untersuchung kann in Abbildung 40 betrachtet werden. Bezogen auf die Anzahl aller Wörter, die nach der Stammformreduktion noch übrig waren, konnte das Vokabular der Kochbeschreibungen durch die 50 häufigsten Wörter zu 68 % abgedeckt werden. Dies war eine erste Untersuchung ohne Berücksichtigung von Rechtschreibfehlern oder einer speziell angepassten Stopp-Wort-Liste. Des Weiteren wurden Synonyme wie „mix“ (von „to mix“ – mischen) und „combin“ (von „to combine“ – vermischen) nicht zusammengefasst. Durch etwas mehr Arbeit ließe sich der Abdeckungsgrad durch ein relativ kleines Vokabular weiter erhöhen. Die Konsequenz eines so kleinen Vokabulars für Tasknamen ist, dass der Datenfluss an Bedeutung gewinnt und aus diesem Grunde wurde die Kochdomäne zum Design des Datenflusses herangezogen.

Bei der Modellierung erster Kochworkflows anhand textueller Beschreibungen wurde festgestellt, dass viel implizites Wissen über den Kochvorgang vorausgesetzt wird und die Granularität der Rezeptbeschreibungen stark schwankt (vgl. [MinorEtAl10b]). Dies lässt sich darauf zurückführen, dass die Sprachökonomie von jedem Menschen durch viele unterschiedliche Faktoren, wie dem sozialen Status, geprägt wird [La80]. So existieren Rezepte für Nudelgerichte in denen explizit steht, wie die Nudeln zuzubereiten sind, während in anderen Rezepten lediglich erwähnt wird, dass die Soße nun über die Nudeln gegossen werden kann. Die Umsetzung solcher Modellierungen liegt in der Hand des Workflowmodellierers und ist häufig eine subjektive Entscheidung. Die Erfahrung des Autors dieser Arbeit ist es, dass es sehr schwierig ist, für eine gewisse Anzahl von Workflows, eine stringente Granularität bei der Modellierung durchzuziehen, da es quasi eine beliebige Anzahl von Nuancen innerhalb von textuellen Beschreibungen gibt, die hinsichtlich ihrer Aufnahme in die Modellierung neu bewertet werden müssen. Die Fallbasis, die in der Evaluation genutzt wurde, beinhaltet recht einfach modellierte Kochworkflows mit Nudelgerichten. Die Zutaten wurden so modelliert, dass sie nur durch einen String repräsentiert werden. Auf Attribute, die ihren Verarbeitungszustand (geschnitten, gekocht, etc.) beschreiben, wurde ebenso verzichtet wie auf die Modellierung von Aggregaten. Die Tasknamen beschränken sich auf das oben beschriebene Vokabular für Kochaktivitäten. In Abschnitt 4.2.1 wurde schon vorweggenommen, wie der Datenfluss mit dem Kontrollfluss verbunden wird und warum die Modellierung der Kochrezepte vereinfacht wurde. Eine detaillierte Modellierung müsste

nicht nur die Datalinks vollständig setzen sondern auch bei der Bearbeitung von Datenobjekten in Tasks dafür sorgen, dass die Eigenschaften der Datenobjekte bearbeitet werden.

5.2 Erweiterung von CAKE I

Um das Konzept zur automatischen Adaption von Workflows auf seine Realisierbarkeit hin zu überprüfen, wurde das CAKE I System zeitgleich erweitert. Dabei spielten die identifizierten Anforderungen aus Abschnitt 4.1 und die Modellierungserfahrungen von Kochprozessen zur Integration eines Datenflusses eine wesentliche Rolle. Die Modellierungsversuche zeigten, wie der Datenfluss mit dem Kontrollfluss verbunden werden kann und dass es im Datenfluss zur Bildung von Aggregaten kommen kann. Die aufgestellten Anforderungen zur Fallrepräsentation waren umfangreich. Um eine geeignete Repräsentationsform für Fälle zu implementieren, musste das Retrieval auf Workflows und generisch modellierten Change Requests ermöglicht werden. Da die sonst üblichen Ähnlichkeitsmaße und Objektstrukturen von CAKE I auf Attribut-Wert-Paaren beruhen, musste eine neue Datenstruktur entwickelt werden, die die Struktur eines Workflows abbilden kann und auf der neue Ähnlichkeitsmaße angewendet werden können. Für den Change Request wie für die Datenobjekte im Datenkontext gilt, dass sie beliebig modellierbar sein sollen. Im Lösungsteil eines Falls müssen die ADD- und DELETE-Listen adäquat abgebildet werden.

5.2.1 Implementierung der Fallrepräsentation

Vor der Implementierung der entworfenen Fallrepräsentation (vgl. Abschnitt 4.2) lag es nahe, das CAKE I System dahingehend zu untersuchen, inwiefern bestehende Komponenten wiederverwendet werden können. Infolgedessen wurde der Entschluss gefasst, das CAKE I Datenmodell für die Modellierung generischer Change Requests und Datenobjekte weiterzuverwenden. Die Modellierung des Change Requests und der Datenobjekte als CAKE I Benutzerklasse hat zwei Vorteile. Indem auf die Systemklassen zurückgegriffen wird, können Benutzerklassen mittels XML spezifiziert werden. Dadurch lässt sich das CAKE III System an eine andere Domäne anpassen, ohne dass zunächst Änderungen am Quellcode vollzogen werden müssen. Zum anderen lassen sich die vorhandenen Ähnlichkeitsmaße weiterbenutzen. CAKE nutzt für seine Systemklassen über 31 Ähnlichkeitsmaße (vgl. Abschnitt 3.1.3), die beliebig in Aggregaten kombinierbar

sind[23]. Um das Adaptionsergebnis abzubilden und um den Informationsverlust des Retrieval Formats zu kompensieren (vgl. Abschnitt 4.2.1), werden der ursprüngliche und der adaptierte Workflow im CAKE II Format in einem Fall erfasst. Für das Retrieval von Adaptation Cases müssen somit nur noch Datenstrukturen geschaffen werden, die das Retrieval Format umsetzen und die ADD- und DELETE-Listen realisieren. Mit diesen Überlegungen wird ein Adaptation Case innerhalb der CAKE III Plattform wie folgt umgesetzt:

Im Problemteil:

1. Der Change Request kann als CAKE I Benutzerklasse modelliert werden. Die zum Stand dieser Arbeit verwendete Benutzerklasse beinhaltet ein einziges Attribut vom Typ String.
2. Der Retrieval Workflow.

Im Lösungsteil:

3. Das Adaptionsergebnis des Falls. Dabei handelt es sich um den adaptierten Workflow im CAKE II Format.
4. Die ADD- und DELETE-Listen.

Zusätzlich:

5. Der ursprüngliche Workflow im CAKE II Format.

Der Adaptation Case selbst wird als neue Systemklasse in CAKE I integriert (siehe Abbildung 41). Er erbt dabei nicht von der Aggregate-Klasse, um Systemkonflikte zu vermeiden. Die neue Adaptation Case Klasse soll ausschließlich für die Adaption von Workflows genutzt werden. Der ursprüngliche und der adaptierte Workflow liegen für die CAKE III Plattform komprimiert in der Fallbasis vor. Um die im Zip-Format komprimierten Workflows textuell abspeichern zu können, werden sie mit dem Base64[24] Verfahren umgewandelt. Während des Einlesens der Fallbasis und der damit verbundenen Umwandlung des XML Streams in CAKE I Datenobjekte werden die CAKE II Workflows dekomprimiert und als normale Strings in einem Adaptionsfall gespeichert. Dieser zusätzliche Zwischenschritt der Kompression kann das Einlesen der Fallbasis

[23] Die vorhandenen Ähnlichkeitsmaße können wohl eher bei Datenobjekten wiederverwendet werden als beim Change Request. Die Semantik des Change Requests lässt sich über die vorhandenen Ähnlichkeitsmaße nicht erfassen. Datenobjekte hingegen können zusätzlich über Attribute verfügen, die bspw. einen numerischen Charakter besitzen. Die vorhandenen CAKE I Ähnlichkeitsmaße sind syntaktische Ähnlichkeitsmaße.

[24] Ein Base64 Encoder ist Bestandteil von CAKE I und ermöglicht es binäre Daten in XML Dateien abzuspeichern.

beschleunigen, wenn man die Eigenschaften moderner Betriebssysteme berücksichtigt[25]. Für die exakte Spezifikation eines Falles sei an dieser Stelle auf das XML Schema aus Anhang 1 verwiesen.

5.2.2 Implementierung des Retrievalformats für Workflows

Der nächste Schritt in der Erweiterung des CAKE I Systems bestand darin neue Datenstrukturen zu implementieren, die sowohl für das Retrieval von Workflows als auch für das Mapping von Ankern genutzt werden können. Für die Datenobjekte des Datenflusses werden zukünftig, wie oberhalb beschrieben, CAKE Benutzerklassen angelegt, die mit Hilfe der Systemklassen in XML definiert werden können. Dadurch ist die Modellierung der Datenobjekte generisch gehalten und es muss kein neues Datenmodell für Datenobjekte entworfen werden. Die Nutzung des CAKE Datenmodells für Datenobjekte löst aber noch nicht das Problem, dass Datenobjekte aggregiert werden können sollen. Es existiert zwar unter den Systemklassen die Klasse Aggregate, diese erlaubt jedoch keine beliebige Aggregation von Datenobjekten, sondern definiert genau ein Datenobjekt als Aggregat. Ein weiteres Problem für die Integration von Workflows in CAKE war die neue Anforderung, dass nun eine ähnlichkeitsbasierte Suche zwischen Workflows ermöglicht werden muss. Mit den bisherigen CAKE Systemklassen lässt sich das nicht bewerkstelligen, da diese keine Ähnlichkeitsberechnung zwischen Strukturen ermöglichen, sondern sich auf Attribut-Wert-Paare beschränken. Aus diesem Grunde wurden die Systemklassen erweitert (siehe Abbildung 41).

[25] Dies ist der Tatsache geschuldet, dass alle gängigen PC-Betriebssysteme über einen sog. Scheduler und Dispatcher verfügen. Diese regeln den Zugriff nebenläufiger Prozesse. Bei Prozessen wird unterschieden, ob ein Prozess Daten anfordert oder etwas auf der CPU berechnen muss. Ein Prozess der auf einen langsamen Datenträger, wie z.B. eine Festplatte zugreifen muss, verlangsamt das System, da lesende Zugriffe auf Sekundärspeicher viel langsamer sind, als Rechenoperationen auf der CPU. Durch die Kompression der CAKE II Workflows lässt sich das Gewicht auf die Seite der CPU-Prozesse (CPU-Bursts) verlagern. Je leistungsstärker die CPU ist, desto schneller ist der Punkt erreicht an dem eine vollständige oder teilweise komprimierte Fallbasis schneller verarbeitet werden kann, als eine nicht komprimierte Fallbasis. [GS06] Gegen eine vollständige Komprimierung spricht jedoch, dass die Fallbasis danach nicht mehr lesbar ist.

Neben der Adaptation Case Klasse wurde eine neue ***Workflowklasse*** geschaffen, die drei Subklassen kennt: ***Sequence***, ***Task*** und ***Node***. Mit diesen vier Klassen lässt sich die Struktur jedes beliebigen Workflows aus der CAKE II Engine mappen. Dazu werden alle Elemente der CAKE Modellierungssprache mit inneren Sequenzen (XOR, AND und Loop) auf die Klasse Node gemappt, während die übrigen Elemente Task und Placeholder Task auf die Klasse Task übertragen werden. Meilensteine und Breakpoints sind zurzeit nicht über das CAKE I Workflowmodell abbildbar, da sie bei der Entwicklung des Konzepts

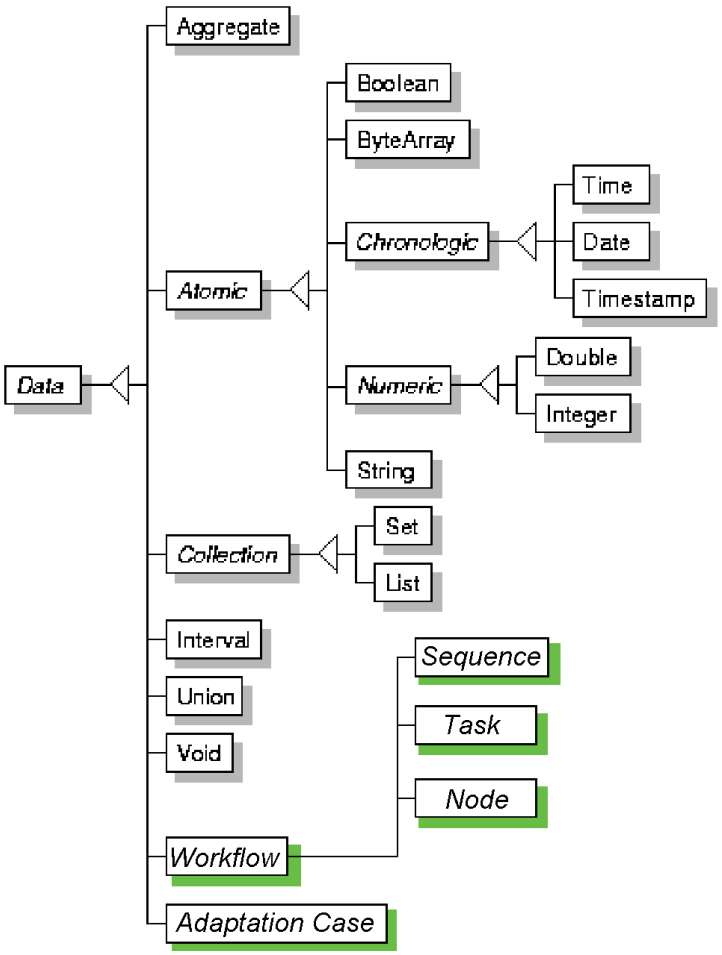


Abbildung 41 - Erweiterung der CAKE Systemklassen, Quelle: Erweiterung von [Ma06]

bisher keine Rolle spielten. Dies liegt zum einen daran, dass es keine ausreichend große Fallbasis gab, die auch suspendierte Workflows umfasste und zum anderen, dass die untersuchten Anwendungsdomänen Elemente wie Meilensteine nicht in Anspruch nahmen. Die neuen Workflowklassen verfügen über Methoden, die es erlauben, die gesamte Struktur – ähnlich einer DOM Struktur – zu durchwandern und die Workflow-elemente zu beschreiben. So kann ein Node durch die Methode „setType(String type)“

als AND, XOR oder Loop beschrieben werden. Der aktuelle Stand der API für Workflowelemente ist der Abbildung 42 zu entnehmen.

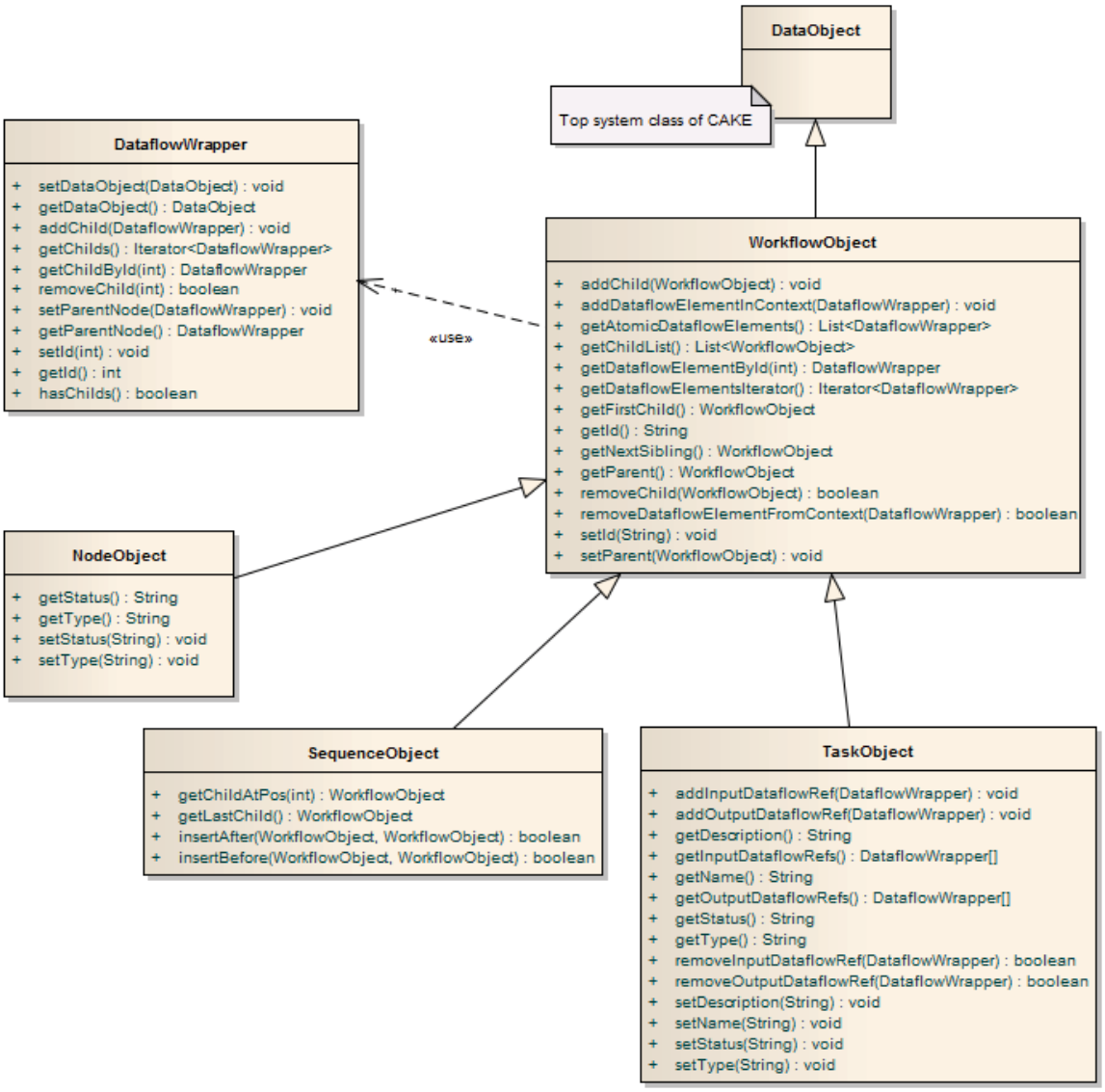


Abbildung 42 - UML Diagramm der Workflow Klassen, Quelle: eigene Erstellung

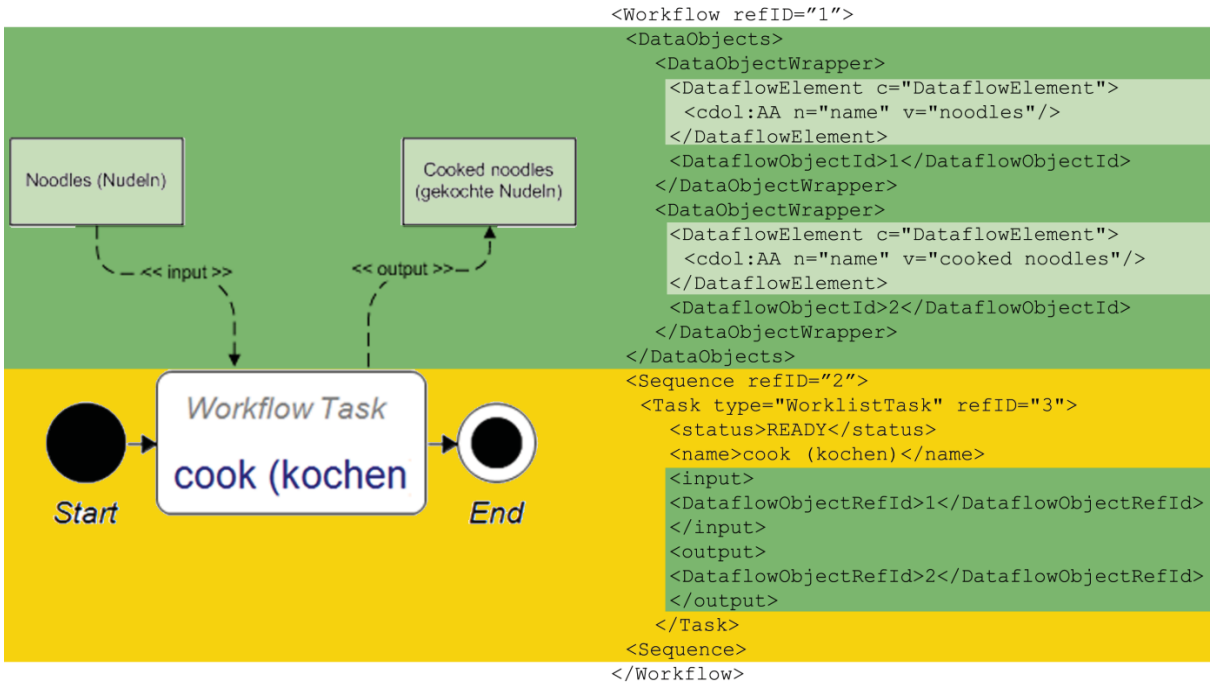
Da das CAKE II WfMS noch keinen Datenfluss kennt, wurden die Überlegungen aus Abschnitt 4.2.1 und 5.1.2 in die neuen Workflow-Systemklassen integriert. Der Datenkontext kann durch die Methoden innerhalb eines WorkflowObjects verwaltet werden (vgl. z. B. Abbildung 42 „addDataflowElementInContext()“ in WorkflowObject). Um die Aggregation von Datenobjekten zu ermöglichen, wurde eine einfache

Wrapperklasse [26] geschrieben, in die die Datenobjekte eingebunden werden. Die "***DataflowWrapper***" (siehe Abbildung 42) besitzen eine eindeutige ID und erlauben den Aufbau von Bäumen. Der Datenkontext besteht somit aus einer Menge von in Wrapper eingebetteten Datenobjekten, die in atomarer oder aggregierter Form vorliegen. Um die Datenobjekte nun mit Tasks zu verknüpfen, besitzt jeder Task Input- und Outputcontainer, die Referenzen auf die Datenobjekte im Datenkontext besitzen (Abbildung 42 „addInputDataflowRef()“ und „addOutputDataflowRef()“ in TaskObject). Der aktuelle Methodenumfang der Workflowsystemklassen umfasst nur eine Teilmenge der Eigenschaften, der ursprünglichen CAKE II Workflowelemente. Ein TaskObject wird zum Stand dieser Arbeit bspw. nur durch seine Namen, seine Beschreibung und seinen Typ (Task oder Placeholder) charakterisiert. Für erste Experimente mit dem System war dies jedoch ausreichend (vgl. [MinorEtAl10a] [MinorEtAl10b]). Durch die Klassen-hierarchie erben TaskObjects, die Blattelementen in einem Baum entsprechen, scheinbar überflüssige Methoden, weil sie darunterliegende Elemente implizieren (z. B. „getFirstChild()“). Da aber ein Placeholder Task gleichfalls als TaskObject abgebildet wird, müssen hierarchische Verbindungen zwischen Workflows beachtet werden. Ebenso kann ein Placeholder Task für Late Binding (vgl. Abschnitt 1.3.1) genutzt werden und auf diese Weise mehrere Kinder beinhalten. Es ist die Aufgabe des Adaptation Managers die Methoden von Workflowobjekten korrekt aufzurufen.

Neben der Entwicklung neuer Systemklassen musste parallel ein XML Format entworfen werden, das für eine persistente Speicherung der Workflows in einer Fallbasis sorgt. Abbildung 43 zeigt exemplarisch, wie der Datenfluss und der Kontrollfluss in XML repräsentiert werden. Die Abbildung ist horizontal und vertikal in zwei Bereiche unterteilt. In der Horizontalen steht links in der Abbildung der graphisch modellierte Workflow. Der Workflow beinhaltet nur einen Task („cook (kochen)“), der für seine Abarbeitung ein Datenobjekt benötigt („Noodles (Nudeln)“) und ein Datenobjekt neu erzeugt („Cooked noodles (gekochte Nudeln)“). Auf der rechten Seite der Abbildung befindet sich die entsprechende XML Darstellung für den kleinen Kochworkflow. In der Vertikalen wird zwischen Datenobjektkontext und der Kontrollflussstruktur des Workflows unterschieden. Der untere, gelbe Bereich beinhaltet Informationen über den Kontrollfluss während im oberen, grünen Bereich die Datenobjekte für den Datenfluss ihren Platz finden. Anhand der Grafik wird nun die Verbindung zwischen Datenobjekten und dem Kontrollfluss auf technischer Ebene ersichtlich. Innerhalb des XML Elements „Task“ befinden sich „input“ und „output“ Abschnitte, die eindeutig auf „DataflowObjectWrapper“ im oberen Abschnitt verweisen. Der Task besitzt in seinem "input" Element genau ein "DataflowObjectRefId", das in seinem Textknoten den Wert

[26] Der Begriff Wrapper stammt aus der Softwareentwicklung und gehört zu der Kategorie der Strukturmuster (Structural Patterns). Der Wrapper findet dann Verwendung, wenn eine Klasse nicht über die notwendigen Schnittstellen verfügt, um mit anderen Klassen zu kommunizieren. Der Wrapper bildet eine Hülle um eine Klasse, um die notwendigen Schnittstellen zu übersetzen (vgl. [GHJV94]]).

"1" stehen hat. Im oberen Datenobjektkontext existiert genau ein Wrapper, der den Wert "1" als "DataflowObjectId" besitzt. Es wird also mit Referenzen auf „DataflowObjectWrapper“ gearbeitet. Innerhalb der „DataflowObjectWrapper“ steht das „DataflowElement“. Es ist frei durch eine CAKE Benutzerklasse definierbar. In diesem einfachen Beispiel beinhaltet das „DataflowElement“, welches intern wie eine Aggregate-Systemklasse behandelt wird, nur das Attribut „name“, das mit einer beliebigen Zeichenkette gefüllt werden kann. Die XML Workflowelemente in der Fallbasis werden wie bei anderen CAKE kompatiblen Fallbasen beim parsen der Fallbasis eingelesen und in die oben vorgestellten Systemklassen umgewandelt. An dieser Stelle sei nochmals auf Anhang 1 verwiesen, der das komplette Schema für Adaptation Cases als auch für das Retrieval Format beinhaltet.




```
<Workflow refID="1">
 <DataObjects>
   <DataObjectWrapper>
    <DataflowElement c="DataflowElement">
     <cdol:AA n="name" v="noodles"/>
    </DataflowElement>
    <DataflowObjectId>1</DataflowObjectId>
   </DataObjectWrapper>
   <DataObjectWrapper>
    <DataflowElement c="DataflowElement">
     <cdol:AA n="name" v="cooked noodles"/>
    </DataflowElement>
    <DataflowObjectId>2</DataflowObjectId>
   </DataObjectWrapper>
 </DataObjects>
 <Sequence refID="2">
  <Task type="WorklistTask" refID="3">
    <status>READY</status>
    <name>cook (kochen)</name>
    <input>
    <DataflowObjectRefId>1</DataflowObjectRefId>
    </input>
    <output>
    <DataflowObjectRefId>2</DataflowObjectRefId>
    </output>
   </Task>
 <Sequence>
</Workflow>
```

Abbildung 43 – Die XML Repräsentation der Workflowklassen, Quelle: eigene Erstellung

5.2.3 Implementierung der ADD- und DELETE-Listen

Bei der Implementierung der ADD- und DELETE-Listen muss ebenso wie im Konzept zwischen drei Typen von Listen unterschieden werden (vgl. Abschnitt 4.4.1): Listen, die den Kontrollfluss ändern (Typ 1), Listen, die die den Datenkontext eines Workflows verändern (Typ 2) und Listen, die Datalinks setzen (Typ 3). Aufgrund der ersten Evaluation, die nur die Verwaltungsdomäne ohne Datenfluss betraf [MinorEtAl10a] und der parallelen Entwicklung von Konzept und Testimplementierung existiert zurzeit nur

ein Interface, das alle drei Typen von Listen abbilden muss (siehe Abbildung 44). Unter Verwendung der Methoden „isStructureBlock()“ (Typ 1), „isDataflowObject()“ (Typ 2) und „isDatalink()“ (Typ 3) kann der Adaptation Manager unterscheiden, um welchen Listentyp es sich handelt. Eine überarbeitete Implementierung sollte dieses Interface aufspalten und für jeden Listentyp eine eigenes Interface mit Implementierung

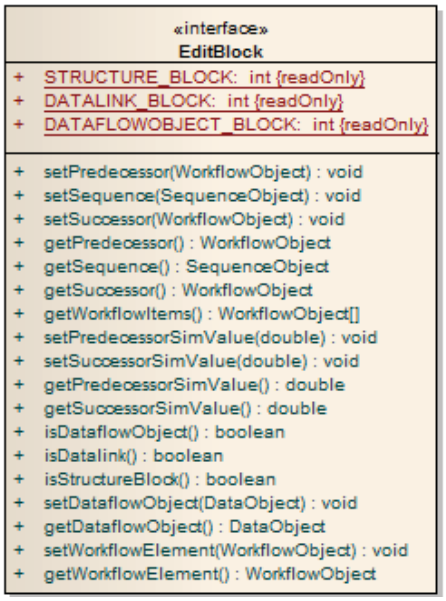


Abbildung 44 - Interface für ADD- und DELETE-Liste, Quelle: eigene Erstellung

erzeugen. Dadurch würde eine falsche Nutzung der Methoden verhindert. Um die ADD- und DELETE-Listen in einem Adaptation Case zu realisieren, wurden die neuen Workflowklassen für das Retrievalformat wiederverwendet. Bei Typ 1 Listen kann als Vorgänger- oder Nachfolgeranker ein beliebiges WorkflowObject verwendet werden. In der Regel handelt es sich dabei um einen einzelnen Task oder ein Blockelement wie z. B. ein AND. Für die Kette kann ein SequenceObject eingefügt werden, welches die einzufügenden oder zu entfernenden Workflowelemente beinhaltet. Bei Typ 3 Listen existiert keine innere Kette. Als Vorgänger- oder Nachfolgeranker kann genau ein Workflowelement („setWorkflowElement“) und genau ein Datenobjekt („setDataflowObject“) gesetzt werden. Bei Typ 2 Listen wird nur ein Datenobjekt in die innere Kette aufgenommen („setDataflowObject“) während die Anker leer bleiben.

5.3 Der Adaptation Manager

Der Adaptation Manager ist die Softwarekomponente, die geschaffen wurde, um den automatischen Adaptionsprozess von Workflows durchzuführen und dabei die vorhandenen CAKE Systeme zu nutzen. Das CAKE I System wird von ihm genutzt um Adaptation Cases aus einer Fallbasis verarbeiten zu können und um mittels vorhandener Ähnlichkeitsmaße die Anker von ADD- oder DELETE-Listen auf Positionen in einem neuen Workflow zu übertragen. Das CAKE II System wird von ihm genutzt um Änderungen, die durch erfolgreich adaptierte ADD- oder DELETE-Listen entstehen, auf einen Workflow anzuwenden. In Abbildung 45 ist ein UML Klassendiagramm des Adaptation Managers dargestellt. Bei näherer Betrachtung ist ersichtlich, dass er im Wesentlichen aus zwei Komponenten besteht: zum einen aus einer Factory („AdaptationManagerFactory“), die es erlaubt während der Laufzeit des Systems zwischen unterschiedlichen Implementierungen für den Adaptionsprozesses zu wählen und zum anderen aus der „WorkflowLogic“, die in jeder Implementierung eines Adaptionsprozesses genutzt werden sollte. Das Factory-Pattern (vgl. [GHJV94]) ermöglicht es für unterschiedliche Anwendungsdomänen unterschiedliche Adaptionsvarianten zu implementieren. Da eine serviceorientierte Architektur angestrebt wird, können so für das Modellierungstool unterschiedliche Adaptation-Services zur Verfügung gestellt werden.

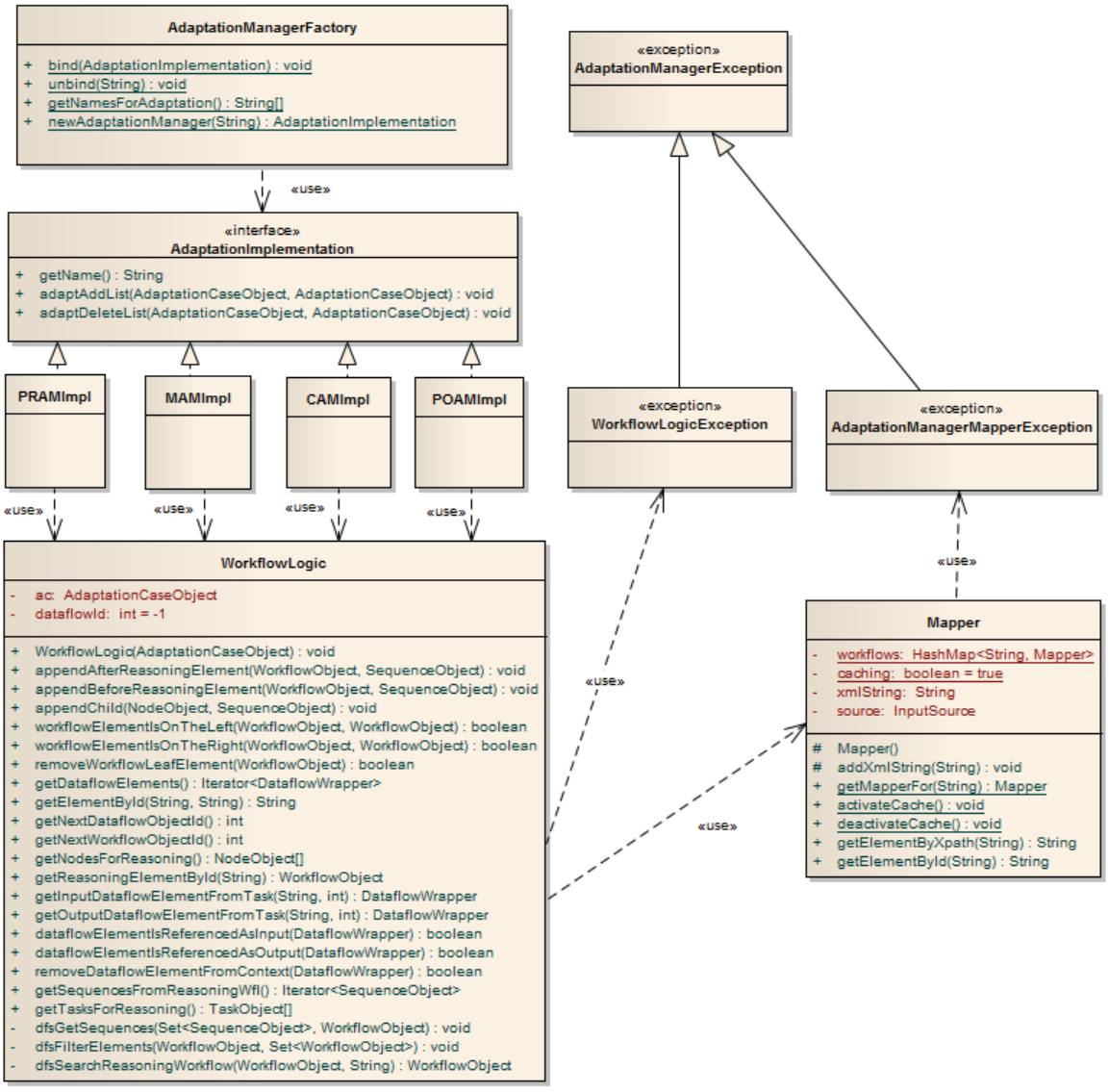


Abbildung 45 - Die Klassen des Adaptation Managers, Quelle: eigene Erstellung

Die in Abschnitt 5.2.2 vorgestellten neuen Workflow-Systemklassen von CAKE beinhalten nur Methoden, um die Struktur des Workflows zu durchwandern und um Zugriff auf Workflowelemente und Datenobjekte zu erhalten. Dies ist jedoch nicht genug, um Adaptionen auf einem Workflow durchzuführen. Aus diesem Grunde existiert die WorkflowLogic. Sie bettet ein Adaptation Case Objekt in sich ein und liefert Methoden nach außen, um Änderungen an einem Workflow vorzunehmen. Bei dem dabei verwendeten Adaptation Case Objekt handelt es sich um die Anfrage, die an das System gestellt wurde, um durch eine Adaptionsvariante angepasst zu werden. Die WorkflowLogic verleiht den Workflow-Systemklassen somit im Nachhinein eine eigene Logik. Die WorkflowLogic nutzt zur Umsetzung ihrer Aufgaben eine weitere Klasse. Die „Mapper“ Klasse hat ihre Daseinsberechtigung vor allem in der Testphase des Systems

und neuer Anwendungsdomänen. Sie wird genutzt, um den Informationsverlust des Retrievalformats zu kompensieren. Dazu stellt sie Methoden bereit, die den serialisierten CAKE II Workflow mittels Xpath-Ausdrücken durchsuchen können. Auf diese Weise kann jedes Attribut oder jeder Textwert des ursprünglichen Workflows ausgelesen werden. Um den DOM für Xpath-Anfragen nicht immer wieder neu aufbauen zu müssen, besitzt der Mapper einen internen Cache. Während der Arbeit mit den Adaptionsvarianten und dem Retrievalformat stellte sich heraus, dass trotz Caching der Zugriff mittels Xpath um ein vielfaches langsamer ist als das direkte Auslesen eines Wertes. Aus diesem Grunde wurden alle für die Evaluation notwendigen Attribute eines Workflows in die Workflow-Systemklassen übertragen.

5.4 Die Interaktion der CAKE Systeme

In diesem Abschnitt wird die Kommunikation der Systeme miteinander genauer beschrieben unter Bezugnahme auf die neun Schritte des Zyklus, der im vorherigen Kapitel in Abschnitt 4.3 vorgestellt wurde. Die Kommunikation läuft zwischen den vier bekannten Komponenten ab:

1. Der Modellierungs-GUI. Diese wird als „Wfl-Man“ illustriert, da hier mit einem Menschen interagiert wird.
2. Der Adaptation Manager. Dieser beinhaltet die Logik für die Workflow-adaption.
3. Der Retrieval-Engine, die Bestandteil von CAKE I ist.
4. Das Workflow Management System CAKE II (LT Engine).

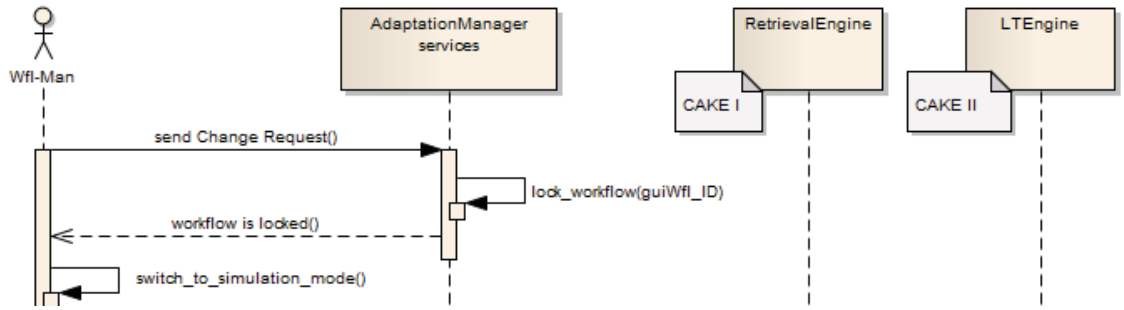


Abbildung 46 - Sequenzdiagramm von Schritt 1 des Zyklus

Anhang 2 beinhaltet ein selbst erstelltes UML Sequenzdiagramm über die Kommunikationswege von CAKE III während einer automatischen Adaption. Aufgrund

der Größe des Diagramms wird es hier ausschnittsweise vorgestellt und erläutert. Abbildung 46 visualisiert Schritt (1) des Zyklus. Der Benutzer schickt seinen Change Request an denAdaptation Manager. Daraufhin sperrt der Adaptation Manager den Workflow. Das Sperren (engl. „locking“) eines Workflows sorgt dafür, dass der Benutzer einen exklusiven Zugriff auf den Workflow erhält. Niemand außer dem Benutzer darf dann Änderungen an dem Workflow vornehmen. Dadurch wird verhindert, dass Inkonsistenzen durch Änderungen anderer Benutzer entstehen [27]. Sobald die Modellierungs-GUI das Signal erhält, dass der Workflow gesperrt wurde, öffnet sich der

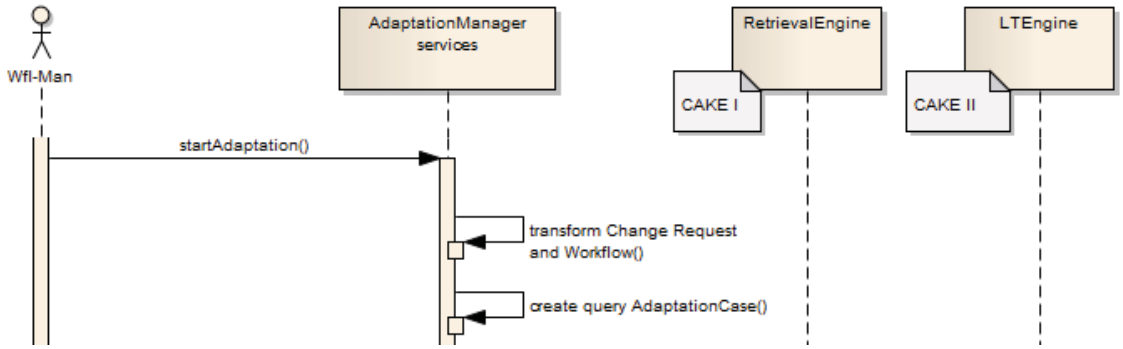


Abbildung 47 - Sequenzdiagramm von Schritt 2 und 3 des Zyklus

Simulationsmodus der GUI. In Abbildung 47 werden die Schritte (2) und (3) des Zyklus dargestellt. Der Benutzer kann nun die Adaption starten. Der Adaptation Manager transformiert den Change Request und den Workflow, um daraus einen Adaptation Case zu erstellen, der als Anfrage dient. Der Change Request wird in eine CAKE Benutzerklasse transformiert, während für die Umwandlung des Workflows in das Retrieval Workflow Format ein XSLT Script benutzt wird (siehe Anhang 3). Der durch die Umwandlung gewonnene XML String im Retrieval Format wird durch einen SAX Parser in die neuen CAKE Workflowobjekte aus Abschnitt 5.2.2 überführt. Aus dem so gewonnenen Workflowobjekt und dem Change Request, der nun in einer CAKE Benutzerklasse abgebildet ist, wird ein Adaptation Case generiert, der als Anfrage an die Fallbasis genutzt wird.

[27] Das Locking kann jedoch keine Inkonsistenzen verhindern, die zwischen dem simulierten und dem ursprünglichen Workflow durch die Ausführung des Workflows entstehen. Ist der Kontrollfluss während der Verweildauer im Simulationsmodus so weit fortgeschritten, dass sich die Änderungen aus dem Simulationsmodus auf die Vergangenheit des Workflows beziehen würden, so dürfen die simulierten Änderungen nicht angewendet werden.

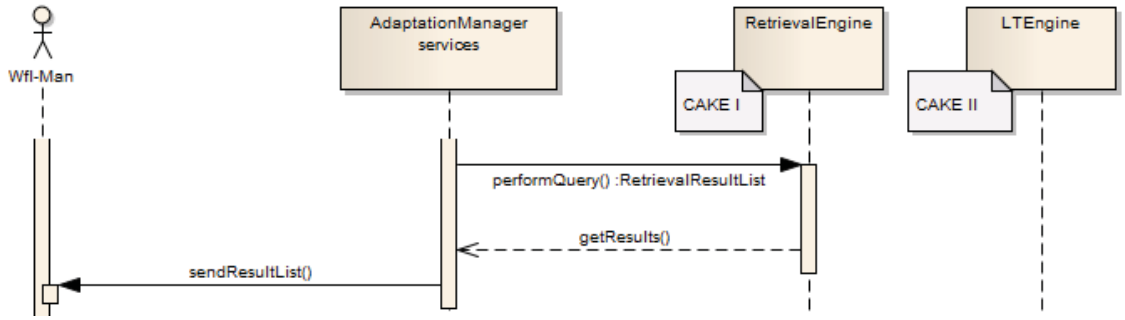


Abbildung 48 - Sequenzdiagramm von Schritt 5 bis 9 des Zyklus

Abbildung 48 zeigt Schritt (4) des Zyklus. Die Fallbasis wird angefragt und die Fälle werden hinsichtlich ihrer Ähnlichkeit zum erstellten Adaptation Case bewertet. Das hierfür verwendete Ähnlichkeitsmaß wurde in Abschnitt 4.3.3 erörtert. Das Ergebnis der Anfrage wird als Liste an die Modellierungs-GUI geschickt, sodass der Benutzer nun Fälle anhand einer Kurzbeschreibung oder der Identifikationsnummer auswählen kann. Klickt der Benutzer auf ein Element der Liste und wünscht eine automatische Adaption (dies entspricht Schritt (5) des Zyklus), startet der Adaptionsprozess im Adaptation Manager mit dem ausgewählten Fall. Dies wird in Abbildung 49 veranschaulicht. Wie in Schritt (6) und (7) beschrieben werden vom Adaptation Manager die ADD- und DELETE-Listen eingelesen und adaptiert. Dazu wird versucht die Anker der Listen auf Elemente des Workflows zu mappen. Da die ADD- und DELETE-Listen auf der Implementierungsebene

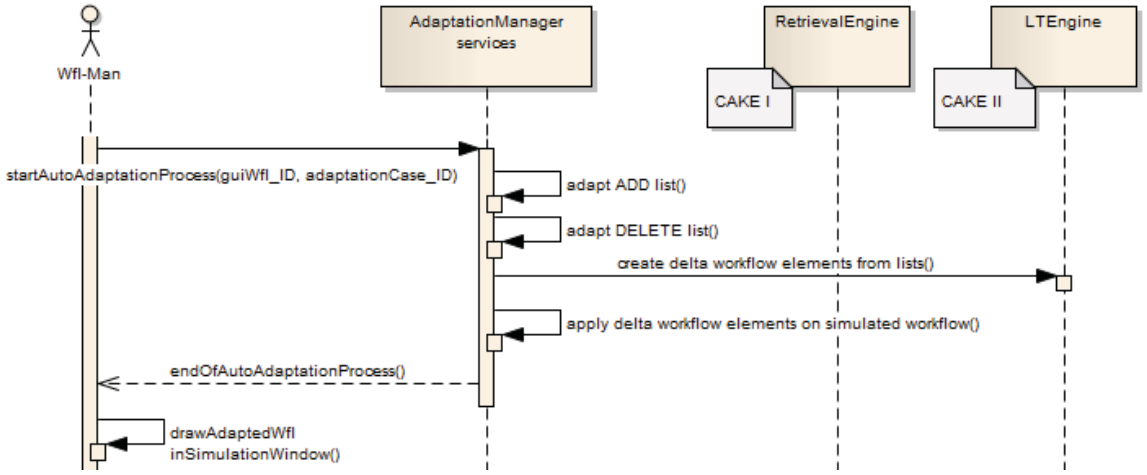


Abbildung 49 - Sequenzdiagramm von Schritt 4 des Zyklus

auf Elemente des Retrievalworkflows zeigen, muss der gesamte Adaptation Case aus der Fallbasis vor dem Adaptionsprozess geklont werden. Jede durchgeführte Änderung an einem Adaptation Case aus der Fallbasis bleibt so lange erhalten, bis das System neu startet und die Fallbasis somit neu eingelesen wird. Wird der Adaptation Case nicht

geklont, so steht er in einer späteren Reuse-Phase nicht in seiner ursprünglichen Form zur Verfügung, da seine ADD- und DELETE-Listen bereits zuvor adaptiert wurden, um auf eine andere Anfrage übetragen werden zu können. Im Folgenden wird also eine geklonte Version des Adaptation Cases genutzt, um den Adaptionsprozess durchzuführen. Um zu bestimmen ob ein Mapping zulässig ist, wird wie beim Retrieval der Fälle ein Ähnlichkeitsmaß verwendet. Ist die Ähnlichkeit zwischen Anker und Workflowelement hoch genug, darf der Anker dem Workflowelement zugeordnet werden. Durch das Mapping wird die Position im Workflow bestimmt, die von einer Änderung betroffen wäre. Anschließend muss noch festgestellt werden, ob die Kette ganz oder nur zum Teil übernommen wird. Bei DELETE-Listen stellt sich hier bspw. die Frage, ob alle Elemente der Kette gelöscht werden oder wie mit verschachtelten Elementen (AND- oder XOR-Block) umgegangen werden soll, die mehrere Elemente beinhalten. Die Ketten, der adaptierten ADD- und DELETE-Listen, werden anschließend als elternlose Elemente in die CAKE II Datenbank geschrieben[28]. Diese Workflowelemente sind die berechnete Differenz zwischen dem Workflow, der sich im Simulationsmodus befindet, und den geplanten Adaptionen. Zuletzt werden die berechneten Workflowelemente in den simulierten Workflow eingefügt oder gelöscht, abhängig davon, ob sie aus einer ADD- oder DELETE-Liste stammen.

[28] An dieser Stelle sei daran erinnert, dass CAKE II Workflows als Bäume mit inneren Knoten intern repräsentiert werden (vgl. Abschnitt 3.2.5).

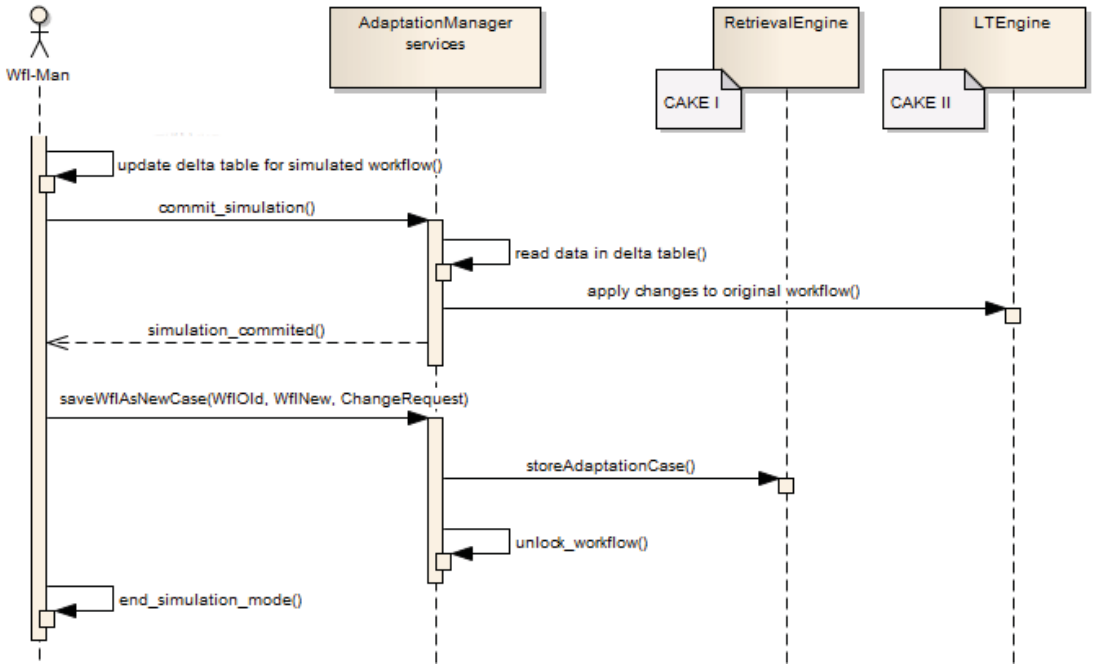


Abbildung 50 - Sequenzdiagramm optionaler Schritte

Damit wäre das Ende eines Adaptionszyklus erreicht. Abbildung 50 zeigt optionale Schritte, die nach einer solchen Adaption einsetzen könnten. So könnte der Benutzer weitere Änderungen am simulierten Workflow vornehmen. Die daraus resultierenden Ummodellierungen erzwingen ebenso eine Berechnung von Differenzelementen, die zumindestens temporär in der CAKE II Datenbank existieren müssen. Entschließt sich der Benutzer, die Änderungen aus dem Simulationsmodus zu übernehmen, werden alle berechneten Differenzelemente, die zuvor elternlos waren, auf den ursprünglichen Workflow übertragen. Tritt dabei ein Fehler auf, werden alle Änderungen aus dem Simulationsmodus verworfen. Ein solcher Fehler könnte durch einen vorzeitigen Abbruch einer laufenden Instanz verursacht worden sein. Falls alle Änderungen erfolgreich übertragen wurden, könnte zusätzlich eine Retain-Phase eingeleitet werden, die anhand des ursprünglichen Change Requests und den angewendeten Modifikationen einen neuen Fall erstellt. Das abschließende Entsperren des Workflows beendet den Simulationsmodus und gibt den Workflow zur Bearbeitung durch andere Benutzer frei.

Anhand des Sequenzdiagramms wurde die Retrieve-, Reuse- und die Retain-Phase einer Workflowadaption durchgesprochen. Während jeder Phase kommunizierte nur der Benutzer der Modellierungs GUI mit dem Adaptation Manager. Sogar in der Retain-Phase kann wegen der Verwendung von ADD- und DELETE-Listen auf einen CBR Experten verzichtet werden. Damit erfüllt das System die Anforderung, dass neues Wissen durch Interaktion mit dem Benutzer geschaffen werden kann.

5.5 Implementierung des CAM Algorithmus

Der vollständige Pseudocode, der CAM Methode, befindet sich in Anhang 4. Um ihn besser zu verstehen, wird er nach und nach in diesem Abschnitt erläutert.

1. **input**:
2. W: target workflow a set of workflow elements with a precedence relation,
   A: set of transformational operations [structural changes, dataflow elements, datalinks; operation type],
   Θ: several validity thresholds

3. **global**:
4. usedPreAnchors ← ∅
5. usedPostAnchors ← ∅

Die Zeilen 1-5 initialisieren den Algorithmus. Als Eingabeparameter existiert ein Workflow **W**, der als Anfrage adaptiert werden soll, eine Menge **A**, die aus ADD- und DELETE-Listen aus einem Fall besteht und einer Menge **Θ** (Theta), die den Algorithmus mit mehreren Gültigkeitsschwellen parametrisiert. Des Weiteren werden zwei globale Mengen angelegt, die verhindern, dass Workflowelemente als pre oder post anchor doppelt benutzt werden.

Die Methode in Zeile 6-32 ist der Grundblock für die Adaption der ADD-Listen. Es wird eine Menge „mappedOperations“ angelegt, die am Ende alle Listen aus den ADD-Listen beinhaltet, die auf den Workflow W übertragbar sind. Die einzelnen Listen „op“ aus den ADD-Listen sind, wie in Abschnitt 4.4.1 beschrieben, vorsortiert. Sie besitzen auf der technischen Ebene eine Struktur, die in Abbildung 44 beschrieben wurde. Bei der Adaption der ADD-Listen wird überprüft, ob es sich bei einer Liste um eine strukturelle Änderung am Kontrollfluss, am Datenobjektkontext oder um einen Datalink handelt:

- Handelt es sich um eine Kontrollflussänderung, wird zunächst ein Mapping der Anker in „adaptAnchors“ durchgeführt. Dann werden die Ähnlichkeitswerte kumuliert. Liegt der kumulierte Ähnlichkeitswert über der Gültigkeitsschwelle, wird die Menge mappedOperations um die adaptierte Kette erweitert und die verwendeten Anker werden gestrichen. Dadurch stehen sie nicht mehr als Kandidat zur Verfügung. Die Menge benutzter Kandidaten wird in pre und post Mengen unterteilt, da ein benutzter pre anchor Kandidat später immer noch als post anchor Kandidat benutzt werden kann und umgekehrt.
- Handelt es sich um ein Datenobjekt, so wird es ohne Überprüfung in den Datenobjektkontext von W übernommen. Die aktuelle Implementierung erlaubt Duplikate im Datenobjektkontext.
- Handelt es sich um einen Datalink, werden die bisher adaptierten Ketten zunächst auf den Workflow übernommen, da die eingefügten Elemente Ankerkandidaten für Datalinks sein könnten. Anschließend wird für das

Workflowelement und das Datenobjekt des Datalinks eine Menge gültiger Kandidaten bestimmt. Die beiden besten Kandidaten bilden den adaptierten Datalink.

Abschließend werden die gemappten Datalinks auf den Workflow angewendet und W wird zurückgegeben. Der modifizierte Workflow W ist die Grundlage zur Berechnung der Delta-Elemente, die als elternlose Subtrees in die CAKE II Datenbank geschrieben werden, bevor sie endgültig auf dem Workflow angewendet werden.

```
6.  begin „adaptation of add list“
7.  mappedOperations ← ∅
8.  for each op ∈  A such that op.operationType = ADD do
9.  if ( op.isStructuralChange )  {
10.         op ← adaptAnchors( op )
11.         sim1 ← sim( op.preAnchor ) + sim( op.postAnchor )
12.         if sim1 ≥ Θ do
13.                 mappedOperations ← mappedOperations ∪ { op }
14.                 usedPreAnchors ← usedPreAnchors ∪ { op.preAnchor }
15.                 usedPostAnchors ← usedPostAnchors ∪ { op.postAnchor }
16.         end
17. }
18. if ( op.isDataflowObject ) {
19.         mappedOperations ← mappedOperations ∪ { op } // insertion without check for duplicates
20. }
21. if ( op.isDatalink ) {
22.         applyOnWorkflow( mappedOperations ) // any previous changes must be applied
23.         wflCandidates ← getValidCandidates( op.wflElement, Θ )
24.         dataflowCandidates ← getDataflowCandidates( op.dataflowElement, Θ )
25.         op.wflElement ← best( wflCandidates ) // determine best of all candidates
26.         op.dataflowElement ← best( dataflowCandidates )
27.         mappedOperations ← mappedOperations ∪ { op }
28. }
29. end
30.         applyOnWorkflow( mappedOperations ) // insert datalinks and dataflow objects
31. return W;
32. end
```

Die Zeilen 33-66 beschreiben das Ankermapping für eine Kette nach der CAM Methode. Zunächst wird eine Menge “validPairs” initialisiert, die am Ende alle gültigen Ankerpaare beinhaltet. Handelt es sich bei der Kette um eine Änderung am Kontrollfluss, wird zunächst versucht tight pairs zu finden. Bei der Bestimmung eines tight pairs muss unterschieden werden, ob die Kette aus einer ADD- oder DELETE-Liste stammt und ob eine Substitution vorliegt (siehe Abschnitt 4.4.1). Wie im Abschnitt 4.4.2 beschrieben, handelt es sich um ein tight pair für Anker in einer ADD-Liste genau dann, wenn der pre anchor direkt vor dem post anchor Kandidaten im Workflow W liegt. Bei einer DELETE-Liste kann auch berechnet werden, ob es sich bei den Ankerkandidaten um tight pairs handelt. Dies relativiert die vorherige Beschreibung der CAM Methode. Bei den zuvor beschriebenen non-tight pairs für Ketten aus der DELETE-Liste handelt es sich um tight pairs, deren Abstand der Länge der Kette entspricht, die gelöscht werden soll. Durch die Überprüfung der tight pair Eigenschaft wird gleichzeitig die Konsistenz des Ankerpaars sichergestellt. Nachdem die Menge von tight pairs in validPairs aufgenommen wurde,

werden die übrigen Ankerkandidaten einzeln der Menge der validPairs hinzugefügt. Dies ist notwendig, da es sein kann, dass ein einzelner Kandidat an der Position des pre oder post anchors eine höhere Ähnlichkeit besitzt, als die kumulierte Ähnlichkeit aus einem tight pair. Das Ankerpaar mit der höchsten Ähnlichkeit wird abschließend für die Kette übernommen.

```
33. begin „adaptAnchors(Operation op )“
34. validPairs ← ∅
35. if ( op.isStructuralChange ) {
36.         preCandidates ← getValidCandidates( op.preAnchor, Θ )
37.         postCandidates ← getValidCandidates( op.postAnchor, Θ )
38.         for each pre ∈ preCandidates do
39.             if pre not ∈ usedPreAnchors {
40.             for each post ∈ postCandidates do
41.             if post not ∈ usedPostAnchors {
42.                 areTight ← false
43.                 if op.operationType = DEL or isSubstitution( op ) {
44.             areTight ← ( W.getElement( pre ).siblingAtPos( op.innerSequence.length ) = post)
45.                 } else {
46.                     areTight ← ( W.getElement( pre ).nextSibling = post )
47.                 }
48.                 if areTight  {
49.                     validPairs ← validPairs ∪ { ( pre, sim(pre), post, sim(post) ) }
50.                 }
51.             }
52.             end
53.             }
54.         end
55.
56. for each pre ∈ preCandidates such that pre not ∈ usedPreAnchors do
57.         validPairs ← validPairs ∪ { ( pre, sim(pre), NIL, 0 ) }
58. end
59. for each post ∈ postCandidtes such that post not ∈ usedPostAnchors do
60.         validPairs ← validPairs ∪ { ( NIL, 0, post, sim(post) ) }
61. end
62.         bestPair ← best(validPairs)
63.         op.preAnchor ← bestPair.pre
64.         op.postAnchor ← bestPair.post
65. }
66. end
```

Die Zeilen 67-84 beschreiben das Auffinden von Kandidaten für Workflowelemente und Datenobjekte. In der Methode „getValidCandidates“ für Workflowelemente werden zunächst die Workflowelemente aus W gefiltert, sodass nur noch die Elemente mit dem Status „ready“ und „active“ als Kandidaten in Betracht kommen. Anschließend werden sie durch eine Gültigkeitsschwelle aus der Menge Θ auf ihre Validität überprüft. Das dabei verwendete Ähnlichkeitsmaß zur Bestimmung der Gültigkeit wurde in Abschnitt 4.4.3 vorgestellt. Das gleiche geschieht in der Methode „getDataflowCandidates“ für Datenobjekte. In dieser Methode existiert lediglich kein Filterkriterium für die Kandidatenbestimmung. Da die Datenobjekte als CAKE I Benutzerklassen definiert werden, können die in CAKE I vorhandenen Ähnlichkeitsmaße für die Kandidatenbestimmung bei Datenobjekten verwendet werden.

```
67. begin „getValidCandidates ( workflowElement, threshold Θ )“
68.         cands ← ∅
69.         pos ← set of workflow elements ∈ W with execution state “READY” or “ACTIVE”
70.         for each e ∈ pos do
71.                 sim1 ← sim(workflowElement, e)
72.                 if sim1 ≥ Θ do cands ← cands ∪ { [workflowElement, e, sim1] } end
73.         end
74. return cands;
75. end
76. begin „getDataflowCandidates( dataflowElement, threshold Θ )“
77.         cands ← ∅
78.         dataObjects ← set of dataflow elements ∈ W
79.         for each e ∈ dataObjects do
80.                 sim1 ← sim(dataflowElement, e)
81.                 if sim1 ≥ Θ do cands ← cands ∪ { [dataflowElement, e, sim1] } end
82.         end
83. return cands;
84. end
```

Die Zeilen 85-128 beschreiben den Grundblock zur Adaption einer DELETE-Liste. Dabei wird analog vorgegangen wie beim Grundblock zur Adaption der ADD-Listen mit dem Unterschied, dass die Listen in anderer Reihenfolge abgearbeitet werden müssen. Beim Entfernen eines Datenobjektes und Workflowelements wird zusätzlich überprüft, ob noch Referenzen auf die Elemente bestehen (Zeile 103 und 119). Nur Elemente, die keine Referenz besitzen, dürfen gelöscht werden. Beim Löschen von Workflow-elementen wird zusätzlich sichergestellt, dass die gemappten Elemente der Kette im Workflow als direkte Nachbarn vorliegen (Zeile 114). Dadurch wird garantiert, dass Ketten nur vollständig und nicht partiell gelöscht werden. An dieser Stelle wären auch andere Lösungen vorstellbar, wie bspw. die teilweise Anwendung einer Kette oder die Untersuchung der Löschbarkeit von Elementen, die innerhalb eines Blocks liegen.

```
85. begin „adaptation of delete list“
86. mappedOperations ← ∅
87.         for each op ∈  A such that op.operationType = DELETE do
88.                 if ( op.isDatalink ) {
89.                         wflCandidates ← getValidCandidates( op.wflElement, Θ )
90.                         dataflowCandidates ← getDataflowCandidates(op.dataflowElement, Θ )
91.                         op.dataflowElement ← best( dataflowCandidates )
92.                         for each wflCandidate ∈ wflCandidates do
93.                                 op.wflElement ← wflCandidate
94.                                 sim1 ← sim( op.dataflowElement ) + sim( op.wflElement )
95.                                 if wflCandidate.hasInputContainer( dataflowCandidate ) and
96.                                 sim1 ≥ Θ do mappedOperations ← mappedOperations ∪ { op }
97.                         end
98.                 }
99.                 if ( op.isDataflowObject ) {
100.                applyOnWorkflow( mappedOperations ) // any previous changes must be applied
101.                dataflowCandidates ← getDataflowCandidates( op.dataflowElement, Θ )
102.                op.dataflowElement ← best( dataflowCandidates )
103.                if noDatalinkIsReferencingDataflowElement( op.dataflowElement ) {
104.                        mappedOperations ← mappedOperations ∪ { op }
105.                }
106.                }
107.                if ( op.isStructuralChange )  {
108.                        applyOnWorkflow( mappedOperations )
109.                        pos ← 0
110.                        for each wflElement ∈ op.innerSequence do
```

```
111.                                    wflCandidates ← getValidCandidates( wflElement, Θ )
112.                                    op.innerSequenceAtPos( pos++ ) ← best( wflCandidates )
113.                            end
114.                    if isTightPair(op.innerSequence) { // check mapped elements in inner sequence
115.                            op ← adaptAnchors( op )
116.                            if posIsBefore( op.preAnchor, op.innerSequence ) and
117.                            posIsAfter( op.postAnchor, op.innerSequence ) {
118.                                    sim2 ← sim(op.preAnchor) + sim(op.postAnchor)
119.                                    if sim2 ≥ Θ and noDatalinksIsReferencing(op.innerSequence)  {
120.                                    mappedOperations ← mappedOperations ∪ { op }
121.                                    applyOnWorkflow( mappedOperations )
122.                                    }
123.                            }
124.                    }
125.                    }
126.            end
127.return W;
128.end
```

Zeile 129-152 behandelt die Methode „applyOnWorkflow“, die als Zwischenschritt häufig aufgerufen wird. Dadurch wird sichergestellt, dass z. B. gelöschte Elemente nicht mehr als Kandidat in Frage kommen und neu eingefügte Elemente als Kandidaten beachtet werden.

```
129. begin „applyOnWorkflow( Operation operations )“ // mapped operations
130.            for each tmp ∈ operations do
131.                    op ← tmp.next
132.                    operations ← operations \ { tmp } // so no operation is repeated
133.            if ( op.isStructuralChange )  {
134.                    usedPreAnchors ← usedPreAnchors ∪ { op.preAnchor }
135.                    usedPostAnchors ← usedPostAnchors ∪ { op.postAnchor }
136.                    if op.operationType = DEL {
137.                            for each wflElement ∈ op.innerSequence do W ← W \ { wflElement }
138.                    }
139.                    if op.operationType = ADD {
140.                            // insert between anchors
141.                            for each wflElement ∈ op.innerSequence do W ← W ∪ { wflElement }
142.                    }
143.            }
144.            if ( op.isDataObject ) {
145.            if op.operationType = DEL {W.dataContext ← W.dataContext  \ {op.dataflowElement } }
146.            if op.operationType = ADD {W.dataContext ← W.dataContext  ∪ {op.dataflowElement }}
147.            }
148.            if ( op.isDatalink ) {
149.            if op.operationType = DEL {removeInputElement(op.workflowElement, op.dataflowElement
      ) }
150.            if op.operationType = ADD { addInputElement( op.workflowElement, op.dataflowElement )}
151.            }
152.            end
153. End
```

5.6 Zusammenfassung und Beurteilung

Dieses Kapitel hat gezeigt welche Komponenten für die CAKE III Plattform geschaffen wurden, um die automatische Adaption von Workflows zu verwirklichen. Um die Adaptation Cases mit einer CBR-Engine nutzen zu können, musste das CAKE I System erweitert werden. Es wurden neue Systemklassen implementiert, die das CAKE I Datenmodell erweitern, und ein einfaches Interface zur Umsetzung der ADD- und DELETE-Listen entworfen. Die modifizierte CAKE I CBR-Engine und das CAKE II-WfMS werden fortan vom Adaptation Manager genutzt, um den automatischen Adaptionsprozess von Workflows durchzuführen. Der Adaptation Manager beinhaltet die Adaptionsalgorithmen und eine zusätzliches Logikmodul. Das Logikmodul erlaubt Editieroperationen auf einem Retrieval Workflow aus einem Adaptation Case. Dies ist notwendig, um für einen Workflow, der adaptiert werden soll, anwendbare ADD- und DELETE-Listen zu erhalten. Nachdem alle relevanten Softwarekomponenten vorgestellt wurden, konnte die Kommunikation innerhalb der CAKE III Plattform erklärt werden. In dem dafür verwendeten UML Aktivitätsdiagramm wurden die Aufgaben der beteiligten Systeme während des Adaptionsprozesses festgelegt. Abschließend wurde unter Beachtung der vorgestellten Implementierungsdetails der CAM Algorithmus detailliert beschrieben.

Wie bereits in der Zusammenfassung des Konzeptteils beschrieben, befindet sich die CAKE III Plattform noch in der Entwicklung. Es existiert noch keine Serviceschicht für den Adaptation Manager, die die Adaptionsalgorithmen als Webservice verfügbar macht. Dies wäre der nächste Schritt um die webbasierte Modellierungsoberfläche mit dem Adaptation Manager zu verknüpfen. Durch diese Verbindung wäre es dann möglich neue Anwendungsdomänen leichter zu erfassen, da die Adaptation Cases automatisch erstellt werden könnten. Bisher wurde nur die Kochdomäne für die Realisierung eines Datenflusses näher untersucht, da Kochworkflows nicht von Experten erstellt werden müssen. Neue Anwendungsdomänen könnten dazu führen, dass das Konzept wechselseitig mit der Implementierung verbessert werden muss. Die Arbeit mit größeren Fallbasen aus anderen Anwendungsdomänen wird dann womöglich aufwendigere Adaptionsalgorithmen mit verbesserten Ähnlichkeitsmaßen verlangen.

Kapitel 6 - Schlussbemerkungen

Jede Aktivität, die zielorientiert und wiederholt durchgeführt wird, kann als Prozess erfasst werden. Die Experimente an Kochprozessen haben gezeigt, dass prozessorientiertes Handeln nicht immer ein betriebswirtschaftliches Ziel verfolgen muss. Inwiefern der Gedanke der Prozessorientierung auch in einem nicht-betriebswirtschaftlichen Umfeld genutzt wird, hängt auf der einen Seite von der Akzeptanz und der Vorteilhaftigkeit der Prozessorientierung ab und auf der anderen Seite wohl auch vom Grad der Affinität der Menschen für neue Denkweisen. Neben der Erfassung von Kochrezepten lassen sich auch andere Anwendungsdomänen finden, die neuen gesellschaftlichen Bedürfnissen entspringen und als Prozess festgehalten werden können. Das Bedürfnis der Menschen nach gesunder Ernährung und körperlicher wie geistiger Gesundheit ist in den letzten Jahren gestiegen[29]. In diesem Zusammenhang werden sportliche Ziele stärker analysiert und durch Trainings- und Ernährungspläne unterstützt. Nicht nur professionelle Trainer, sondern auch zunehmend Laien probieren sich am Aufbau solcher Pläne. Sie stützen sich dabei auf selbst angeeignetes Wissen, das z. B. innerhalb von Internetforen gesammelt wird. Ein entscheidender Faktor auf dem Weg einer prozessorientierten Erfassung solcher Pläne liegt wohl darin, wie stark solche Gruppen ein WfMS nutzen möchten. Dabei spielt die Schnittstelle zwischen den Benutzern und dem WfMS eine entscheidende Rolle. Es ist fraglich, ob die Abbildung eines Prozesses durch einen Kontrollflusses für alle Benutzergruppen ein gutes Modellierungswerkzeug darstellt. Im ersten Kapitel wurde bereits auf den klinischen Sektor Bezug genommen. Für die Behandlung von Patienten in Krankenhäusern existieren Prozesse, die für die Kalkulation der Behandlungskosten genutzt werden. In der Langzeitbetreuung von Patienten in privaten Praxen ist eine solche Erfassung nicht üblich. In der Regel wird für jeden Patienten eine Karteikarte angelegt, die die Krankengeschichte der Person beinhaltet. Auch hier lässt sich diskutieren, ob das Assistenzpersonal auf die klassische Workflowmodellierung zurückgreifen möchte. Außerhalb eines technischen oder betriebswirtschaftlichen Umfelds sollten den Menschen Modellierungswerkzeuge an die Hand gegeben werden, die sie zunächst von der prozessorientierten Sicht fernhalten und die sich von der ursprünglichen Form der Informationserfassung nur wenig unterscheiden. Wenn solche abstrahierenden Schnittstellen für eine Anwendungsdomäne entwickelt werden können, kann auf der technische Ebene immer noch eine automatische Adaption eines Workflows durchgeführt werden, die durch einen Änderungswunsch des Benutzers ausgelöst wird. Es könnte durchaus erstrebenswert sein, die Behandlung eines Patienten in der Langzeittherapie zu formalisieren, um sich bei der Behandlung eines anderen Patienten

[29] Für viele Artikel zu diesem Thema sei hier auf die Gesundheitsberichterstattung des Bundes hingewiesen. Link: http://www.gbe-bund.de

Optionen aufzeigen zu lassen. Der Wissensbestand einiger Foren ist durch die Kollaboration seiner Benutzer zu einer beachtlichen Größe herangewachsen. Könnte dieses Wissen zur Lösung konkreter Probleme durch neue Modellierungswerkzeuge oder eine Zwischenschicht als Prozess erfasst werden, würde auch hier die Möglichkeit entstehen, den Menschen eine neue Form der Entscheidungsunterstützung zu bieten. Konkret im Sportbereich könnte den Benutzern eines solchen Systems alternative Trainingsmöglichkeiten offeriert werden, die bspw. schonender oder effektiver für bestimmte Bewegungsabläufe sind. Die Mühe und der Aufwand, die hinter der Entwicklung solcher neuer Schnittstellen steht, kann damit gerechtfertig werden, dass das hier vorgestellte Konzept einer automatischen Adaption von Workflows direkt von der Größe der Fallbasis abhängig ist. Wenn die Kollaboration von Internet-Communities genutzt werden kann, dann würde jeder Benutzer zur Erweiterung der Fallbasis beitragen. Je mehr Fälle mit unterschiedlichen Problemlösungen in der Fallbasis vorhanden sind, desto höher ist der Nutzen für alle Benutzer.

In Unternehmen werden Prozesse schon seit Langem eingesetzt und zum Teil auch implizit gesetzlich gefordert[30]. Die Hürde zur Verständlichkeit eines Kontrollflusses ist hier niedriger anzusehen, da sich die Unternehmen und damit ihre Mitarbeiter in der Regel schon mit Prozessen auseinandersetzen mussten. Je nach Größe verfügt ein Unternehmen mittlerweile auch über eigene Prozessmodellierer und/oder Prozess-manager. In Unternehmen existieren also Prozesse, die mit semantischem Wissen angereichert werden können und somit für spätere Modellierungen als Wissensbasis dienen können. Der Vorteil bei der Akquirierung von Prozessen steht bei Unternehmen jedoch im Gegensatz zur freien Verfügbarkeit dieser Prozesse. Weitere Forschungen am Ankerprinzip durch seinen Einsatz in verschiedenen Unternehmensbranchen können nur durch Kooperation von Unternehmen gelingen. Diese müssen abwägen zwischen der Herausgabe von Firmengeheimnissen[31] und den Vorteilen durch die Entscheidungs-unterstützung während der Modellierungsphase des Prozesses und dessen Abarbeitung. Die Anpassung des Konzepts an eine Anwendungsdomäne erfordert jedoch, dass ein Unternehmen zumindest eine Teilmenge seiner Prozesse zur Verfügung stellt. Anhand dieser Prozesse kann dann festgestellt werden, ob die vorhandene Prozessmodellierung nutzbar ist für eine automatische Adaption oder ob sie zuerst umgewandelt werden muss. Anschließend können die Ähnlichkeitsmaße so konstruiert werden, dass sie während des Retrievals der Adaptation Cases genutzt werden können und die Anker auf sinnvolle Positionen gemappt werden. Ohne vorherige Untersuchungen kann einem Unternehmen keine fertige Komplettlösung offeriert werden. Eine weiteres Hindernis für

[30] Aktiengesellschaften müssen seit dem 1. Mai 1998 durch das "Gesetz zur Kontrolle und Transparenz im Unternehmensbereich" (KonTraG) und seit dem 29. Mai 2009 zusätzlich durch das "Bilanzrechts-modernisierungsgesetz" (BilMog) ein aktives Risikomanagement durchführen müssen.

[31] In der Regel werden Verschwiegenheitsvereinbarungen (engl. Non-disclosure agreements) getroffen, die bei einem Vertragsbruch kodifizierte Vertragsstrafen nach sich ziehen.

den Einsatz dieses Konzeptes in Wirtschaft und Industrie stellt die verwendete Software der Unternehmen zur Prozessmodellierung und -ausführung dar. Viele Unternehmen nutzen Business Prozess Management Systeme (BPMS) zur Modellierung und Ausführung ihrer Prozesse, die gar keine Unterstützung für dynamische Prozesse bieten. Vermutlich würden große Unternehmen ihre IT-Infrastruktur nicht komplett umstellen, um ein agiles WfMS wie CAKE II zu nutzen. Die optimalen Grundvoraussetzungen zur Nutzung des Konzeptes sind wohl dann gegeben, wenn es sich um ein mittelständisches Unternehmen handelt, dass erst damit angefangen hat Prozesse festzuhalten und eine Innovationskultur besitzt, die Neues schnell integrieren kann. Die Nutzung herkömmlicher BPMS führt dazu, dass der Prozessmodellierer nur während der Workflowkomposition unterstützt werden kann, da das darunterliegende WfMS keine Änderungen zur Laufzeit mittels Ad-hoc Änderungen unterstützt. Es gibt allerdings bereits Ansätze, die verbreitete BPMS nachträglich um agile Workflowtechniken erweitern [WEH08]. Die Integration agiler Workflowtechniken in bestehende Softwarelösungen würde dazu führen, dass das hier vorgestellte Konzepte zur automatischen Workflowadaption auch von großen Unternehmen genutzt werden kann, ohne dass diese ihre IT-Infrastruktur umstellen müssen.

Bei der Anpassung von Workflows durch Adaptation Cases gehören die Größe der verfügbaren Fallbasis und die Güte der Ähnlichkeitsmaße zu den wichtigsten Faktoren, um dem Benutzer ein brauchbares System zu liefern. Weitere Forschungen an dem Konzept könnten Ontologie-basierte Ähnlichkeitsmaße für die Kochdomäne beinhalten, um daraus Ernährungsvorschläge bei Diäten oder auf ein Training abgestimmte Essenspläne zu liefern. Die Frage, die dabei zu klären wäre, ist die Art der Präsentation des Adaptionsergebnisses für den Benutzer. Der Vorteil dieser Forschung läge darin, dass es keine Abhängigkeit zu einem externen Unternehmen gäbe. Die Kehrseite dieser Forschung liegt in der Frage, ob die Schnittstelle zwischen Benutzer und System so geschaffen werden kann, dass sie auch wirklich genutzt wird. Falls ein Unternehmen Interesse an einer automatischen Adaption von Workflows hat, kann eher davon ausgegangen werden, dass das System auch eingesetzt werden soll. Es sind wohl die neugegründeten und wissensintensiven Unternehmen, die die größten Vorteile daraus ziehen, wenn sie ein agiles WfMS einsetzen in Kombination mit einem System zur automatischen Adaption von Workflows, wie es hier vorgestellt wurde. Die prinzipielle Realisierbarkeit des Konzeptes wurde hier gezeigt.

Anhang

Anhang 1 - XML Schema für Adaptation Cases

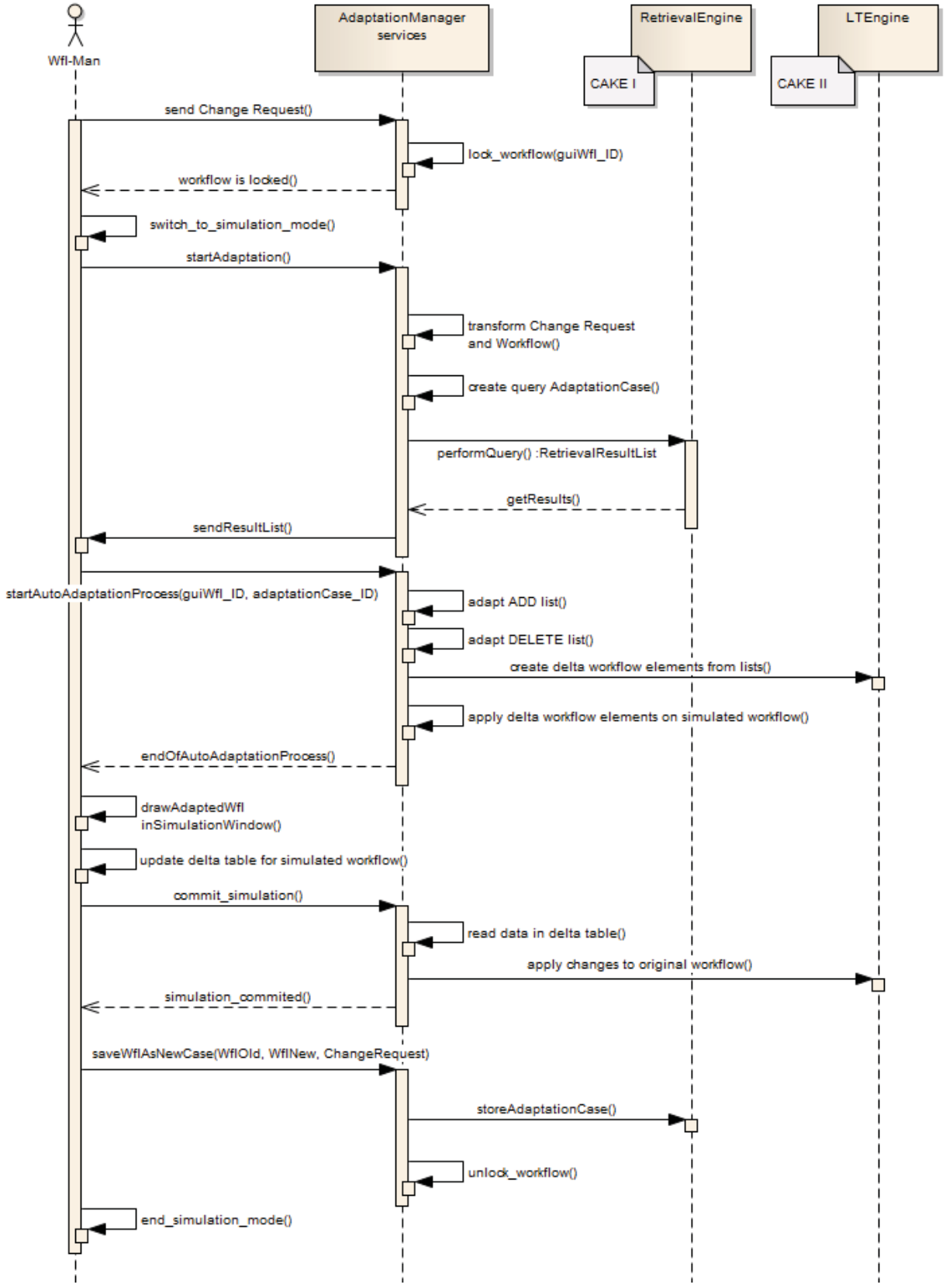
```
1.  <schema xmlns="http://www.w3.org/2001/XMLSchema"
2.  targetNamespace="http://cake.wi2.uni-trier.de/xml/rwfl"
3.  xmlns:rwfl="http://cake.wi2.uni-trier.de/xml/rwfl"
4.  xmlns:cdol="http://cake.wi2.uni-trier.de/xml/cdol"
5.  elementFormDefault="qualified">
6.  <import namespace="http://cake.wi2.uni-trier.de/xml/cdol"
7.  schemaLocation="cdol.xsd"/>
8.  <element name="Repository">
9.  <complexType>
10.         <choice minOccurs="0" maxOccurs="unbounded">
11.             <element ref="rwfl:AdaptationCase"/>
12.         </choice>
13. </complexType>
14. </element>
15. <element name="AdaptationCase">
16. <complexType>
17.         <sequence maxOccurs="1" minOccurs="0">
18.             <element ref="rwfl:ChangeRequest" minOccurs="0" maxOccurs="1" />
19.             <element name="XmlWflBefore" minOccurs="0" maxOccurs="1" />
20.             <element name="XmlWflAfter" minOccurs="0" maxOccurs="1" />
21.             <element ref="rwfl:Workflow" minOccurs="1" maxOccurs="1"/>
22.             <element ref="rwfl:ADDList" minOccurs="0" maxOccurs="1"/>
23.             <element ref="rwfl:DELList" minOccurs="0" maxOccurs="1"/>
24.         </sequence>
25. </complexType>
26. </element>
27. <element name="ChangeRequest">
28. <complexType>
29.         <complexContent>
30.             <extension base="cdol:ObjectType">
31.                 <sequence minOccurs="0" maxOccurs="unbounded">
32.                 <choice>
33.                 <element ref="cdol:AA"/>
34.                 <element ref="cdol:OA"/>
35.                 </choice>
36.                 </sequence>
37.             </extension>
38. </complexContent>
39. </complexType>
40. </element>
41. <element name="ADDList">
42. <complexType>
43.         <sequence>
44.             <element ref="rwfl:Block" minOccurs="0" maxOccurs="unbounded"/>
45.             <element ref="rwfl:DataflowElement" minOccurs="0" maxOccurs="unbounded"/>
```

```
46.                     <element ref="rwfl:DataLink" minOccurs="0" maxOccurs="unbounded" />
47.             </sequence>
48.     </complexType>
49.     </element>
50.     <element name="DELList">
51.     <complexType>
52.             <sequence>
53.                     <element ref="rwfl:DataLink" minOccurs="0" maxOccurs="unbounded"/>
54.                     <element ref="rwfl:Block" minOccurs="0" maxOccurs="unbounded"/>
55.                     <element ref="rwfl:DataflowElement" minOccurs="0" maxOccurs="unbounded"/>
56.             </sequence>
57.     </complexType>
58.     </element>
59.     <element name="DataLink">
60.     <complexType>
61.             <sequence>
62.                     <element ref="rwfl:DataflowElement" minOccurs="1" maxOccurs="1"/>
63.                     <element name="WorkflowElement">
64.                     <complexType>
65.                             <choice>
66.                             <element name="Task" minOccurs="0" maxOccurs="1"
    type="rwfl:TaskType"/>
67.                             <element name="Node" minOccurs="0" maxOccurs="1"
    type="rwfl:NodeType"/>
68.                             </choice>
69.                     </complexType>
70.                     </element>
71.             </sequence>
72.     </complexType>
73.     </element>
74.     <element name="Block" type="rwfl:StructureBlockType" />
75.     <complexType name="StructureBlockType">
76.     <sequence>
77.             <element name="pre" minOccurs="1" maxOccurs="1">
78.             <complexType>
79.                     <choice>
80.                             <element name="Task" minOccurs="0" maxOccurs="1"
    type="rwfl:TaskType"/>
81.                             <element name="Node" minOccurs="0" maxOccurs="1"
    type="rwfl:NodeType"/>
82.                     </choice>
83.             </complexType>
84.             </element>
85.             <element ref="rwfl:Sequence" minOccurs="0" maxOccurs="1"/>
86.             <element name="succ" minOccurs="1" maxOccurs="1">
87.                     <complexType>
88.                     <choice>
89.                             <element name="Task" minOccurs="0" maxOccurs="1"
    type="rwfl:TaskType"/>
90.                             <element name="Node" minOccurs="0" maxOccurs="1"
    type="rwfl:NodeType"/>
91.                     </choice>
92.                     </complexType>
93.             </element>
94.     </sequence>
95.     </complexType>
96.     <element name="Workflow">
97.     <complexType>
98.             <sequence>
99.                     <element ref="rwfl:DataObjects"  minOccurs="0" maxOccurs="1"/>
100.                    <element ref="rwfl:Sequence" minOccurs="1" maxOccurs="1"/>
```

```
101.            </sequence>
102.            <attribute name="refID" use="required" type="int" />
103. </complexType>
104. <key name="elementID">
105.            <selector xpath=".//rwfl:Task | .//rwfl:Node | ." />
106.                    <field xpath="@refID" />
107. </key>
108. <key name="DataObjectID">
109.            <selector xpath=".//rwfl:DataflowObjectId" />
110.                    <field xpath="." />
111. </key>
112. <keyref refer="rwfl:DataObjectID" name="DataObjectRefIdCheck">
113.            <selector xpath=".//rwfl:DataflowObjectRefId"/>
114.                    <field xpath="."/>
115. </keyref>
116. </element>
117. <element name="Sequence">
118. <complexType>
119.            <choice maxOccurs="unbounded" minOccurs="0">
120.                    <element name="Task" type="rwfl:TaskType" />
121.                    <element name="Node" type="rwfl:NodeType" />
122.            </choice>
123.            <attribute name="refID" use="optional" type="int" />
124. </complexType>
125. </element>
126. <element name="DataObjects">
127.            <complexType>
128.                    <sequence minOccurs="0" maxOccurs="unbounded">
129.                            <element ref="rwfl:DataObjectWrapper" />
130.                    </sequence>
131.            </complexType>
132. </element>
133. <element name="DataObjectWrapper">
134. <complexType>
135.            <sequence minOccurs="1" maxOccurs="1">
136.                    <element ref="rwfl:DataflowElement" minOccurs="1" maxOccurs="1" />
137.                    <element name="DataflowObjectId" type="string" minOccurs="1" maxOccurs="1"/>
138.                    <element name="childs" maxOccurs="1" minOccurs="0">
139.                    <complexType>
140.                    <sequence>
141.                    <element ref="rwfl:DataObjectWrapper" minOccurs="0" maxOccurs="unbounded"/>
142.                    </sequence>
143.                    </complexType>
144.                    </element>
145.            </sequence>
146. </complexType>
147. </element>
148. <element name="DataflowElement">
149. <complexType>
150.            <complexContent>
151.                    <extension base="cdol:ObjectType"><!-- Object ist z.B. Agg -->
152.                    <sequence minOccurs="0" maxOccurs="unbounded">
153.                            <choice>
154.                                    <element ref="cdol:AA"/>
155.                                    <element ref="cdol:OA"/>
156.                            </choice>
157.                    </sequence>
158.                    </extension>
159.            </complexContent>
160. </complexType>
161. </element>
```

```
<complexType name="NodeType">
<sequence maxOccurs="unbounded" minOccurs="0">
        <element name="status" type="string" maxOccurs="1" minOccurs="0" />
        <element ref="rwfl:Sequence" />
</sequence>
<attribute name="type" use="required" type="rwfl:enumNodeType" />
<attribute name="refID" use="required" type="int" />
</complexType>
<complexType name="TaskType">
<sequence>
        <element name="status" type="string" maxOccurs="1" minOccurs="0" />
        <element name="name" type="string" maxOccurs="1" minOccurs="0" />
        <element name="description" type="string" minOccurs="0" maxOccurs="1" />
        <element name="input" type="rwfl:DataflowObjectIdSet" minOccurs="0" maxOccurs="1" />
        <element name="output" type="rwfl:DataflowObjectIdSet" maxOccurs="1" minOccurs="0" />
</sequence>
<attribute name="type" use="required" type="rwfl:enumTaskType" />
<attribute name="refID" use="required" type="int" />
</complexType>
<complexType name="DataflowObjectIdSet">
        <sequence minOccurs="0" maxOccurs="unbounded">
                <element name="DataflowObjectRefId" minOccurs="1" maxOccurs="1" type="string"/>
        </sequence>
</complexType>
<simpleType name="enumNodeType">
        <restriction base="string">
                <enumeration value="AND" />
                <enumeration value="XOR" />
                <enumeration value="Loop" />
        </restriction>
</simpleType>
<simpleType name="enumTaskType">
        <restriction base="string">
                <enumeration value="WorklistTask" />
                <enumeration value="ServiceTask" />
                <enumeration value="PlaceholderTask" />
        </restriction>
</simpleType>
</schema>
```

Anhang 2 - CAKE III Sequenzdiagramm einer automatischen Adaption



Anhang 3 - XSLT zur Erzeugung des Retrieval Formats für Workflows

```
1.  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2.  xmlns:wfl="cake.ltworkflow.core.workflow" version="1.0">
3.  <xsl:template match="text()" />
4.  <xsl:output indent="yes"/>
5.  <xsl:template match="wfl:workflow">
6.          <Workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
7.                  <xsl:attribute name="refID"><xsl:value-of select="@wfl:id"/></xsl:attribute>
8.                  <xsl:apply-templates />
9.          </Workflow>
10. </xsl:template>
11. <xsl:template match="wfl:sequence/wfl:child_elements">
12.         <Sequence>
13.                 <xsl:apply-templates />
14.         </Sequence>
15. </xsl:template>
16. <xsl:template match="wfl:task">
17.         <Task>
18.                 <xsl:attribute name="type">WorklistTask</xsl:attribute>
19.                 <xsl:attribute name="refID">
20.                         <xsl:value-of select="@wfl:id"/>
21.                 </xsl:attribute>
22.                 <status>
23.                         <xsl:value-of select="./wfl:engine_data/wfl:status/text()"/>
24.                 </status>
25.                 <name>
26.                         <xsl:value-of select="./wfl:gui_data/wfl:name/text()"/>
27.                 </name>
28.                 <description>
29.                         <xsl:value-of select="./wfl:work_description/text()"/>
30.                 </description>
31.         </Task>
32. </xsl:template>
33. <xsl:template match="wfl:placeholdertask">
34.         <Task>
35.                 <xsl:attribute name="type">PlaceholderTask</xsl:attribute>
36.                 <xsl:attribute name="refID">
37.                         <xsl:value-of select="@wfl:id"/>
38.                 </xsl:attribute>
39.                 <status>
40.                         <xsl:value-of select="./wfl:engine_data/wfl:status/text()"/>
41.                 </status>
42.                 <name>
43.                         <xsl:value-of select="./wfl:gui_data/wfl:name/text()"/>
44.                 </name>
45.         </Task>
46. </xsl:template>
47. <xsl:template match="wfl:and">
48.         <Node>
49.                 <xsl:attribute name="type">AND</xsl:attribute>
50.                 <xsl:attribute name="refID"><xsl:value-of select="@wfl:id"/></xsl:attribute>
51.                 <status>
52.                         <xsl:value-of select="./wfl:engine_data/wfl:status/text()"/>
53.                 </status>
```

```
54.                     <xsl:apply-templates select="wfl:child_elements" />
55.             </Node>
56.     </xsl:template>
57.     <xsl:template match="wfl:xor">
58.             <Node>
59.                     <xsl:attribute name="type">XOR</xsl:attribute>
60.                     <xsl:attribute name="refID"><xsl:value-of select="@wfl:id"/></xsl:attribute>
61.                     <status>
62.                             <xsl:value-of select="./wfl:engine_data/wfl:status/text()"/>
63.                     </status>
64.                     <xsl:apply-templates select="wfl:child_elements" />
65.             </Node>
66.     </xsl:template>
67.     <xsl:template match="wfl:loop">
68.             <Node>
69.                     <xsl:attribute name="type">Loop</xsl:attribute>
70.                     <xsl:attribute name="refID"><xsl:value-of select="@wfl:id"/></xsl:attribute>
71.                     <status>
72.                             <xsl:value-of select="./wfl:engine_data/wfl:status/text()"/>
73.                     </status>
74.                     <xsl:apply-templates select="wfl:child_elements" />
75.             </Node>
76.     </xsl:template>
77.     </xsl:stylesheet>
```

Anhang 4 - Pseudocode des Composite Anchor Mappings

```
1.  input:
2.          W: target workflow a set of workflow elements with a precedence relation,
            A: set of transformational operations [structural changes, dataflow elements, datalinks;
            operation type],
            Θ: several validity thresholds

3.  global:
4.          usedPreAnchors ← ∅
5.          usedPostAnchors ← ∅
6.  begin „adaptation of add list“
7.  mappedOperations ← ∅
8.  for each op ∈  A such that op.operationType = ADD do
9.  if ( op.isStructuralChange )  {
10.         op ← adaptAnchors( op )
11.         sim1 ← sim( op.preAnchor ) + sim( op.postAnchor )
12.         if sim1 ≥ Θ do
13.                 mappedOperations ← mappedOperations ∪ { op }
14.                 usedPreAnchors ← usedPreAnchors ∪ { op.preAnchor }
15.                 usedPostAnchors ← usedPostAnchors ∪ { op.postAnchor }
16.         end
17. }
18. if ( op.isDataflowObject ) {
19.         mappedOperations ← mappedOperations ∪ { op } // insertion without check for duplicates
20. }
21. if ( op.isDatalink ) {
22.         applyOnWorkflow( mappedOperations ) // any previous changes must be applied
23.         wflCandidates ← getValidCandidates( op.wflElement, Θ )
24.         dataflowCandidates ← getDataflowCandidates( op.dataflowElement, Θ )
25.         op.wflElement ← best( wflCandidates ) // determine best of all candidates
26.         op.dataflowElement ← best( dataflowCandidates )
27.         mappedOperations ← mappedOperations ∪ { op }
28. }
29. end
30.         applyOnWorkflow( mappedOperations ) // insert datalinks and dataflow objects
31. return W;
32. end
33. begin „adaptAnchors(Operation op )“
34. validPairs ← ∅
35. if ( op.isStructuralChange ) {
36.         preCandidates ← getValidCandidates( op.preAnchor, Θ )
37.         postCandidates ← getValidCandidates( op.postAnchor, Θ )
38.         for each pre ∈ preCandidates do
39.                 if pre not ∈ usedPreAnchors {
40.                 for each post ∈ postCandidates do
41.                 if post not ∈ usedPostAnchors {
42.                         areTight ← false
43.                         if op.operationType = DEL or isSubstitution( op ) {
44.                 areTight ← ( W.getElement( pre ).siblingAtPos( op.innerSequence.length ) = post)
45.                         } else {
46.                                 areTight ← ( W.getElement( pre ).nextSibling = post )
47.                         }
48.                         if areTight  {
49.                                 validPairs ← validPairs ∪ { ( pre, sim(pre), post, sim(post) ) }
50.                         }
51.                 }
52.                 end
53.                 }
```

```
54.             end
55.
56.  for each pre ∈ preCandidates such that pre not ∈ usedPreAnchors do
57.             validPairs ← validPairs ∪ { ( pre, sim(pre), NIL, 0 ) }
58.  end
59.  for each post ∈ postCandidtes such that post not ∈ usedPostAnchors do
60.             validPairs ← validPairs ∪ { ( NIL, 0, post, sim(post) ) }
61.  end
62.             bestPair ← best(validPairs)
63.             op.preAnchor ← bestPair.pre
64.             op.postAnchor ← bestPair.post
65.  }
66.  end
67.  begin „getValidCandidates ( workflowElement, threshold Θ )“
68.             cands ← ∅
69.             pos ← set of workflow elements ∈ W with execution state “READY” or “ACTIVE”
70.             for each e ∈ pos do
71.                         sim1 ← sim(workflowElement, e)
72.                         if sim1 ≥ Θ do cands ← cands ∪ { [workflowElement, e, sim1] } end
73.             end
74.  return cands;
75.  end
76.  begin „getDataflowCandidates( dataflowElement, threshold Θ )“
77.             cands ← ∅
78.             dataObjects ← set of dataflow elements ∈ W
79.             for each e ∈ dataObjects do
80.                         sim1 ← sim(dataflowElement, e)
81.                         if sim1 ≥ Θ do cands ← cands ∪ { [dataflowElement, e, sim1] } end
82.             end
83.  return cands;
84.  end
85.  begin „adaptation of delete list“
86.  mappedOperations ← ∅
87.             for each op ∈  A such that op.operationType = DELETE do
88.                         if ( op.isDatalink ) {
89.                                     wflCandidates ← getValidCandidates( op.wflElement, Θ )
90.                                     dataflowCandidates ← getDataflowCandidates(op.dataflowElement, Θ )
91.                                     op.dataflowElement ← best( dataflowCandidates )
92.                                     for each wflCandidate ∈ wflCandidates do
93.                                                 op.wflElement ← wflCandidate
94.                                                 sim1 ← sim( op.dataflowElement ) + sim( op.wflElement )
95.                                                 if wflCandidate.hasInputContainer( dataflowCandidate ) and
96.                                                 sim1 ≥ Θ do mappedOperations ← mappedOperations ∪ { op }
97.                                     end
98.                         }
99.                         if ( op.isDataflowObject ) {
100.                        applyOnWorkflow( mappedOperations ) // any previous changes must be applied
101.                        dataflowCandidates ← getDataflowCandidates( op.dataflowElement, Θ )
102.                        op.dataflowElement ← best( dataflowCandidates )
103.                        if noDatalinkIsReferencingDataflowElement( op.dataflowElement ) {
104.                                    mappedOperations ← mappedOperations ∪ { op }
105.                        }
106.                        }
107.                        if ( op.isStructuralChange )  {
108.                                    applyOnWorkflow( mappedOperations )
109.                                    pos ← 0
110.                                    for each wflElement ∈ op.innerSequence do
111.                                                wflCandidates ← getValidCandidates( wflElement, Θ )
112.                                                op.innerSequenceAtPos( pos++ ) ← best( wflCandidates )
113.                                    end
114.                        if isTightPair(op.innerSequence) { // check mapped elements in inner sequence
115.                                    op ← adaptAnchors( op )
116.                                    if posIsBefore( op.preAnchor, op.innerSequence ) and
```

```
                                posIsAfter( op.postAnchor, op.innerSequence ) {
                                    sim2 ← sim(op.preAnchor) + sim(op.postAnchor)
                                      if sim2 ≥ Θ and noDatalinksIsReferencing(op.innerSequence)  {
                                        mappedOperations ← mappedOperations ∪ { op }
                                         applyOnWorkflow( mappedOperations )
                                      }
                                }
                        }
                        }
        end
return W;
end
begin „applyOnWorkflow( Operation operations )“ // mapped operations
        for each tmp ∈ operations do
                op ← tmp.next
                operations ← operations \ { tmp } // so no operation is repeated
        if ( op.isStructuralChange )  {
                usedPreAnchors ← usedPreAnchors ∪ { op.preAnchor }
                usedPostAnchors ← usedPostAnchors ∪ { op.postAnchor }
                if op.operationType = DEL {
                        for each wflElement ∈ op.innerSequence do W ← W \ { wflElement }
                }
                if op.operationType = ADD {
                        // insert between anchors
                        for each wflElement ∈ op.innerSequence do W ← W ∪ { wflElement }
                }
        }
        if ( op.isDataObject ) {
        if op.operationType = DEL {W.dataContext ← W.dataContext  \ {op.dataflowElement } }
        if op.operationType = ADD {W.dataContext ← W.dataContext  ∪ {op.dataflowElement }}
        }
        if ( op.isDatalink ) {
        if op.operationType = DEL {removeInputElement(op.workflowElement, op.dataflowElement
    ) }
        if op.operationType = ADD { addInputElement( op.workflowElement, op.dataflowElement )}
        }
        end
End
```

Abbildungsverzeichnis

Literaturverzeichnis

[AlthoffEtAl01] Althoff Klaus-Dieter, Decker Björn, Hartkopf Susanne, Jedlitschka Andreas, Nick Markus, Rech Jörg, „Experience Management: The Frauenhofer IESE Experience Factory“, 2001, Proc. Industrial Conference Data Mining, July 24-25, Institute for Computer Vision and applied Computer Sciences, Leipzig

[AP94] Aamodt A., Plaza E., “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches”, 1994, AI Commun., Vol. 7, p. 39-59

[Be02] Bergmann Ralph, “Experience Management : foundations, development methodology, and Internet based applications”, 2002, Springer-Verlag

[BergmannEtAl06] Bergmann R., Freßmann A., Maximini K. Maximini R., Sauer T., “Case-Based Support for Collaborative Business”, T.R. Roth-Berghofer et al. (Eds.): ECCBR 2006, LNAI 4106, pp. 519–533, Springer-Verlag Berlin Heidelberg 2006

[BergmannEtAl09] Bergmann Ralph, Althoff Klaus-Dieter, Minor Mirjam, Reichle Meike, Bach Kerstin, “Case-Based Reasoning - Introduction and Recent Developments”, 2009, BöttcherIT Verlag, Bremen, Künstliche Intelligenz, Heft 1/2009, p. 5-12

[Carbonell83] Carbonell J.G., “Derivational Analogy and Its Role in Problem Solving”, 1983, Proceedings of the National Conference on Artificial Intelligence. Washington, D.C., August 22-26, 1983, p.64-69

[CELP09] Chinthaka E., Ekanayake J., Leake D., Plate B., “CBR Based Workflow Composition Assistant”, 2009, Proceedings of the 2009 Congress on Services, IEEE Computer Society, pp. 352-355

[Fi07] Fischer L., “BPM and Workflow Handbook”, 2007, Published in Association with the Workflow Management Coalition (WfMC)

[Gö09] Görg S. “Erkenntnisse der Neuroökonomie für Leadership”, 2009, Grin Verlag

[GHJV94] Gamma E., Helm R., Johnson R., Vlissides J., “Design Patterns. Elements of Reusable Object-Oriented Software”, 1994, Addison-Wesley Longman, Amsterdam

[GreinerEtAl04] Greiner U., Ramsch J., Heller B., Löffler M., Müller R., Rahm E., “Adaptive Guideline-based Treatment Workflows with AdaptFlow”, 2004, Computer-based Support for Clinical Guidelines and Protocols. Proceedings of the Symposium on Computerized Guidelines and Protocols (CGP 2004). Prague. IOS Press. p. 113-117

[Gr93] Gruber T.R., “A Translation Approach to Portable Ontology Specifications.”, 1993, Knowledge Acquisition 6(2), Special Issue: Current issues in knowledge modeling, pp. 199–221

[GS06] Garrido J., Schlesinger R., “Principles of Modern Operating Systems”, 2006, Class Publishing

[Hu02] Humpl B., “Transfer von Erfahrungen”, 2002, Dissertation Technische Universität Granz, Deutscher Universitäts-Verlag

[HJM04] in Hyun Son, Jung Sun Kim, Myoung Ho Kim, “Extracting the workflow critical path from the extended well-formed workflow schema”, 2004, Journal of Computer and System Sciences 70 (2005), p. 86-106, Elsevier

[Ke09] Kempin. S, “Fallstudie zur Weiterentwicklung von CAKE II in der Verwaltungsdomäne“, Diplomarbeit, Universität Trier, 2009

[La80] Labov W., "Einige Prinzipien linguistischer Terminologie. In: Sprache im sozialen Kontext. Eine Auswahl von Aufsätzen", 1980, Königstein, Athenäum Verlag

[LiEtAl08] Li, C., Reichert, M., Wombacher, A. 2008. On Measuring Process Model Similarity Based on High-Level Change Operations. ER 2008. LNCS 5231. Springer, 248-264.

[LKM08] Leake D., Kendall-Morwick J., “Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance”, 2008, Springer-Verlag, ECCBR 2008, LNAI 5239, pp. 269-283

[Ma06] Maximini Rainer, “Advanced Techniques for Complex Case-Based Reasoning Applications”, 2006, Dr. Hut Verlag

[Mi06] Minor M., „Erfahrungsmanagement mit fallbasierten Assistenzsystemen“, 2006, Dissertation, Humboldt-Universität zu Berlin

[MinorEtAl10a] Minor M, Bergmann R, Görg S., Walter K., “Towards Case-Based Adaptation of Workflows”, 2010, In Stefania Montani, and Isabelle Bichindaritz, editors, ICCBR 2010, LNAI 6176, pages 421-435, Springer-Verlag

[MinorEtAl10b] Minor M, Bergmann R, Görg S., Walter K., “Adaptation of Cooking Instructions Following the Workflow Paradigm”, 2010, In Cindy Marling, editor, ICCBR 2010 Workshop Proceedings

[MSB08] Minor M., Schmalen D., Bergmann R., “XML-based Representation of Agile Workflows”, 2008, In Martin Bichler, Thomas Hess, Helmut Krcmar, Ulrike Lechner, Florian Matthes, Arnold Picot, Benjamin Speitkamp, and Petra Wolf, editors,

Multikonferenz Wirtschaftsinformatik, MKWI 2008, München, Proceedings, pp. 439-440, GITO Verlag

[MSK09] Minor M., Schmalen D., Kempin S., “Demonstration of the Agile Workflow Management System CAKE II Based on Long-Term Office Workflows”, 2009, CEUR Workshop Proceedings, Vol. 489, Paper 5

[MSKB07] Minor M., Schmalen D., Koldehoff A., Bergmann R., „Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning”, 2007, 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007)

[MTSB08] Minor, M., Tartakovski, A., Schmalen, D., Bergmann, R., “Agile Workflow Technology and Case-Based Change Reuse for Long-Term Processes.”, International Journal on Intelligent Information Technologies 4(1), p.80-98 (2008)

[Mü05] Müller J., „Workflow-based Integration“, 2005, Springer-Verlag

[MZM04] Madhusudan, T., Zhao, J.L., Marshall, B., “A case-based reasoning framework for workflow model management “, 2004, Data and Knowledge Engineering, 50: pp. 87-115, 2004.

[Ol04] Olbrich A., „ITIL kompakt und verständlich“, 2004, 3. Auflage Juli 2006, Friedr. Vieweg & Sohn Verlag |GWV Fachverlage GmbH

[OMG] Object Management Group, BPMN Implementors and Quotes, URL: http://www.bpmn.org/BPMN_Supporters.htm , Abgerufen am: 18.04.2010

[RD98] Reichert M, Dadam P., ADEPTflex, 1998, “supporting dynamic changes of workflows without losing control”, Journal of Intelligent Information Systems 1998; 10: 93-129.

[RvdAH06] Russel N., van der Aalst W.M.P., Hofstede ter A., „Workflow Exception Patterns“, 2006, E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 288-302, Springer-Verlag

[RvHS04] Richter-von Hagen C., Stucky W., „Business-Process- und Workflow-Management: Prozessverbesserung durch Prozess-Management“, 2004, Vieweg+Teubner Verlag, 1. Auflage

[RRD03] Reichert M., Rinderle S., Dadam P., “ADEPT Workflow Management System: Flexible Support for Enterprise-Wide Business Processes - Tool Presentation -“, 2003, Springer-Verlag, Berlin

[Sa97] Sadiq W., Orlowska M. E., “On Correctness Issues in Conceptual Modeling of Workflows”, 1997, In Proceedings of the 5th European Conference on Information Systems, Cork, Ireland

[SW02] Schuschel H, Weske M., „Fallbehandlung: Ein Neuer Ansatz zur Unterstützung Prozessorientierter Informationssysteme“, Desel, Weske (Hrsg.) Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen. GI-Edition Lecture Notes in Informatics Nr. P-21. pp 52-63. Bonn: Gesellschaft für Informatik 2002

[TDG07] Tayler Ian J., Deelman E., Gannon D.B., “Workflows for e-science: scientific workflows for grids”, 2007, Springer-Verlag

[vdA03] Van der Aalst W.M.P., „Workflow Patterns“, 2003, Distributed and Parallel Databases, 14, 5-51, 2003, Kluwer Academic Publishers

[vdA98] Van der Aalst W.M.P., „The Application of Petri Nets to Workflow Management”, 1998, The J. Circuits, Systems and Computers, vol. 8, no. 1, pp. 21-66

[vdAP06] Van der Aalst W.M.P., Pesic M. “DecSerFlow: Towards a Truly Declarative Service Flow Language”, 2006, Springer-Verlag

[vdAvHK04] Van der Aalst W.M.P., van Hee Kees M., “Workflow management: models, methods, and systems”, 2004, MIT Press, New Edition

[We07] Weske M., “Business Process Management - Concepts, Languages, Architectures”, 2007, Springer-Verlag

[We08] Weidlich J., „Design and Implementation of a Workflow Enactment Service for Agile Processes”, Diplomarbeit, Universität Trier 2008

[WEH08] Wörzberger R., Ehses N., Heer T., "Adding Support for Dynamics Patterns to Static Business Process Management Systems", 2008, Software composition: 7th international symposium, SC 2008, Budapest, März 29.-30., 2008

[Wi07] Wilhelm R., „Prozessorganisation“, 2007, Oldenbourg Wissenschaftsverlag GmbH

[WFMC08] WFMC-TC-1011 Version 3 Terminology and Glossary English, 17/08/2008, URL: http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html , Abgerufen am: 01.02.2010

[WP08] Wimmel H., Priese L., „Petri-Netze“, 2008, Springer-Verlag, 2.Auflage

[WWB04] Weber B., Wild W., Breu R., “CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning”, 2004, Springer-Verlag, ECCBR 2004, LNAI 3155, pp. 434-448

[Wo02] Wooldridge, M.: Intelligent Agents: The Key Concepts, 2002, Proceedings of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications IISelect, Springer-Verlag