Collaborative Agent-Based Knowledge Engine

Anforderungen – Konzept – Lösungen

Technical Report

Universität Trier
Lehstuhl für Wirtschaftsinformatik II
D-54286 Trier

Dipl.-Inform. Kerstin Maximini
Dr. Rainer Maximini

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Seit dem Jahr 2005 wird im Rahmen unterschiedlicher Forschungsprojekte und Dissertationen am Lehrstuhl für Wirtschaftsinformatik II[1] das Framework *Collaborative Agent-based Knowledge Engine (CAKE)* [FMS05, FMMS05a, FMMS05b, Max06, SMMB06, BFM+06] entwickelt. Ziel war es, eine gute Basis für Unterstützungssysteme in sehr verschiedenen Anwendungsdomänen zu realisieren. Seit der Fertigstellung des ersten Prototypen Ende 2005 wird CAKE kontinuierlich erweitert und um weitere Funktionalitäten ergänzt. Die Grundfunktionalitäten bleiben dabei jedoch unverändert und werden im Folgenden zusammenfassend dargestellt.

Dieser Report beschreibt in Abschnitt 2 die Beweggründe und die Motivation für die Entscheidung zur Entwicklung eines generischen, domänenunabhängigen Frameworks, das in mehreren Forschungsprojekten und Dissertationen Verwendung finden sollte. Hier galt es, den Mehraufwand zur Konzeption und Implementierung einer generischen Lösung gegen den Aufwand zur Realisierung mehrerer, spezialisierter Einzellösungen gegeneinander abzuwägen. Abschnitt 3 stellt dann die betrachteten Anwendungsdomänen vor und leitet schließlich die hauptsächlichen Anforderungen an ein generisches Framework ab. Diese wurden sämtlichst bei der Konzeption und Entwicklung von CAKE berücksichtigt. Den Schwerpunkt dieses Kapitels stellt Abschnitt 4 dar, der die Konzepte und Funktionalitäten von CAKE beschreibt. Hier wird insbesondere den Funktionalitäten, die zur Entwicklung von Appear benötigt und genutzt wurden, entsprechende Aufmerksamkeit gewidmet. Das Kapitel schließt in Abschnitt 5 mit einer knappen Vorstellung von Lösungen für die in Abschnitt 3 vorgestellten Anwendungsdomänen.

[1] http://www.wi2.uni-trier.de

2 Motivation

Unter dem Schlagwort *Rechnergestützte Gruppenarbeit (engl. Computer supported cooperative work, CSCW)* [Gru94, SSU01, CS03] beschäftigt sich seit den 80er Jahren eine ganze Forschungsgemeinde mit der Frage, wie die Zusammenarbeit mehrerer Menschen in einem Team durch eine geeignete IT-Unterstützung verbessert werden kann. Das übergeordnete Ziel dieser Forschung ist es, die Geschäftsprozesse einer Organisation durch geeignete CSCW-Umgebungen so zu verbessern, dass im Idealfall die Qualität des beim Geschäftsprozess entstehenden Endproduktes verbessert und die Kosten für dessen Erzeugung verringert werden können [SSU01].

Auch wenn die im Laufe der Zeit betrachteten Teams sehr unterschiedlicher Natur waren und sich dadurch auch die Einzellösungen zur Unterstützung dieser Teams natürlicherweise stark unterschieden, konnten drei Hauptaufgaben identifiziert werden, die von einer Arbeitsumgebung, die kollaborative Zusammenarbeit unterstützen soll, abgedeckt werden müssen: es handelt sich um die Unterstützung der *Kommunikation* der Teammitglieder, um die Verbesserung der *Koordination* der Teammitglieder und um die Erleichterung der *Kooperation* der Teammitglieder. Um eine erfolgreiche Arbeitsumgebung bereitstellen zu können, müssen die Kommunikations- Koordinations- und Kooperations-Prozesse außerdem systematisch in den Geschäftsprozess der betrachteten Organisation integriert sein. In jüngster Zeit hat sich außerdem die Auffassung verfestigt, dass eine erfolgreiche Kollaborationsunterstützung nicht bei den Teammitgliedern, also den Menschen, aufhören darf: sie muss vielmehr auch *elektronische Wissensquellen nahtlos integrieren* und Möglichkeiten zur Entscheidungsunterstützung bieten, die auf elektronisch verfügbarem Wissen beruhen. Das Verhalten eines solchen Systems sollte außerdem unter *Verwendung früherer Erfahrungen* gesteuert und gelenkt werden können. Beispielsweise können frühere Erfahrungen hilfreich zur Auswahl einer geeigneten Wissensquelle sein oder sie können genutzt werden, um früher bereits erfolgreiche Kollaborationsstrategien in einer konkreten Situation vorzuschlagen und umzusetzen. So fließen in aktuelle CSCW-Lösungen auch immer mehr Techniken ein, die traditionell den Forschungsbereichen des Wissens- und Erfahrungsmanagements und der Entscheidungsunterstützung zuzuordnen sind [SSU01]. Die verschiedenen Forschungsrichtungen wachsen bei der Entwicklung kollaborativer Arbeitsumgebungen also mehr und mehr zusammen.

Kommunikation
Koordination
Kooperation

Integration elektronischen Wissens
Verwendung früherer Erfahrungen

Die Kehrseite dieser Entwicklung ist jedoch, dass dadurch die Unterstützungssysteme immer komplexer werden und dass somit deren Entwicklung immer länger dauert. Bei der Entwicklung müssen einerseits Anforderungen aus betriebswirtschaftlicher Sicht und andererseits aus technischer Sicht erho-

ben und durch das System befriedigt werden: Aus betriebswirtschaftlicher Sicht können die Anforderungen an ein neues System nur nach einer eingehenden Analyse der Art und Weise wie die Teammitglieder aktuell zusammenarbeiten, um ihr Geschäftsziel zu erreichen, und welche Wissensquellen sie aktuell zur Entscheidungsunterstützung nutzen, erhoben werden. Daraus können dann Anforderungen aus technischer Sicht abgeleitet werden, auf welche Weise die Kommunikation, die Koordination und/oder die Kooperation unterstützt werden muss und welche elektronischen Wissensquellen ins System integriert werden müssen. Zur Implementierung eines Unterstützungssystems sind dann zwei Lösungen denkbar: entweder entwickelt man eine auf das Problem zugeschnittene *Individuallösung*, oder man verwendet ein möglichst allgemein gehaltenes Framework, das man für das konkrete Problem konfigurieren und anpassen kann (*Customizing*).

Individuallösung

Vorteil der ersten Variante ist es, dass nur für die Probleme, die aktuell in der Organisation vorkommen, Lösungen programmiert werden müssen. Spielt also beispielsweise die Unterstützung der Kooperation in einem konkreten Geschäftsprozess nur eine untergeordnete Rolle, so muss auch nur wenig Arbeit in die Programmierung einer entsprechenden Lösung investiert werden. Außerdem müssen auch nur die elektronischen Systeme eingebunden werden, die aktuell vorkommen und verwendet werden. Nachteil dieser Variante ist jedoch, dass sie nur im betrachteten Einsatzszenario verwendet werden kann und sie in anderen Szenarien nicht wiederverwendet werden kann.

Customizing eines Frameworks

Vorteil der zweiten Variante ist, dass ein gutes Framework bereits viele Basisfunktionalitäten bereitstellt, die dann sehr einfach auf die bestehende Situation angepasst werden können. Beispielsweise kann ein Framework Techniken zur Kommunikation oder Techniken zur Einbindung einer Datenbank bereitstellen. Unter Zuhilfenahme eines solchen Frameworks können daher die Entwicklungszeit und somit auch die Entwicklungskosten reduziert werden. Nachteil dieser Variante ist jedoch, dass die Entwicklung eines guten Frameworks, das flexibel und generisch aufgebaut ist um in möglichst vielen Szenarien eingesetzt werden zu können, sehr zeitaufwändig und damit teuer ist. Es stellt ggf. Funktionalitäten bereit, die in konkreten Anwendungen nicht benötigt werden und somit „überflüssig“ sind.

Zusammenfassend lohnt sich die Entwicklung eines Frameworks nur dann, wenn es in mehreren Projekten eingesetzt werden soll und wenn der Aufwand des Customizings deutlich geringer ist als der Aufwand der Programmierung einer individuellen Lösung. Außerdem müssen durch das Framework möglichst alle Anforderungen aus betriebswirtschaftlicher und technischer Sicht erfüllbar sein. Zu guter letzt müssen in den verschiedenen Projekten ähnliche An-

forderungen, hauptsächlich aus technischer Sicht, vorkommen, d.h. die Schnittmenge benötigter Funktionalitäten sollte möglichst groß sein.

Am Lehrstuhl für Wirtschaftsinformatik II wurden unter diesem Gesichtspunkten gegen Ende des Jahres 2004 die Anforderungen aus verschiedenen Forschungsprojekten und Dissertationen zusammengetragen und analysiert [CAK04, BFM+06]. Hierbei handelte es sich um die Medizin, um Geografisches Informationsmanagement, um Notfalleinsätze der Feuerwehr und um Agiles Software Engineering. In allen Anwendungsdomänen bestand ein großer Bedarf zur Koordination kollaborativer Aktivitäten und zur Integration verschiedenartigster elektronischer Wissensquellen, dessen Wissen während der Ausführung der Aktivitäten einbezogen werden sollte. Außerdem zeichneten sich die Geschäftsprozesse selbst durch ein hohes Maß an Flexibilität aus, d.h. sie müssen zwar in einer ähnlichen Art und Weise immer wieder ausgeführt werden, jedoch gibt es bei jeder Ausführung in Abhängigkeit der aktuellen Situation gewisse Variationen. Da die Schnittmenge benötigter Funktionalitäten als sehr hoch eingeschätzt wurde, ist schließlich die Entscheidung zugunsten eines generischen Frameworks gefallen, an dessen Entwicklung sich alle Mitarbeiter beteiligt haben. Dieses Framework wird seit dem Jahr 2005 unter dem Namen *Collaborative Agent-based Knowledge Engine (CAKE)* [FMS05, FMMS05a, FMMS05b, Max06, SMMB06, BFM+06] entwickelt.

3 Anforderungen aus den Anwendungsdomänen

In diesem Abschnitt werden die vier Anwendungsdomänen beschrieben, die im Rahmen der Konzeption von CAKE zu Beginn des Jahres 2005 berücksichtigt wurden. Hierbei handelt es sich neben der Medizin um Geografisches Informationsmanagement, um Notfalleinsätze der Feuerwehr und um Agiles Software Engineering. In allen Anwendungsdomänen sind die in diesen Domänen arbeitenden Personen tagtäglich mit einer ganzen Reihe an Problemen konfrontiert, die die Kommunikation, die Koordination und/oder die Kooperation betreffen. Die Aufgabe bestand daher in allen Anwendungsdomänen in der Entwicklung eines Systems, das die Kommunikation, die Koordination und die Kooperation der in diesen Domänen arbeitenden Personen verbessern kann.

Die folgenden vier Abschnitte beschreiben die vier Anwendungsdomänen und fokussieren dabei auf ausgewählte Probleme vor der Einführung eines auf CAKE-basierenden Unterstützungssystems. In Abschnitt 3.5 werden schließlich Anforderungen an ein Framwork formuliert, das als Grundlage zur Entwicklung von Unterstützungssystemen in allen zuvor beschriebenen Domänen diesen kann.

3.1 Medizin

Eine wichtige Frage, mit der sich Mediziner tagtäglich auseinandersetzen müssen ist die Frage, ob ihre Patienten gemäß verfügbarer Leitlinien und klinischer Pfade behandelt werden oder ob sie besser behandelt werden könnten. Diese Frage kann aktuell fast nur von einem Menschen beantwortet werden, der die textuellen Beschreibungen der Leitlinien kennt und (papierbasierte) Handakten sowie elektronisch verfügbare Informationen eines Patienten analysiert und auswertet. Da dieses Vorgehen äußerst zeitintensiv ist, wäre eine IT-Unterstützung hilfreich. Damit ein IT-System individuelle Patienten-Behandlungsverläufe mit klinischen Pfaden vergleichen kann, ist als Voraussetzung sowohl eine elektronische Repräsentation von Patienten-Behandlungsverläufen als auch eine elektronische Repräsentation klinischer Pfade notwendig. Die automatische Ableitung individueller Patienten-Behandlungsverläufe und die Akquisition elektronischer Repräsentationen von klinischen Pfaden für konkrete Krankheitsbilder stellen die beiden Hauptprobleme bei der Entwicklung einer solchen Lösung dar.

Ableitung von Patienten-Behandlungsverläufen

In heutigen Krankenhäusern wird zur patientenbezogenen Dokumentation meist ein hybrider Ansatz aus papierbasierter und elektronischer Akte eingesetzt. Informationen, die nur papierbasiert vorliegen, sind von einem IT-System nicht verarbeitbar, so dass diese Informationen bei der automatischen Ablei-

tung eines Patienten-Behandlungsverlaufs in jedem Fall „verloren“ gehen. Aber auch die elektronisch verfügbaren Informationen können ohne weitergehende Analysen oft nicht direkt genutzt werden: die meisten Krankenhäuser verwenden eine Vielzahl an IT-Systemen, beispielsweise für administrative Aufgaben, zur Speicherung von Befunden oder zur Dokumentation von Krankheitsverläufen. Jedes dieser Systeme beinhaltet Informationen, die zur Ableitung eines Patienten-Behandlungsverlaufs wichtig sind, aber keines dieser Systeme enthält alle relevanten Informationen. Zudem sind die Systeme oft sehr schlecht integriert und enthalten häufig widersprüchliche Informationen.

Akquisition Klinischer Pfade

Leitlinien und klinische Pfade sind üblicherweise wenig formalisierte Texte. Damit sie zum Vergleich mit einem Patienten-Behandlungsverlauf überhaupt herangezogen werden können, müssen sie formalisiert werden. Die Modellierung von Hand ist dabei sehr zeitaufwändig und sowohl für Mediziner, als auch für Informatiker schwer durchzuführen: Mediziner haben häufig Probleme bei der Formalisierung von Prozessen und Informatikern fehlt meist das nötige medizinische Wissen. Denkbar ist daher, eine solche Formalisierung nicht von Grund auf aufzubauen, sondern das in individuellen Patienten-Behandlungsverläufen implizit beinhaltete Prozesswissen auszunutzen: So könnten mehrere individuelle Patienten-Behandlungsverläufe von Patienten mit gleichem Krankheitsbild generalisiert werden, um zu einer initialen Version eines klinischen Pfades zu gelangen. Diese könnte als Basis dienen, die von Medizinern durch weitere Prozessschritte verfeinert werden könnte.

3.2 Geografisches Informationsmanagement

GIS

Die Firma rjm business solutions GmbH [rjm] erstellt im Auftrag einer hessischen Landesbehörde seit 2004 im Rahmen des eGovernment-Projekts *DenkXweb* ein *Geographisches Informationssystem (GIS)* zum Nachweis von denkmalschutzbezogenen Fachinformationen auf Gebäude- und Flurstücksebene für das gesamte Bundesland Hessen. Ziel des Projekts ist es, die Fachinformationen grafisch und textuell so aufzubereiten, dass diese Planungsträgern wie Architekten und Hauseigentümern über ein Internet-Portal[2] bereitgestellt werden können: Für ein beliebiges Grundstück in Hessen wird es ermöglicht, einen Liegenschaftsplan im Maßstabsbereich 1:500-1:2500 abzurufen, sowie eine Begründung, warum eine spezifische Auszeichnung der dort dargestellten Gebäude oder Gesamtanlagen notwendig ist. Bei der Projektdurchführung werden die Mitarbeiter der Firma mit zahlreichen Problemen konfrontiert, von denen in diesem Abschnitt lediglich die gravierendsten knapp vorge-

[2] http://www.denkmalpflege-hessen.de/denkxweb.

stellt werden. Genauere Informationen sind verschiedenen Veröffentlichungen [BOP+05, SMMB06, SM06, BFM+06] zu entnehmen.

Liegenschaftskataster
ALK
ALB
Fachinformationen

Die hauptsächliche Datengrundlage des Internet-Portals bilden eine digitale Version des *Liegenschaftskatasters* (das aus der *Automatisierten Liegenschaftskarte (ALK)* und dem *Automatisierten Liegenschaftsbuch (ALB)* besteht), gedruckte Karten mit handschriftlichen Markierungen denkmalgeschützter Objekte sowie digitale *Fachinformationen*. In allen Datenquellen dient die Flurstücksnummer als Primärschlüssel zur Zuordnung von CAD-Daten sowie Meta- und Fachinformationen. Ausgehend vom Liegenschaftskataster wird schrittweise das gesamte Gebiet des Bundeslandes Hessen mit Fachinformationen ausgezeichnet. Die Gesamtfläche wird anhand der üblichen Aufteilung in Kreisen (41) und Gemeinden (426) untergliedert, innerhalb der Gemeinden werden dann spezifische Ortslagen wie Altstadtkerne oder Villengebiete betrachtet. Innerhalb jeder Ortslage werden die Fachinformationen schließlich für einzelne Flurstücke und Gebäude auf CAD-Layern eingetragen. Dazu werden die gedruckten Karten mit den handschriftlichen Markierungen sowie die Fachinformationen selbst berücksichtigt. Dieser Erfassungsprozess läuft zwar relativ standardisiert ab, jedoch kann er nicht immer vollständig wie geplant ausgeführt werden. Die vier gravierendesten Probleme sind im Folgenden knapp zusammengefasst.

Priorisierung

Es ist häufig notwendig, eine festgelegte Priorisierung der Bearbeitung von Kreisen und Gemeinden abzuändern. Dies führt zu einem „Einfrieren“ aktueller Bearbeitungen, die oft erst nach mehreren Monaten wieder aufgenommen wird.

Inkonsistenzen

Inkonsistenzen zwischen den ALK-Daten, den ALB-Daten, den Karten mit den handschriftlichen Markierungen sowie den Fachinformationen sind sehr häufig. Typisch ist das Scheitern der Zusammenführung von ALK- und ALB-Daten wegen fehlender oder widersprüchlicher Beschreibungen oder wegen unterschiedlicher Granularität. Diese Probleme können erst erkannt werden, wenn die Bearbeitung einer konkreten Ortslage stattfindet und sie ziehen eine Rückfrage beim zuständigen Amt nach sich. Eine anderes typisches Problem ist das Fehlschlagen der Übertragung der denkmalschutzbezogenen Daten wegen Unterschieden in der ALK und der gedruckten Karte. In einem solchen Fall müssen Vergleichsinformationen zusammengestellt und in einer Anfrage an die entsprechende Fachbehörde übermittelt werden. Da die Bearbeitung auf regional eng begrenzten Flächen stattfindet, ergeben sich pro Gemeinde leicht mehrere hundert Bearbeitungsstellen. Zufriedenstellende Antworten der angefragten Behörde lassen oft bis zu sechs Monate auf sich warten, so dass die Freigabe einer ganzen Gemeinde oder gar eines ganzen Kreises dadurch verzögert werden kann. Außerdem wird durch diese oft sehr langen Laufzeiten die Zuordnung von Ergebnissen aus Rückfragen erschwert.

Dokumentation

Zur internen Projektorganisation und zur Dokumentation des Projekts und

insbesondere auftretender Fehler werden eine Reihe von Programmen und Datenbanken verwendet, u.a. ein Zeiterfassungs-Programm, ein Wiki-System und verschiedene Excel-Dateien[3]. Diese heterogene Datenhaltung macht die Bestimmung des aktuellen Bearbeitungszustandes äußerst schwierig. Ein schneller Überblick zu jeder Zeit wäre aber für die Projektverantwortlichen sehr wichtig, um Verzögerungen frühzeitig zu erkennen und ggf. gegensteuern zu können. Außerdem sollten auch Anfragen der Behörden bzgl. der Fertigstellung einer Gemeinde oder eines Kreises möglichst zügig und ohne großen Aufwand beantwortet werden können.

Software-Fehler

Bei Software-Fehlern, die in den CAD- und Web-Komponenten auftreten, müssen evtl. weitere Prüfschleifen eingefügt werden, um eine korrekte Präsentation im Internet sicherzustellen. Korrekte Informationen sind sehr wichtig, da die im Internet-Portal angebotenen Fachinformationen Gesetzescharakter haben.

3.3 Notfalleinsätze der Feuerwehr

Rettungskräfte leisten tagtäglich großartige Arbeit bei der Bekämpfung von Bränden, dem Absichern von Unfallorten oder dem Transport von Verletzten in Krankenhäuser und Kliniken. Allen Notfalleinsätzen ist gemein, dass sie plötzlich und unangekündigt auftreten, dass sie unverzüglich nach Meldung des Notfalls beginnen und zügig abgearbeitet werden müssen, dass meist viele Menschen an unterschiedlichen Orten für den Erfolg eines Einsatzes verantwortlich sind und dass diese Menschen eine große Menge an Erfahrung und Wissen zur Erledigung ihrer Aufgaben benötigen.

AMIRA

Ziel des durch die europäische Union geförderten Projektes *AMIRA (Advanced Multi-modal Intelligence for Remote Assistance)*[4] war es, eine IT-Lösung zur Unterstützung zeit- und geschäftskritischer Prozesse wie Notfalleinsäte zu entwickeln [FMS05, FMMS05a, Max06, BFM+06, FBT+07]. Dabei sollten die kollaborativ arbeitenden Menschen zu jeder Zeit (z.T. unaufgefordert) Wissen zur Verfügung gestellt bekommen, das sie in ihren jeweils aktuellen Situationen benötigen um ihre Aufgaben möglichst optimal zu erfüllen. Das in einer konkreten Situation erforderliche Wissen befindet sich üblicherweise in einer ganzen Reihe an elektronischen Wissensquellen wie Datenbanken, Adressbüchern, oder Sammlungen unstrukturierter Daten, in papierbasierten Wissensquellen wie al-

[3] Excel ist ein Tabellenkalkulationsprogramm von Microsoft.

[4] Das AMIRA Projekt wurde von Juli 2004 bis Juni 2006 unter der Fördernummer IST-2003-511740 gefördert. Projektpartner waren Kaidara Software, Fast DataSearch, DaimlerChrysler RIC, UniversitätTrier, Fire Service College, West Midlands Fire Service, and Avon Fire & Rescue.

ten Einsatzprotokollen, Straßenkarten oder Handbüchern und „in den Köpfen“ der Einsatzkräfte. Als besonders problematisch bei der Entwicklung von Unterstützungssystemen erweisen sich neben der Verteiltheit und den z.T. eingeschränkten Zugriffsmöglichkeiten auf eine Wissensquelle auch die unterschiedlichen Abstraktionsgrade und die unterschiedlichen Strukturen der Wissensquellen. Das explizite Abspeichern von erfolgsversprechenden Suchstrategien und Vorgehensweisen zur Mischung von Ergebnissen aus unterschiedlichen Wissensquellen sollte hier eingesetzt werden. Eine weitere Anforderung an das AMIRA-System war außerdem, dass es sprachbasiert gesteuert werden kann. Dies ist sehr wichtig, da beispielsweise Feuerwehrleute während der Brandbekämpfung ihre Uniformen tragen und kein Laptop oder PDA zur Interaktion mit dem System bedienen können. Gefordert war daher eine Sprachdialog-Komponente als Schnittstelle zum System, damit Feuerwehrleute über Headsets untereinander und mit der Einsatzleitung kommunizieren können.

3.4 Agiles Software Engineering

Im Software Engineering stellen „traditionelle“ Workflow Systeme Standard-Werkzeuge zur Koordination und Kooperation von Projektmitarbeitern dar. Sie übernehmen die Strukturierung von Aktivitäten wie dem Bereitstellen von Projektplänen und der Generierung von ToDo-Listen, verwenden zuvor modellierte Task Beschreibungen häufig vorkommender Aktivitäten effizient wieder, sorgen für eine möglichst gleich verteilte Last der einzelnen Projektmitarbeiter und legen eine priorisierte Ausführungsreihenfolge der einzelnen Aktivitäten fest. Problematisch ist bei dieser Art der Unterstützung jedoch die mangelnde Flexibilität herkömmlicher Workflow Systeme: Software Entwicklungsprozesse lassen sich oft nicht im Voraus vollständig planen, Pläne müssen häufig geändert werden und „neue“ Aktivitäten, die bisher niemals vorkamen, sind an der Tagesordnung. Die notwendigen Entscheidungen zur Änderung von Plänen können nur unter Berücksichtigung unterschiedlicher Wissensquellen (elektronische Wissensquellen und Projektmitarbeiter) gefällt werden. All diese Probleme sind insbesondere bei Agilen Entwicklungsmethoden [Boe02] gravierend.

Ziel des Projektes AgileSE war es, eine IT-Lösung zur Unterstützung Agiler Software-Entwicklungsprozesse zu entwickeln [FSB05, BFM+06]. Die Lösung sollte auf die Kollaboration der Projektmitarbeiter (*menschliche Agenten*) unter Berücksichtigung *elektronischer Agenten* wie Bug-Tracking Systeme und automatisierte Test-Programme fokussieren. Software-Entwicklungsprozesse sollten im Voraus zumindest grob modelliert werden können, jedoch sollten sie während ihrer Ausführung jederzeit abgeändert oder verfeinert werden können.

Das AgileSE-System sollte außerdem eine Möglichkeit bieten, so genannte *Normen* [LSDN01], die in vielen Unternehmen definiert sind um die gewünschte Arbeitsweise der Mitarbeiter zu beschreiben, zu modellieren und bei der Unterstützung der Kollaboration zu verwenden. Während der Unterstützung eines Software-Entwicklungsprozesses sollte das System dafür Sorge tragen, dass jede Aktivität möglichst vom jeweils kompetentensten Agenten ausgeführt wird: kann eine Aktivität von einem elektronischen Agenten ausgeführt werden, so wird immer der kompetenteste gewählt; bei menschlichen Agenten wird ebenfalls der kompetenteste gewählt, sofern dieser nicht überlastet ist. Aufgabe des AgileSE-Systems ist also auch die Optimierung der ToDo-Listen menschlicher Agenten, so dass jeder Agent entsprechend seiner Fähigkeiten optimal eingesetzt wird, ohne jedoch überlastet oder unterfordert zu sein.

3.5 Anforderungen an CAKE

In einer ausführlichen Studie [CAK04] wurden die Probleme und Herausforderungen der in den letzten vier Abschnitten beschriebenen Anwendungsdomänen analysiert. Daraus konnten zunächst applikationsspezifische Anforderungen abgeleitet werden, die sich jedoch gut zu allgemeinen Anforderungen generalisieren ließen. Diese Anforderungen lassen sich in vier Gruppen unterteilen [CAK04, BFM+06]: Anforderungen bezüglich der *Integration von Informationen*, Anforderungen bezüglich der *Integration von Personen*, Anforderungen bezüglich der *Integration von (Geschäfts-)Prozessen* sowie Anforderungen an möglichst vielseitige *Suchmöglichkeiten*. Diese Anforderungen wurden bei der Entwicklung von CAKE berücksichtigt und sind im Folgenden dargestellt.

Integration von Informationen

In allen Anwendungsdomänen ist das zur Lösung einer Kollaborationsaufgabe notwendige Wissen über heterogene Wissensquellen verteilt. Es gibt die unterschiedlichsten Formen elektronischer Wissensquellen wie beispielsweise Datenbanken, Kalender, Adressbücher, Projektmanagement-Tools, oder Dateien in unterschiedlichen Dateiformaten. Häufig liegt das zur Kollaboration benötigte Wissen auch papierbasiert, z.B. in Form von Büchern, Handakten oder Katasterplänen, oder gar nur implizit in den Köpfen der Kollaborationspartner vor.

Ziel des CAKE Frameworks ist daher die Bereitstellung einer Möglichkeit, alle denkbaren Wissensquellen anzubinden. Insbesondere soll die Lösung so flexibel gehalten werden, dass nicht nur die in den vier betrachteten Anwendungsdomänen vorkommenden Wissensquellen angebunden können, sondern Wissensquellen im Allgemeinen. Nur so kann gewährleistet werden, dass CAKE zukünftig auch in völlig anderen Domänen eingesetzt werden kann. Ein auf

CAKE basierendes System soll nahtlos in die bestehende IT-Infrastruktur eingebunden werden können. Dies erhöht gerade in der Startphase eines Projektes die Akzeptanz des Systems bei den durch das System zu unterstützenden Personen, da sie ihre „alten" Systeme weiterverwenden können. Dies bedeutet, dass insbesondere die Struktur der elektronischen Wissensquellen unverändert bleiben sollte, so dass sie nach wie vor mit bestehenden Anwendungen erzeugt, gewartet und gepflegt werden können. „Menschlichen" Wissensquellen muss eine möglichst einfache Möglichkeit angeboten werden ihr Wissen zu teilen. Lediglich papierbasierte Wissensquellen müssen zu Projektstart in geeigneter Weise digitalisiert werden und es sollte immer eine Lösung erarbeitet werden, diese bei längerem Einsatz des Systems zu ersetzen.

Zusammenfassend kann man diese Anforderung auch als das Schaffen von Interoperabilität zwischen existierenden Anwendungen, Funktionen und Diesnten bezeichnen.

Integration von Personen

In allen Anwendungsdomänen arbeiten Menschen kollaborativ zusammen: Ärzte und Pflegepersonal behandeln einen Patienten, Einsatzkräfte löschen ein Feuer, und Software-Entwickler entwickeln gemeinsam ein neues Software Produkt. Typischerweise hat jeder menschliche Benutzer im Verlauf eines Kollaborationsprozesses mal selbst Informationsbedürfnisse und mal kann er die Informationsbedürfnisse anderer Kollaborationspartner befriedigen. CAKE muss daher menschliche Benutzer nahtlos in das System integrieren können und ihnen insbesondere Interaktionsmöglichkeiten anbieten, die beiden Rollen gerecht werden.

Integration von Prozessen

Alle vier Anwendungsdomänen sind durch komplexe, wissensintensive Geschäftsprozesse charakterisiert, die zwar in ähnlicher Art und Weise immer wieder ablaufen, die jedoch durch unterschiedliche Einflussfaktoren bedingt jeweils (leicht) variieren können: Zwar werden bei einem Schlaganfallpatienten fast immer die Anamnese, der neurologische Befund, eine Computertomographie und Laborwerte erhoben, in Abhängigkeit der gefundenen Ergebnisse und des Zustands des Patienten werden weitere Maßnahmen jedoch individuell geplant und können somit bei zwei verschiedenen Schlaganfallpatienten stark variieren. Auch die Erfassung einer Gemeinde im GIS-Szenario läuft prinzipiell immer nach dem gleichen Schema ab, jedoch ergeben sich bedingt durch auftretende Fehler sehr unterschiedliche Ausprägungen dieser Prozesse. Diese beiden Beispiele zeigen außerdem, dass viele Prozesse bei ihrem Start nicht vollständig geplant werden können, sondern „Platzhalter" benötigen, die erst im Verlauf des Prozesses konkretisiert werden können.

Eine Anforderung an CAKE ist die Bereitstellung einer Workflow Technologie, die diese Art flexibler Prozesse modellieren und ausführen kann. So sollen in einem Prozessmodell zwar typische Abläufe formal beschrieben werden

können, bei ihrer Ausführung soll aber das Hinzufügen, das Weglassen und das Austauschen einzelner Prozessschritte oder die Konkretisierung eines Platzhalters, z.B. durch das Starten eines spezifischen Unter-Prozesses, möglich sein. Entscheidungen über das Abändern eines Workflows sollen vom System automatisch getroffen werden können ohne einen menschlichen Benutzer zu involvieren. Beispielsweise soll das System im Falle dass eine im Prozessmodell vorgesehene Wissensquelle aktuell nicht verfügbar ist automatisch eine andere Wissensquelle befragen können. Um in diesem Fall entscheiden zu können, welche Wissensquelle alternativ befragt werden kann muss CAKE Zugriff auf entsprechendes Meta-Wissen über die Wissensquellen erhalten.

Die Workflow Technologie soll es außerdem ermöglichen, Kollaborationsstrategien in Form von „best practices“ abzulegen. Eine solche Kollaborationsstrategie könnte eine Formalisierung einer Informationsbeschaffungsaufgabe sein und beschreiben, welche Wissensquellen in welcher Reihenfolge angefragt werden sollten um ein bestimmtes Informationsbedürfnis zu befriedigen. Sie sind insbesondere für neue Mitarbeiter bei der Ausführung ihrer Aufgaben wertvoll.

Suchmöglichkeiten

Um die Anforderungen der Integration von Personen und Prozessen erfüllen zu können ist der Bedarf möglichst vielseitiger Suchstrategien offensichtlich: Wissensquellen, Prozessmodelle und Kollaborationsstrategien müssen situationsabhängig gesucht werden können. Nur dann ist es dem System möglich, in jeder Situation den geeignetsten Geschäftsprozess oder die geeignetste Kollaborationsstrategie auszuführen oder den geeignetsten, aktuell verfügbaren Kollaborationspartner zu finden. Um dies zu ermöglichen, müssen Wissensquellen und Prozessmodelle mit Meta-Wissen annotiert werden, das ihre Kompetenzen beschreibt. Dieses Meta-Wissen sollte mit Hilfe einer unscharfen Suchtechnologie durchsucht werden, da nicht in jeder Situation eine 100%ig geeignete Wissensquelle oder Kollaborationsstrategie bzw. 100%ig geeignetes Prozessmodell ein gefunden werden kann.

4 Das Konzept von CAKE

CAKE

Die *Collaborative Agent-Based Knowledge Engine (CAKE)*[5] ist ein domänenunabhängiges Framework, das zur Entwicklung von Unterstützungssystemen in verschiedenen Domänen eingesetzt werden kann. Ziel der Entwicklung war es, die Kommunikation, Koordination und Kooperation menschlicher Akteure zu realisierten sowie elektronische Wissensquellen nahtlos in einen solchen Kollaborationsprozess zu integrieren. Anders ausgedrückt war es das Ziel, eine Integrationsplattform zu schaffen, die Personen, Informationen und Prozesse möglichst einheitlich integrieren kann. Damit ist die Zielsetzung ähnlich zu der kommerzieller Entwicklungen, die heute unter dem Schlagwort *Service-Orientierte Architekturen (SOA)* geführt werden: hier sind beispielsweise SAP NetWeaver [SAP-NW] der Firma SAP oder IBM WebSphere [IBM-WS] der Firma IBM zu nennen. Zur Erreichung dieses Ziels kombiniert CAKE drei Schlüsseltechnologien, nämlich *Agenten Technologie*, *Workflow Technologie* und *CBR Technologie*.

SOA

Agenten Technologie

Die *Agenten Technologie* wird zur Integration heterogener, elektronischer Wissensquellen sowie zur Integration von Personen verwendet. Dabei stellt das Agentenframework Wrapper zur Verfügung, die aus Systemsicht vom konkreten Typ des Agenten (menschlicher Agent oder Computersystem) abstrahieren, so dass beliebige Agenten miteinander kollaborieren können. Insbesondere müssen die dem System bekannten Agenten nicht zu jeder Zeit tatsächlich verfügbar sein, sondern das System verhält sich robust gegenüber Änderungen am Agentenpool und unterstützt zu jedem Zeitpunkt die möglichst optimale Kollaboration zwischen den aktuell verfügbaren Agenten.

Workflow Technologie

Mit Hilfe der *Workflow Technologie* können sowohl Geschäftsprozesse als auch Kollaborationsstrategien verschiedener Agenten modelliert und ausgeführt werden. Die Prozessmodelle sind äußerst flexibel gehalten, so dass die ausgeführten Prozesse in konkreten Situationen flexibel geändert und angepasst werden können. Außerdem können zur Lösung einer Aufgabe auch mehrere unterschiedliche Kollaborationsstrategien modelliert werden und im Rahmen einer konkreten Kollaboration wird dann die geeignetste Strategie angewendet.

CBR Technologie

Um dynamische Agentenpools und flexible Workflows zu ermöglichen, sind ausgeklügelte Suchmöglichkeiten unerlässlich. Beim Entwurf von CAKE wurde hier eine fallbasierte Suche (engl. *Case-Based Reasoning, CBR*) [Ber02, BAB+03] als erfolgsversprechendsten angesehen. Daher bildet die *CBR Technologie* die dritte Schlüsseltechnologie. Sie wird einerseits verwendet, um auf Workflow und Agenten-Charakterisierungen zu suchen und dadurch den in einer konkreten Situation passendsten Workflow bzw. den geeignetsten Kollaborationspartner

[5] CAKE-WWW-Seite: http://cake.wi2.uni-trier.de

zu finden, andererseits kann sie auch im Sinne einer elektronischen Wissensquelle genutzt werden.

CAKE Datenrepräsentation

Alle drei Schlüsseltechnologien verwenden die *CAKE Datenrepräsentation*. Sie basiert auf einem objekt-orientierten [Boo06, Mey97] Ansatz und verwendet Datenklassen und zugehörige Datenobjekte: das *CAKE Datenmodell* stellt domänen-unabhängige Datenklassen (*Systemklassen*) zur Verfügung, die unter Zuhilfenahme der Abstraktionskonzepte *Aggregation* und *Spezialisierung* zur Spezifikation domänenabhängiger Datenklassen (*Benutzerklassen*) verwendet werden können. Diese Benutzerklassen bilden dann gemeinsam mit den Systemklassen ein domänenabhängiges Datenmodell. Von allen Klassen dieses Datenmodells können unter Verwendung der *Instanziierung* Instanzen gebildet werden, die als *CAKE Datenobjekte* bezeichnet werden.

CAKE Architektur

Einen Überblick über die *CAKE Architektur* und insbesondere die Interaktion der drei Schlüsseltechnologien liefert Abbildung 1. Das Zugrunde liegende Datenmodell und die drei Schlüsseltechnologien werden in den folgenden vier Abschnitten intensiv erläutert. Abschnitt 4.5 erläutert schließlich die von CAKE zur Verfügung gestellte Modellierungsumgebung namens CAKE Editor. Weiterführende Informationen zu CAKE können auch verschiedenen Veröffentlichungen des Lehrstuhls für Wirtschaftsinformatik II [CAK05, FMS05, FMMS05a, FMMS05b, Max06, SMMB06, BFM+06] entnommen werden.

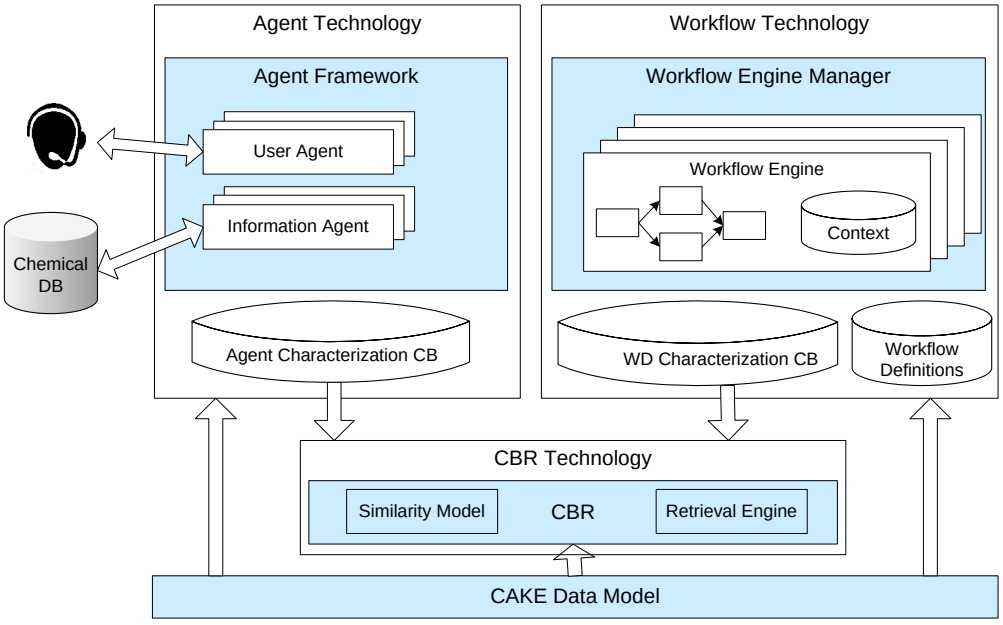

Abbildung 1: CAKE Architektur

4.1 CAKE Datenrepräsentation

Alle Komponenten innerhalb von CAKE nutzen eine einheitliche Repräsentation ihrer Daten. Durch diese einheitliche Repräsentation wird die Flexibilität gewährleistet, die in allen Anwendungsdomänen notwendig ist. Als Repräsentationsform wird ein objekt-orientierter Ansatz [Boo06, Mey97] verwendet, der mit Hilfe des *CAKE Datenmodells* und zugehöriger *CAKE Datenobjekte* realisiert wird.

CAKE Datenmodell

CAKE Datenmodell

Das *CAKE Datenmodell* ist ein objekt-orientiertes Datenmodell, das von den Abstraktionskonzepten *Aggregation* und *Spezialisierung* zur Spezifikation von Datenklassen Gebrauch macht. Die gemeinsame Oberklasse aller Datenklassen ist die abstakte Datenklasse *Data*. „Abstrakt" bedeutet in diesem Zusammenhang, dass von ihr keine Instanzen, also keine CAKE Datenobjekte, gebildet werden können, sondern sie lediglich als Oberklasse anderer Systemklassen dient. Von ihr sind atomare Datenklassen wie Boolean, Integer und Double sowie Datenklassen zur Repräsentation von Daten und Uhrzeiten abgeleitet. Ebenso sind komplexe Datentypen wie Intervalle, Aggregate oder Mengen Unterklassen von Data. Abbildung 2 zeigt alle Datenklassen, die von CAKE bereits zur Verfügung gestellt werden. Sie sind domänen-unabhängig und werden als *Systemklassen* bezeichnet. Abstrakte Datenklassen sind in der Grafik kursiv geschrieben; instanziierbare Datenklassen sind „normal" gesetzt. Unter Verwendung der Spezialisierung können von diesen Systemklassen domänen-abhängige Unterklassen gebildet werden, die als *Benutzerklassen* bezeichnet werden. Selbstverständlich können die Benutzerklassen ihrerseits auch weiter spezialisiert werden. Gemeinsam bilden die Systemklassen und die Benutzerklassen das domänenabhängige Datenmodell.

Systemklassen

Benutzerklassen

Aggregate

Die Datenklasse *Aggregate* ist eine abstrakte Datenklasse, die als Oberklasse von Benutzerklassen dienen kann. Sie verwendet das Abstraktionskonzept der Aggregation um mehrere unterschiedliche Datenklassen in einer neuen Datenklasse zu kombinieren. Für jede eingebettete Datenklasse gibt es ein Attribut, das über seinen Namen und die Datenklasse des eingebetteten Datenobjektes beschrieben ist. Diese Attribute sind also vergleichbar mit Instanzenvariablen objektorientierter Programmiersprachen. Da beliebige Datenklassen, also insbesondere auch weitere Unterklassen von Aggregate, eingebettet werden können, ist eine beliebig tiefe Aggregationshierarchie möglich.

Atomic

Auch die Datenklasse *Atomic* ist eine abstrakte Datenklasse. Gemeinsam ist all ihren Unterklassen, dass deren Datenobjekte jeweils einen einzelnen Wert

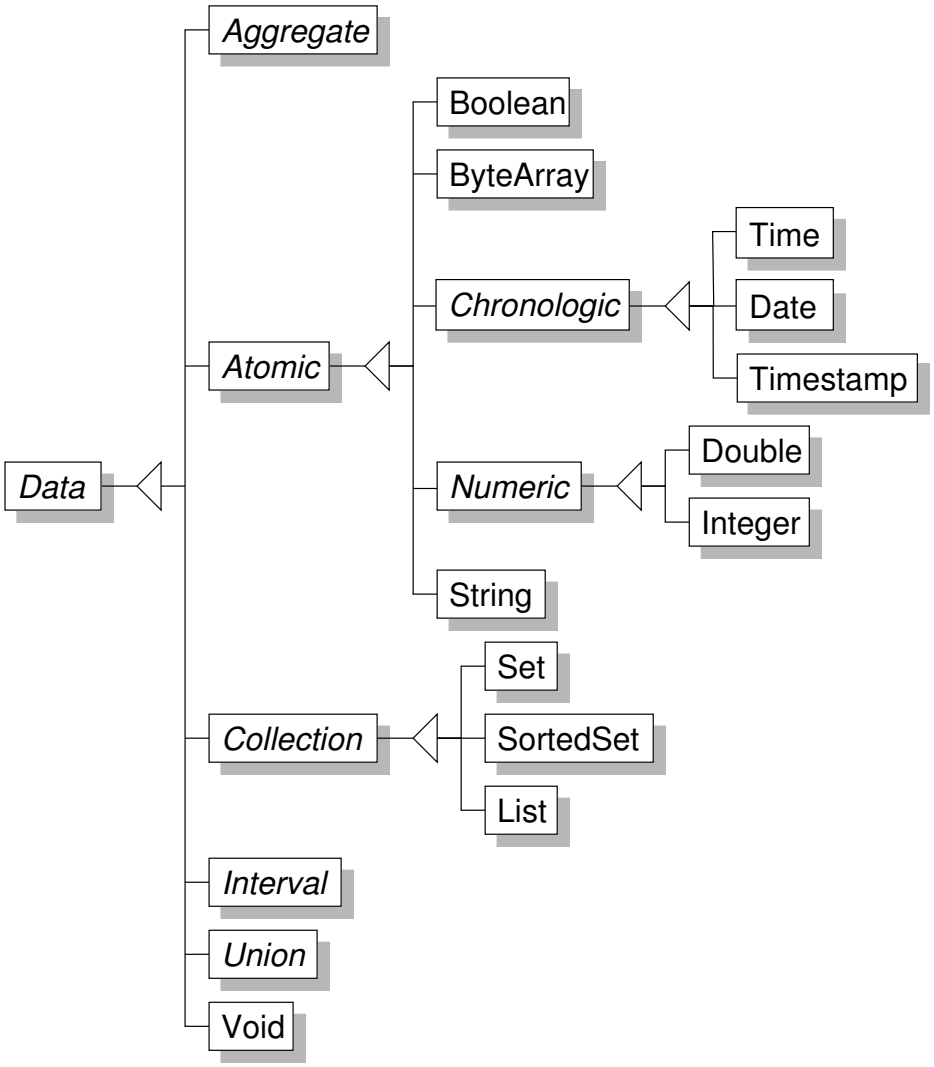

Abbildung 2: Systemklassen des CAKE Datenmodells, in Anlehnung an [Max06]

speichern können. Atomare Datentypen sind aus typisierten Programmiersprachen bekannt und die Bezeichnungen im CAKE Datenmodell halten sich an die üblichen Namenskonventionen: „Boolean“ bezeichnet eine Datenklasse, deren Datenobjekte zwei Werte (wahr/true bzw. falsch/false) annehmen können. Mit Hilfe der Datenklasse „Integer“ können ganze Zahlen und mit Hilfe der Datenklasse „Double“ Gleitkommazahlen repräsentiert werden. Diese Datenklassen sind von der gemeinsamen, abstrakten Oberklasse „Numeric“ abgeleitet. Es gibt eine weitere abstrakte Oberklasse namens „Chronologic“, von der die Datenklassen „Time“ zur Repräsentation von Uhrzeiten, „Date“ zur Repräsentation von Daten und „Timestamp“ zur Repräsentation von Zeitstempeln, also einer Kombination aus Datum und Uhrzeit, abgeleitet sind. Die Datenklasse „String“ ist zur Darstellung einer Zeichenkette notwendig und die Datenklasse ByteArray kann genutzt werden, wenn in dem entsprechenden Datenobjekt ein komplexes Objekt wie ein Bild, ein Dokument oder ein Video abgelegt werden soll. Eine Spezialisierung einer atomaren Datenklasse besteht entweder in einer Beschränkung des möglichen Wertebereichs oder in einer Aufzählung aller möglichen Werte.

Beschränkung des Wertebereichs: Man kann beispielsweise davon ausgehen, dass alle im Krankenhaus behandelten Personen zwischen 1900 und 2100 geboren sind, und somit kann man eine Benutzerklasse „Geburtsjahr“ als Unterklasse von „Integer“ modellieren, deren Datenobjekte nur Werte im Intervall [1900..2100] annehmen können. Diese Art der Beschränkung ist nur für total geordnete Datenklassen möglich, d.h. für Unterklassen von Boolean und ByteArray ist dieses Konzept nicht anwendbar.

Aufzählung aller Werte: Mit Hilfe von Aufzählungen kann eine endliche Menge möglicher Werte für die Datenobjekte vorgegeben werden. Beispielsweise könnte man eine Benutzerklasse „InkonsistenzTyp“ anlegen, deren Datenobjekte lediglich Beschreibungen von dem System bekannten Inkonsistenzen, also im Falle von Appear „Duplikat“, „Zeitstempel“, „Fehlende Leistung“, „Lange Zeitspanne“ und „Falsche Reihenfolge“ enthalten können. Die Elemente der Menge können entweder total geordnet oder taxonomisch angeordnet werden.

Collection

Die abstrakte Datenklasse *Collection* kann zur Beschreibung von Mengen, also zur Zusammenfassung mehrerer Datenobjekte gleichen Datentyps, genutzt werden. CAKE unterscheidet drei verschiedene Mengen-Arten, für die jeweils eigene Unterklassen im CAKE Datenmodell enthalten sind: Die Datenklassen „Set“ und „SortedSet“ beschreiben jeweils Mengen, die beliebig viele Objekte des gleichen Datentyps aufnehmen können; verboten sind lediglich Duplikate.

Diese beiden Datenklassen unterscheiden sich dahingehend, dass die Elemente einer „Set“ wahllos angeordnet sind und sie keinem Sortierkriterium unterliegen, während die Elemente einer „SortedSet“ sortiert sind. Die Datenklasse „List“ beschreibt eine Menge, die beliebig viele Elemente des gleichen Datentyps und insbesondere Duplikate aufnehmen kann. Auf die einzelnen Elemente kann mittels eines Index zugegriffen werden. Somit sind Datenobjekte dieser Datenklasse vergleichbar mit aus Programmiersprachen bekannten Arrays. Da man jede beliebige Datenklasse, insbesondere Data, als Datentyp der Elemente bestimmen kann, können beliebige Mengen realisiert werden.

Interval Mit Hilfe der abstrakten Datenklasse *Interval* können Intervalle total geordneter Datentypen modelliert werden. Bei der Spezifikation der Unterklasse muss dazu die Datenklasse des total geordneten Datentyps angegeben werden. Beispielsweise könnte zur Repräsentation des Referenzbereichs einer Laboruntersuchung eine Datenklasse „ReferenzbereichInterval“ als Unterklasse von Interval definiert werden, deren Datenobjekte Intervalle von Gleitkommazahlen (z.B. [0,84-1,45]) darstellen.

Union Die Datenklasse *Union* ist aus konzeptioneller Sicht nicht unbedingt nötig, jedoch kann sie in bestimmten Situationen das domänenabhängige Datenmodell vereinfachen und das zu entwickelnde System sicherer machen. Von einer Union-Klasse werden keine Instanzen gebildet, sondern sie dient lediglich als „Kapselung“ von Datenobjekten unterschiedlichen Datentyps. Beispielsweise ist es denkbar, dass ein Attribut eines Datenobjektes vom Datentyp Aggregate sowohl Integer-Werte als auch Integer-Intervalle aufnehmen können soll oder dass in einer Menge sowohl Integer-Werte als auch Integer-Intervalle abgelegt werden können sollen. In beiden Szenarien wäre natürlich eine prinzipielle Modellierung der Integer-Werte als Intervalle denkbar, d.h. jeder Integer-Wert x könnte als Intervall $[x..x]$ aufgefasst werden. Jedoch macht diese Lösung die Modellierung z.T. unnötig schwerfällig. Im Falle der Menge müsste man außerdem die gemeinsame Oberklasse von Integer und Interval, also „Data“ als Datentyp der Elemente modellieren. Damit wäre es allerdings auch möglich Strings oder Bilder in der Menge abzulegen, was nicht gewünscht ist. Um solche Situationen zu vermeiden, kann eine Unterklasse von Union modelliert werden, die festlegt, von welchen Datenklassen die Werte der Union abgeleitet sein dürfen. Diese Union-Datenklasse kann dann als Datentyp des Attributs oder als Basisklasse einer Menge angegeben werden.

Void Die letzte Systemklasse heißt *Void* und mit ihrer Hilfe können unspezifizierte Werte dargestellt werden. Sie wird vor allem im Rahmen der CBR Technologie verwendet, um vom Benutzer nicht-spezifizierte Attribute der Anfrage zu repräsentieren. Alle Attribute konkreter Datenobjekte vom Datentyp Aggregate, für die kein Wert spezifiziert ist, bekommen intern ein Datenobjekt der Daten-

klasse Void zugewiesen.

Das domänenabhängige Datenmodell besteht aus den oben beschriebenen Systemklassen und allen vom Benutzer definierenden Benutzerklassen. Die Benutzerklassen werden in einem XML-Format namens *CAKE Data Model (CDM)* repräsentiert. Eine detaillierte Beschreibung der als Grammatik dienenden XML Schema Definition ist von Maximini [Max06] beschrieben. Als veranschaulichendes Beispiel ist in Listing 1 ein Ausschnitt aus dem Appear Datenmodell angegeben, nämlich die Repräsentation der Benutzerklasse „LorenzoPatienten-Information“ zur Speicherung der in LORENZO verfügbaren Patientendaten. Zu ihrer Spezifikation werden die Systemklasse „String“ und drei weitere Benutzerklassen benötigt: Die Benutzerklasse „Geschlecht“, die die beiden Werte „männlich“ und „weiblich“ annehmen kann, die Benutzerklasse „Geburtsjahr“, deren Instanzen ganzzahlige Werte zwischen 1900 und 2100 annehmen können, und die Benutzerklasse „FallnummernSet“, deren Datenobjekt eine Menge mit allen Fallnummern des Patienten ist. Alle vier Benutzerklassen sind dem Appear Datenmodell entnommen.

CDM

```
<AtomicClass name="Geschlecht" superClass="String">
  <ValueEnumeration>
    <Value v="männlich"/>
    <Value v="weiblich"/>
  </ValueEnumeration>
</AtomicClass>

<AtomicClass name="Geburtsjahr" superClass="Integer">
  <ValueInterval lowerBound="1900" upperBound="2100"/>
</AtomicClass>

<CollectionClass name="FallnummernSet" superClass="Set">
  <ElementClass name="String"/>
</CollectionClass>

<AggregateClass name="LorenzoPatientenInformation"
  superClass="Aggregate">
  <Attribute name="patientennummer" class="String"/>
  <Attribute name="vorname" class="String"/>
  <Attribute name="nachname" class="String"/>
  <Attribute name="geburtsjahr" class="Geburtsjahr"/>
  <Attribute name="geschlecht" class="Geschlecht"/>
  <Attribute name="krankenhausfälle" class="FallnummernSet"/>
</AggregateClass>
```

Listing 1: Ausgewählte Benutzerklassen des Appear Datenmodells

CAKE Datenobjekte

CAKE Datenobjekt

Mit Ausnahme der Union-Datenklassen können alle CAKE Datenklassen zu *CAKE Datenobjekten* instanziiert werden. Im Umkehrschluss ist von jedem CAKE Datenobjekt seine Datenklasse eindeutig festgelegt und damit auch die erlaubten Attribute, Wertebereiche, Datentypen, etc. dieser Objekte. Die Objekt-Hierarchie ist identisch zur Hierarchie der Datenklassen (siehe Abbildung 2). Als veranschaulichendes Beispiel ist in Listing 2 ein Datenobjekt der Benutzerklasse „LorenzoPatientenInformation“ angegeben. Mehrere CAKE Datenobjekte können in einem *CAKE Datenobjekt-Pool* zusammengefasst werden. Zur persistenten Speicherung eines solchen Pools wurde ein eigenes XML-Format namens *CAKE Data Object Pool (CDOP)* entwickelt. Eine detaillierte Beschreibung der als Grammatik dienenden XML Schema Definition ist von Maximini [Max06] beschrieben.

CAKE Datenobjekt-Pool

CDOP

```
<Agg c="LorenzoPatientenInformation">
    <AA n="patientennummer" v="SAB-01"/>
    <AA n="vorname" v="Peter"/>
    <AA n="nachname" v="Mustermann"/>
    <AA n="geburtsjahr" v="1950"/>
    <AA n="geschlecht" v="männlich"/>
    <OA n="krankenhausfälle">
        <C>
            <A v="SAB-01-01"/>
            <A v="SAB-01-02"/>
        </C>
    </OA>
</Agg>
```

Listing 2: Datenobjekte der Benutzerklasse „LorenzoPatientenInformation“

4.2 CAKE CBR Technologie

CBR

Unter *Fallbasiertem Schließen* (engl. *Case-Based Reasoning, CBR*) [Ber02, BAB+03] versteht man den Prozess des Lösens eines neuen Problems unter Berücksichtigung früherer Lösungen zu ähnlichen Problemen. Die Grundidee lehnt sich dabei an menschliche Problemlösungsstrategien an: Ein Automechaniker, der ein ihm bisher unbekanntes Auto repariert, verwendet CBR, indem er sich an Reparaturen früherer Autos mit ähnlichen Symptomen erinnert und die früheren

Schritte zur Reparatur adaptiert und durchführt. Ein Anwalt, der sich Urteile zu früheren, ähnlich gelagerten Fällen ansieht und somit die Strategie für einen aktuellen Fall erarbeitet, verwendet ebenfalls CBR. Diese einfache Grundidee wurde in den 80er Jahren aufgegriffen, um „intelligente“ Computersysteme zu entwickeln [Sch82, Kol83a, Kol83b]. Mit Hilfe dieser Systeme sollten Computer in die Lage versetzt werden, Probleme zu lösen, auch wenn die Lösungen selbst oder eine Lösungsstrategie nicht explizit programmiert wurden.

Der 4Re-Zyklus

Im Jahr 1994 beschrieben Aamodt und Plaza [AP94] den aus vier Schritten bestehenden *4Re-Zyklus*, der bis heute als typisches Vorgehensmodell für CBR-Applikationen dient. Die englischen Bezeichnungen der Schritte beginnen alle mit den Buchstaben „Re“, nämlich *Retrieve*, *Reuse*, *Revise* und *Retain*. Der 4Re-Zyklus ist in Abbildung 3 dargestellt und die vier Schritte sind im Folgenden knapp erläutert. Für detaillierte Informationen sei an dieser Stelle auf einschlägige Literatur [Ber02, BAB+03] verwiesen.

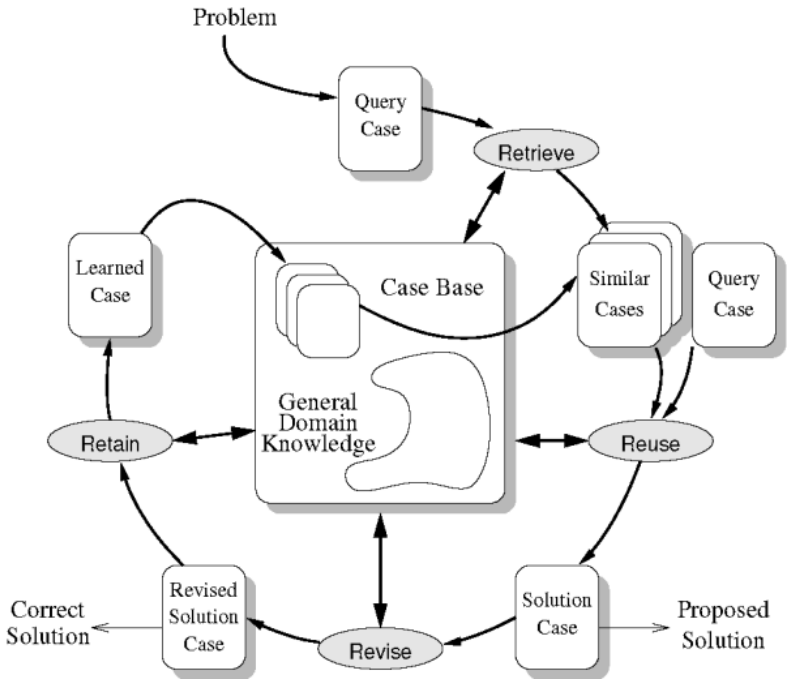

Abbildung 3: Der 4RE-Zyklus, in Anlehnung an [AP94]

Das Wort „Retrieve“ kann im Deutschen mit den Worten „Wiederauffinden“ **Retrieve**

oder „Abfragen“ übersetzt werden. Als Eingabe dieses Schritts dient ein neues Problem, das man als *Query* bezeichnet. In Abhängigkeit von dieser Query werden mehrere *Fälle* aus einer so genannten *Fallbasis (engl. Case Base)* herausgesucht, die zur Lösung des neuen Problems nützlich sein könnten. Dabei besteht ein solcher Fall mindestens aus einer Beschreibung eines Problems und einer Beschreibung seiner Lösung. Um möglichst nützliche Fälle aus der Fallbasis finden zu können muss es möglich sein, das in der Query spezifizierte Problem mit den Problembeschreibungen der Fälle in der Fallbasis zu vergleichen und somit die Fälle der Fallbasis nach ihrer Nützlichkeit für das aktuelle Problem zu sortieren. Dies wird mit Hilfe mathematischer Funktionen realisiert, die als *Ähnlichkeitsmaße* bezeichnet werden. Ähnlichkeitsmaße liefern als Ergebnis immer einen Wert aus dem Intervall [0..1], wobei 0 für völlige Unähnlichkeit und 1 für absolute Übereinstimmung steht. Das Ergebnis wird auch als *Ähnlichkeit* bezeichnet.

Reuse Das Wort „Reuse“ kann im Deutschen mit dem Wort „Wiederverwenden“ übersetzt werden. In diesem Schritt werden die im Retrieve-Schritt gefundenen Lösungen der ähnlichsten Fälle betrachtet, um daraus schließlich eine Lösung für das neue Problem abzuleiten. Die neue Lösung muss also keinesfalls exakt mit einer früheren Lösung übereinstimmen; vielmehr werden die Erfahrungen aus alten Lösungen als Grundlage genommen, um ganz neue Lösungen zu entwickeln.

Revise Das Wort „Revise“ kann im Deutschen mit den Worten „Überprüfen“ oder „Bereinigen“ übersetzt werden. In diesem Schritt wird die im Reuse-Schritt entwickelte Lösung verifiziert. Man wendet sie also auf das neue Problem an (oder simuliert deren Anwendung) und bewertet im Anschluss, ob die Lösung tatsächlich anwendbar ist und ob sie das Problem Zufrieden stellend lösen kann. Falls die Lösung nicht gut anwendbar ist oder falls bei deren Anwendung Verbesserungsmöglichkeiten auffallen, kann die Lösung in diesem Schritt auch noch einmal abgeändert und verbessert werden.

Retain Das Wort Retain kann im Deutschen mit den Worten „Aufbewahren“ oder „Behalten“ übersetzt werden. Wenn eine Lösung im Revise-Schritt als anwendbar und wertvoll eingestuft wurde, so soll sie als Erfahrung für zukünftige Problemlösungen aufbewahrt werden. Dies bedeutet nichts anderes, als dass ein neuer Fall, bestehend aus der Problembeschreibung der Query und der im Revise-Schritt verifizierten Lösung, erzeugt und in der Fallbasis abgelegt wird. Damit ist sie für zukünftige Anfragen auffindbar und kann zur Lösung zukünftiger Probleme herangezogen werden.

Umsetzung des 4RE-Zyklus in CAKE

In CAKE ist der 4Re-Zyklus vollständig implementiert [CAK06, CAK05]. Bei der Implementierung des Zyklus werden verschiedene Arten von Wissen benötigt, beispielsweise im Retrieve-Schritt Wissen darüber, wann ein Fall ähnlich zu einer Query ist oder im Reuse-Schritt Wissen darüber wie eine gefundene Lösung für ein neues Problem angepasst werden kann. Ganz allgemein ist das von einem CBR-System verwendete Wissen meist nicht an einer zentralen Stelle und in einem einheitlichen Repräsentationsformat abgelegt. Vielmehr gibt es je nach Art des Wissens unterschiedliche Repräsentationsformen, die für ihre entsprechende Anwendung besonders geeignet sind und jede einzelne Wissens-Art wird in einem eigenen *Wissenscontainer* abgelegt. Im Jahre 1995 hat Richter [Ric95] vier Wissenscontainer für CBR-Systeme identifiziert, nämlich das *Vokabular*, die *Fallbasis*, das *Ähnlichkeitsmodell* und das *Adaptionswissen*. Diese vier Wissenscontainer werden im Folgenden jeweils zunächst allgemeingültig beschrieben und im Anschluss wird dargelegt, wie ihre Umsetzung in CAKE realisiert wurde.

Vokabular

Das Vokabular wird zur Repräsentation aller in der Anwendungsdomäne des CBR-Systems vorkommenden Entitäten verwendet und ist im Kontext der wissensbasierten Systeme auch unter dem Begriffsbezeichner *Domänenmodell* bekannt. Es dient als Basis der drei übrigen Wissenscontainer.

In CAKE wird das Vokabular durch das CAKE Datenmodell umgesetzt. Durch die Spezifikation von Benutzerklassen können die notwendigen Strukturen definiert werden, mit denen konkretes Domänenwissen repräsentiert werden kann. So dient als Datenklasse eines Falls, also einer wieder verwendbaren Erfahrung, meist eine Unterklasse von Aggregate, deren Attribute die Probleme und deren Lösungen charakterisieren. Auch eine Query ist ein CAKE Datenobjekt dieser Datenklasse.

Fallbasis

Die Fallbasis dient zur Speicherung der bisherigen Erfahrungen des CBR-Systems. Es ist also eine endliche Menge von Fällen. Da Fälle mit Hilfe des Vokabulars repräsentiert sind, benötigt dieser Wissenscontainer das Vokabular.

In CAKE ist die Fallbasis eine Instanz eines CAKE Datenobjekt-Pools und beinhaltet somit eine endliche Menge von CAKE Datenobjekten.

Ähnlichkeits-modell

Um im Retrieve-Schritt die Query mit den Fällen der Fallbasis zu vergleichen, sind *Ähnlichkeitsmaße* vonnöten, die die Nützlichkeit eines Falls für die Lösung eines neuen Problems approximieren können. Ähnlichkeitsmaße sind mathematische Funktionen, die die *Ähnlichkeit* zwischen einer Query und einem Fall, also einen Wert aus dem Intervall [0..1], als Ergebnis zurückliefern können. Aus der Literatur [Ber02] sind verschiedene mathematische Funktionen bekannt, die als Ähnlichkeitsmaße fungieren können, beispielsweise der Ham-

mingabstand, der Simple-Matching-Coeffizient und die Sigmoid-Funktion. Welche dieser Funktionen jedoch in welcher Situation die nützlichsten Ergebnisse liefern kann, hängt entscheidend von der konkreten Anwendung und von der gewählten Fallrepräsentation ab. Daher können Ähnlichkeitsmaße prinzipiell parametrisiert und auf konkrete Anwendungen angepasst werden. Ganz allgemein sind alle Ähnlichkeitsmaße, die einem CBR-System bekannt sind, im Ähnlichkeitsmodell abgelegt. Dieses hängt u.a. vom Vokabular ab, da die Fallrepräsentation berücksichtigt werden muss.

In CAKE werden Ähnlichkeitsmaße also dazu verwendet, zwei CAKE Datenobjekte zu vergleichen und ihre Ähnlichkeit als Ergebnis zurückzuliefern. Als domänenunabhängiges Framework wird in CAKE ein Ansatz verfolgt, der den Benutzer bei der Definition applikationsspezifischer Ähnlichkeitsmaße unterstützt: Im *CAKE Ähnlichkeitsmodell* stellt CAKE bereits die 24 am häufigsten verwendeten Ähnlichkeitsmaße zur Verfügung. Diese sind teilweise spezialisiert für bestimmte Datenklassen, also beispielsweise für atomare Datenklassen wie String oder Numeric, aber auch für komplexe Datenklassen wie Aggregate und Collection-Klassen. Diese Ähnlichkeitsmaße können einfach und komfortabel für eine konkrete Datenklasse parametrisiert und angepasst werden. Dadurch ist es dann möglich, spezialisierte Ähnlichkeitsmaße für applikationsspezifische Benutzerklassen zu spezifizieren. Eine detaillierte Erläuterung der 24 implementierten Ähnlichkeitsmaße und wie diese für konkrete Benutzerklassen parametrisiert werden können ist in der Dokumentation des CAKE-Systems [CAK06] angegeben. Das Ähnlichkeitsmodell wird in einem XML-Format na-

CDSM

mens *CAKE Data Similarity Model (CDSM)* repräsentiert. Eine detaillierte Beschreibung der als Grammatik dienenden XML Schema Definition ist von Maximini [Max06] beschrieben. Als veranschaulichendes Beispiel ist in Listing 3 die Angabe eines Ähnlichkeitsmaßes für die Benutzerklasse „LorenzoPatientenInformation“ des Appear Datenmodells angegeben. Zum Vergleich zweier CAKE Datenobjekte der Benutzerklasse „LorenzoPatientenInformation“ sind die Attribute „patientennummer", „vorname“ und „nachname“ irrelevant, da sie nichts über die Ähnlichkeit zweier Patienten aussagen. Auch die reine Anzahl der Krankenhausaufenthalte spielt keine Rolle; wichtiger wäre eher, ob die Patienten wegen der gleichen Krankheit behandelt wurden, was in dieser Benutzerklasse jedoch nicht abgelegt ist. Daher werden nur die Attribute „geburtsjahr“ und „geschlecht“ in die Ähnlichkeitsbetrachtung einbezogen. Da das Alter eines Patienten eine wichtigere Rolle spielt als das Geschlecht, wurde dem Attribut „geburtsjahr“ ein höheres Gewicht zugewiesen als dem Attribut „geschlecht".

```
<StringEqual class="Geschlecht" caseSensitive="true"
             name="default" default="true"/>
```

```
<NumericLinear class="Geburtsjahr"
               name="default" default="true"/>

<AggregateAverage class="LorenzoPatientenInformation"
                  name="default" default="true">
    <AggWeight att="patientennummer" weight="0.0"/>
    <AggWeight att="vorname" weight="0.0"/>
    <AggWeight att="nachname" weight="0.0"/>
    <AggWeight att="geburtsjahr" weight="6.0"/>
    <AggWeight att="geschlecht" weight="4.0"/>
    <AggWeight att="krankenhausfälle" weight="0.0"/>
</AggregateAverage>
```

Listing 3: Ähnlichkeitsmaß der Benutzerklasse „LorenzoPatientenInformation“

Neben den Ähnlichkeitsmaßen enthält das CAKE Ähnlichkeitsmodell zusätzlich *Completion Rules*. Dies sind Regeln die beispielsweise dazu verwendet werden können, eine Query zu vervollständigen. Completion Rules sind in Skripten repräsentiert, die von der *CAKE Script Engine* ausgeführt werden.

Adaptionswissen

Adaptionswissen wird im Reuse-Schritt des 4Re-Zyklus angewendet, um die Lösung des zu einer Query ähnlichsten Falls oder die Lösungen der zu ihr ähnlichsten Fälle anzupassen und somit eine Lösung für den betrachteten neuen Fall abzuleiten. In CAKE wird dieses Wissen in Form von Skripten repräsentiert und von der *CAKE Script Engine* ausgeführt.

Die drei Wissenscontainer Vokabular, Ähnlichkeitsmodell und Adaptionswissen werden während des Entwurfs und der Entwicklung eines CBR-Systems (*build time*) spezifiziert und bleiben immer konstant. Im Gegensatz dazu kann die Fallbasis während der Laufzeit eines CBR-Systems (*run time*) durch das Hinzufügen von Fällen im Retain-Schritt, aber auch durch das Löschen veralteter Fälle in einem Wartungsschritt, manipuliert werden. Eine weitere Charakteristik der vier Wissenscontainer ist, dass im Prinzip jeder von ihnen das gesamte benötigte Wissen eines CBR-Systems aufnehmen könnte. Insbesondere ist es also fast immer möglich, Wissen eines Wissenscontainers in einen anderen zu überführen. Würde beispielsweise die Fallbasis alle nach dem Vokabular potentiell möglichen Fälle beinhalten, so wäre kein Adaptionswissen und lediglich ein sehr einfaches Ähnlichkeitsmaß, das nur auf Gleichheit überprüfen kann, notwendig. Es ist im Gegensatz dazu aber auch möglich, hauptsächlich Adaptionswissen abzulegen, das so mächtig ist, dass nahezu jeder Fall als Grundlage zur Lösungsgenerierung herangezogen werden kann. Im Extremfall würden dadurch sogar eine Fallbasis und ein Ähnlichkeitsmodell überflüssig werden,

jedoch würde dadurch die Idee des Fallbasierten Schließens ad-absurdum geführt werden.

Verwendung der CBR Technologie in CAKE

Bei der Entwicklung eines auf CAKE basierenden Systems wird die CBR Technologie für zwei hauptsächliche Einsatzzwecke genutzt: zum einen im „traditionellen“ Sinne als Wissensquelle für gemachte Erfahrungen, zum anderen zur situationsabhängigen Auswahl eines geeigneten verfügbaren Agenten bzw. zur Auswahl eines am geschicktesten zu startenden Workflows.

Wissensquelle

Zur Verwendung der CBR Technologie als Wissensquelle muss zunächst das Vokabular, also das domänenabhängige Datenmodell festgelegt werden. Dies geschieht durch die Definition von Benutzerklassen, die die Systemklassen des CAKE Datenmodells erweitern. Im Anschluss muss das Ähnlichkeitsmodell definiert werden, damit Datenobjekte der Benutzerklassen ähnlichkeitsbasiert verglichen werden können. Dies geschieht durch die Parametrisierung geeigneter Ähnlichkeitsmaße. Sind bereits zur Entwicklungszeit Fälle bekannt, so kann ein CAKE Datenobjekt-Pool als initiale Fallbasis aufgebaut. Als letzter Schritt muss ein Agent definiert werden, der in der Lage ist, auf die Fallbasis zuzugreifen und diese ggf. zu manipulieren. Wird dieser Agent dann dem Agentenpool hinzugefügt, so steht die Fallbasis dem System als Wissensquelle zur Verfügung.

Auswahl von Agenten und Workflows

Zur situationsabhängigen Auswahl eines Agenten bzw. eines Workflows kann die CBR Technologie ebenfalls eingesetzt werden. Durch ihren Einsatz kann die Flexibilität des Systems erreicht werden, die in den Anforderungen gefordert wurde. Hier sind vor allem die bereits in Abschnitt 4 angesprochenden dynamische Agentenpools und die flexiblen Workflows zu nennen. Zur Umsetzung werden Agenten und Workflows durch spezielle CAKE Datenobjekte, deren Struktur der Benutzer im domänenabhängigen Datenmodell definieren kann, charakterisiert. Dabei beschreibt die Charakterisierung eines Agenten, für welche Aufgaben der Agent kompetent ist und die Charakterisierung eines Workflows, in welchen Situationen der Workflow erfolgsversprechend ausgeführt werden kann. Die Charakterisierungen werden in zwei verschiedenen Fallbasen, nämlich der *Agent Characterization Case Base* und der *WD Characterization Case Base* (siehe Abbildung 1) abgelegt. Diese beiden Fallbasen sind also feste Bestandteile jeder auf CAKE basierenden Anwendung. CAKE Datenobjekte der gleichen Benutzerklassen werden zur Laufzeit des Systems außerdem genutzt, um die aktuelle Situation, die auch als *Kontext* bezeichnet wird, zu repräsentieren. Somit können durch einen ähnlichkeitsbasierten Vergleich von Charakterisierungen und Elementen des aktuellen Kontexts Agenten und Workflows situationsabhängig ausgewählt werden.

4.3 CAKE Workflow Technologie

Das Ziel der Workflow Technologie liegt in der Unterstützung der Ausführung flexibler *Geschäftsprozesse*. Als Geschäftsprozess wird dabei eine Zusammenfassung mehrerer koordiniert ablaufender Aktivitäten verstanden, deren Ziel es ist, ein für einen Auftraggeber nützliches Endprodukt zu schaffen. Ein koordinierter Ablauf der Aktivitäten bedeutet, dass diese sequentiell oder parallel angeordnet sein können sowie nur unter bestimmten Bedingungen oder auch iterativ ausgeführt werden können. Eine Aktivität kann außerdem selbst wiederum die Ausführung eines weiteren Geschäftsprozesses anstoßen [Wiki, Galileo]. Geschäftsprozesse laufen „in der realen Welt“ ab und ihre Unterstützung ist das primäre Ziel eines auf CAKE basierenden Unterstützungssystems. Damit ein solches Unterstützungssystem Geschäftsprozesse flexibel unterstützen kann, werden „interne“ Prozesse benötigt, die beispielsweise Teilprodukte des Endprodukts erzeugen oder Hintergrundinformationen beschaffen, die zur Ausführung des Geschäftsprozesses notwendig sind. Zu diesen internen Prozessen zählen *Kollaborationsmuster (engl. Collaboration Patterns* [FSB05, FMMS05a]) und *Unterstützungsprozesse*. Mit Hilfe von *Kollaborationsmustern* können Agenten-Kollaborationsstrategien modelliert werden, d.h. basierend auf Erfahrungen können „best-practices“ festgehalten werden. Ein typisches Kollaborationsmuster, das in einer mittelgroßen Firma modelliert sein könnte, wäre eine Formalisierung folgender Strategie: „Tritt am Deinem PC ein Fehler auf, den Du Dir nicht erklären kannst, so suche zunächst in der im Intranet verfügbaren FAQ nach einer Lösung. Ist Dein Problem dort nicht beschrieben, so wende Dich an Herrn Müller (Durchwahl 123) oder Frau Schulz (Durchwahl 456). Nur wenn diese beiden Dir nicht helfen können, schicke eine E-Mail an den Leiter der IT-Abteilung“. Kollaborationsmuster bestehen also aus mehreren Aktivitäten und können insbesondere weitere Prozesse anstoßen. Als *Unterstützungsprozesse* werden CAKE-spezifische Prozesse bezeichnet, die aus technischer Sicht für interne Aufgaben notwendig sind, jedoch auf konzeptioneller Ebene keine Rolle spielen. Ein Beispiel für einen Unterstützungsprozess ist ein Prozess, der auf initiale Benutzereingaben wartet, um schließlich in Abhängigkeit dieser Eingaben ein Kollaborationsmuster auszuführen.

Geschäftsprozess

Kollaborationsmuster

Unterstützungsprozess

In CAKE wird zur Modellierung von Geschäftsprozessen, Kollaborationsmustern und Unterstützungsprozessen der gleiche Repräsentationsmechanismus verwendet, d.h. die aus konzeptioneller Sicht wichtige Unterscheidung dieser drei Prozesstypen spielt bei der technischen Umsetzung keine Rolle. Alle Prozesstypen werden in *CAKE Workflow Definitionen* beschrieben, die gemeinsam in einem *CAKE Workflow-Definitionsmodell* abgelegt sind. Workflow Definitionen können zur Laufzeit situationsabhängig instanziiert und ausgeführt wer-

CAKE Workflow Definition

CAKE Workflow-Definitionsmodell

CAKE Workflow Kontext

den; solche laufenden Instanzen werden als *CAKE Workflows* bezeichnet. Jeder laufende Workflow hat einen eigenen *Kontext*, der jeweils seine aktuelle Situation repräsentiert. Ein Kontext ist wie ein Wörterbuch strukturiert, d.h. unter frei wählbaren Schlüsselworten können beliebige CAKE Datenobjekte hinterlegt werden. Beispielsweise könnte ein während des Ablaufs erzeugtes Ergebnis unter einem Schlüsselwort „Ergebnis“ abgelegt werden.

CAKE Workflow-Definitionsmodell

CWDM

Ein applikationsabhängiges Workflow-Definitionsmodell enthält CAKE Workflow Definitionen für alle in der Applikation notwendigen Prozesse. Es wird in einem XML-Format namens *CAKE Workflow Definition Model (CWDM)* repräsentiert. Eine detaillierte Beschreibung der als Grammatik dienenden XML Schema Definition ist von Maximini [Max06] beschrieben.

Jede CAKE Workflow Definition besteht aus einer *Workflow Charakterisierung*, einer *Menge an Aktivitäten*, einem *Kontrollfluss* zwischen diesen Aktivitäten und *initialen Werten für den Kontext* eines laufenden Workflows. Diese vier Teile einer Workflow Definition sind im Folgenden beschrieben. Als durchgängiges Beispiel zur Beschreibung der einzelnen Teile einer Workflow Definition im CWDM-Format wird ein Kollaborationsmuster namens „EinfacheDBAnfrage“ angenommen, das in der Lage ist, in Abhängigkeit vom Datentyp einer Query zu entscheiden, welcher Agent diese am besten beantworten kann, dann diesem Agenten die Query weiterzuleiten und schließlich das Ergebnis zurückzuliefern. Dieses Kollaborationsmuster ist dem Appear Workflow-Definitiosmodell entnommen.

Workflow Charakterisierung

Die Workflow Charakterisierung ist ein spezielles CAKE Datenobjekt, das eine semantische Beschreibung des Workflows beinhaltet. Genauer beschreibt sie, in welchen Situationen der Workflow erfolgsversprechend ausgeführt werden kann. Sie wird in der *WD Characterization Case Base* (siehe Abbildung 1) abgelegt, so dass eine situationsabhängige, fallbasierte Suche nach sinnvollerweise auszuführenden Workflow Definitionen ermöglicht wird. Die Charakterisierung in Listing 4 sagt aus, dass dieser Workflow Queries der angegebenen Benutzerklassen zu beantworten vermag.

```
<Characterization>
   <co:Agg c="WorkflowCharakterisierung">
      <co:AA n="name" v="EinfacheDBAnfrage"/>
      <co:OA n="kompetenz">
         <co:C>
            <co:A v="Referenzworkflow"/>
            <co:A v="LaborbefundInformation"/>
```

```
            <co:A v="CCNeurologischerBefund"/>
            <co:A v="CCAnamnese"/>
            <co:A v="LorenzoKrankenhausfall"/>
            <co:A v="LorenzoPatientenInformation"/>
            <co:A v="DRGInformation"/>
            <co:A v="ICD10Information"/>
            <co:A v="OPSInformation"/>
        </co:C>
    </co:OA>
  </co:Agg>
</Characterization>
```

Listing 4: CDWM: Workflow Charakterisierung

Menge an Aktivitäten

Task Beschreibung

Die Aktivitäten eines Workflows werden als *Tasks* bezeichnet. Jeder Task eines Workflows wird durch seine *Task Beschreibung* beschrieben. Eine solche Beschreibung besteht aus dem Namen des Tasks, dem Namen der ihn ausführenden JAVA-Klasse und einer Menge an Parametern, die zur Ausführung des Tasks notwendig sind. Damit nicht jeder Task aufwändig in Java programmiert werden muss, werden von CAKE bereits eine Menge von Tasks zur Verfügung gestellt, die für die meisten Anwendungen ausreichen. Zur Realisierung anwendungsspezifischer Tasks können entweder eigene Java Klassen programmiert werden oder aber Tasks verwendet werden, die die CAKE Script Engine benutzen und vom System bereits zur Verfügung gestellt werden. In diesem Fall genügt die Angabe eines entsprechenden Skriptes, das die Anwendungslogik enthält. In beiden Fällen können durch einen Task sowohl einfache Aufgaben wie die Addition zweier Zahlen als auch komplexe Aufgaben wie das Zusammentragen und Bewerten von Informationen realisiert werden. Insbesondere kann ein Task fallbasiert nach Workflow Charakterisierungen suchen und ggf. den entsprechenden Workflow starten, so dass eine hierarchische Dekomposition von Prozessen möglich ist.

Das Kollaborationsmuster „EinfacheDBAnfrage“ besteht aus zwei Tasks, nämlich einem Task zur Auswahl eines Agenten (WaehleAgent) und einem Task zur Ausführung der Anfrage (FuehreAnfrageAus). Beide Tasks sind als Skripte realisiert. Die Task Beschreibung des Tasks „WaehleAgent“ ist in Listing 5 angegeben.

```
<Task name="WaehleAgent" templateName="JS"
      class="cake.workflow.task.defaultImpl.JavaScript">
  <Parameter type="boolean" name="reload">
     true
  </Parameter>
  <Parameter type="string" name="script">
```

```
      wd_einfachedb.js
    </Parameter>
    <Parameter type="string" name="failureKey">
      result
    </Parameter>
</Task>
```

Listing 5: CDWM: Task Beschreibung

Kontrollfluss

Mit Hilfe des Kontrollflusses können die Tasks sequentiell, parallel, oder iterativ angeordnet werden. Es gibt immer einen ausgezeichneten Start-Task und einen ausgezeichneten Ende-Task einer Workflow Definition. Zu erwähnen ist an dieser Stelle, dass nur der Kontrollfluß, also die Reihenfolge der einzelnen Tasks, explizit modelliert wird, nicht jedoch der Datenfluss. Der Datenaustausch geschieht unter Verwendung des Kontextes, auf den alle Tasks eines Workflows zugreifen können. CAKE verwendet zum Datenaustausch also das *Blackboard-Paradigma* [Cor03]. Die beiden Tasks des Kollaborationsmusters „EinfacheDBAnfrage“ werden sequentiell ausgeführt. Die entsprechende Repräsentation in CDWM zeigt Listing 6.

```
<FirstConnection destinationTask="WaehleAgent"
                 destinationPort="in"/>
<Connection sourceTask="WaehleAgent"
            sourcePort="out"
            destinationTask="FuehreAnfrageAus"
            destinationPort="in"/>
<FinalConnection sourceTask="FuehreAnfrageAus"
                 sourcePort="out"/>
```

Listing 6: CDWM: Kontrollfluss

Initiale Kontext-Werte

Jeder laufende Workflow hat einen Kontext, der jeweils die aktuelle Situation des Workflows repräsentiert. Sollen beim Start eines Workflows bereits initiale Werte im Kontext gesetzt werden, so können diese bereits in der Workflow Definition spezifiziert werden. Ein Beispiel eines solchen Kontext-Eintrags ist in Listig 7 zu sehen: unter dem Schlüsselwort „timeout“ wird initial die Anzahl an Sekunden angegeben, die bei Netzwerkverbindungen gewartet werden soll.

```
<ContextEntry key="timeout">
  <co:A v="30" c="Integer"/>
</ContextEntry>
```

Listing 7: CDWM: Initialer Kontext-Wert

CAKE Workflows

Workflow Engine Manager

Workflow Engine

Zur Laufzeit initialisiert der *Workflow Engine Manager* einen Workflow als Instanz einer Workflow Definition. Jeder Workflow wird von einer eigenen *Workflow Engine* kontrolliert, die die entsprechende Workflow Definition interpretiert und die Tasks in der spezifizierten Reihenfolge ausführt. Die explizite Trennung von Workflow Definition und Ausführung ermöglicht flexible Änderungen, also beispielsweise das Hinzufügen oder Weglassen von Tasks, während der Ausführung eines Workflows, ohne dass die Workflow Definition angepasst werden muss. Eine Workflow Engine ist außerdem zuständig für die Verwaltung des Kontexts des von ihm kontrollierten Workflows. Jede Workflow Engine besitzt einen eigenen Kontext, auf den jeder in der Workflow Engine ablaufende Task lesend und schreibend zugreifen darf. Zusätzlich gibt es die Möglichkeit, auf übergeordnete Kontexte zuzugreifen, d.h. wird eine Workflow Engine von einer anderen Workflow Engine gestartet, so haben die Tasks der gestarteten Engine lesenden Zugriff auf den Kontext der übergeordneten Engine. Die daraus entstehende Hierarchie der Kontexte spiegelt somit die Hierarchie der Workflow Engines wider.

4.4 CAKE Agenten Technologie

Aufgabe der Agenten Technologie ist die Realisierung der Integration von Personen und Informationen, damit sämtliche während eines durch die Workflow Technologie beschriebenen Kollaborationsprozesses auftretenden Informationsbedürfnisse erfüllt werden können. Dabei kommen als Informationslieferanten sowohl elektronische Wissensquellen als auch menschliche Wissensquellen, also Kollaborationspartner, in Betracht. Informationen anfragen können ebenfalls sowohl menschliche Kollaborationspartner als auch Computersysteme: beispielsweise könnte eine elektronische Wissensquelle zur Beantwortung einer Anfrage ihrerseits weitere Informationen benötigen. Ziel der Agenten Technologie ist es daher, vom konkreten Typ des Agenten (menschlicher Agent oder Computersystem) zu abstrahieren und die Kollaboration sehr verschiedener Agenten zu ermöglichen. Dazu ist aus Systemsicht eine einheitliche Schnittstelle vonnöten, die die konkrete Ausprägung des Agenten kapselt.

CAKE Agentenframework

Agenten-framework

Das *Agentenframework* stellt eine einheitliche Schnittstelle zur Verfügung, über die beliebige Agenten an das System angebunden werden können. Aus Systemsicht spielt dabei der konkrete Typ des Agenten keine Rolle. Aus technischer

Sicht werden jedoch *Informationsagenten* und *Benutzeragenten* unterschieden: Ein

Informationsagent

Informationsagent ist ein Informationslieferant, d.h. seine Aufgabe ist die Bereitstellung von Informationen während eines Kollaborationsprozesses. Mit Hilfe von Informationsagenten werden also beispielsweise Datenbanken, Suchmaschinen, fallbasierte Systeme, gemeinsam genutzte Kalender, Dialog-Strategie-Tools oder menschliche Kollaborationspartner integriert. Gegebenenfalls kann ein Informationsagent auch die von ihm bereitgestellten Informationen manipulieren, z.B. durch das Einfügen oder Löschen eines Datenbanktupels in seine Datenbank, eines Falls in seine Fallbasis oder eines Termin in seinen Terminkalender. Ein *Benutzeragent* fragt Informationen an, hat also Informationsbedürfnisse, die erfüllt werden sollen. Beispiele sind menschliche Kollaborationspartner, Dialog-Strategie-Tools oder Suchmaschinen, die zur Beantwortung einer Anfrage zunächst bei anderen Informationslieferanten Informationen beziehen müssen. Wichtig ist es festzuhalten, dass ein Agent sowohl Informationsagent als auch Benutzeragent sein kann: beispielsweise kann ein menschlicher Benutzer im Rahmen eines Kollaborationsprozesses mal Informationslieferant und mal Anfragender sein.

Benutzeragent

Jeder Agent ist konzeptionell durch eine *Technologie Komponente*, einen *Wrapper* und eine *Agenten Charakterisierung* aufgebaut. Als *Technologie Komponenten* werden die externen Systeme bezeichnet, die in CAKE integriert werden sollen, also die menschlichen Kollaborationspartner und die benötigten Computersysteme. Um die Kommunikation zwischen CAKE und diesen Technologie Komponenten zu ermöglichen werden *Wrapper* als Mediatoren eingesetzt. Sie vermitteln einerseits zwischen den von den externen Systemen verwendeten Ontologien und dem domänenabhängigen Datenmodell und realisieren andererseits die technische Schnittstelle zur Anbindung der Technologie Komponente. Beispielsweise realisieren Wrapper im Falle einer Datenbank ein OR-Mapping zwischen dem Datenbankschema und dem domänenabhängigen Datenmodell und ermöglichen es Anfragen an das Datenbanksystem über eine Datenbankschnittstelle zu stellen. Im Falle eines menschlichen Benutzers werden grafische Benutzeroberflächen oder Sprachverarbeitungssoftware als Schnittstellen verwendet. Die Eingaben des Benutzers werden dann vom Wrapper in Instanzen des domänenabhängigen Datenmodells überführt, so dass sie vom System weiterverarbeitet werden können. Die *Agenten Charakterisierung* ist ein spezielles CAKE Datenobjekt, das eine semantische Beschreibung der Kompetenz des Agenten beinhaltet, also beispielsweise, welche Arten von Anfragen ein Informationsagent beantworten kann. Sie wird in der *Agent Characterization Case Base* (siehe Abbildung 1) abgelegt, so dass eine fallbasierte Suche nach dem kompetentensten Informationsagenten ermöglicht wird.

Technologie Komponente

Wrapper

Agenten Charakterisierung

Eine Besonderheit des CAKE Systems ist die Verwaltung eines *flexiblen Agen-*

Flexibler Agentenpool

tenpools, d.h., dass nicht zu jedem Zeitpunkt jeder Agent dem System zur Verfügung stehen muss [vEDA04]. Dazu unterscheidet CAKE zwischen *bekannten* Agenten und *verfügbaren* Agenten: Jeder Agent, der prinzipiell durch das System integriert werden soll, muss durch eine Agenten Charakterisierung beschrieben werden und ist dadurch dem System bekannt. Es kann aber vorkommen, dass ein solcher Agent nicht zur Verfügung steht: dies gilt natürlich hauptsächlich für menschliche Agenten, die nicht 24 Stunden am Tag und 7 Tage die Woche verfügbar sind, aber auch für elektronische Agenten, die beispielsweise wegen des Neustarts des Servers, auf dem sie laufen, temporär nicht erreichbar sind. Die Menge der verfügbaren Agenten ist daher eine Teilmenge der bekannten Agenten. Agenten können sich jederzeit am Agentenframework anmelden und abmelden. Durch die Charakterisierungen der Agenten und der fallbasierten Suche auf diesen Charakterisierungen kann das System aber immer gewährleisten, dass zu jedem Zeitpunkt die optimale Kollaboration zwischen den aktuell verfügbaren Agenten unterstützt wird.

CAKE Agenten-Definitionspool

CAKE Agenten-Definitionspool

Ein applikationsabhängiger *CAKE Agenten-Definitionspool* enthält Agenten Definitionen für alle in der Applikation bekannten Agenten. Zur persistenten Speicherung eines solchen Pools wurde ein eigenes XML-Format namens *CAKE Agent Definition Pool (CADP)* entwickelt. Eine detaillierte Beschreibung der als Grammatik dienenden XML Schema Definition ist in der Dokumentation des CAKE Systems [MFS+05] beschrieben.

CADP

Eine CAKE Agenten Definition besteht aus vier Teilen, nämlich einer Menge an *Attributen*, einer *Agenten Charakterisierung*, einer Menge an *externen Bibliotheken (Libraries)* und einer Menge an *Parametern*. Diese vier Teile einer Agenten Definition sind im Folgenden beschrieben. Als durchgängiges Beispiel zur Beschreibung der einzelnen Teile einer Agenten Definition im CADP-Format wird der aus dem Appear-System stammende LORENZO-Agent verwendet.

Attribute

Die wichtigsten Informationen zu einem Agenten sind sein Name, die JAVA-Klasse die seine Funktionalität implementiert und die Festlegung, ob es sich um einen Informationsagenten und/oder einen Benutzeragenten handelt. Diese drei Informationen werden in Attributen des Tags <AgentDefinition> abgelegt, das eine Agenten-Definition einleitet. Wie in Listing 8 zu sehen ist, trägt der LORENZO-Agent den Namen „Lorenzo", ist ein Informationsagent und wird durch die Klasse „cake.appear.agents.Lorenzo“ implementiert.

```
<AgentDefinition name="Lorenzo"
    class="cake.appear.agents.Lorenzo"
    type="information">
```

```
  . . .
</AgentDefinition>
```

Listing 8: Attribute des LORENZO-Agenten

Agenten Charakterisierung

Die Agenten Charakterisierung ist ein spezielles CAKE Datenobjekt, das eine semantische Beschreibung des Agenten beinhaltet. Die Charakterisierung in Listing 9 sagt aus, dass der Lorenzo-Agent in der Lage ist auf administrative Patienteninformationen zuzugreifen.

```
<co:Agg c="AgentCharakterisierung">
    <co:AA n="name" v="Lorenzo"/>
    <co:AA n="typ" v="DB"/>
    <co:OA n="kompetenz">
        <co:C>
            <co:A v="Administrative␣Patienteninformationen"/>
        </co:C>
    </co:OA>
</co:Agg>
```

Listing 9: Charakterisierung des LORENZO-Agenten

Externe Bibliotheken

Mit Hilfe des <Library>-Tags können beliebig viele externe Bibliotheken angegeben werden, die von der Implementierung des Agenten benötigt werden. Soll ein Informationsagent beispielsweise Zugriff auf eine Datenbank haben, so kann hier eine Datenbankschnittstelle eingebunden werden. Das CAKE Agentenframework kapselt die einzelnen Agenten voneinander ab, so dass in einer Anwendung von verschiedenen Agenten unterschiedliche Versionen der gleichen externen Bibliothek problemlos verwendet werden können. Somit agiert das Agentenframework als Applikationsserver. Als externe Bibliothek können jar-Dateien, Java Klassen oder ganze Verzeichnisse angegeben werden. Wie in Listing 10 zu sehen ist benötigt der LORENZO-Agent eine jar-Datei, die Implementierungen applikationsspezifischer Agenten enthält, und einen Postgres-Treiber zur Vereinfachung der Implementierung des Wrappers.

```
<Library name="lib/appear-agents.jar"/>
<Library name="lib/postgresql-8.1-407.jdbc3.jar"/>
```

Listing 10: Benötigte Bibliotheken des LORENZO-Agenten

Parameter

Eine Agenten-Definition kann beliebig viele Parameter spezifizieren, die von der den Agenten implementierenden Klasse ausgewertet werden müssen. Die Agenten-Definition legt dabei die Parameternamen fest und ob die Angabe der Parameter bei der Registrierung eines Agenten unbedingt erforderlich oder lediglich optional sind. Jeder Parameter kann durch eine textuelle Beschreibung

näher erläutert werden. In Listing 11 sind die vier Parameter des LORENZO-Agenten zu sehen: Damit der Agent auf die LORENZO-Datenbank zugreifen kann, müssen beispielsweise der Benutzername und das Passwort eines berechtigten Benutzers spezifiziert sein.

```
<Parameter name="connectionURI" required="true">
   <Description>Connection URI to the database</Description>
</Parameter>
<Parameter name="user" required="true">
   <Description>user for the database</Description>
</Parameter>
<Parameter name="password" required="true">
  <Description>password to the database</Description>
</Parameter>
<Parameter name="driverName" required="true">
   <Description>jdbc driver name</Description>
</Parameter>
```

Listing 11: Parameter des LORENZO-Agenten

CAKE Agenten Registrierung und Deregistrierung

Um einen bekannten Agenten verfügbar zu machen, muss er beim CAKE Agent Manager registriert werden. Bei der Registrierung wird angegeben, welche Agenten Definition dem Agenten zugrunde liegt und dementsprechend müssen auch zumindest die erforderlichen Parameter konkret spezifiziert werden. Bei der Registrierung eines Agenten kann außerdem eine weitere Agenten Charakterisierung angegeben sein, die sich von der Charakterisierung in der Agenten Definition unterscheiden kann. Meist ist die bei der Registrierung angegebene Agenten Charakterisierung spezifischer als die in der Agenten Definition. Falls diese angegeben ist, wird sie bei der Registrierung in die Agent Characterization CB übernommen, ansonsten wird quasi als Fallback die Charakterisierung aus der Definition übernommen. Listing 12 zeigt die XML-Repräsentation der Registrierung eines LORENZO-Agenten, der kompetent ist, Anfragen nach Krankenhausfällen eines Patienten oder nach Informationen zu einem Patienten zu beantworten. Die Informationen sind in einer Datenbank namens „codes“ gespeichert und können von einem Benutzer namens „codes“ ausgelesen werden. Zur Verbindung mit der Datenbank wird der angegebene Postgres-Treiber verwendet.

```
<RegisterAgent name="Lorenzo" definition="Lorenzo">
   <co:Agg c="AgentCharakterisierung">
```

```
        <co:AA n="name" v="Lorenzo"/>
        <co:AA n="typ" v="DB"/>
        <co:OA n="kompetenz">
            <co:C>
                <co:A v="LorenzoKrankenhausfall"/>
                <co:A v="LorenzoPatientenInformation"/>
            </co:C>
        </co:OA>
    </co:Agg>
    <Parameter name="connectionURI">
        jdbc:postgresql:codes
    </Parameter>
    <Parameter name="user">codes</Parameter>
    <Parameter name="password">codes</Parameter>
    <Parameter name="driverName">
        org.postgresql.Driver
    </Parameter>
</RegisterAgent>
```

Listing 12: Registrierung des LORENZO-Agenten

CAMC

Alle Agenten, die bei Systemstart automatisch registriert werden sollen, können in einer XML-Datei namens *CAKE Agent Manager Configuration (CAMC)* zentral festgelegt werden. Eine CAMC-Datei enthält also eine Menge an <RegisterAgent>-Elementen. Eine detaillierte Beschreibung der als Grammatik dienenden XML Schema Definition ist in der Dokumentation des CAKE Systems [MFS+05] beschrieben.

Über den CAKE Agent Manager können verfügbare Agenten zur Laufzeit des Systems auch jederzeit wieder deregistriert werden.

CAKE Datenbankagent

In vielen Anwendungen dienen relationale Datenbanken als Datenquellen, um benötigte Informationen persistent zur Verfügung zu stellen. Einer der ersten Agenten im Standard-Repertoire von CAKE dient daher zur Kommunikation mit relationalen Datenbanken. Dieser Datenbankagent stellt eine generische Schnittstelle zur Verfügung, so dass ein Anwendungsentwickler ihn zu dessen Verwendung lediglich instanziieren und ein Mapping zwischen den domänenabhängigen CAKE Datenklassen und den Tabellen und Attributen der relationalen Datenbank angeben muss.

In Abbildung 4 ist der grundsätzliche Ablauf der Kommunikation über das CAKE Agentenframework als UML-Sequenzdiagramm dargestellt: Der Benut-

zer (*User*; stellvertretend für eine beliebige Komponente) sendet eine Anfrage an das CAKE Agentenframework. Dieses analysiert die Anfrage und wählt aus den aktuell registrierten Agenten den zur Beantwortung der Anfrage kompetentesten (Datenbank-)Agenten aus. Diesem sendet es die Anfrage weiter. Der Datenbankagent konvertiert die Anfrage in ein äquivalentes SQL-Statement und sendet dieses an die von ihm angebundene Datenbank. Nach dem Empfang der Ergenbistabelle konvertiert der Datenbankagent dieses wieder in ein CAKE Datenobjekt, das über das Agentenframework schließlich den Benutzer erreicht.

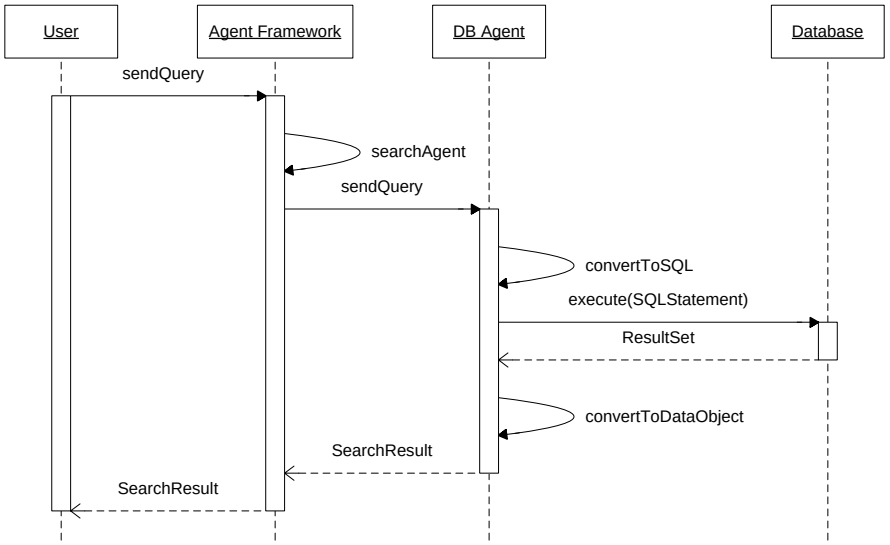

Abbildung 4: UML-Sequenzdiagramm des CAKE Datenbankagenten

4.5 CAKE Editor

Der *CAKE Editor* stellt eine grafische Modellierungsumgebung zur Verfügung, mit dessen Hilfe alle Daten, die für eine domänenabhängige Anwendung definiert werden müssen, modelliert werden können. Der CAKE Editor basiert auf dem Eclipse Framework und bietet verschiedene Perspektiven für die zu modellierenden Daten.

Data Model

In der *Data Model*-Perspektive kann das domänenabhängige Datenmodell modelliert werden. In einem Klassen-Baum werden initial alle Systemklassen angezeigt, die zu Benutzerklassen spezialisiert werden können. Sämtliche Eigenschaften, die durch das Konzept der CAKE Datenrepräsentation angeboten werden wie die Einschränkung von Wertebereichen bei atomaren Datenklassen, die Angabe einer taxonomischen Ordnung bei String-Datenklassen, oder die

Definition von Attributen und ihren Datenklassen bei Aggregate-Datenklassen können hier grafisch modelliert werden. Auch spezifische Ähnlichkeitsmaße für Benutzerklassen werden in der *Data Model*-Perspektive definiert. Sind das domänenabhängige Datenmodell und das domänenabhängige Ähnlichkeitsmodell konsistent modelliert, können entsprechende CDM- und CDSM-Dateien gespeichert werden. Abbildung 5 zeigt einen Screenshot des CAKE Editors. Zu sehen ist die *Data Model*-Perspektive, in der die Benutzerklasse „LorenzoPatientenInformation“ definiert wird.

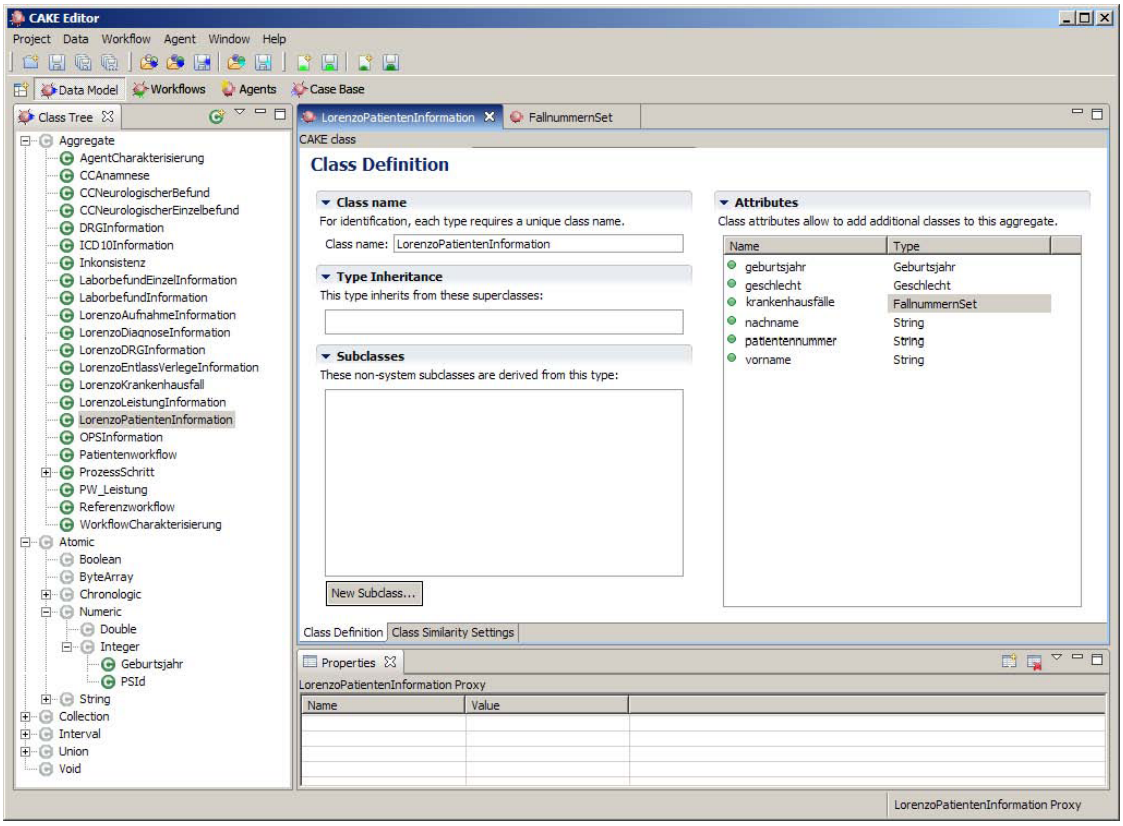

Abbildung 5: CAKE Editor, *Data Model*-Perspektive

Case Base In der *Case Base*-Perspektive können initiale Fallbasen angelegt werden. Dazu muss zuerst das domänenabhängige Datenmodell definiert worden sein und in der Fallbasis können dann CAKE Datenobjekte einer Benutzerklasse instanziiert und abgelegt werden. Eine so angelegte Fallbasis wird schließlich als CDOP-Datei (siehe Abschnitt 4.1) abgespeichert.

Workflows In der *Workflows*-Perspektive kann das CAKE Workflow-Definitionsmodell modelliert werden. Die Definition eines solchen Modells beginnt mit dem Anle-

gen der Workflow-Definitionen und den sie aufbauenden Tasks. Auch die einzelnen Task Beschreibungen sowie die Workflow Charakterisierungen können in dieser Perspektive definiert werden. den Hauptteil dieser Perspektive nimmt ein Editor ein, mit dem der Kontrollfluss zwischen den Tasks grafisch spezifiziert werden kann. Ist das Workflow-Definitionsmodell vollständig modelliert, kann eine CWDM-Datei abgespeichert werden.

Agents

In der *Agents*-Perspektive können alle bekannten Agenten beschrieben werden. Insbesondere können hier die Attribute, Agenten Charakterisierungen, externen Bibliotheken und Parameter jeder Agenten Definition festgelegt werden. Sind alle verfügbaren Agenten beschrieben, kann eine CADP-Datei (siehe Abschnitt 4.4) geschrieben werden. Die Perspektive bietet außerdem die Möglichkeit, die beim Systemstart automatisch zu registrierenden Agenten zu beschreiben und kann auch eine entsprechende CAMC-Datei zu generieren.

5 Lösungen für die Anwendungsdomänen

In diesem Abschnitt werden Lösungen für die Problemstellungen der in Abschnitt 3 vorgestellten Anwendungsdomänen skizziert. Dabei wird insbesondere betont, welche Technologien des CAKE Frameworks in den einzelnen Anwendungen auf welche Weise eingesetzt werden. Die einzelnen Lösungen sind in zahlreichen Veröffentlichungen des Lehrstuhls für Wirtschaftsinformatik II ausführlicher beschrieben; Referenzen auf Veröffentlichungen mit weitergehenden Informationen zu den einzelnen Anwendungsdomänen sind jeweils in den Abschnitten angegeben.

5.1 Medizin

Das *Appear-System*[6] *(**A**nalyse von **P**atientenbehandlungs-**P**rozessen zur **E**rkennung von **A**bweichungen und **R**egelmäßigkeiten)* wird vorraussichtlich im Frühjahr 2007 fertig gestellt sein. Detailierte Informationen werden nach Fertigstellung veröffentlicht.

5.2 Geografisches Informationsmanagement

Die erste Version eines auf CAKE basierenden Unterstützungssystems für das GIS-Szenario wurde im Jahr 2005 durch eine studentische Forschungsgruppe unter dem Namen *GIS-DOKU*[7] [BOP+05, SMMB06] realisiert. GIS-DOKU unterstützt die Projektmitarbeiter während der Erfassung der Kreise, Gemeinden und Ortslagen durch eine Dokumentation der Anfragen an Behörden und eine standardisierte Fehlerbehandlung und liefert den Entscheidungsträgern durch diese automatisierte Dokumentation die Möglichkeit, den aktuellen Bearbeitungszustand jedes Gebiets abzufragen. Das GIS-DOKU Workflow-Definitionsmodell macht intensiven Gebrauch der Möglichkeit zur hierarchischen Dekomposition von Workflows: der allgemeinste Workflow ist ein Workflow zur Erfassung eine Gemeinde, der in Abbildung 6 angegeben ist. Für jeden der drei Tasks wurden weitere Workflow Definitionen realisiert, die während des Ablaufs eines Workflows angestoßen werden können. Zusätzlich wurden Workflow Definitionen modelliert, die die Behandlung häufiger und typischer Fehler repräsentieren und deren Instanzen nur dann in einen Workflow eingefügt werden, wenn der entsprechende Fehler auftritt. So verwaltet GIS-DOKU insgesamt 426

[6] http://www.wi2.uni-trier.de/de/cms/projects/Appear/
[7] http://www.wi2.uni-trier.de/de/cms/projects/SP_rjm_04/

Workflows jeweils unterschiedlichen Ausführungsstatus: Gemeinden, bei denen die Erfassung noch nicht begonnen hat, bei denen die Erfassung läuft und bei denen die Erfassung bereits abgeschlossen wurde. Zwar sind alle Workflows Instanzen der Top-Level Workflow Definition, jedoch unterscheiden sich ihre Sub-Strukturen durch die verschiedenen während des Prozesses aufgetretenen Fehler. Die Informationen zur Ableitung aktueller Bearbeitungszustände bezieht GIS-Doku aus ausgewählten, im Unternehmen eingesetzten Wissensquellen (ALK und ALB Datenbanken, Zeiterfassung, Wiki, etc.), die mit Hilfe des Agenten Frameworks eingebunden wurden.

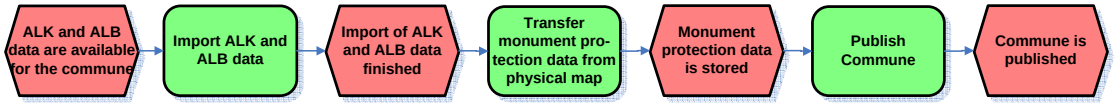

Abbildung 6: Top-Level Workflow Definition: Erfassung einer Gemeinde

Nach Abschluss des studentischen Forschungsprojektes wurden Erweiterungen und Verbesserungen am Unterstützungssystem vorgenommen [SMMB06, SM06, BFM+06]: Lag es in GIS-DOKU noch in der Hand des Benutzers, bei einem auftretenden Fehler die passendste Workflow Definition zur Behandlung dieses Fehlers auszuwählen und in den laufenden Workflow einzufügen, wird dieser Prozess mittlerweile fallbasiert unterstützt. Dazu wurden einerseits die Workflow Charakterisierungen verbessert und zusätzlich wurde eine Fallbasis angelegt die Hilfestellungen zur Auswahl einer Workflow Definition liefern kann: Als Problembeschreibung ist eine Workflow Charakterisierung und ein Zahlenwert angegeben, der anzeigt, wie oft der entsprechende Workflow in der gleichen Gemeinde bereits ausgeführt wurde. Falls also beispielsweise die Rückfrage bei einem Ansprechpartner mehrfach erfolglos blieb, kann die Anfrage schließlich zu dessen Vorgesetzten eskaliert werden. Die kontextabhängige, fallbasierte Suche nach geeigneten Workflow Definitionen läuft für den Nutzer des Systems transparent ab. Weitere Verbesserungen werden aktuell im Bereich der Ableitung aktueller Bearbeitungszustände vorgenommen. Hier wird die Erkennung aktueller und bereits abgearbeiteter Prozessschritte durch die Beobachtung von Bearbeitungsergebnissen optimiert. Dazu müssen neben den durch GIS-DOKU betrachteten Wissensquellen weitere Wissensquellen des Unternehmens mit Hilfe des Agenten Frameworks eingebunden werden.

5.3 Notfalleinsätze der Feuerwehr

Im Rahmen des AMIRA-Projektes[8] wurden eine ganze Reihe unterschiedlicher Szenarien entworfen, wie ein Notfalleinsatz der Feuerwehr durch ein IT-System unterstützt werden kann. Diese Szenarien können grundsätzlich in drei unterschiedliche Gruppen unterteilt werden [FMS05, Max06]:

- **Single Person Request:** Eine Einsatzkraft hat während eines Einsatzes ein konkretes Informationsbedürfnis, stellt eine Anfrage an das AMIRA-System und erwartet die bestmögliche Antwort aus allen verfügbaren Wissensquellen.
- **Pro-active Context-based Information Support:** Das AMIRA-System stellt verschiedenen Einsatzkräften proaktiv Informationen zur Verfügung, die im jeweiligen Zustand der Kollaboration gerade nützlich sind.
- **Collaborative A-posteriori Analysis:** Das AMIRA-System unterstützt die Analyse eines vergangenen Einsatzes und hilft bei der Erstellung eines Abschlussreports, der in zukünftigen Einsätzen als Informationsquelle zur Verfügung steht.

Alle Szenarien wurden im auf CAKE basierenden AMIRA-System realisiert [FMS05, FMMS05a, Max06, BFM+06, FBT+07]. Als Beispiel sei an dieser Stelle die Umsetzung des Single Person Requests skizziert: Zunächst wurden alle verfügbaren Wissensquellen bezüglich ihres Inhalts und ihrer Kompetenz analysiert und anschließend mittels der Agenten Technologie als Informationsagenten in das AMIRA-System integriert: elektronische Wissensquellen konnten durch den Einsatz geeigneter Wrapper direkt, papierbasierte Wissensquellen erst nach einem Digitalisierungs-Schritt integriert werden; menschliche Experten wurden über eine Benutzerschnittstelle (Web-Interface oder Sprachdialog-Komponente) eingebunden. Erfolgsversprechende Suchstrategien wurden dann mit Hilfe von Kollaborationsmustern umgesetzt. Dabei wurde meist die Strategie verfolgt, dass zunächst nur elektronische Wissensquellen zur Beantwortung einer Anfrage herangezogen werden sollten und dass nur falls diese keine Antwort hinreichender Qualität liefern können ein menschlicher Experte angefragt wird. Zum Vergleich und zur Bewertung der Ergebnisse verschiedener Informationsagenten wurden weitere Kollaborationsmuster entwickelt. Diese sind außerdem für die Generierung einer endgültigen Antwort an den Anfragenden verantwortlich, die Wissen von verschiedenen Informationsagenten kumuliert enthalten kann. Eine typische Anfrage läuft dann folgendermaßen ab: eine Einsatzkraft formuliert seine Anfrage in natürlicher

[8] http://www.wi2.uni-trier.de/de/cms/projects/Amira/

Sprache und setzt sie über die Sprachdialog-Komponente ab. Der Text wird vom Wrapper des Benutzeragenten analysiert und in ein CAKE Datenobjekt einer passenden Benutzerklasse überführt. Dann werden geeignete Kollaborationsmuster gestartet, die die Anfrage beantworten können und schließlich als Antwort ein CAKE Datenobjekt mit allen wichtigen Informationen an den Benutzeragenten zurückliefern. Dieser analysiert das CAKE Datenobjekt und liest dem Benutzer schließlich alle relevanten Informationen vor.

5.4 Agiles Software Engineering

An envisaged scenario within an agile development [Boe02] is a follows: the team has proceeded to the first iteration, and an intermediate system has been deployed at the customer site. Negotiations with the customers throughout the first iteration have resulted in the customer preference that an e-commerce system should be added to the system. Now, the team members playing management roles have to settle on the most basic workflow, which will control the very next steps. The team starts to instantiate the very first business process and suitable team members playing the roles of a domain expert and a software developer are assigned. Thereby, organization-specific constraints have to be taken into account (by querying a human resources management information agent providing availability data and a Wiki system containing tentatively planned leaves). These team members can now autonomously decide how to proceed for fulfilling the tasks. E.g., the software developer starts work on the assigned task by adapting the actual business process. Because of not being familiar to the actual project the software developer is searching comprehensive information about this project with respect to similar tasks, to relevant Java references, and to relevant design rationales. That is, relevant information is presented to the developer that he/she can be familiarized to the work at issue. Additionally, the developer gets a link for searching a Java expert for booking systems if more explicit information is necessary. Primarily the idea is to support the developer through computerized agents but if this is not enough there is an option to ask human experts. Therefore, a search facility organizes retrieval on human Java experts, returning information about how to get in contact to such an expert. Surely, the developer can proceed further in adapting the assigned task. Either he/she uses best practices in terms of collaboration which already includes useful domain knowledge or he/she carries out own adaptations on the task for achieving the envisaged task artefact.

Software development using a Scrum-like agile method is supported by CAKE in the following way: The development team starts by instantiating a baseli-

ne workflow definition (long-term workflow) for the first iteration. The team adds „pre-game“ , „sprint“ and „post-game“ sub workflow definitions to it, with each triggering a CAKE sub workflow of their own in sequence. The pre-game workflow definition covers the negotiations with the customer like „meetings on site“ or „writing recommendation reports“ . Using SCBR, workflows outlining best practice learned from previous projects or iterations can be identified to ease workflow modelling. The „sprint“ workflow definition acts as the Scrum backlog: While executing the tasks included within pre-game, the team will add tasks to the sprint workflow required to actually produce the system. For instance, after having executed pre-game, the sprint workflow would include tasks like „design database schema“ or „create input forms“ . For each task, risk is estimated, and by placing control flow relationships between the tasks, the most risky tasks will be executed first. The risk estimation itself is knowledge-intense, thus CAKE supports this as well: By using an information agent which connects to a bug database, information about the reliability of a specific component may be retrieved.

Literaturverzeichnis

[AP94] Agnar Aamodt and Enric Plaza. Case-based reasoning: foundational issues, methodological variations, and system approacches. *AI Communiactions*, 7(1):39–59, 1994.

[BAB+03] Ralph Bergmann, Klaus-Dieter Althoff, Sean Breen, Mehmet Göker, Michel Manago, Ralph Traphöner, and Stefan Wess. *Developing Industrial Case-Based Reasoning Applications: The INRECA Methodology*, volume 1612 of *LNAI*. Springer Berlin, Heidelberg, New York, Hong Kong, London, Milan, Paris, Tokyo, 2 edition, 2003.

[Ber02] Ralph Bergmann. *Experience Management - Foundations, Development Methodology, and Internet-Based Applications*, volume 2432 of *LNAI*. Springer Berlin, Heidelberg, New York, Hong Kong, London, Milan, Paris, Tokyo, 2002.

[BFM+06] Ralph Bergmann, Andrea Freßmann, Kerstin Maximini, Rainer Maximini, and Thomas Sauer. Case-based support for collaborative business. In *Proceedings of the 8th European Conference on Case-Based Reasoning (ECCBR 2006)*, Ölüdeniz/Fethiye, Turkey, September 2006.

[Boe02] Barry Boehm. Get ready for the agile methods, with care. *IEEE Computer*, 35(1):64–69, 2002.

[Boo06] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison Wesley, 3rd edition, 2006.

[BOP+05] Janette Birkner, Maryja Ognyanova, Jacques Pütz, Annabelle Rücker, and Yangshen Yao. Workflow-Unterstützung für Geoinformation. Technical report, Universität Trier, September 2005. Endbericht.

[CAK04] Andrea Freßmann, Kerstin Maximini, Rainer Maximini, and Thomas Sauer. *Anforderungen aus verschiedenen Forschungsprojekten und*

Dissertationen. University of Trier, Department of Business Information Systems II, 54286 Trier, 2004.

[CAK05] Rainer Maximini. *CAKE Developer Manual*. University of Trier, Department of Business Information Systems II, 54286 Trier, 2005.

[CAK06] Andrea Freßmann, Kerstin Maximini, Rainer Maximini, Mirjam Minor, and Daniel Schmalen. *CAKE System Documentation*. University of Trier, Department of Business Information Systems II, 54286 Trier, 2006.

[Cor03] Daniel D. Corkill. Collaborating software: Blackboard and multi-agent systems & the future. In *Proceedings of ILC 03 International Lisp Conference*, New York, 2003.

[CS03] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work: New challenges to systems design. In Kenji Itoh, editor, *Handbook of Human Factors/Ergonomics*, pages 619–636. Asakura Publishing, Tokyo, 2003.

[FBT+07] Andrea Freßmann, Ralph Bergmann, Brian Taylor, Ben Diamond, and Gary Carr-Smith. Mobile knowledge management support in fire service organisations. In *Tagungsband der 8. Internationalen Tagung der Wirtschaftsinformatik*. Karlsruhe University Press (accepted to be published), 2007.

[FMMS05a] Andrea Freßmann, Kerstin Maximini, Rainer Maximini, and Thomas Sauer. CBR-based execution and planning support for collaborative workflows. In *Workshop "Similarities - Processes - Workflows" on the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, Chicago, Illinois (USA), August 2005.

[FMMS05b] Andrea Freßmann, Kerstin Maximini, Rainer Maximini, and Thomas Sauer. Collaborative agent-based knowledge support for empirical and knowledge-intense processes. In Torsten Eymann, Franziska Klügl, Winfried Lamersdorf, Matthias Klusch, and Michael Huhns, editors, *Proceedings of the 3rd German Conference on Multi-agent System Technologies (MATES 2005)*, volume 3550 of *LNAI*, pages 235–236, Koblenz, Germany, September 2005. Springer-Verlag.

[FMS05] Andrea Freßmann, Rainer Maximini, and Thomas Sauer. Towards collaborative agent-based knowledge support for time-critical and

business-critical processes. In Klaus-Dieter Althoff, Andreas Dengel, Ralph Bergmann, Markus Nick, and Thomas Roth-Berghofer, editors, *Professional Knowledge Management*, volume 3782 of *LNAI*, pages 421–430, Kaiserslautern, Germany, April 2005. Springer-Verlag.

[FSB05] Andrea Freßmann, Thomas Sauer, and Ralph Bergmann. Collaboration patterns for adaptive software engineering processes. In H. Czap, R. Unland, C. Branki, and H. Tianfield, editors, *Self-Organization and Autonomic Informatics (I)*, volume 135, pages 304–312, Amsterdam, Netherlands, November 2005. Frontiers in Artificial Intelligence and Applications (FAIA), IOS Press.

[Galileo] http://www.galileocomputing.de/glossar. Galileo Compution – Glossar.
Verifiziert im November 2006.

[Gru94] Jonathan Grudin. Computer-supported cooperative work: Its history and participation. *IEEE Computer*, 27(5):19–26, 1994.

[IBM-WS] http://www-306.ibm.com/software/de/websphere/ index.html. IBM Deutschland GmbH.
Verifiziert im November 2006.

[Kol83a] Janet L. Kolodner. Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science*, 7:243–280, 1983.

[Kol83b] Janet L. Kolodner. Reconstructive Memory : A Computer Model. *Cognitive Science*, 7:281–328, 1983.

[LSDN01] Kecheng Liu, Lily Sun, Alan Dix, and Mohan Narasipuram. Norm-based agency for designing collaborative information systems. *Information Systems Journal*, 11:229–247, 2001.

[Max06] Rainer Maximini. *Advanced Techniques for Complex Case-Based Reasoning Applications*. Dissertation, Fachbereich IV der Universität Trier, April 2006.

[Mey97] Bertrand Meyer. *Object-oriented Software Construction*. Sams, 2nd bk&cd edition, 1997.

[MFS+05] Rainer Maximini, Andrea Freßmann, Thomas Sauer, Kerstin Maximini, and Ralph Bergmann. CAKE - Collaborative Agent-based

Knowledge Engine. Technical report, University of Trier, Department of Business Information Systems II, July 2005.

[Ric95] Michael M. Richter. The knowledge contained in similarity measures. Invited Talk on the International Conference on Case-Based Reasoning 1995 (ICCBR-95) in Sesimbra, Portugal, 1995.

[rjm] http://www.rjm.de/cms/. rjm Medienservice GmbH und rjm Business Solutions GmbH.
Verifiziert im November 2006.

[SAP-NW] http://www.sap.com/germany/plattform/netweaver/ index.epx. SAP Deutschland AG & Co. KG.
Verifiziert im November 2006.

[Sch82] Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, 1982.

[SM06] Thomas Sauer and Kerstin Maximini. Using workflow context for automated enactment state tracking. In Mirjam Minor, editor, *Workshop Proceedings: 8th European Conference on Case-Based Reasoning, Ölüdeniz/Fethiye, Turkey, Workshop: Case-based Reasoning and Context Awareness (CACOA 2006)*, pages 300–314. Universität Trier, Department of Business Information Systems II, September 2006.

[SMMB06] Thomas Sauer, Kerstin Maximini, Rainer Maximini, and Ralph Bergmann. Supporting collaborative business through integration of knowledge distribution and agile process management. In *Multikonferenz Wirtschaftsinformatik (MKWI 2006), Teilkonferenz Collaborative Business*, 2006.

[SSU01] Gerhard Schwabe, Norbert A. Streitz, and Rainer Unland. *CSCW-Kompendium: Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten*. Springer Berlin, Heidelberg, New York, Hongkong, London, Mailand, Paris, Singapur, Tokio, 2001.

[vEDA04] L. van Elst, V. Dignum, and A. Abecker. Towards agent-mediated knowledge management. In *Agent-Mediated Knowledge Management, International Symposium AMKM 2003, March 24-26, 2003, Revised and Invited Papers*, pages 1–30, Stanford CA, USA, 2004. Springer-Verlag.

[Wiki] http://www.wikipedia.de. Wikipedia – die freie Enzyklopädie. Verifiziert im August 2005.

Index