

Komponentenbasierte Anpaßbarkeit von Groupware

Oliver Stiemerling

Institut für Informatik III, Universität Bonn

Zusammenfassung

Aufgrund der Dynamik und Verschiedenartigkeit von Anforderungen in kooperativen Arbeitsumgebungen ist Anpaßbarkeit eine zentrale Eigenschaft von Groupwaresystemen. Dabei besteht ein Spannungsfeld zwischen der Mächtigkeit der Anpassungsmechanismen und ihrer einfachen und effizienten Anwendbarkeit. In diesem Beitrag wird der Ansatz der komponentenbasierten Anpaßbarkeit vorgestellt. Der grundlegende Gedanke dieses Ansatzes ist, durch hierarchisch geschachtelte Softwarekomponenten eine beliebig skalierbare Flexibilität zu schaffen. Eine Anwendung kann so auf verschiedenen Ebenen von Abstraktion und Komplexität nicht nur betrachtet sondern auch effektiv manipuliert werden. Es werden eine Reihe von Fragestellungen diskutiert, die sich aus dem Ansatz ergeben und eine Implementation einer Laufzeit- und Anpassungsumgebung vorgestellt, die auf Java und dem JavaBeans-Komponentenmodell basiert. Diese Umgebung wurde im POLiTeam-Projekt benutzt, um das Suchwerkzeug im eingesetzten Groupwaresystem LINKWORKS anpaßbar zu gestalten.

1 Einleitung

Anpaßbarkeit spielt eine wesentliche Rolle bei der Produktentwicklung für große Marktsegmente [Henderson und Kyng 91], der menschengerechten und sozialverträglichen Gestaltung von Software [Oppermann 89; Friedrich 90] und der Entwicklung komplexer Systeme [Aksit et al. 96]. Insbesondere Gruppenarbeitssysteme sollten aufgrund der erhöhten Bedeutung der Dynamik und Verschiedenartigkeit des Anwendungskontextes anpaßbar sein [Malone et al. 95; Oberquelle 94].

Die Motivation für die Entwicklung der komponentenbasierten Anpaßbarkeit ist durch Erfahrungen aus dem POLITeam Projekt [Klöckner et al. 95] begründet, in dem ein Gruppenarbeitssystem bei verschiedenen Stellen der öffentlichen Verwaltung und einem Autohersteller eingeführt wird. Die unterschiedlichen und wechselnden Anforderungen auf der Ebene von Organisation, Gruppe und Individuum haben die Anpaßbarkeit des Systems zu einem wichtigen Erfolgskriterium gemacht. Die im Projekt gewonnenen Erfahrungen und die einschlägige Literatur zeigen folgende Anforderungen auf:

- Anpassungen sollten auf einer Vielzahl von Ebenen und Mächtigkeitsgraden möglich sein, um unterschiedliche Rollen und Fähigkeiten zu unterstützen (z.B. Endbenutzer, Systemadministratoren und Entwickler). Dabei sollte der Übergang auf die nächst mächtigere Ebene mit möglichst geringem Qualifizierungsaufwand verbunden sein [Bentley & Dourish 95].
- Anpassung ist häufig eine kooperative Aktivität, bei der neben erfahreneren Kollegen und der EDV-Abteilung auch externe Berater und Entwickler mitwirken [Mackay 90]. Auf der Systemebene sollte das dadurch berücksichtigt werden, daß Anpassungen verteilbar, archivierbar und automatisch auf Verträglichkeit mit dem aktuellen System überprüfbar sind.
- Besonders bei Mehrbenutzersystemen wie z.B. Gruppenarbeitssystemen sollten Anpassungen im laufenden Betrieb durchführbar sein, um möglichst wenig Störungen zu verursachen. Außerdem sollte das Laufzeitverhalten nicht schlechter als bei starr implementierten Systemen sein.

Die Herausforderung bei der Gestaltung von anpaßbaren Systemen liegt darin, daß die Implementation des Systems den oben formulierten zusätzlichen Anforderungen unterliegt. In herkömmlichen Systemen kann die Implementation von den Entwicklern vor allen anderen Beteiligten „versteckt“ werden, die mit dem System nur über *Nutzungsschnittstellen* interagieren. In anpaßbaren Systemen muß ein ausgesuchter Teil der Implementation offengelegt werden (vgl. *open implementation*, [Kiczales 96]). Der offene Teil der Implementation stellt die *Anpassungsschnittstelle* dar. Dabei besteht die Anpassungsschnittstelle aus den offengelegten Implementationskonzepten (z.B. dem Objektmodell des Systems) und den Manipulationsmöglichkeiten (Transformationen oder Erweiterungen der bisherigen Implementation, z.B. Parametrisierung oder Änderungen am Source-Code, vgl. [Mørch 97] oder [Henderson & Kyng 91]). Gegenstand dieses Beitrags ist die Gestaltung der Anpassungsschnittstelle basierend auf der hierarchischen Dekomposition einer Anwendung in einzelne Komponenten.

Im nächsten Abschnitt werden drei auf Anpaßbarkeit abzielende Gruppenarbeitssysteme aus der aktuellen Literatur im Hinblick auf die oben formulierten Anforderungen disku-

tiert. Dann wird der Ansatz der komponentenbasierten Anpaßbarkeit vorgestellt. Im Rahmen dieses Ansatzes werden eine Reihe von Fragestellungen aufgeworfen und ihre Beantwortung in der Implementation einer generischen Laufzeit- und Anpassungsumgebung vorgestellt (Abschnitt 3). Im vierten Abschnitt wird die Anwendung des Ansatzes auf das Suchwerkzeug im POLiTeam Projekt beschrieben. Die dabei gewonnenen Erfahrungen werden im fünften Abschnitt evaluiert und es wird ein Ausblick auf zukünftige Arbeiten gegeben.

2 Stand der Forschung

In der CSCW-Literatur werden eine Reihe von Forschungssystemen vorgestellt, die mit dem Ziel der Anpaßbarkeit entwickelt wurden. In diesem Abschnitt werden kurz drei Systeme diskutiert, die unterschiedliche Ansätze repräsentieren.

OVAL von [Malone et al. 95] ist ein Groupware-„Baukasten“, aus dessen vier Primitiven *Objects*, *Views*, *Agents* und *Links* kooperationsunterstützende Systeme zusammengebaut werden können. Die Autoren beschreiben, wie sie mit diesen vier Elementen eine Reihe von existierenden Groupwaresystemen (u.a. spezielle LOTUSNOTES-Applikationen) implementiert haben.

Das System PROSPERO von [Dourish 96], das auf der Basis einer reflektiven Programmiersprache (CLOS, vgl. [Kiczales et al. 91]) ein hochgradig anpaßbares CSCW-Toolkit implementiert, ist im Gegensatz zu OVAL ein Beispiel für einen Ansatz, der mehr auf Groupware-Entwickler als auf weniger technisch qualifizierte Benutzer ausgerichtet ist.

ARIADNE von [Simone und Schmidt 98] ist ein recht abstraktes Modell eines anpaßbaren Groupwaresystems. Es besteht aus drei Schichten: der α -, β - und γ -Schicht. Die letztere definiert dabei die grundlegende „Grammatik“, d.h. die Ausdrucksstärke der Sprache, mit der auf der β -Schicht Koordinationsmechanismen definiert werden können. Die α -Schicht beschreibt das laufende System, d.h. eine Menge von Instanzen von Koordinierungsmechanismen.

Alle drei Systeme haben den Nachteil, daß die jeweilige Anpassungssprache im Mächtigkeits- und Abstraktionsgrad nicht beliebig skalierbar ist (vgl. erste Anforderung). Reicht beispielsweise die Funktionalität der vier Primitiven in OVAL nicht aus, so muß auf Systemebene programmiert werden, was im praktischen Betrieb ein erhebliches Handicap darstellt. PROSPERO, dessen Anpaßbarkeit a priori nicht für Endanwender konzeptioniert ist, greift auf CLOS zurück, wenn es darum geht, die vorhandene Groupwarefunktionalität an spezielle Anforderungen anzupassen. In ARIADNE ist eine einfache Skalierung vorstellbar, wobei zum Beispiel die Entwickler eine Reihe von Sprachen auf der γ -Schicht definieren, die von Systemadministratoren (auf der β -Schicht) genutzt werden können, um Koordinationsmechanismen zu definieren, die von Endbenutzern auf der α -Schicht je nach Bedarf instanziiert werden. Die Anzahl der Schichten ist jedoch fest und der Übergang zur nächst tieferen Schicht ist offensichtlich mit einem erheblichen Qualifizierungsaufwand verbunden. Die beiden anderen Anforderungen (Unterstützung von kooperativen Anpassungen und Anpassungen zur Laufzeit) werden von den vorgestellten Systemen (bis auf die einfache Austauschbarkeit von Spezifikationen der β -Schicht bei ARIADNE) nicht

berücksichtigt. Gegenstand der vorgestellten Arbeit ist die Entwicklung eines Ansatzes zur Anpaßbarkeit, der die gestellten Anforderungen erfüllt und insbesondere eine fein skalierbare Flexibilität erlaubt.

3 Komponentenbasierte Anpaßbarkeit

Der Ansatz besteht darin, das bisher hauptsächlich in der Entwicklungsphase angewandte Konzept der *Softwarekomponente* in die Nutzungsphase zu übertragen und so im Bezug auf die formulierten Anforderungen adäquate Anpassungsmöglichkeiten zu schaffen. Dabei wird eine Softwarekomponente als ein prinzipiell unabhängiger Teil eines (laufenden) Softwaresystems angesehen, dessen Abhängigkeiten vom Rest des Systems explizit als Schnittstellen spezifiziert sind. Softwarekomponenten können geschachtelt sein, d.h. selber wieder aus anderen Komponenten bestehen.

Die Möglichkeit der Schachtelung von Komponenten erlaubt die Betrachtung und Anpassung des Systems auf unterschiedlichen Ebenen (vgl. erste Anforderung). Vorstellbar sind anwendungsnahe Komponenten (z.B. eine Buchhaltungskomponente), die sich auf einer tieferen Ebene aus systemnahen Komponenten (z.B. TCP/IP-Komponenten) zusammensetzen. Die Schachtelungstiefe kann dabei (im Gegensatz zu ARIADNE) beliebig gewählt werden. Ein solcher Systemaufbau erlaubt dem Anpassenden einen abgestuften Zugang zum System, bei dem der Übergang von einer Ebene zur anderen lediglich das Verständnis einiger neuer Komponenten voraussetzt, während Komposition als grundlegendes Strukturprinzip erhalten bleibt. Ein weiterer Vorteil der Schachtelbarkeit ist die Möglichkeit, das System hierarchisch so zu dekomponieren, daß Anpassungen, die eine große Zahl von atomaren Komponenten betreffen, auf einer höheren Ebene einfach, d.h. mit wenigen Transformationen spezifiziert werden können.

Die zweite Anforderung wird durch den Komponentenansatz ebenfalls unterstützt, da industrielle Standards für Softwarekomponenten mit der Zielsetzung der einfachen Wiederverwendbarkeit (speziell durch Dritte) konzipiert werden. Aus diesem Grund sind Archivformate und Selbstbeschreibungsfähigkeit in einigen Standards bereits berücksichtigt, was einfache Verteilbarkeit und automatische Verträglichkeitsprüfung möglich macht.

Einige Modelle (z.B. das JavaBeans Komponentenmodell, siehe JavaSoft 1997) erhalten die durch die Softwarekomponenten gegebene Struktur auch noch während der Laufzeit, so daß dynamische Änderungen ohne größere Performanzverluste unterstützt werden (vgl. dritte Anforderung).

Die Benutzung von geschachtelten Softwarekomponenten zur anpaßbaren Gestaltung von Groupwareanwendungen macht es erforderlich, eine Reihe von Fragen neu zu überdenken oder aufzuwerfen. Die Softwareentwicklung mit Komponenten zielt auf andere Benutzergruppen ab, findet unter anderen Umständen statt und macht zumeist die während der Entwicklung (im Source-Code) vorhandene modulare Struktur in der ausgelieferten – monolithischen – Anwendung wieder zunichte. Insbesondere der letzte Punkt legt die Frage nach einer integrierten Laufzeit- und Anpassungsumgebung nahe.

3.1 Systemarchitektur einer Laufzeit- und Anpassungsumgebung

An eine solche Umgebung wird die Anforderung gestellt, daß sie die komponentenbasierte Struktur der Anwendung während der Laufzeit in manipulierbarer Form erhält. Es müssen grundlegende Transformationsoperationen wie zum Beispiel das dynamische Instanzieren einer neuen Komponente, das Verbinden von Komponenten, das Lösen von Verbindungen und das Entfernen von Komponenten unterstützt werden. Außerdem muß die Beschreibung der hierarchischen Struktur einer Anwendung nachgehalten werden. Dazu wird eine *ADL* (*Architecture Description Language*, vgl. [Garlan & Perry 95]) benötigt, die von der Umgebung beim Start ausgewertet wird und nach deren Spezifikation die Anwendung aus primitiven Komponenten „zusammengebaut“ wird. Wir benutzen dazu eine Abwandlung von DARWIN, einer Sprache zur Beschreibung der Architektur verteilter Systeme (vgl. [Magee et al. 95]). Unsere Abwandlung heißt *CAT* (*Component Architecture for Tailorability*, beschrieben in [Stiemerling 97]) und erlaubt die hierarchische Komposition einer Anwendung auf der Basis von primitiven, in einer beliebigen Sprache (hier Java) implementierten Komponenten. In Abbildung 1 ist schematisch die Architektur der Laufzeit- und Anpassungsumgebung dargestellt.

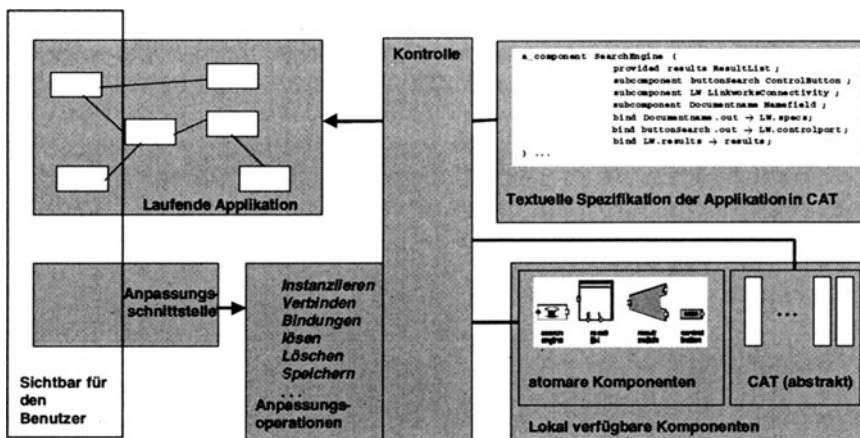


Abbildung 1: Architektur der integrierten Laufzeit- und Anpassungsumgebung

Oben links ist die eigentliche Anwendung abgebildet, wobei einige ihrer Komponenten für den Benutzer sichtbar und andere unsichtbar sind. Beim Starten des Systems wird eine *CAT-Datei* (oben rechts) ausgewertet, indem die hierarchische Spezifikation der Anwendung top-down durchlaufen wird, wobei die unterste Ebene aus primitiven Komponenten besteht, die nicht in CAT spezifiziert sind. Die primitiven (oder atomaren) Komponenten sind in Java als JavaBeans definiert und werden in einem lokalen *Komponentenpool* (unten rechts) vorgehalten. Die Auswertung der CAT-Datei ist abgeschlossen, wenn alle Komponenten der Anwendung entsprechend der CAT-Spezifikation instanziiert und verbunden sind. Das *Kontrollmodul* der Umgebung (in der Mitte) stellt jetzt eine Reihe von einfachen Transformationsoperationen zur Verfügung, mit deren Hilfe die Anwendung

auf allen Ebenen der Komponentenhierarchie dynamisch verändert werden kann. Es ist beispielsweise möglich, Teile der Anwendung als CAT-Dateien im Komponentenpool (unten rechts) abzuspeichern. Auf diese Weise werden verschiedene neu konstruierte Alternativen nachgehalten, die bei Bedarf in der laufenden Anwendung ausgetauscht werden können.

Die bisher beschriebenen Elemente des Systems machen die Anpaßbarkeit auf technischer Ebene aus. Nun stellt sich die Frage, wie die Anpassung durchgeführt wird.

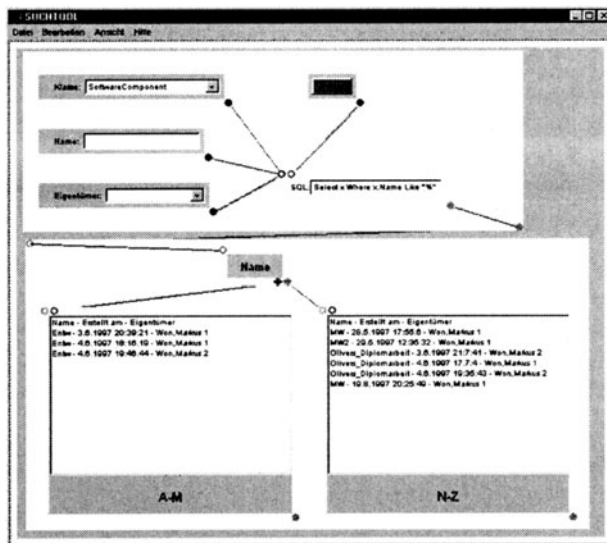


Abbildung 2: Prototyp einer graphischen Anpassungsschnittstelle

3.2 Benutzerschnittstellen für Anpassungen

Im POLiTeam Projekt wurden insbesondere Ansätze der *Adaptierbarkeit* untersucht, im Gegensatz zu Ansätzen der automatischen Anpassung (*Adaptivität*, vgl. [Kühme et al. 92]). Letztere sind nur dann sinnvoll, wenn eine Sensorik für wechselnde Anforderungen vorhanden ist und diese in direkten Zusammenhang mit einer Anwendungsalternative gestellt werden können. Bei den von uns untersuchten Beispielen (vgl. Suchwerkzeug in Abschnitt 4) waren diese Voraussetzungen nicht erfüllt. Bei der Anpassung durch die Benutzer sind im Rahmen unseres Ansatzes eine Reihe von Alternativen vorstellbar:

- Der anpassende Benutzer manipuliert die *textuelle* Repräsentation der Anwendung im CAT-Format. Diese Alternative erschien uns aufgrund der Möglichkeit von Syntaxfehlern als zu kompliziert.
- Die hierarchische Struktur der Anwendung wird *graphisch* dargestellt und ist mit Hilfe von einfachen Mausoperationen manipulierbar. Diese Alternative wurde im-

plementiert. Abbildung 2 zeigt die Benutzerschnittstelle, die in Anlehnung an Verfahren der visuellen Programmierung (vgl. [Myers 90] oder [Nardi 93]) gestaltet wurde.

- Alternativen für verschiedene Aspekte der Applikation werden aus einer *Auswahl-liste* in ein Formular eingetragen. Diese Möglichkeit des *form-based end-user programming* wird als besonders einfach angesehen (z.B. [Nardi 93, S. 66]) und erscheint uns geeignet für kooperative Anpassungsmodelle, wobei ein höher qualifizierter Benutzer (zum Beispiel ein Systemadministrator) für weniger qualifizierte Kollegen Alternativen konstruiert, die diese durch einfache Auswahl von Alternativen (vgl. [Henderson & Kyng 91]) nutzen können (noch nicht implementiert).

Abbildung 2 stellt das als Beispiel implementierte Suchtool (eine Erklärung der Semantik des einzelnen Komponenten erfolgt in Abschnitt 4) auf der untersten Ebene der Hierarchie dar. Anhand der zwei hellen Umrahmungen läßt sich die hierarchische Struktur der Anwendung erkennen. Es ist möglich, den graphischen Editor auf unterschiedliche Ebenen der Hierarchie einzustellen und so unterschiedlich komplexe Anpassungen vorzunehmen.

3.3 Die hierarchische Dekomposition einer Anwendung

Im Entwicklungsprozeß komponentenbasierter anpaßbarer Software muß auf der Basis des vorhandenen Wissens um Dynamik und Verschiedenartigkeit von Anforderungen eine angemessene hierarchische Dekomposition der Anwendung gefunden werden. Dieses Wissen kann offensichtlich nur durch heuristische Methoden und nur für eine bestimmte Anwendung gewonnen werden (vgl. [Stiemerling et al. 97]). Ein Vorteil des komponentenbasierten Ansatzes ist, daß anwendungsspezifische Designentscheidungen bezüglich der Anpaßbarkeit lediglich in die Dekomposition der Anwendung eingehen, wobei die Transformationen (wie z.B. Instanzieren und Verbinden) und damit auch die Laufzeit- und Anpassungsumgebung im Idealfall generisch, d.h. auf ganz unterschiedliche Probleme anwendbar sind.

Durch die Dekomposition wird das „Vokabular“ der Anpassungssprache auf den verschiedenen Ebenen der Hierarchie definiert. Unter Berücksichtigung des Wissens über Dynamik und Verschiedenartigkeit von Anforderungen kommen folgende qualitative Dekompositionskriterien (vgl. [Parnas 72]) zum Tragen:

- Auf hohen Ebenen der Hierarchie (hoch im Bezug auf Nähe zur Wurzel der Hierarchie) sollten die abstrakten Komponenten *anwendungsnahe und dem Benutzer bekannte Konzepte* reflektieren. In Abbildung 2 ist zu sehen, daß eine abstrakte Komponente (unten) für die Darstellung der Suchergebnisse verantwortlich ist. Diese Dekomposition wurde gewählt, weil die POLITeam-Benutzer häufig eine andere Darstellung der Suchergebnisse in Abhängigkeit von ihrer momentanen Arbeitsaufgabe forderten. Durch die Zusammenfassung aller für die Darstellung verantwortlichen primitiven Komponenten ist ein einfaches Austauschen der gesamten Darstellungskomponente auf einer dem normalen Benutzer verständlichen Ebene möglich.
- Die unteren Ebenen der Hierarchie sollten *systemnahen Konzepten* vorbehalten bleiben, wie zum Beispiel der Suchmaschine (die mit „SQL“ bezeichnete Kompo-

nente in Abbildung 2), die die Verbindung zur Datenbank herstellt und die eigentliche Abfrage zusammenstellt und durchführt. Anpassungen auf dieser Ebenen sind eher für Systemadministratoren oder qualifiziertere Benutzer vorstellbar.

- Die *Granularität der atomaren Komponenten* der untersten Ebenen bestimmt die Mächtigkeit der Anpassungssprache. Sie sollte offensichtlich so gewählt sein, daß alle antizipierten (oder extrapolierten) Anforderungsalternativen unterstützt werden können. Es gibt Ansätze der visuellen Programmierung, die generische Komponenten sehr feiner Granularität mit – in Komposition – der Mächtigkeit einer klassischen Programmiersprache wie PASCAL zur Verfügung stellen (vgl. [Myers 90]). Dabei ist allerdings fraglich, ob eine so feine Granularität bei dynamisch anpaßbaren Systemen nicht zu Laufzeiteinbußen führt. In Rahmen dieses Ansatzes halten wir Komponenten auf der untersten Ebenen für sinnvoll, die hauptsächlich anwendungsspezifische (z.B. Suchwerkzeug-) Aufgaben erfüllen. Grundlegende Schnittstellenelemente wie Schaltflächen stellen dabei eine Ausnahme dar.

An dieser Stelle sei noch einmal ausdrücklich darauf hingewiesen, daß die angemessene Dekomposition einer Anwendung aufgrund von Unsicherheit und offensichtlich unvollständigem Wissen über zukünftige Anforderungen mehr eine Kunst als eine Methode darstellt. Jedoch hat notwendigerweise jeder Ansatz zur Anpaßbarkeit mit dieser Natur des Problems zu kämpfen. Der Vorteil des komponentenbasierten Ansatzes ist, daß bei Irrtümern im „künstlerischen“ Teil des Entwicklungsprozesses lediglich die Dekomposition (eventuell sogar nur einzelne Komponenten) betroffen ist, während die Anpassungsmechanismen und Schnittstellen unverändert bleiben.

4 Anwendung des Ansatzes auf das Suchwerkzeug

Die Benutzer in den POLITeam-Anwendungsfeldern müssen häufig nach Dokumenten in gemeinsam benutzten Arbeitsbereichen und in den umfangreichen Dokumentenbeständen des Systems suchen. Für diese Aufgabe stellt das im POLITeam-Projekt verwendete Groupwaresystem LinkWorks ein Suchwerkzeug zur Verfügung.

Das Suchwerkzeug besteht im wesentlichen aus zwei Fenstern. In einem Fenster spezifiziert der Benutzer die gesuchten Dokumente und erhält nach dem Suchvorgang im anderen Fenster eine Liste der gefundenen Dokumente. Das Suchwerkzeug ist eigentlich eine sehr einfache Anwendung, verdeutlicht aber gerade dadurch die Notwendigkeit der Anpaßbarkeit von Groupware. Obwohl das Suchwerkzeug alleine nicht als eigenständige Groupware gelten kann, so stammt doch ein wesentlicher Teil der Anforderungen an die Anpaßbarkeit dieser Software von ihrem Einsatz als Teil des POLITeam-Systems. Denn schon kurze Zeit nach Projektbeginn wurde deutlich, daß das ursprüngliche Suchwerkzeug den Anforderungen der POLITeam-Anwendungsfelder nicht entsprach. Zum einen war die Benutzerschnittstelle viel zu komplex (es wurden alle Suchmöglichkeiten in einem Fenster dargestellt). In den Anwendungsfeldern wurde immer nur nach einer kleinen Teilmenge von Kriterien gesucht (Name des Dokuments und Erstellungsdatum). Außerdem verletzte das bestehende Suchtool die Privatheit der Anwender, indem es Suchen auf den (virtuellen) Schreibtischen anderer Kollegen erlaubte. Weiterhin war die Übernahme der Ergebnisse unbefriedigend, da statt einer Kopie des gefundenen Dokuments ein Ver-

weis erzeugt wurde, so daß Veränderungen am gefundenen Dokument gleichzeitig auch andere Kollegen betreffen konnten, was zu erheblichen Konflikten führte. Auch äußerten Benutzer in Workshops und Interviews unterschiedliche Anforderungen bezüglich der Darstellung der Suchergebnisse. Konkret wurde eine Gruppierung der Ergebnisse nach unterschiedlichen Kriterien (Datum oder Fundort) gewünscht.

Für dieses einfache Beispiel ließen sich bestimmt auch noch eine Reihe anderer Anpassungskonzepte finden, aber in diesem Beitrag geht es gerade darum, die Möglichkeiten und Probleme der komponentenbasierten Anpaßbarkeit an einem übersichtlichen Beispiel zu diskutieren.

4.1 Dekomposition des Suchwerkzeugs in atomare Komponenten

Basierend auf den beschriebenen Anforderungen und im Hinblick auf die vorher formulierten Kriterien zur Dekomposition wurde das Suchwerkzeug in eine Reihe in Java implementierter Komponenten zerlegt. In Abbildung 2 sind die wesentlichen Arten von Komponenten zu sehen. Im oberen Teil gibt es eine Reihe von *visuellen Komponenten*, die die Spezifikation der gesuchten Dokumente z.B. über den Namen erlauben. Die *Suchmaschinenkomponente* (mit „SQL“ markiert), die die Verbindung zu LinkWorks herstellt, ist während der normalen Nutzung hingegen unsichtbar. Am Ausgang der Suchkomponente stehen nach Beendigung der Suche die Ergebnisse zur Verfügung. In Abbildung 2 werden die Ergebnisse durch eine (normalerweise unsichtbare) *Weichenkomponente* nach dem Namen aufgespalten und in zwei *Listenkomponenten* abgebildet. Die Ergebnisübernahme erfolgt durch spezielle *Schaltflächen*, die in Abbildung 2 nicht zu sehen sind, aber an die Ergebnislisten angehängt werden können. Durch Wahl unterschiedlicher Schaltflächen kann so beispielsweise bestimmt werden, ob die Ergebnisse als Verweis oder als Kopie übernommen werden sollen. Der vollständige Satz von Komponenten (engl.: *Framework*) steht dem Benutzer während des Anpassens in einem Werkzeugkasten zur Verfügung.

Mit Hilfe dieser Komponenten können alle von den POLIteam-Benutzern geforderten unterschiedlichen Anzeigarten, Ergebnisübernahmen und Suchspezifikationen konstruiert werden, wobei besonders häufig durchgeführte Anpassungen, wie zum Beispiel der Wechsel zu einer anderen abstrakten Darstellungskomponente durch einfachen Austausch einer Komponente auf einer hohen Ebenen möglich sind. Neue Darstellungskomponenten können beispielsweise von erfahrenen Kollegen oder Systemadministratoren auf einer tieferen Ebenen konstruiert werden. Obwohl das Suchwerkzeug ein recht einfaches Anwendungsbeispiel ist, zeigt sich schon hier der Nutzen einer hierarchischen Schachtelung. Die abstrakten Darstellungskomponenten können als separate CAT-Datei gespeichert (vgl. Abbildung 1, unten rechts) und beispielsweise per Email oder über einen gemeinsamen Arbeitsbereich verteilt werden. Zudem ist es möglich, das Suchwerkzeug während der Laufzeit anzupassen. Im Anpassungsmodus (siehe Abbildung 2) sind die Komponenten weiterhin voll funktionstauglich.

5 Bisherige Ergebnisse und zukünftige Arbeiten

Auf der *softwaretechnischen Ebene* hat sich gezeigt, daß die Benutzung des JavaBeans-Standards eine angemessene Basis für den Ansatz darstellt. Es wurde deutlich, daß durch das flexible Ereignismodell (Ereignisse sind Objekte und können so beliebige Daten transferieren) eine Vielfalt von Interaktionsformen auch zwischen unsichtbaren Komponenten (z.B.: Suchmaschine und Weiche) unterstützt wird. Durch Java-RMI (*Remote Method Invocation*) können JavaBeans auf verschiedenen Rechnern Ereignisse senden und empfangen (beim Suchwerkzeug wurden noch TCP/IP-Sockets verwendet). Verteilte Komponenten sind insbesondere für Groupware von zentraler Bedeutung. Da im Suchwerkzeug lediglich ein einfaches Client/Servermodell unterstützt werden mußte, ist als nächster Schritt geplant, ein erweitertes Verteilungsmodell zu entwickeln, das eine beliebige Verteilung von hierarchisch geschachtelten Komponenten in einem Rechnernetz erlaubt.

Auf der *ergonomischen Ebene* weist der Ansatz bisher noch ein prinzipielles geometrisches Problem auf. Die Existenz von vielen unsichtbaren Komponenten führt im normalen Benutzungsmodus dazu, daß auch bei geschickter Verteilung und Skalierung immer noch leere Stellen auf der graphischen Oberfläche zu sehen sind. Zur Lösung dieses Problems entwickeln wir im Moment einen Ansatz der auf der Anwendung von Methoden der 3-D Modellierung basiert. Dabei werden die sichtbaren Komponenten auf einer Ebene im Raum (dem "sichtbaren" Fenster) plziert. Die unsichtbaren Komponenten können im Raum "hinter" der Ebene verteilt werden. Der Benutzer kann sich dann mit Hilfe eines VRML-fähigen Browsers frei durch die Anwendung bewegen und entsprechende Anpassungen vornehmen.

Von zentraler Bedeutung für den Ansatz ist die *angemessene Dekomposition* der anpaßbar zu gestaltenden Anwendung. Wir haben bei der Zerlegung des Suchwerkzeugs die Erfahrung gemacht, daß der Designschritt von der in Interviews und Workshops gewonnen Empirie zu einer sinnvollen Zerlegung eine übersichtliche Darstellung der verschiedenartigen Anforderungen und deren antizipierten Dynamik erfordert. Bisherige Vorgehensweisen der Softwareentwicklung basieren auf der Eliminierung von Alternativen und sind aus diesem Grund ungeeignet, um die erforderliche Anpaßbarkeit zu modellieren. Lediglich in der *change case* Erweiterung [Ecklund et al. 96] der *use case* Methodik [Jacobsen et al. 92] können antizipierte Veränderungen explizit dargestellt werden. An dieser Stelle besteht noch Forschungsbedarf. Auch müssen Maße zur genaueren Definition des Begriffs "angemessene Dekomposition" entwickelt werden.

Ein weiteres Problem tritt auf, wenn eine gewählte Dekomposition geändert werden muß und Komponenten von Drittanbietern Teil des Frameworks sind. Da solche Komponenten in der Regel nur im Binärformat, d.h. ohne Source-Code vorliegen, sind nach dem heutigen Stand der Technik Änderungen der Implementierung zumeist unmöglich. Auf der Ebene der Programmiersprache könnten flexiblere Konzepte wie *Delegation* (vgl. z.B. [Kniesel 98]) einen Lösungsansatz darstellen.

Zusammenfassend kann gesagt werden, daß der Ansatz die in der Einleitung formulierten Anforderungen weitestgehend erfüllt. Seine Anwendung auf das POLITeam-Suchwerkzeug hat gezeigt, daß dynamische und verschiedenartige Anforderungen eines realen Anwendungsfeldes unterstützt werden können. Trotz der Einfachheit des Beispiels wurde der

Wert einer hierarchischen Schachtelung anhand des differenzierten Anpassungsmodells bezüglich der abstrakten Darstellungskomponente deutlich.

Danksagung

Der Prototyp wurde von Markus Won implementiert. Dank gebührt außerdem Armin B. Cremers und den Kollegen im Projektbereich Software-Ergonomie und CSCW am Institut für Informatik III für anregende Diskussionen und den anonymen Rezensenten für die wertvollen konstruktiven Hinweise.

Literatur

- [Aksit et al. 96] Aksit, M., Tekinerdogan, B., und Bergmans, L., "Achieving adaptability through separation and composition of concerns," in: *Special Issues in Object-Oriented Programming: Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP '96*, Linz, M. Mühlhäuser, Ed. Heidelberg: dpunkt Verlag, 1996, pp. 37-42.
- [Bentley und Dourish 95] Bentley, R. und Dourish, P., "Medium versus Mechanism: Supporting Collaboration through Customisation", in: *Proceedings of ECSCW 95*, H. Marmolin, Y. Sundblad, und K. Schmidt, Eds. Stockholm, Sweden: Kluwer, 1995, pp. 133-148.
- [Dourish 96] Dourish, P., "Open implementation und flexibility in CSCW toolkits," Ph.D. Thesis, University College London, 1996.
- [Ecklund et al. 96] Ecklund, E. F., Delcambre, L. M. L., und Freiling, M. J., "Change Cases: Use Cases that Identify Future Requirements", in: *Proceedings of OOPSLA '96*, CA, USA: ACM Press, 1996, pp. 342-358.
- [Garlan und Perry 95] Garlan, D. und Perry, D. E., "Introduction to the Special Issue on Software Architecture," *IEEE Transactions on Software Engineering*, vol. 21, pp. 269-274, 1995.
- [Henderson und Kyng 91] Henderson, A. und Kyng, M., "There's No Place Like Home: Continuing Design in Use," in: *Design At Work*, J. Greenbaum und M. Kyng, Eds. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 1991, pp. 219-240.
- [Jacobsen et al. 92] Jacobsen, I., Christerson, M., Jonsson, P., und Övergaard, G., *Object-Oriented Software Engineering. A Use Case Driven Approach*: ACM Press, 1992.
- [JavaSoft 97] JavaSoft, "JavaBeans 1.0 API Specification", . Mountain View, California: SUN Microsystems, 1997.
- [Kahler 95] Kahler, H., "From Taylorism to Tailorability: Supporting Organizations with Tailorable Software and Object-orientation", in: *Proceedings of HCI '95*, Y. Anzai, K. Ogawa, und H. Mori, Eds.: Elsevier, 1995, pp. 995-1000.
- [Kiczales 96] Kiczales, G., "Beyond the Black Box: Open Implementation," *IEEE Software*, vol. 13, 1996.
- [Kiczales et al. 91] Kiczales, G., Rivières, J. d., und Bobrow, D. G., *The Art of the Metaobject Protocol*: MIT Press, 1991.

- [Klößner et al. 95] Klößner, K., Mambrey, P., Solenkamp, M., Prinz, W., Fuchs, L., Kolvenbach, S., Pankoke-Babatz, U., und Syri, A., "POLITeam --- Bridging the Gap between Bonn und Berlin for und with the Users", in: Proceedings of ECSCW '95, H. Marmolin, Y. Sundblad, und K. Schmidt, Eds. Stockholm, Sweden: Kluwer, 1995, pp. 17-32.
- [Kniesel 98] Kniesel, G., "Type-safe delegation for dynamic component adaptation", in: Proceedings of ECOOP 98 Workshop on Component-Oriented Programming (WCOP 98), Brussels, Belgium, 1998.
- [Kühme et al. 92] Kühme, T., Dieterich, H., Malinowski, U., und Schneider-Hufschmidt, M., "Approaches to Adaptivity in User Interface Technology: Survey und Taxonomy", in: Proceedings of IFIP '92, 1992.
- [Mackay 90] Mackay, W. E., "Patterns of sharing customizable software", in: Proceedings of CSCW '90. Los Angeles, CA: ACM Press, 1990, pp. 209-221.
- [Magee et al. 95] Magee, J., Dulay, N., Eisenbach, S., und Kramer, J., "Specifying Distributed Software Architectures", in: Proceedings of 5th European Software Engineering Conference. Barcelona, 1995.
- [Malone et al. 95] Malone, T. W., Lai, K.-Y., und Fry, C., "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work," *ACM Transactions on Information Systems*, vol. 13, pp. 177-205, 1995.
- [Mørch 97] Mørch, A., "Method und Tools for Tailoring of Object-oriented Applications: An Evolving Artifacts Approach", PhD-Thesis, *Department of Computer Science*. Oslo: University of Oslo, 1997.
- [Myers 90] Myers, B. A., "Taxonomies of Visual Programming and Program Visualization," *Journal of Visual Languages und Computing*, 1, pp. 97-123, 1990.
- [Nardi 93] Nardi, B. A., *A Small Matter of Programming - Perspectives on End User Programming*. Cambridge, Massachusetts: The MIT Press, 1993.
- [Oberquelle 94] Oberquelle, H., "Situationsbedingte und benutzerorientierte Anpaßbarkeit von Groupware," in: *Menschengerechte Groupware - Software-ergonomische Gestaltung und partizipative Umsetzung*, A. Hartmann, T. Herrmann, M. Rhode, und V. Wulf, Eds. Stuttgart: Teubner, 1994, pp. 31-50.
- [Oppermann 89] Oppermann, R., "Individualisierte Systemnutzung," in: *GI - 19. Jahrestagung*, vol. 1, *GI Jahresberichte*, M. Paul, Ed.: Springer-Verlag, 1989, pp. 131-145.
- [Parnas 72] Parnas, D. L., "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, pp. 1053-1058, 1972.
- [Simone und Schmidt 98] Simone, C. und Schmidt, K., "Taking the distributed nature of cooperative work seriously", in: Proceedings of 6th Euromicro Workshop on Parallel und Distributed Processing. Madrid: IEEE-Press, 1998, pp. 295-301.
- [Stiemerling 97] Stiemerling, O., "CAT - Component Architecture for Tailorability," University of Bonn, Department of Computer Science, Bonn, Working Paper, 1997.
- [Stiemerling et al. 97] Stiemerling, O., Kahler, H., und Wulf, V., "How to Make Software Softer - Designing Tailorable Applications", in: Proceedings of DIS '97. Amsterdam: ACM Press, 1997.