Markus Won und Volker Wulf

Anpassungsumgebung für komponentenbasierte Software: Kooperativ und lernförderlich

A cooperative and easy-to-learn Tailoring Environment for component-based architectures

CSCW_Groupware_Anpassbarkeit_Lernförderlichkeit_Komponentenarchitekturen_Integritätsprüfung

Zusammenfassung. Der Aufsatz beschreibt eine komponentenbasierte Anpassungsumgebung, die Adaptionen der Software zur Laufzeit auf verschiedenen Komplexitätsebenen erlaubt. Die Anpassungsumgebung unterstützt auch den Austausch von angepassten Artefakten zwischen Nutzern. Erläutert wird die Anpassungsumgebung am Beispiel eines Suchwerkzeugs einer Groupware, das Benutzer entsprechend ihren Anforderungen in verschiedener Weise modifizieren können. Nutzungsprobleme ergaben sich bei der Einführung der Anpassungsumgebung in eine Organisation der öffentlichen Verwaltung sowohl bezüglich der eigenständigen Komposition von Suchwerkzeugen als auch bei von Kollegen übernommenen Suchwerkzeugen. Basierend auf dieser Problemanalyse wurden verschiedene Konzepte entwickelt, die sowohl das individuelle als auch das kooperative Erlernen der neuen Anpassungsmöglichkeiten erleichtern. Im Bereich des individuellen Lernens wird das endbenutzer-orientierte Anpassen vor allem durch semi-automatische Überprüfung der Integrität der Komposition unterstützt. Auf der kooperativen Ebene helfen Explorationsmöglichkeiten und Annotationen. Beide Konzepte unterstützen die Kommunikation zwischen Benutzern über die Bedeutung von Kompositionen.

Summary. The paper describes a tailoring environment which allows for run-time adaptions of component-based software. The tailoring language can be used on different levels of complexity. Cooperative aspects like the exchange of tailoring artifacts between users or descriptions are integrated. A search tool for groupware was implemented where users can tailor their own tool according to their needs. During a field study in a public administration organization we got to know about problems the users had when building their own tools or using tools which were built by colleagues. Based on this analysis we developed four different concepts that ease the individual as well as the collaborative learning of our tailoring language and the tailoring environment. The individual learning was supported by a semi-automatic integrity check. In order to support collaborative learning aspects we introduced an exploration environment and the possibility to annotate and therefore describe own compositions. Both concepts are designed to enforce the communication between users during the learning and tailoring process.

1. Einleitung

Anwendungen sollten Nutzern Möglichkeiten bieten, diese zur Laufzeit anzupassen. Speziell bei Standard-Software und Groupware sind solche Anpassungsmöglichkeiten von zentraler Bedeutung (vgl. Bentley und Dourish 1995, Oberquelle 1993 und 1994, Henderson und Kyng 1991). Anpassungen einer Software werden von den Benutzern als zusätzliche Aufgabe im Rahmen ihrer normalen Arbeit durchgeführt. Um diese Aktivitäten zu fördern, müssen Anpassungsumgebungen möglichst leicht zu erlernen sein und evtl. den Anpassungsakt selbst technisch unterstützen.

Da es unter den Nutzern einer Software typischerweise große Unterschiede im Hinblick auf deren Systemkenntnisse und Anpassungsfähigkeiten gibt, werden solche Anpassungen häufig kooperativ durchgeführt. Lokale Experten spielen in diesem Zusammenhang eine sehr wichtige Rolle, da sie – anders als die Mitglieder der IT-Abteilungen – auch mit der konkreten Arbeitspraxis vertraut sind (vgl. Mackay 1990, Nardi 1993, Wulf 1999). Der Gedanke, Anpassungsaktivitäten als kollaborative Arbeit zu verstehen, ist bisher in der technik-gestaltenden Forschung noch kaum vertieft aufgegriffen worden. Eine erste sys-

tematische Untersuchung liefert Kahler (2001).

Im Folgenden sollen vier unterschiedliche Konzepte vorgestellt werden, die sowohl das kollaborative Anpassen unterstützen als auch das individuelle Erlernen von Anpassungssprachen erleichtern. Diese Konzepte werden im Rahmen einer visuellen Anpassungsumgebung diskutiert, die auf einem komponentenbasierten Ansatz basiert. Als Beispiel stellen wir ein Suchtool für ein Groupware-System vor, bei dessen Entwicklung neuartige Konzepte zur Förderung der Erlernbarkeit von Anpassungsumgebungen umgesetzt wurden und werden.

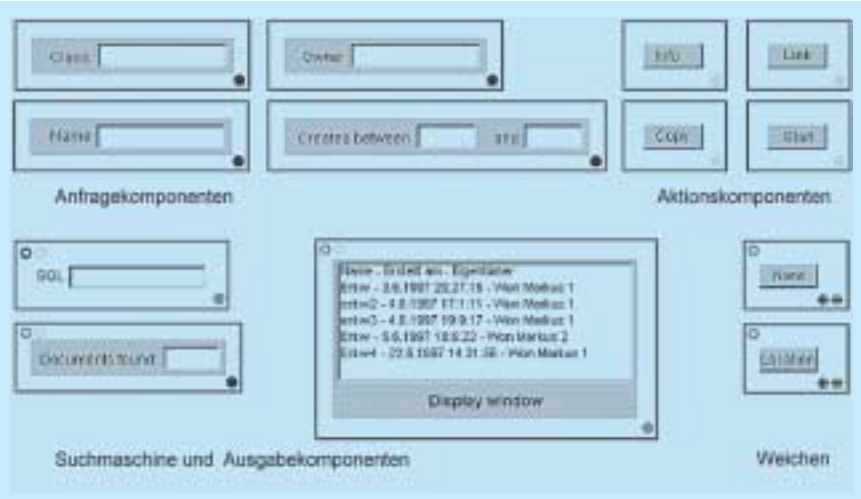

Bild 1: Basis-Komponenten für das Suchtool

2. Komponentenbasierte visuelle Anpassung

Um Anwendungen anzupassen, werden verschiedenartige Anpassungssprachen (grafisch, textuell etc.) verwendet.[1] Geeignete Anpassungsumgebungen dienen dann dazu, Benutzer beim Verändern bestehender Artefakte zu unterstützen. Anpassungen können – so sie von der Anpassungssprache unterstützt wird – auf unterschiedlichen Komplexitätsniveaus ausgeführt werden. Henderson und Kyng (1991) unterscheiden drei Komplexitätsniveaus:

- Auswahl zwischen Alternativen unterschiedlichen Verhaltens,
- Erstellung neuen Verhaltens aus bestehenden Teilen,
- Veränderung des Artefakts (z. B. Neuprogrammierung).

Offensichtlich steigt mit wachsender Komplexität der Anpassungssprachen oder der Operationen deren Ausdruckskraft. Dementsprechend müssen sie gezielt im Hinblick auf die erforderliche Ausdruckskraft entwickelt werden. Wenn ein hoher Grad an Anpassungskomplexität erforderlich ist, sollten Operationen auf unterschiedlichen Komplexitätsstufen für eine gleichmäßige Steigerung („gentle Slope") sorgen in dem Sinne, dass Benutzer mit einfachen Operationen erste Schritte unternehmen können und sich allmählich in Bezug auf die Komplexität ihrer Anpassungen steigern können (MacLean et al. 1990).

Trotzdem besteht bei den meisten Ansätzen eine große Kluft zwischen einfachen Operationen wie der Auswahl von Alternativen und der Modifikation des Artefakts durch Re-Programmierung. Als Ansatz zur Lösung dieses Problemkreises haben wir komponentenbasierte Anpassungsumgebungen konzipiert und implementiert. Dieser Ansatz versucht die Prinzipien komponentenorientierten Software-Entwicklung (vgl. Szyperski 1997) auf die Gestaltung von Anpassungsumgebungen zu übertragen.

Die Idee wird konzeptionell und technisch in Stiemerling und Cremers (1998) und (Stiemerling 2000) dargestellt. In der FreEvolve-Plattform[2] wird die Möglichkeit der Komponentenkomposition während der Laufzeit durch ein auf dem JavaBeans Komponentenmodell basierenden Framework geschaffen. Die FreEvolve-Plattform basiert auf der Idee der hierarchischen Komponentenarchitektur, die Operationen zur Verknüpfung von Komponenten auf verschiedenen Abstraktionslevels zur Verfügung stellt. Es gibt einfache Komponenten (z. B. Buttons oder Eingabetasten) und zusammengesetzte Komponenten (z. B. eine komplette Suchanfrage-Box), die durch Zusammensetzung und Verknüpfung einiger einfacher (oder zusammengesetzter) Komponenten erstellt werden können. Benutzer können dann sowohl zusammengesetzte, als auch einfache Komponenten benutzen, um ihre eigenen Anwendungen zu erstellen. Auf einem niedrigeren Level der Anpassungskomplexität bedeutet dies, dass die Benutzer eventuell nur einige zusammengesetzte Komponenten verbinden müssen, die für ihre Anpassungsaufgabe relevant sind. Auf dem höheren Level der Anpassungskomplexität steht ein umfangreichere Menge feingranularer Komponenten zur Auswahl.

Die FreEvolve-Plattform basiert auf der Idee, dass die hier entwickelte Anpassungssprache aus zwei Elementen besteht: die Operationen, die ausgeführt werden können und die Operanden, die durch die Operation verändert werden sollen. Wie oben beschrieben, gelten die Anpassungsoperationen (Einfügen und Entfernen von Komponenten, Verändern der Verknüpfung innerhalb einer Komposition) durchweg, wohingegen die Operanden in verschiedene Komplexitätslevels eingeordnet werden können. Dadurch wird das Lernen

[1] Die Bezeichnung „Anpassungssprachen" dient hier als Oberbegriff für alle Interaktionstechniken, mit denen Nutzer das Verhalten von Computeranwendungen zur Laufzeit verändern können. Anpassungssprachen müssen sowohl den aktuellen Zustand einer Anwendung in den Nutzern verständlicher Weise darstellen als auch intuitiv verständliche Interaktionstechniken zur Veränderung des Zustands anbieten. Bis jetzt fehlt eine wissenschaftliche Aufarbeiten des Gestaltungsraumes von Anpassungssprachen und Erkenntnisse über deren geeignete Anwendung für konkrete Anpassungsprobleme.

[2] Die FreEvolve-Plattform wurde an der Universität Bonn in der Diplomarbeit von Ralph Hinken und der Disseration von Oliver Stiemerling entwickelt unter der ursprünglichen Bezeichnung Evolve (vgl. Stiemerling 2000). Die Software ist öffentlich verfügbar unter GNU Public License. Sie wird aktuell in einer Reihe von Forschungsvorhaben weiterentwickelt und evaluiert.

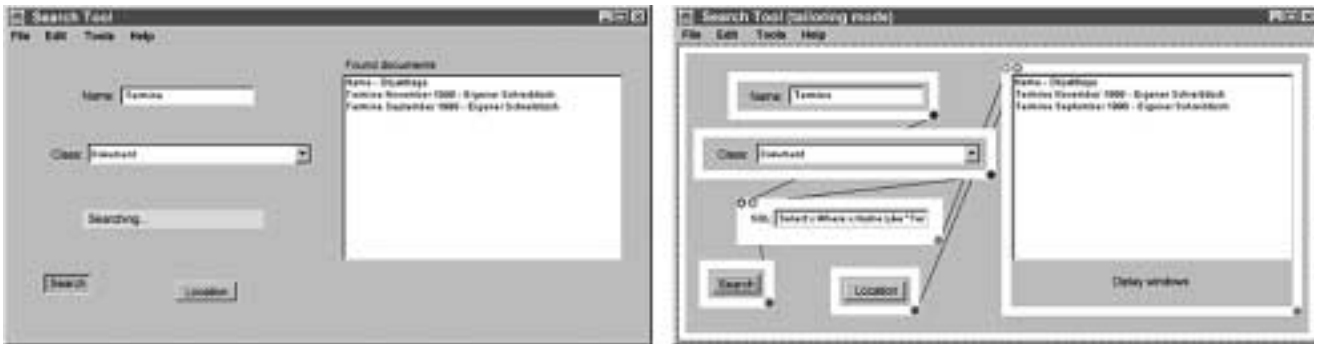

Bild 2: Suchtool zur Laufzeit und im Anpassungsmodus

solcher Anpassungsumgebungen erleichtert. Weitere Unterstützungstechniken zum Erlernen komponentenbasierter Anpassbarkeit sollen im Folgenden diskutiert werden.

Eine Beispielapplikation, die in FreEvolve realisiert worden ist, ist ein hochgradig anpassbares Suchwerkzeug für Groupware, anhand dessen im Folgenden das oben beschriebene Konzept kurz verdeutlicht werden soll. Die Zerlegung in Basiskomponenten und deren Design basieren auf empirischen Erhebungen bezüglich Suchgewohnheiten in vier verschiedenen Organisationen und einer Auswertung eines ersten Prototyps (vgl. Kahler 1996). Daraus ergab sich eine Aufteilung in vier Arten von Basiskomponenten (siehe Bild 1): Anfragekomponenten (oben links), Aktionskomponenten (oben rechts), Output Komponenten (einen Zähler und ein Displayfenster) und Datenflusskomponenten (die Suchmaschine unten links und die Schalter unten rechts) (Engelskirchen 2000).

Um Komponenten auszuwählen, verfügt die Anpassungssprache über entsprechende Operationen. Weitere Operationen ermöglichen das Verbinden zweier Komponenten über typgleiche Ein- bzw. Ausgangsports. In der Anpassungsumgebung sind diese als leere und gefüllte Kreise kenntlich gemacht, die Farben entsprechen den Typen der Ports. Die Operation wird mit der Maus ausgeführt, wobei eine Linie zwischen die entsprechenden Ports gezogen wird.

Bild 2 zeigt ein mögliches Suchtool zur Laufzeit und im Anpassungsmodus. Im zweiten Teilbild sind auch die Komponenten ohne GUI-Repräsentation wie die Suchmaschine selbst (Mitte) und die Weiche (unten, „Location") sichtbar. Die Verbindungen werden – wie oben erwähnt – als Linien zwischen den durch Kreise kenntlich gemachten Ports dargestellt.

Das Konzept der hierarchischen Anpassungssprachen wurde hier so umgesetzt, dass man zusammengesetzte, so genannte abstrakte Komponenten vorkonstruieren kann. Diese lassen sich speichern. Eine Möglichkeit bestände darin, jeweils eine abstrakte Komponente für die Eingabe und eine für die Ausgabe zu konstruieren. Weniger qualifizierte Benutzer können dann aus eben diesen beiden Komponenten ein Suchwerkzeug zusammensetzen, ohne sich um Details (jede abstrakte Komponente enthält dann mehrere Basiskomponenten und Verbindungen zwischen ihnen) zu kümmern.

Ein Menüpunkt in der Anpassungsumgebung erlaubt es, verschiedene vorkonfigurierte Suchtools entsprechend den jeweiligen konkreten Suchbedürfnissen der Nutzer auszuwählen. Weiterhin lassen sich im Anpassungsmodus komplett neue Suchwerkzeuge bauen und abspeichern. Dazu stehen die oben erwähnten Basiskomponenten ebenso zur Verfügung wie schon konstruierte abstrakte Komponenten. Um den Austausch von Kompositionen zu erleichtern, sind in der Anpassungsumgebung Funktionen integriert, mittels derer neue Kompositionen inkl. deren Beschreibungen für andere Benutzer verfügbar gemacht werden können.

Im Folgenden sollen nun die Konzepte vorgestellt werden, die das Erlernen der Anpassungssprache der FreEvolve-Plattform erleichtern. Diese Konzepte wurden entwickelt auf Basis von Literaturstudien und Diskussionen mit Nutzern während der oben angesprochenen Evaluationsphasen im Feld.

3. Erlernbarkeit in Anpassungsumgebungen

Bei Betrachtung der Eigenschaften, die das Erlernen von Anpassungssprachen unterstützen, können wir auf Erfahrungen bezüglich einfacher Funktionen in einzelnen Benutzeranwendungen zurückgreifen (vgl. Paul 1994). Anpassungsumgebungen für Benutzer ohne Programmierkenntnisse sollten konsistent mit der gewöhnlichen Funktionalität (vgl. MacLean et al. 1990, Nardi 1993, Wulf und Golombek 2001) entwickelt werden. Komponentenbasierte Anpassungssprachen ermöglichen das Verbinden eines Komponentensets mit verschiedenen Verknüpfungs- und Auswahloperationen.

Auf einem individuellen Level stimuliert die hierarchische Struktur schrittweises Erlernen der verschiedenen Sprachebenen (vgl. MacLean et al. 1990) durch sich langsam steigenden Umfang und Komplexität der verwendeten Komponenten. Um kooperatives Lernen zu ermöglichen, müssen Benutzer ohne Programmierkenntnisse in der Lage sein, von lokalen Experten bereit gestellte Anpassungsartefakte (z. B. die Menüleisten, die ein lokaler Experte erstellt hat) zu verstehen. Eigenschaften, die das Erlernen von Software fördern, erlauben das Strukturieren, Beschreiben, Experimentieren mit und das beispielhafte Erläutern von einzelnen Funktionen (vgl. Herrmann 1986, Caroll 1987, Caroll und Carrithers 1984, Paul 1994, Wulf 1999). Diese Eigenschaften werden während der Software-Entwicklung von den Programmierern für die Nutzer bereit gestellt. Im Folgenden werden wir die oben erwähnten Konzepte auf die Gestaltung von Anpassungsumgebun-

gen beziehen. Dabei müssen sie das Erlernen sowohl der Anpassungssprache selbst, als auch das der Anpassungsartefakte, die andere Nutzern oder lokalen Experten bereitstellen, fördern.

3.1 Benachrichtigungen und aktive Nutzerführung

Integritätsprüfungen, die während des Anpassungsvorgangs ein Feedback an der Benutzerschnittstelle erstellen, können Benutzer davon abhalten, leicht erkennbare Fehler zu machen oder nicht funktionstüchtige Anwendungen zu erstellen (vgl. Won 2000). Aufgrund der Tatsache, dass traditionelle Anwendungen weniger komplexe Anpassungsmechanismen bieten, bestand bisher wenig Bedarf nach dieser Art technischen Supports. Die hier durchgeführte Studie deutet an, dass Hilfen in Form von Ratschlägen oder sogar aktiver Unterstützung auf Basis von Integritätsüberprüfungen zu besserem Erlernen der Anpassungssprache beitragen können.

Die Idee ist nun, einen Integritätscheck zu integrieren, der die Komposition (aus Sicht des Benutzers die angepasste Applikation) während oder nach dem Anpassungsvorgang prüft und gegebenenfalls Feedback gibt oder Änderungen vorschlägt, um die Konstruktion „sinnloser" oder nicht funktionsfähiger Applikationen zu verhindern. Diese Prüfung läuft auf verschiedenen Ebenen ab:

- Typkorrektheit der Verbindungen zwischen Komponenten
- Eventfluss-Integrität
- Parametererisierungs-Constraints
- Komponentenauswahl-Constraints

Die ersten beiden Ansätze beziehen sich auf die Verbindungen zwischen Komponenten. Die typkorrekte Verbindung zweier Komponenten wird schon auf der Ebene der Programmiersprache oder innerhalb der Laufzeitumgebung überprüft. Sie unterbindet beispielsweise Verbindungen zwischen typfremden Ports. Beispielsweise ist es nicht zulässig eine Suchspezifizierungskomponente (z. B. „Name des Dokuments") mit einem Ausgabefenster statt mit der Suchmaschine zu verbinden. Da eben diese Prüfung innerhalb der FreEvolve-Plattform jedoch erst zur Laufzeit vorgenommen werden kann, wurde eine entsprechende Überprüfung integriert. Die Eventfluss-Integrität reicht noch weiter.

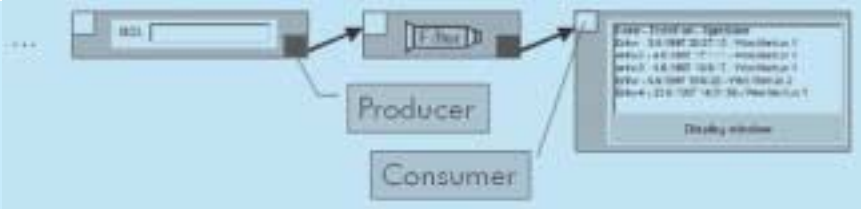

Bild 3: Eventfluss-Integrität am Beispiel eines Suchtools

Es werden nicht nur typkorrekte Verbindungen geprüft. Vielmehr enthalten die Basiskomponenten selbst Informationen, ob ein von ihnen bereitgestellter Port (Ausgang oder Eingang) optional ist oder eine korrekte Verwendung der Komponente vorsieht, dass diese Ports verwendet werden bzw. die von der Komponente erzeugten Nachrichten weiterverarbeitet werden müssen. Am Beispiel eines Ausgangsports lässt sich das recht gut veranschaulichen:

In Bild 3 ist der schematische Ausschnitt eines Suchwerkzeugs zu sehen. Hierbei sind drei Komponenten miteinander verbunden: die Suchmaschine, die eine Zugriffsmöglichkeit auf die darunter liegende Datenbanken hat, eine Filterkomponente und ein Ausgabefenster. Die Suchmaschine hat einen definitiv zu verwendenden Ausgangsport. Dieser wurde in der Beschreibung der Komponente definiert. Für das Kompositionswerkzeug bedeutet das, dass überprüft werden muss, ob dieser Ausgangsport (Producer) an einen „sinnvollen" Eingangsport angeschlossen ist, der in der Lage ist, die übermittelten Nachrichten zu konsumieren. Das Ausgabefenster ist ein solcher Konsument. Die oben beschriebene Konfiguration ist also gültig in dem Sinne, dass Suchmaschine und Ausgabefenster transitiv miteinander verbunden sind. Der Filter reicht ein Suchergebnis lediglich durch, ist im Sinne unserer Beschreibung kein Konsument.

Die Kombination des hier skizzierten Ansatzes mit dem der hierarchischen Komposition ist problemlos möglich, da die durch den Eventfluss definierten Nebenbedingungen an eine Komposition an eine „darüber liegende" abstrakte Komponente weitergereicht werden. Dieser Ansatz lässt sich nun noch verfeinern in der Form, dass es auch definitiv benötigte Eingangsports geben kann, dass einige Ports mehrere (in der Anzahl) Eingangswerte benötigen bzw. als Ausgangsports mehrere andere Komponenten mit Nachrichten versorgen können oder müssen (vgl. Won 2000).

Constraints – wie sie in Datenbanken bekannt sind – überprüfen die Korrektheit der Datenbasis. Sowohl die Parametererisierungs-Constraints als auch die Komponentenauswahl-Constraints werden in Applikationsvorlagen festgelegt. Hier werden für bestimmte Arten von Kompositionen (z. B. ein Suchwerkzeug) festgelegt, welche Komponenten oder Komponentenarten mindestens enthalten sein sollten. Weiterhin lässt sich aus den Beschreibungen ersehen, wie Parameter belegt werden dürfen. Auch Abhängigkeiten zwischen Parametern innerhalb einer Komponente oder zwischen unterschiedlichen Komponenten sind darstellbar.

Semantische Informationen, wie sie sowohl bei der Eventfluss-Integrität als auch bei den Constraints verwendet werden, müssen den Komponenten als externe Informationen beigefügt werden. Sie werden von den Komponentenentwicklern definiert, sind aber von Domänenexperten leicht änderbar (sie liegen in einem gut lesbaren XML-Format vor).

Wichtig für die Umsetzung eines durch einen Integrationscheck erweiterten Anpassungsmodus ist die Interaktion mit den Benutzern des Systems im Falle eines Fehlers oder einer vom System vorgeschlagenen Änderung. Nur durch geeignete Interaktionstechniken können Anpassungsoperationen sinnvoll unterstützt und geführt werden. Kühme et al. (1993) unterscheiden bezüglich des Grades der Automatisierung vier Arten, wie das System Unterstützungsarbeit leisten kann, angefangen von der Visualisierung von Anpassungshinweisen für die Nutzer bis hin zur vollständig automatischen Ausführung einer Änderung des Artefakts. Die hier vorgestellten Arbeiten stellen jedoch die Lernförderlichkeit in den Mittelpunkt, so dass vollständig automatisierte Adaptio-

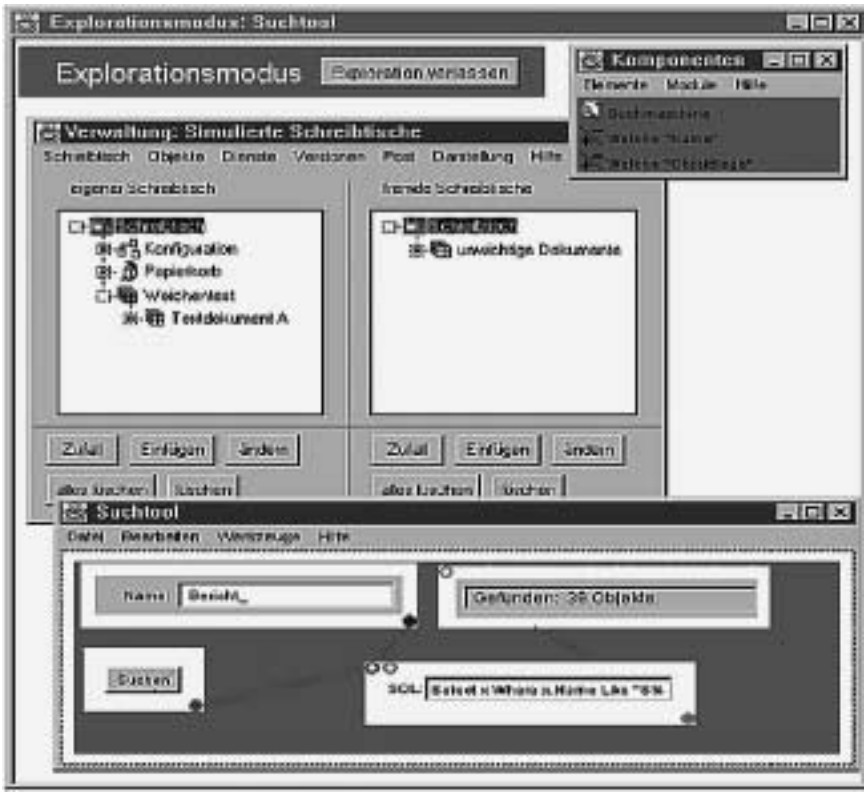

Bild 4: Suchtool im Explorationsmodus

nen des manuell angepassten Artefakts ausscheiden. Stattdessen sollen lediglich die ersten beiden Stufen (nach Kühme et al. 1993) abgedeckt werden: das Auffinden von möglichen Fehlern und die Benachrichtigung der Nutzer. Im ersten Fall, der *expliziten Analyse des Artefakts*, wird die Integritätsprüfung von den Benutzern gestartet. Dabei werden alle oben beschriebenen Bedingungen geprüft, wobei die Fehler direkt im Anpassungsmodus an den betreffenden Komponenten angezeigt werden. Beispielsweise wird das Fehlen von Verbindungen an unbedingt zu belegenden Ports durch Warnfarben und Hinweistexte dargestellt. Die *implizite Unterstützung begleitend zum Anpassungsvorgang* prüft die Integritätsbedingungen zum Zeitpunkt der Ausführung von Anpassungsaktivitäten, so dass Hinweise während der Komposition bereits berücksichtigt werden können.

3.2 Exploration von Anpassungsartefakten

Mackay (1990) ebenso wie Oppermann und Simm (1994) weisen darauf hin, dass das Experimentieren beim Erlernen von Anpassungsumgebungen eine wichtige Rolle spielt. Nichtsdestotrotz berichtet Mackay (1990), dass die Angst, einen funktionierenden Artefakt zu „zerstören", Nutzer vom Anpassen abhalten kann. Oppermann und Simm (1994) argumentieren, dass die aus dem Experimentieren mit Anpassfunktionen resultierenden Effekte schwer zu erkennen sind. Bezogen auf „normale" Funktionen von Einzelplatzanwendungen haben „Undo"-Funktionen, „freezing points", Spieldaten und ein Neutralmodus sich als geeignete Konzepte zur Unterstützung explorativen Lernens erwiesen (vgl. Paul 1994).

Im Hinblick auf die Exploration von anpassbarer Groupware lassen sich diese Konzepte aber nicht direkt übertragen, weil sie darauf beruhen, dass man an der Benutzungsschnittstelle des Aktivators den Systemzustand vor und nach der Ausführung einer Funktion wahrnehmen kann. Dies ist in Groupware typischerweise nicht der Fall. Dort kann der Aktivator entweder nicht den relevanten Systemzustand vor Ausführung einer Funktion wahrnehmen (z. B.: bei einem Suchtool, das in verschiedenen dem Aktivator nicht vollständig zugänglichen Arbeitsbereichen sucht) oder aber den Zustand nach Ausführung der Funktion (z. B.: beim Mail-Versand, die Art der Darstellung von Prioritätsangabe und Attachments beim Empfänger). Diese Probleme erschweren bereits das Erlernen von „normalen" Funktionen, wiegen aber besonders schwer, wenn diese anpassbar gestaltet sind.

Um exploratives Lernen von anpassbarer Groupware zu unterstützen, haben wir das Konzept der Explorationsumgebungen entwickelt (vgl. Wulf 2000). In Explorationsumgebungen wird das Systemverhalten an der Benutzungsschnittstelle des Aktivators in den für das Erlernen relevanten Aspekten simuliert. So wird es möglich, die Benutzungsschnittstelle anderer Nutzer an der des Aktivators darzustellen. Durch Wechseln der Perspektiven zwischen verschiedenen möglichen Nutzern und die Aktivierung anderer in der Explorationsumgebung implementierter Funktionen kann der Aktivator die Effekte der Ausführung der zu explorierenden Funktion auf die Schnittstelle anderer Nutzer erkunden.

Um bei der Simulation den Zustand der im System befindlichen Daten nicht zu verändern, müssen Spieldaten in der Explorationsumgebung erzeugt werden. Die in der Explorationsumgebung realisierten Funktionen führen auf den Spieldaten Zustandsübergänge aus, die analog zu denen der realen Funktionen sind. Auch die in der Explorationsumgebung nachgebauten Dialog- und Ein- und Ausgabeschnittstellen der ursprünglichen Funktionen sollten ihrem realen Vorbild so ähnlich wie möglich sein. Allerdings können aus didaktischen Gründen und auf Grund der für die gleichzeitige Anzeige zweier simulierter Benutzungsschnittstellen (der des Aktivators und der eines anderen Nutzers) begrenzten Ein- und Ausgabemöglichkeiten Vereinfachungen vorgenommen werden.

Die Nutzung der Explorationsumgebung im Rahmen der Anpassungsumgebung des Suchtools ist so einfach wie möglich gestaltet. Durch Aufruf eines Befehls in der Menüliste der Anpassungsumgebung wird in die Explorationsumgebung gewechselt. Die Umgebung nimmt eine andere Farbe an (spezifisch für Explorationsumgebungen), um dem Nutzer die Veränderung des Zustands anzuzeigen. Statt der realen Arbeitsbereiche arbeitet das Suchwerkzeug nunmehr auf simulierten Arbeits-

bereichen, die mit Experimentaldaten (Dokumentensammlungen) bevölkert sind. Innerhalb der Explorationsumgebung kann der Nutzer einzelne Varianten von Suchwerkzeugen im Hinblick auf ihre Wirkung auf die verschiedenen Arbeitsbereiche ausprobieren.

Bild 4 zeigt ein Suchtool im Explorationsmodus. Das große Fenster in der Bildmitte erlaubt die verschiedenen virtuellen Arbeitsbereiche mit zusätzlichen Dokumenten (Spieldaten) zu bevölkern. Das Fenster im unteren Teil des Screen Shots zeigt das zu explorierende Suchtool im Explorationsmodus. Es sieht genauso aus wie das Suchtool im Anpassungsmodus, lediglich die Hintergrundfarbe ist modifiziert.

3.3 Erläuterungen anhand von Beispielen

Beispiele und Erklärungen, die andere Nutzer zusammengestellt haben, sind wichtige „Anreize" zur Anpassung (Wulf 1999). So zeichnen Animationsmaschinen die Abfolge der Interaktionsschritten auf und geben damit Anleitungen, wie bestimmte Funktionen benutzt werden können (vgl. Paul 1994).

Um Nutzern eine Anleitung bei der Komposition von Komponenten zu geben, ermöglicht die Anpassungsumgebung, zu jeder Komponente ein Beispiel-Suchwerkzeug abzuspeichern. Solche Beispielkompositionen veranschaulichen wie die entsprechende Komponente innerhalb einer eingesetzt werden kann. Die Beispielkomposition sollte so konstruiert sein, dass der Nutzen der Komponente gut veranschaulicht wird. Den Basiskomponenten, die von den Entwicklern implementiert werden, werden diese Beispiele direkt beigefügt. Abstrakte Komponenten können dagegen von den jeweiligen Erzeugern mit selbstgebauten Beispiel-Suchwerkzeugen illustriert werden.

Falls eine solche Beispielanwendung aktiviert wird, geschieht dies im Explorationsmodus. Hier können Benutzer das Tool ohne Seiteneffekte auf andere Benutzer testen und anhand der Ergebnisse das Verhalten der Komposition verstehen.

Erprobungen ergaben, dass dieses Konzept für viele Benutzer sehr hilfreich ist, um zu verstehen, wie eine Komponente eingesetzt werden kann. Eine der Testpersonen wählte sogar ein Beispielwerkzeug direkt für seine tägliche Arbeit aus, auch dies ist innerhalb der Anpassungsumgebung möglich. Inwieweit die Annotationsfunktion genutzt wurde, um eigenen Kompositionen Beispiele hinzuzufügen, konnte aufgrund der begrenzten Dauer des Feldtests nicht mehr ermittelt werden (vgl. Wulf 1999).

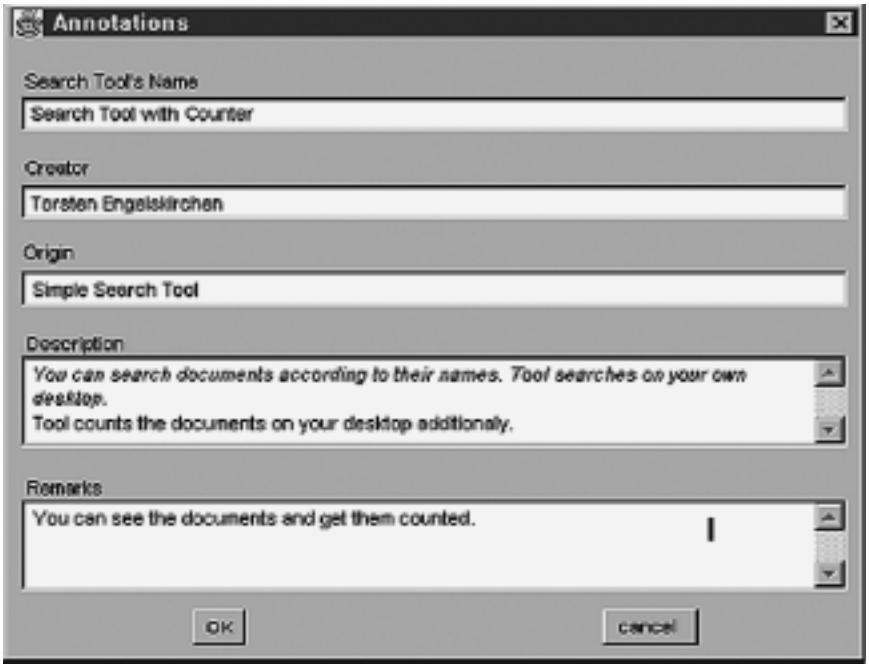

Bild 5: Annotation zu einer abstrakten Komponente

3.4 Beschreibungsmöglichkeiten

Mackay (1990) zeigt, dass das Fehlen einer Dokumentation die Nutzung bzw. das Anpassen verhindert. Handbücher und Hilfetexte gelten als typische Mittel, um die Funktionalität einer Anwendung zu beschreiben. Dementsprechend sollten Nutzer oder lokale Experten, die auf Basis hierarchischen Sprachen eigene Applikationen anpassen oder konstruieren, diese auch dokumentieren können. Die Anpassungsumgebung müssen entsprechende Möglichkeiten zur Verfügung stellen. Speziell unerfahrene Benutzer können mit Hilfe von Annotationen vorgefertigte (von lokalen Experten erstellten) Kompositionen nutzen und verstehen.

In der Feldstudie zeigte sich, dass allein der Name eines Suchwerkzeugs – als einer Komposition aus verschiedenen Komponenten - und dessen Klassifikation in einem Repository Nutzern nur eine vage Vorstellung vermitteln, welche Funktionalität sich darin verbirgt. Selbstangefertigte und den Kompositionen beigefügte Annotationen können hier weiterhelfen. Bild 5 zeigt die für die Anpassungsumgebung gewählte Struktur der Annotationen. Neben einer Beschreibung der Funktionalität hat sich vor allem der Name des Erzeugers der Komposition als wichtige Information erwiesen. Er lässt Rückschlüsse auf die Funktionalität und die Qualität der Komposition zu, bietet die Möglichkeit für Rückfragen und erlaubt es dem Erzeuger, für seine Arbeiten innerhalb der Organisation Anerkennung zu finden (vgl. Wulf 1999). Durch kooperative Elemente, die es auch anderen Benutzern erlauben, den Kompositionen aus Komponenten Annotationen hinzuzufügen, lässt sich die Aussagekraft dieser Beschreibungstechnik weiter erhöhen.

4. Zusammenfassung

Anpassbarkeit ist eine zentrale Anforderung für interaktive Systeme, insbesondere wenn sie in dynamisch sich verändernden Kontexten eingesetzt sind. Wir haben hier einen innovativen Ansatz vorgestellt, Anwendungen in weitreichender Weise anpassbar zu gestalten. Die FreEvolve-Plattform erlaubt, zur Laufzeit Änderungen an der Komponenten-Komposition (der Applikation) vorzunehmen. Am Beispiel eines Suchwerkzeugs für ein Groupware-System wurde dieser

Ansatz evaluiert. Auf Basis der Ergebnisse der Studie wurden verschiedene Konzepte entwickelt, um Benutzer beim Anpassungsvorgang zu unterstützen.

Zunächst wurde die Anpassungsumgebung kooperativ gestaltet, um den Austausch von Anpassungsartefakten zwischen den Nutzern zu fördern. Bei der Förderung von kooperative Anpassungsaktivitäten erscheint die hierarchische Komponentenarchitektur besonders geeignet, weil sie es den verschiedenen Akteuren erlaubt, auf verschiedenen Komplexitätsniveaus Anpassungen vorzunehmen.

Wir haben dann vier Konzepte der Gestaltung der Benutzungsschnittstelle vorgestellt, die die Lernförderlichkeit von Anpassungsumgebungen fördern. Sie unterstützen Nutzer sowohl bei individuellen Anpassungsaktivitäten als auch bei der Nutzung von von anderen Nutzern erstellten Anpassungsartefakten. Insofern stellen sie ein zentrales Element der Gestaltung komponentenbasierter Anpassungsumgebung dar. Diese Konzepte der Benutzungsschnittstelle sind aber auch übertragbar auf Anpassungsumgebungen, die anderen software-technischen Paradigmen folgen.

Danksagung

Diese Veröffentlichung entstand im Kontext des EU Networks of Excellence on „End User Development: Empowering people to flexibly employ advanced information and communication technology“ (Contract Number IST-2001-37470). Wir danken unseren Kollegen Sascha Alda, Torsten Engelskirchen, Björn Golombek, Markus Klann, Matthias Krings, Michael Krüger, Volkmar Pipek und Gunnar Stevens für ihre Unterstützung.

Literatur

Bentley, R.; Dourish, P.: Medium versus Mechanism. In: *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work – ECSCW '95* (Hrsg. Marmolin, H.; Sundblad, Y.; Schmidt, K.). Kluwer, 1995.

Birngruber, D.; Hof, M.: Interactive Decision support for JavaBeans composition. In: *Fifth International Workshop on Component-Oriented Programming.* Cannes (Frankreich), 2000.

Caroll, J. M.: Five Gambits for the advisory Interfacs Dilemma. In: *Psychological Issues of Human Computer Interaction in the Work Place.* (Hrsg.: Frese, M.; Ulich, E.; Dzida, W.), Amsterdam, 1987.

Carroll, J. M.; Carrithers, C.: Training Wheels in a User Interface. In: *Communications of the ACM.* Vol. **27**, No. 8 (1984) 800–806.

Dourish, P.: *Open Implementation and Flexibility in CSCW Toolkits.* PhD Thesis, University College London, 1996.

Engelskirchen, T.: *Exploration anpaßbarer Groupware.* Diplomarbeit, Institut für Informatik III, Universität Bonn, 2000.

Henderson, A.; Kyng M.: There's No Place Like Home. Continuing Design in Use. In: *Design at Work.* Lawrence Erlbaum Associates, Publishers (1991), S. 219-240.

Herrmann, Th.: *Zur Gestaltung der Mensch – Computer – Interaktion. Systemerklärung als kommunikatives Problem.* Niemeyer, Tübingen, 1986.

Kahler, H.: Developing Groupware with Evolution and Participation – A Case Study. In: *Proceedings of the Participatory Design Conference.* Cambridge, MA, S. 173–182, 1996.

Kahler, H.: *Supporting Collaborative Tailoring.* PhD-Thesis, Roskilde University, Dänemark, Roskilde, 2001.

Kühme, T.: A User-centered Approach to Adaptive Interfaces. In: *Proceedings of the International Workshop on Intelligent User Interfaces (IUI '93).* January 4–7, Orlando, FL, ACM-Press, New York, S. 243–245, 1993.

Mackay, W. E.: *Users and customizable Software: A Co-Adaptive Phenomenon.* PhD Thesis, MIT, Boston (MA), 1990.

MacLean, A.; Carter, K.; Lövstrand, L.; Moran, T.: User-tailorable Systems: Pressing the Issue with Buttons. In: *Proceedings of the Conference on Computer Human Interaction (CHI '90).* April 1–5, Seattle (Washington), ACM-Press, New York (1990), S. 175–182.

Nardi, B. A.: *A Small Matter of Programming – Perspectives on end-user computing.* MIT-Press, Cambridge et al, 1993.

Oberquelle, H.: Anpassbarkeit von Groupware als Basis für die dynamische Gestaltung von computergestützter Gruppenarbeit. In: *Benutzeroberflächen in der teilautonomen Arbeit.* (Hrsg. Konradt, U.; Drisis, L.): Köln (1993), S. 37–54.

Oberquelle, H.: Situationsbedingte und benutzerorientierte Anpassbarkeit von Groupware. In: *Menschengerechte Groupware Hartmann.* (Hrsg. Herrmann, Th.; Rohde, M.; Wulf, V.), Stuttgart (1994) S. 31–50.

Oppermann, R.; Simm, H.: Adaptability: User-Initiated Individualization. In: Adaptive User Support – *Ergonomic Design of Manually and Automatically Adaptable Software.* (Hrsg. Oppermann, R.), Lawrence Erlbaum Ass., Hillsdale, New Jersey, 1994.

Paul, H. (1994): *Exploratives Agieren,* Peter Lang, Frankfurt/M.

Stiemerling, O.; Cremers, A.B.: Tailorable Component Architectures for CSCW-Systems. In: *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Programming.* Jan 21–24, IEEE Press, Madrid (Spain), S. 302–308, 1998.

Stiemerling, O.: *Component-based tailorability.* Dissertation, Institut für Informatik III, Universität Bonn, Bonn, 2000.

Szyperski, C.: *Component Software – Beyond object-oriented programming.* Addison-Wesley, 1997.

Won, M.: *Checking integrity of component-based architectures.* CSCW 2000 Philadelphia USA, Workshop on Component-Based Groupware, 2000.

Wulf, V.: „Let's see your Search-Tool!" – Collaborative use of Tailored Artifacts in Groupware. In: *Proceedings of GROUP '99.* ACM-Press, New York, (1999) 50–60.

Wulf, V.: Exploration Environments: Supporting Users to Learn Groupware Functions. In: *Interacting with Computers.* Vol. 13 No. **2** (2000) 265–299.

Wulf, V.; Golombek, B.: *Direct Activation: A Concept to Encourage Tailoring Activities, Behavior and Information Technology,* Vol. 20, No. **4** (2001) 249–263.

1

2

1 Prof. Dr. Volker Wulf, Professor für Wirtschaftsinformatik an der Universität Siegen, Leiter einer Forschungsgruppe am Fraunhofer FIT, St. Augustin. Hauptarbeitsgebiete: Computerunterstützte Gruppenarbeit, computer-unterstütztes Lernen, Wissensmanagement, Mobile Computing und Usability Engineering.
E-Mail: volker.wulf@fit.fraunhofer.de

2 Dipl. Inf. Markus Won, wissenschaftlicher Mitarbeiter des Projektbereichs Software-Ergonomie und CSCW (ProSEC) am Institut für Informatik III der Universität Bonn. Hauptarbeitsgebiete: CSCW, Wissensmanagement auch in der Software-Technik (komponentenbasierte Anpassbarkeit im Bereich Groupware).
E-Mail: won@cs.uni-bonn.de