Der folgende Artikel ist im OBJEKTspektrum 3/99, S. 90-95 erschienen.

Frameworkbasierte Anwendungsentwicklung (Teil 3):

Die Anbindung von Benutzungsoberflächen und Entwicklungsumgebungen an Frameworks

Wolf-Gideon Bleek, Martin Lippert, Stefan Roock, Wolfgang Strunk, Heinz Züllighoven

In OBJEKTspektrum 1/99 haben wir eine Modellarchitektur für große interaktive Anwendungssysteme vorgeschlagen. Zwei wesentliche Komponenten sind dabei Werkzeuge und Materialien. Werkzeuge, die fachlich die Arbeitsmittel im Anwendungsbereich repräsentieren, kapseln softwaretechnisch die Interaktion mit dem Benutzer. In OBJEKTspektrum 2/99 ging es um die frameworkbasierte Konstruktion von Werkzeugen.[1] Diesmal wollen wir zeigen, wie Benutzungsoberflächen frameworkunterstützt an Werkzeuge angebunden werden können. Da Benutzungsoberflächen heute mit Hilfe von Entwicklungsumgebungen erheblich einfacher erstellt werden können als manuell, stellt sich das Problem, wie sich Frameworks solchen Entwicklungsumgebungen gegenüber verhalten sollen. Wir werden diesen Punkt am Beispiel des JWAM-Frameworks diskutieren.

Interaktive Software muß per Definition eine Benutzungsoberfläche besitzen. Heute ist die Verwendung grafischer Benutzungsoberflächen (z. B. MS-Windows) Stand der Kunst. Diese weisen nicht einfach nur eine ‚angenehmere" Präsentation auf als textorientierte Benutzungsoberflächen. Sie gestatten dem Benutzer auch, die Software flexibler zu verwenden. Er kann einzelne Fenster des Softwaresystems in (fast) beliebiger Reihenfolge aktivieren und seine Aufgaben so erledigen, wie es ihm angemessen erscheint. Die breite Palette an Oberflächenelementen erlaubt außerdem eine differenzierte Präsentation und Interaktion. So müssen z. B. keine Kürzel mehr eingegeben werden, um ein Element aus einer Menge auszuwählen. Mit einer grafischen Oberfläche kann eine Auswahlliste mit den möglichen Elemente direkt präsentiert werden.

Die neuen Möglichkeiten grafischer Benutzungsoberflächen haben allerdings auch den Preis erhöhter Komplexität bei der Anbindung der eigentlichen Anwendung. Die vom Betriebssystem bereitgestellten Programmierschnittstellen (Application Programming Interfaces, APIs) sind in der Regel sehr schwierig zu benutzen. Aus diesem Grund bieten verschiedene Hersteller sogenannte GUI-Toolkits (Abk. für Graphical User Interface) an, mit denen grafische Benutzungsoberflächen leicht erstellt und angebunden werden sollen. Sie bieten eine der verwendeten Programmiersprache angemessenere Schnittstelle sowie einen GUI-Builder. Mithilfe des GUI-Builders können Oberflächen visuell komponiert werden. Je nach verwendetem Toolkit erzeugt der GUI-Builder Quellcode oder Ressourcen-Dateien. Letztere werden meist zur Laufzeit vom Programm eingelesen und in eine grafische Benutzungsoberfläche umgesetzt. Häufig bieten die GUI-Toolkits darüber hinaus eine Abstraktion des vom Betriebssystem bereitgestellten API. So kann dieselbe GUI-Anbindung für verschiedene Betriebssysteme genutzt werden. Im Idealfall müssen dann die Programme bei Wechsel des Betriebssystems nur neu kompiliert werden.

Bei näherer Betrachtung der Betriebssystem-APIs sowie der durch die Toolkits und GUI-Builder bereitgestellten Schnittstellen fällt auf, daß diese sich zum Teil erheblich unterscheiden. Dies liegt zum einen daran, daß jedes Betriebssystem eine unterschiedliche Menge an Oberflächenelementen anbietet. So werden grafische Baumdarstellungen erst seit Windows-95 und dem Windows-Explorer auf breiter Ebene verwendet. Andere Betriebssysteme bieten hierfür teilweise keine Oberflächenelemente. Auf der anderen Seite spiegelt sich in jeder Betriebssystemprogrammierschnittstelle und jeder von einem Toolkit bereitgestellten Schnittstelle die dahinterstehende Entwurfsidee wider.

Eine Kapselung dieser Schnittstellen (API oder Toolkit) stellt also eine äußerst anspruchsvoll Aufgabe dar. Aufgrund dieser Komplexität, wird diese Aufgabe häufig gar nicht in Angriff genommen. Man ‚verheiratet"

[1] Beide Artikel finden Sie übrigens auch im Internet unter www.sigs.de.

sich dann mit dem Hersteller der Wahl und hofft, daß dieser lange genug überlebt. In der Folge werden bei der Entscheidung für ein GUI-Toolkit oftmals große Hersteller bevorzugt, auch wenn Produkte kleinerer Anbieter aus einer technischen Perspektive viel geeigneter wären. Allerdings bietet auch dieses Vorgehen keine Garantie, wie sich z. B. an der GUI-Bibliothek ‚zApp" von der Firma Rogue Wave zeigt.[2]

Aber selbst wenn der Hersteller überlebt, können sich Schnittstellen gravierend ändern. Ein Beispiel dafür ist die Entwicklung der GUI-Anbindung im ‚Java Development Kit" (JDK). Zunächst wurde beim Übergang von Version 1.0 auf Version 1.1 die Behandlung von Ereignissen im ‚Abstract Windowing Toolkit" (AWT) vollständig ausgetauscht. Beim aktuellen Übergang von JDK 1.1 auf JDK 1.2 hat das Unternehmen Sun die ‚Swing"-Oberflächenbibliothek eingeführt, die eine vollständig andere Schnittstelle anbietet als das AWT.

Es gilt also, die negativen Auswirkungen des Austauschs eines GUI-Toolkits zu minimieren. Im letzten Teil dieser Serie ([Ble99b]) haben wir einen Mechanismus und die passende Frameworkunterstützung für die Trennung zwischen Funktion und Interaktion vorgestellt. Hierdurch haben wir die Auswirkung von Änderungen am GUI-Toolkit zwar auf die Interaktion beschränkt; wir halten jedoch einen weitergehenden Ansatz für sinnvoll, der vom eingesetzten Toolkit abstrahiert.

Interaktionsformen zur Anbindung von Benutzungsoberflächen

Die Vergangenheit hat gezeigt, daß sich stabile Abstraktionen über GUI-Toolkits nur sehr schwer finden lassen, wenn von einer Reihe vorhandener Toolkits ausgegangen wird. Unser Ansatz zur Kapselung von GUI-Toolkits geht deshalb von den Anforderungen der Anwendungen aus und stellt die durch die Technologie (des GUI-Toolkits) vorgegebene Architektur in den Hintergrund. So können sehr stabile Abstraktionen mit dem richtigen Umfang definiert werden - sie enthalten genau die Anteile, die von der Anwendung benötigt werden.

Diese Abstraktionen des GUI-Toolkits und des Betriebssystem-APIs für Benutzungsoberflächen nennen wir Interaktionsformen (IAF). Wir haben sie zunächst in einem C++-Framework und dann in unserem Java-Framework (siehe [JWAM99]) umgesetzt.

Interaktionsformen vergegenständlichen Benutzerinteraktionen auf einem fachlichen Niveau. Sie orientieren sich am Umgang des Anwenders mit der Benutzungsoberfläche des Programms. So sprechen wir beispielsweise vom Aktivieren einer Funktion, der Auswahl eines Elements oder dem Eintragen eines Werts. Abbildung 1 zeigt eine beispielhaft einige Interaktionsformen.

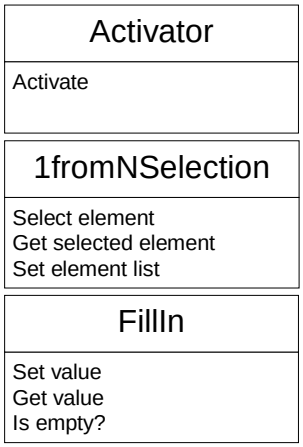

Abbildung 1: Einige beispielhafte Interaktionsformen mit den möglichen Operationen

Die Interaktionsformen stehen in keinem fixierten Zusammenhang mit den Oberflächenelementen. Sie müssen aber mit Oberflächenelementen verbunden werden, um sie dem Benutzer zu präsentieren. Erst durch die konkrete Realisierung kann der Benutzer eine angebotene Interaktionsform handhaben. Das zu einer Interaktionsform passende Element aus einem GUI-Toolkit wird durch eine Präsentationsform gekapselt. Die Interaktions- form kennt nur die ihr zugeordnete Präsentationsform, nicht jedoch das gekapselte Oberflächenelement. Abbildung 2 zeigt die Verknüpfung von Interaktions- mit Präsentationsformen.

Wie aus der Abbildung leicht zu erkennen ist, kann eine Interaktionsform mit mehreren Präsentationsformen verbunden werden. Beispielsweise läßt sich die Interaktion, ein Werkzeug zu schließen, auf unterschiedliche Art und Weise realisieren:

- Das Fenster kann geschlossen werden.
- Im Fenster kann ein Knopf ‚Schließen" gedrückt werden.
- Der Menüeintrag ‚Schließen" kann gewählt werden.

[2] Rogue Wave hat die Entwicklung am Produkt "zApp" eingestellt, und auch einen Support wird es in Kürze nicht mehr geben (siehe z.B.: http://www.roguewave.com/products/zapp.html")

Alle drei Möglichkeiten bedeuten für das Werkzeug die gleiche Interaktion, nämlich: “schließe das Werkzeug".

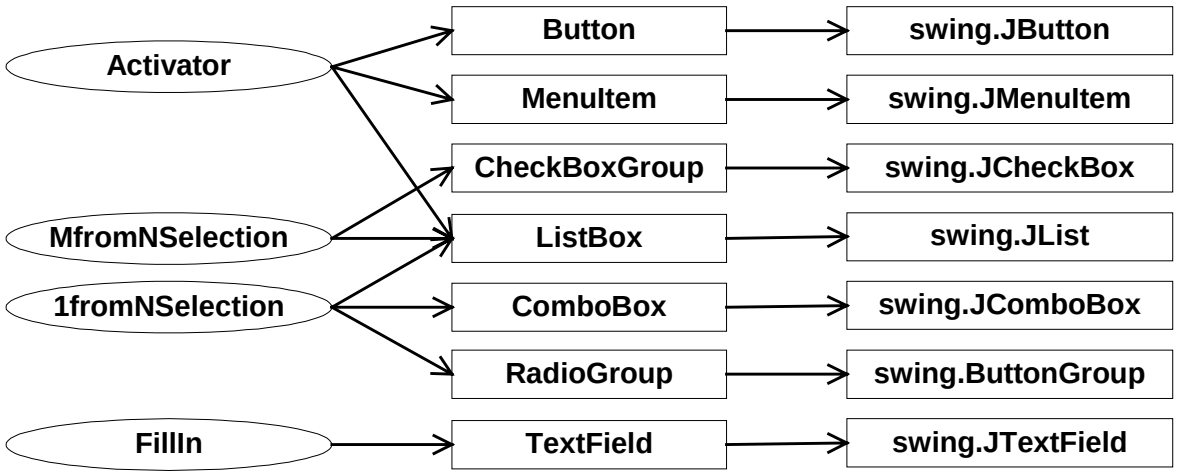

Abbildung 2: Interaktionsformen und zugehörige Präsentationsformen sowie gekapselte Widgets für eine Java-Swing-Anbindung

Eine Präsentationsform kann aber auch mit mehreren Interaktionsformen verbunden sein. So kann beispielsweise die Präsentationsform ListBox die Interaktionsform Activator anbieten (z. B. durch einen Doppelklick auf ein Element in der Liste), aber auch die Interaktionsform 1FromNSelection durch Auswahl eines Elementes aus einer Menge durch Anklicken. Die Aufgaben von Interaktions- und Präsentationsformen sind in Tabelle 1 dargestellt.

	Interaktionsform	**Präsentationsform**
Aufgabe	Vergegenständlichung von Benutzerinteraktion auf der konzeptionellen Ebene	Realisierung der Interaktion mit dem Benutzer durch graphische Ein- und Ausgabe
Bindeglied für	Anbindung Präsentationsform an das Programm	Anbindung an ein Toolkit- oder die Betriebssystem-API
Informationen über	die Art und Weise der Interaktion mit dem Programm	die konkrete Handhabung und Visualisierung

Tabelle 1: Konzept von Interaktions- und Präsentationsform

Passende Interaktionsformen für Softwarewerkzeuge können z. B. Systemvisionen entnommen werden. Systemvisionen beschreiben den möglichen Umgang mit dem zu entwickelnden System. Ein Ausschnitt eines Beispiels findet sich im Kasten 1.

... der Berater öffnet dann seinen Beratungsordner mit einem Doppelklick, wählt zunächst das Produkt über das sichtbare Inhaltsverzeichnis, wählt anschließend in einem zweiten Verzeichnis die gewünschte Variante mit einem Doppelklick und läßt sich dadurch gleich die zugehörige Verkaufshilfe anzeigen.

Kasten 1: Beispiel für Systemvision

Mit der Trennung von Interaktion und Präsentation haben wir ein Konzept bereitgestellt, um die Unabhängigkeit des Programms von der Benutzungsoberfläche zu verstärken. Bei der Implementierung des Konzepts können wir verschiedene Wege gehen:

- Interaktions- und Präsentationsformen lassen sich in zwei getrennten Klassenhierarchien realisieren. Zur Laufzeit werden Objekte beider Typen unabhängig voneinander erzeugt. Diese Variante beschreiben wir im übernächsten Abschnitt genauer.
- Alternativ können Interaktions- und Präsentationsform in derselben Klassen zusammengefaßt werden. Versteht man Interaktions- und Präsentationsform als verschiedene Rollen desselben Objekts (ausgedrückt z. B. durch getrennte Interfaces in Java), spricht nichts gegen eine Zusammenfassung der Funktionalität in einer Klasse und somit eine Reduzierung der Objektanzahl zur Laufzeit.

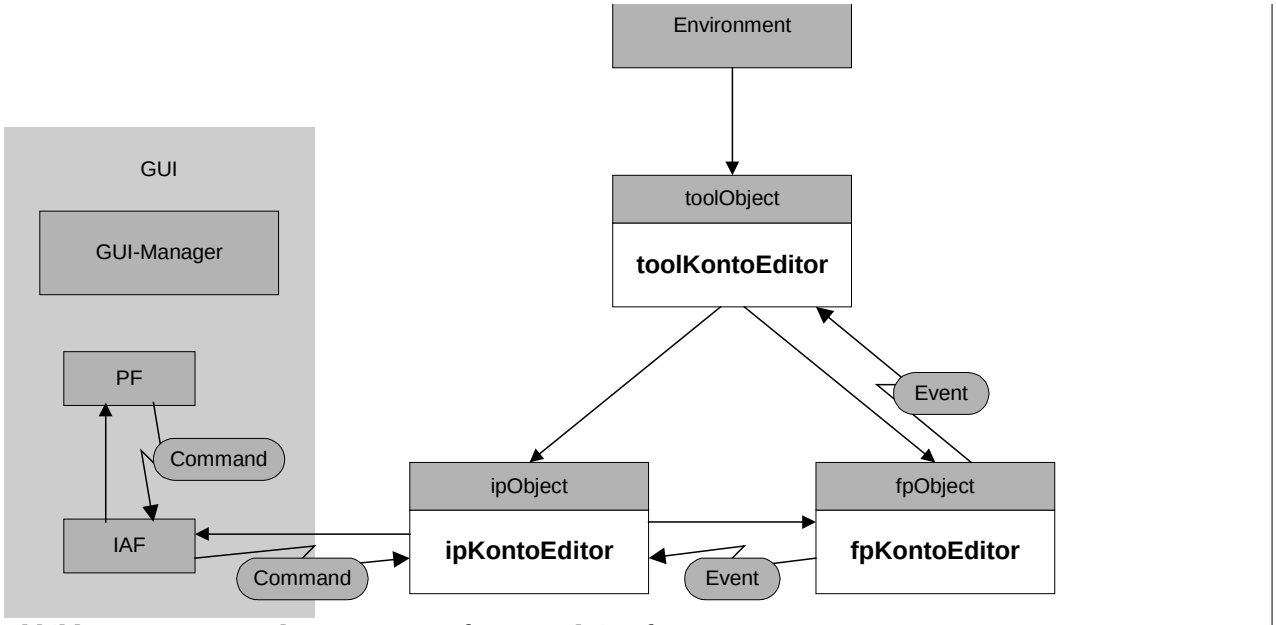

Abbildung 3: Frameworkunterstützung für Interaktionsformen

Frameworkunterstützung für Interaktionsformen

Interaktions- und Präsentationsformen können vollständig als Blackbox-Frameworks realisiert werden. Für die Verwendung müssen keine Klassen beerbt werden, und die Komplexität der Oberflächenanbindung bleibt hinter den Interaktionsformen verborgen. Die Interaktionsformen stellen eine deutlich schlankere Schnittstelle für die Oberflächenanbindung zur Verfügung, als dies bei GUI-Toolkits der Fall ist.

Damit werden auch die Manipulationsmöglichkeiten eingeschränkt. So kann über die Schnittstellen der Interaktionsformen nicht direkt die Präsentation (z. B. Farbe, Schriftart) der Oberflächenelemente beeinflußt werden. Für viele Anwendungsbereiche ist eine solche differenzierte Kontrolle der Präsentation nicht notwendig. Möchte der Anwendungsentwickler doch direkt auf Eigenschaften der Präsentation zugreifen, so kann er dies mit Hilfe der Präsentationsformen. Die Teile der Anwendung, die von dieser Möglichkeit Gebrauch machen, werden dadurch toolkit-abhängig. Da in diesen Teilen die Präsentationsformen als Typen verwendet werden, lassen sie sich leicht identifizieren. So kann der Umstellungsaufwand beim Wechsel des GUI-Toolkits gut abgeschätzt werden.

Neben den Interaktions- und Präsentationsformen muß ein geeigneter Callback-Mechanismus bereitgestellt werden. Im JWAM-Framework wird dazu ein modifiziertes Befehlsmuster (siehe [Gam98]) verwendet. Über dieses Befehlsmuster melden Präsentationsformen den zugeordneten Interaktionsformen Aktionen des Benutzers. Die Interaktionsformen wiederum verwenden Kommandos, um diese Aktionen an das Softwarewerkzeug weiterzuleiten. Softwarewerkzeuge können sich bei Interaktionsformen registrieren, wenn sie bei Benutzeraktionen aktiviert werden sollen. Löst der Benutzer eine Aktion aus, wird zunächst das entsprechende Kommando an der Präsentationsform aktiviert, was wiederum zur Auslösung des Kommandos an der Interaktionsform führt. Dieses Kommando führt schließlich zum Aufruf einer Operation in dem Softwarewerkzeug. Die Auswahl des Kommando-Mechanismus hängt allerdings stark von der verwendeten Implementierungsvariante der Interaktions- und Präsentationsformen ab.

Ein Großteil der benötigten Komponenten für die Anbindung der Benutzungsoberfläche kann bereits im Framework implementiert werden. In Abbildung 3 ist dargestellt, welche Teile eines interaktiven Werkzeugs aus dem Framework kommen, und welche für die jeweilige Anwendung zu programmieren sind. Alle grau gekennzeichneten Bestandteile werden vom Framework zur Verfügung gestellt.

Das Verhältnis von Entwicklungsumgebungen zu Frameworks

Der Einsatz von Entwicklungsumgebungen ist heute unabdingbar für die Entwicklung großer Anwendungen. Wie schon erwähnt, erleichtern GUI-Builder die Erstellung von Benutzungsoberflächen erheblich. Allerdings können gerade GUI-Builder großen Einfluß auf die Architektur der Anwendung nehmen. Wenn der verwendete GUI-Builder Quellcode generiert, sind in diesem immer Annahmen über die Art der Verwendung

enthalten. Baut man seine Anwendungen ‚naiv" auf diesem generierten Quellcode auf, so werden sich große Teile der Anwendung in ihrer Struktur nach dem generierten Code richten (müssen). Dadurch beeinflußt der GUI-Builder die Anwendung in einem Maße, das weit über den eigentlichen interaktiven Anteil hinausgeht. Die Anwendung wird insgesamt abhängig von dem verwendeten GUI-Builder.

Ein Framework sollte also die Möglichkeit bieten, prinzipiell jeden GUI-Builder zu verwenden. Dies kann durch eine geeignete Anbindung des generierten Quellcodes erfolgen oder durch das Abspeichern und Einlesen von Ressourcen. Im JWAM-Framework haben wir beide Varianten realisiert. Das Framework unterstützt jeden beliebigen Java-GUI-Builder, indem die Speicherung der im GUI-Builder erzeugten Oberflächen durch ein im Framework mitgeliefertes JavaBean übernommen wird. Schließlich kann der Zugriff auf die erzeugte Oberfläche über Interaktionsformen gekapselt werden. Das Framework übernimmt die Aufgabe, die gespeicherte Oberfläche einzulesen bzw. die Oberfläche aus dem generierten oder manuell programmierten GUI-Quellcode zu erzeugen. Weiterhin stellt das Framework die Mechanismen bereit, mit deren Hilfe die Interaktionsformen mit den passenden Präsentationsformen verbunden werden können. Die Zuordnung erfolgt dabei über Namen. Jede Interaktionsform hat genau einen Namen, und jede Präsentationsform kann - je nach unterstützter Interaktionsform - einen Namen haben.[3]

Das Konzept der Interaktionsformen, wie es im JWAM-Framework implementiert ist, kann nahtlos mit allen gängigen Entwicklungsumgebungen verwendet werden. Die Präsentationsformen sind im Rahmen des Java-Komponentenmodells als Java-Beans entwickelt. So lassen sich die Präsenta-tionsformen entweder im GUI-Builder oder in selbst implementierten Oberflächen einfach benutzen. Grundsätzlich ist dies ein Vorteil des standardisierten Komponentenmodells. Es ermöglicht, Präsentationsformen unabhängig von der einzelnen Entwicklungsumgebung zu entwickeln und zu verwenden. Diese Eigenschaft ist auch auf andere Komponentenmodelle in anderen Programmiersprachen übertragbar.

Implementierung der Interaktionsformen

Im JWAM-Framework haben wir das Konzept der Interaktionsformen mit Java realisiert. Wir haben uns entschieden, für jede Interaktions- und jede Präsentationsform ein Objekt zu erzeugen. Es existiert also eine Klassenhierarchie für Interaktionsformen (siehe Abb. 4) und eine Schnittstellenhierarchie für Präsentationsformen (siehe Abb. 5).

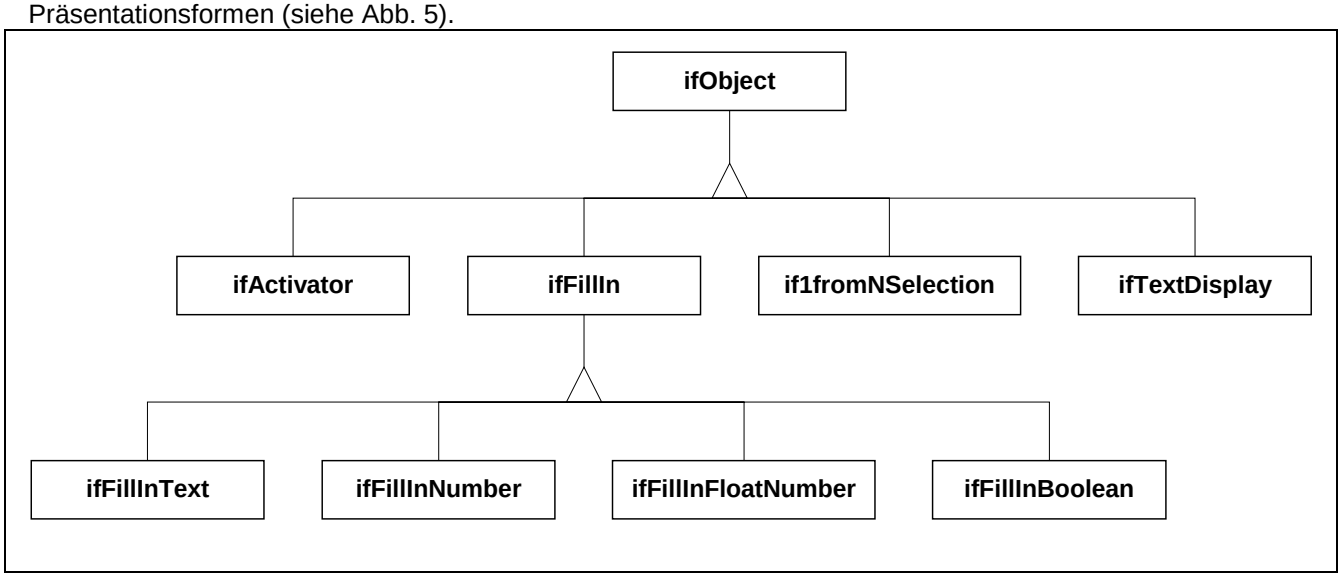

Abbildung 4: Klassenhierarchie der Interaktionsformen im JWAM Framework

[3] Eine ausführlichere Erläuterung der Probleme bei der Anbindung von GUI-Buildern sowie eine sehr einfache, java-spezifische Lösung zeigen wir in der aktuellen Ausgabe des JavaSpektrums (siehe [Fri99]).

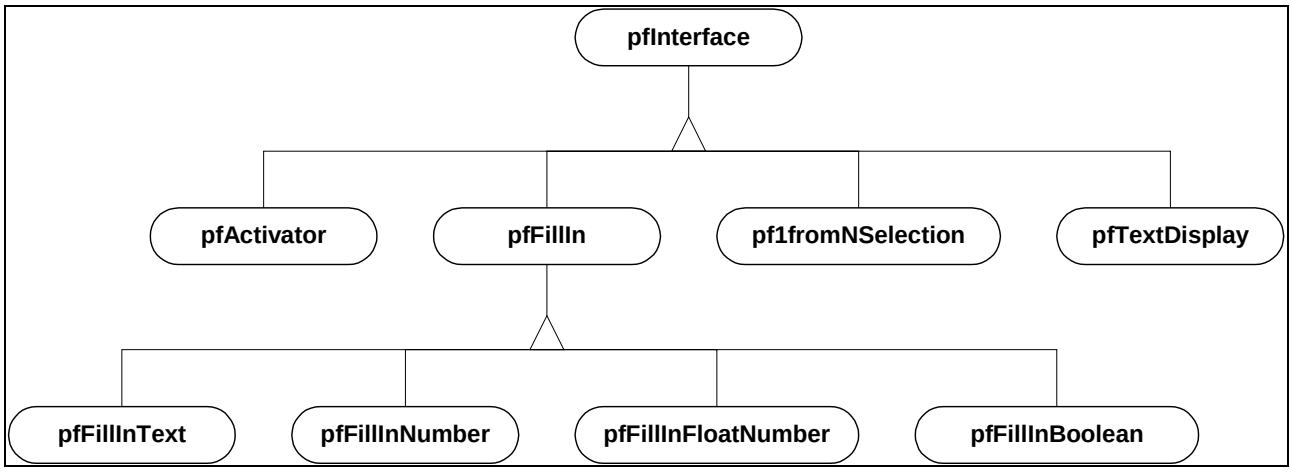

Abbildung 5: Interfacehierarchie für die Anbindung von Präsentationsformen im JWAM Framework

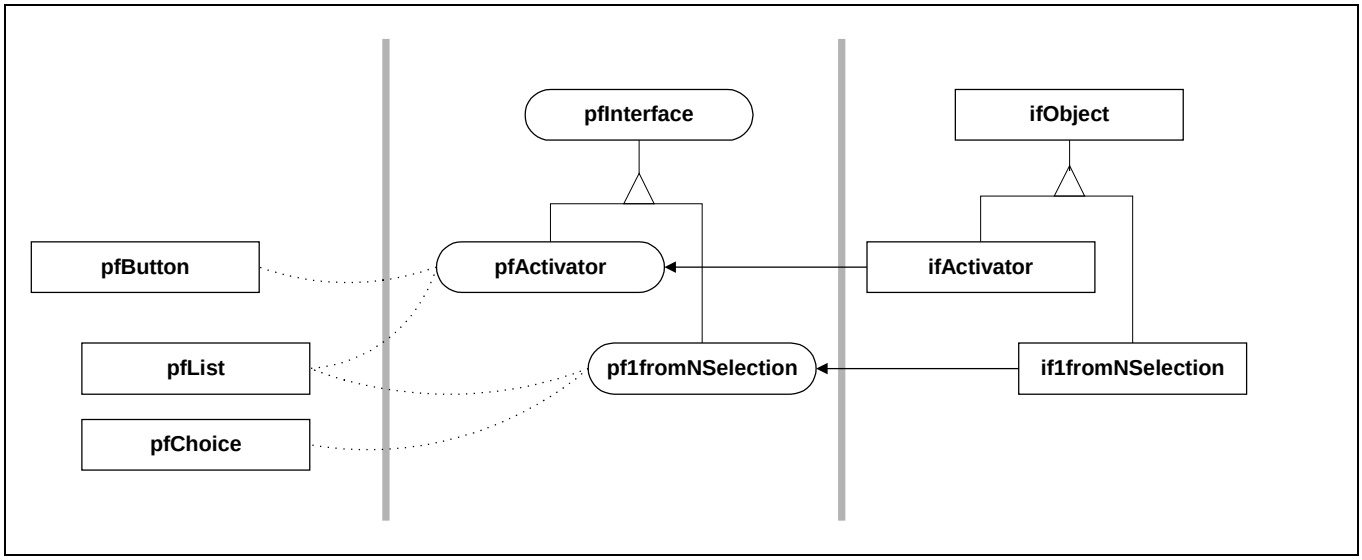

Abbildung 6: Verbindung von Präsentation und Interaktion im JWAM Framework

Die Hierarchie der Interaktionsformen besitzt das Wurzelobjekt ifObject. In ihm ist die allgemeine Schnittstelle der Interaktionsformen definiert. Dies umfaßt insbesondere einen Mechanismus, der später zur Verknüpfung mit Präsentationsformen verwendet wird. Die Präsentationsformen implementieren Interfaces, über die sie mit den Interaktionsformen verbunden werden können. Die Verbindung erfolgt, wie oben beschrieben, über Namen. Diese Verbindung wird beispielhaft in Abbildung 6 gezeigt. Die Abbildung zeigt nur die Benutzt-Beziehung zwischen den Interaktions- und Präsentationsformen. Für den Callback-Mechanismus von der Präsentations- zu der Interaktionsform haben wir ein modifiziertes Befehlsmuster verwendet.

Bewertung

Das Konzept der Interaktionsformen hat im JWAM-Framework vom Übergang des JDK 1.0 auf JDK 1.1 getragen und funktioniert auch bei der aktuell durchgeführten Umstellung von AWT auf Swing sehr gut. Obwohl die Schnittstellen der verwendeten technologischen Basis sich radikal verändert haben, sind die der Interaktionsformen vollständig unverändert geblieben. Wir mußten lediglich die Präsentationsformen entsprechend anpassen. Werden sehr viele Oberflächenelemente unterstützt, müssen auch sehr viele Präsentationsformen angepaßt werden. In der Regel ist die Anzahl der Werkzeuge, die mit dem Framework erstellt wurden, sehr viel größer als die der Präsentationsformen. Daher ist die Änderung der Präsentationsformen mit vertretbarem Aufwand verbunden. Der Zugriff auf die konkrete Darstellung mittels direktem Zugriff auf die Präsentationsformen war nur selten notwendig. Der Großteil der von uns entwickelten Werkzeuge konnte problemlos allein mit Interaktionsformen realisiert werden. Neben den hier

vorgestellten Implementierungen wurde das Konzept in einem universitären und einem industriellen Projekt erfolgreich in C++ implementiert.

Ausblick

In diesem Artikel haben wir grundsätzliche Probleme bei der Anbindung von Benutzungsoberflächen sowie der Anbindung von Entwicklungsumgebungen an Frameworks vorgestellt. Der Aufwand für die Erstellung von Interaktions- und Präsentationsformen scheint auf den ersten Blick erheblich und vor allem für kleinere Unternehmen finanziell nicht gerechtfertigt. Unsere Erfahrung in industriellen Projekten hat aber gezeigt, daß der Aufwand wesentlich höher ist, wenn die Umstellung bereits existierender Anwendungen erfolgen muß, weil der Hersteller der GUI-Bibliothek diese nicht mehr weiterentwickelt. Dies ist in uns bekannten Projekten im Bereich der C++-Toolkits mit ‚OWL", ‚CommonView" und ‚zApp" geschehen. Für in Java implementierte Projekte ohne eine entsprechende Kapselung stellt der Übergang von der ersten AWT-Bibliothek zu der neueren AWT-Version, zu Swing oder zu AFC von Microsoft einen ähnlichen Bruch dar.

Im nächsten Teil dieser Serie werden wir uns mit Fachwerten und deren Zusammenhang mit Frameworks beschäftigen.

Weitergehende Informationen

Das Konzept der Interaktionsformen wurde von einer Gruppe wissenschaftlicher Mitarbeiter und Studenten am Arbeitsbereich Softwaretechnik des Fachbereichs Informatik der Universität Hamburg ausgearbeitet. Diese Gruppe bestand aus Wolf-Gideon Bleek, Thorsten Görtz, Keno Hamer, Carola Lilienthal, Martin Lippert, Stefan Roock, Ulfert Weiß, Wolfgang Strunk, Henning Wolf. Diese Gruppe arbeitet zur Zeit an einer Fachbereichsmitteilung, in dem das Konzept der Interaktionsformen ausführlich beschrieben wird ([Ble99a]). Die Fachbereichsmitteilung kann über die Autoren dieses Artikels bezogen werden.

Literatur

[Bäu98] D. Bäumer, Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme, Dissertationsschrift am FB Informatik der Uni Hamburg, 1/98

[Ble99a] W.-G. Bleek, T. Görtz, C. Lilien-thal, M. Lippert, W. Strunk, S. Roock, H. Wolf, U. Weiß, Interaktionsformen zur flexiblen Anbindung von Benutzungsoberflächen, Mitteilung des FB Informatik FBI-HH-M 285/99, Uni Hamburg. 1999

[Ble99b] W.-G. Bleek, G. Gryczan, C. Lilienthal, M. Lippert, S. Roock, H. Wolf, H. Züllighoven, Frameworkbasierte Anwendungsentwicklung (Teil 1), in: OBJEKTspektrum 1/99

[Fri99] N. Fricke, C. Lilienthal, M. Lippert, S. Roock, H. Wolf, Erlöst von allem Übel? GUI-Builder-Unabhängigkeit, in: JavaSpektrum 2/99

[Gam98] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Entwurfsmuster : Elemente wiederverwendbarer objektorientierter Software, 2. Auflage, Addison-Wesley 1998

[Gry99b] G. Gryczan, C. Lilienthal, M. Lippert, S. Roock, W. Strunk, H. Wolf, H. Züllighoven, Frameworkbasierte Anwendungsentwicklung (Teil 2): die Konstruktion interaktiver Anwendungen, in: OBJEKTspektrum 2/99

[Gör98] T. Görtz, Abstraktion der GUI-Komponente in einem objektorientierten Rahmenwerk, Diplomarbeit, Uni Hamburg, 1998

[JWAM99] JWAM: Java Framework for the Tools and Materials Approach, Uni Hamburg, FB Informatik, Arbeitsbereich Softwaretechnik, 1999 (siehe http://swt-www.informatik.uni-hamburg.de/Software/JWAM)

[Lip97] M. Lippert, Konzeption und Realisierung eines GUI-Frameworks in Java nach der WAM-Metapher, Studienarbeit, Uni Hamburg, 1997 (siehe: http://swt-www.informatik.uni-hamburg.de/~4lippert)

[Zül98] H. Züllighoven, Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz, dpunkt-Verlag, 1998