Der folgende Artikel ist im OBJEKTspektrum 1/99, S. 90 - 99 erschienen.

Frameworkbasierte Anwendungsentwicklung (Teil 1)

Guido Gryczan, Carola Lilienthal, Martin Lippert, Stefan Roock, Henning Wolf, Heinz Züllighoven

Zusammenfassung

Frameworks sind ein wichtiger Bestandteil bei der Entwicklung großer objektorientierter Softwaresysteme. Mit ihrem Einsatz ist die Hoffnung auf gesteigerte Produktivität, kürzere Entwicklungszeiten und höherer Qualität von Anwendungsentwicklungen verbunden. Dieser Beitrag bildet den Auftakt zu einer neuen Serie, in der es darum gehen wird, verschiedene Aspekte der framework-basierten Anwendungsentwicklung - wie die Realisierung von Werkzeugen oder Fragen der Verteilung und der Persistenz - detaillierter zu behandeln. Diesmal wollen wir unsere grundsätzliche Herangehensweise bei der Konstruktion objektorientierter, framework-basierter Anwendungen beschreiben. Die in der Artikelserie verwendeten Begriffe sind in einem Glossar am Ende des Beitrags zusammengestellt.

Motivation

Die Autoren dieses Artikels befassen sich praktisch und konzeptionell mit der professionellen Entwicklung großer Softwaresysteme, wobei interaktive Anwendungssoftware im Vordergrund steht (vgl. [Flo97]). Dabei vertreten wir die Überzeugung, daß Softwareentwicklung und die Anwendung softwaretechnischer Methoden und Techniken für Unternehmen stets nur ein Mittel sein können, um für deren Kunden eine erfolgreiche Dienstleistung zu erbringen. Eine so verstandene Softwareentwicklung und die dazugehörige Technik nennen wir anwendungsorientiert.

Anwendungsorientierung steht heute im Spannungsfeld zwischen individuell zugeschnittener Software und der Forderung nach effizienten und kostensparenden Entwicklungsprozessen. Aus diesem Dilemma kann die Verwendung und die Konstruktion von *Application-Frameworks* heraushelfen. Nun sind Application-Frameworks meist auf einen bestimmten Anwendungsbereich zugeschnitten. Da wir aber in Kooperationsprojekten unterschiedliche Anwendungsbereiche unterstützen müssen, stellt sich für uns die Frage, welche Teile eines Application-Frameworks sich auch ohne einen konkreten Einsatzkontext generisch bereitstellen lassen.

Betrachten wir zunächst die Fragen, die bei der Entwicklung von Application-Frameworks wesentlich sind:

- Wie können wir *Strukturähnlichkeit* von anwendungsfachlichen und technischen Konzepten erreichen, um den Zusammenhang zwischen Gegenständen und Konzepten der Anwendungswelt und denen der technischen Welt zu verdeutlichen?
- Wie können wir sicherstellen, daß der fachliche Kern von Anwendungssoftware möglichst geringfügig durch die Verwendung einer zugrundeliegenden Technologie - wie etwa einem spezifischen Fenster- oder Datenbanksystem - beeinflußt wird?
- Welche Anteile eines Anwendungssystems können wir so weit konstruieren, daß sie als vorgefertigte Komponenten in verschiedenen konkreten Anwendungen wiederverwendet werden können?
- Wie können wir für Entwickler sicherstellen, daß sie in einem großen Application-Framework Entwurfsentscheidungen lokalisieren können? Anders gefragt: Welche Hilfen können wir Entwicklern geben, damit sie feststellen können, wo welche Entwurfsentscheidung getroffen wurde?

Ein einführendes Beispiel soll die Relevanz dieser Fragestellungen für die Anwendungsentwicklung verdeutlichen.

Ein Beispiel

Das fachliche Konzept eines ‚Kontos" im Bankenbereich kann unabhängig von der zugrundeliegenden

Technologie und der Art seiner Handhabung modelliert werden. Dabei sollte das fachliche Modell des Kontos aber strukturähnlich zu seiner technischen Realisierung in einer Klassenhierarchie sein, damit Veränderungen des fachlichen Konzepts ‚Konto" möglichst direkt auf seine technische Realisierung bezogen werden können. Ein so modelliertes Konto soll sowohl am Arbeitsplatz eines Kundenberaters mit Werkzeugen als auch beispielsweise an einem Geldautomaten verwendet werden können.

Grundsätzlich wirken die drei folgenden Größen auf die Anwendungsentwicklung ein:

- Der Anwendungsbereich ist von primärer Bedeutung - er bildet den Ausgangspunkt der Modellierung. Wesentliches Anliegen dabei ist, ein fachlich motiviertes Modell strukturähnlich in Softwarekomponenten zu übertragen. Dies mag am Beispiel des Kontos im ‚Kleinen" einfach und einsichtig sein; im weiteren werden wir die Frage beantworten, wie diese Strukturähnlichkeit auch für große Systeme aufrechterhalten werden kann.
- Die verwendete Technik hat maßgeblichen Einfluß auf unsere Vorstellung von dem, was überhaupt sinnvoll im Anwendungsbereich modellierbar ist. Hier tut sich ein wichtiges Spannungsfeld auf: Einerseits wollen wir das fachliche Modell unabhängig von einer bestimmten Technik erstellen. Andererseits ist die verwendete Technik im Wortsinne der begrenzende Rahmen für die fachliche Modellierung. So ist es z. B. schlicht sinnlos, fachlich die Weitergabe von Unterlagen über Postkörbe zu modellieren, wenn technisch keine Möglichkeit der Verteilung zur Verfügung steht.
- Der Einfluß der Handhabung und Präsentation bei der Gestaltung eines Anwendungssystems steigt. Hier haben sich in den letzten Jahren etwa durch das Internet (Stichwort: Home-Banking) tiefgreifende Änderungen ergeben, die die Architektur von Anwendungssystemen beeinflussen.

Frameworks müssen bei der Anwendungsentwicklung möglichst stabil sein, denn jede Veränderung am Framework beeinflußt potentiell alle damit entwickelten Anwendungssysteme. Gleichzeitig muß sich ein Framework aber ständig an die sich verändernden Umstände anpassen lassen. Wir haben uns daher mit den genannten Einflußgrößen auf die Dimensionen festgelegt, in denen ein Framework möglichst einfach und folgenlos verändert werden kann. Entwickler sollen außerdem einfach feststellen können, wo welche Entwurfs- und Designentscheidungen getroffen worden sind, und wo neue Funktionalität im Framework angesiedelt werden kann.

Eine solche grundlegende Strukturierung eines Softwaresystems nennen wir eine *Modellarchitektur*. Im folgenden Abschnitt stellen wir unsere Modellarchitektur für ein Application-Framework nach dem *Werkzeug/Automat/Material-Ansatz* (Abk. WAM, vgl. [Zül98]) vor. Bei der Konstruktion des Application-Frameworks JWAM (Abk. für Java-Rahmenwerk und WAM) haben wir diese Modellarchitektur verwendet, um die Teile bereitzustellen, die generisch jenseits eines konkreten Anwendungsbereichs sind. Die Modellarchitektur gibt damit Anleitung für die technische Realisierung interaktiver Anwendungssoftware nach dem WAM-Ansatz. Sie weist zudem die ‚Ansatzstücke" aus, an die eine anwendungsspezifische Konkretisierung anknüpfen kann.

Bei der Entwicklung interaktiver Anwendungssoftware ist es sehr hilfreich, wenn Entwickler und spätere Anwender eine gemeinsame ‚Vision" über das Anwendungssystems haben. Um diese Vision herzustellen, verwenden wir die Metaphern ‚Werkzeug" und ‚Material". Dies stimmt mit unserer Alltagserfahrung überein: In den meisten Arbeitssituationen kann intuitiv zwischen den Gegenständen unterschieden werden, die bearbeitet werden (Materialien), und den Gegenständen, die Arbeitsmittel sind (Werkzeug). Ein Werkzeug unterstützt wiederkehrende Arbeitshandlungen. Es kann sinnvoll für verschiedene Zwecke und Aufgaben verwendet werden. Die Handhabung eines Werkzeugs wird immer durch einen Benutzer veranlaßt. Materialien sind in diesem Ansatz die Gegenstände und Ergebnisse unserer Arbeit. Sie stellen die fachliche Funktionalität eines Anwendungs- bereichs dar. Ein Material wird mit dafür geeigneten Werkzeugen bearbeitet. Welche Funktionalität ein Material besitzt, wird durch seine Schnittstelle definiert. Oft verwenden wir aber statt Werkzeugen auch (kleine) Automaten, die uns lästige Routinearbeiten abnehmen. Ein *Leitbild* hilft uns in unserem Ansatz zu verdeutlichen, welche generelle Perspektive wir bei der Anwendungsentwicklung einnehmen. Unser vorherrschendes Leitbild ist der gut ausgestattete Arbeitsplatz für eigenverantwortliche, kooperative Tätigkeiten. An diesem Arbeitsplatz sollen Werkzeuge, Automaten und Materialien so bereitgestellt werden, daß ein qualifizierter Anwender seine Arbeitsaufgaben situationsabhängig erledigen kann.

Eine framework-basierte Modellarchitektur

Zunächst verdeutlichen wir die gerade genannten Forderungen und Prinzipien an der framework-basierten WAM-Modellarchitektur, die wir generell innerhalb des WAM-Ansatzes verwenden. Sie beruht auf umfangreichen Konstruktionserfahrungen im Bankenbereich (vgl. [Bäu98]). Die WAM-Modellarchitektur ist auf die Entwicklung interaktiver Anwendungssoftware unter Verwendung von Frameworks ausgerichtet. Grundsätzlich gilt für uns dabei: Ein Application-Framework kann nicht für beliebige Anwendungen bereitgestellt werden. Vielmehr muß deutlich sein, welche Klasse von Anwendungen durch die Architektur der Application-Frameworks effizient unterstützt werden kann. Häufig spricht man in diesem Zusammenhang auch von Domänen.

Ein Application-Framework ist im Sinne dieser Begriffsbildung also dazu bestimmt, generische Lösungen für eine fachlich eingegrenzte Domäne zur Verfügung zu stellen. Da ein Application-Framework naturgemäß recht komplex sein muß, besteht es selbst aus Frameworks. Nehmen wir ein Beispiel: Das bereits erwähnte Konto ist Teil eines bankfachlichen Application-Frameworks, ebenso wie ein Überweisungswerkzeug oder ein EC-Karten-Automat.

Die allgemeine Konstruktion eines interaktiven Werkzeugs ist aber nicht vollständig anwendungsspezifisch, sondern kann im Rahmen einer bestimmten Vorstellung von der Handhabung und Präsentation eines Anwendungssystems teilweise vorformuliert werden. Dies heißt z. B., daß die Konstruktion interaktiver Werkzeuge unterstützt wird, indem der Zusammenhang der Komponenten eines Werkzeugs bereits in einem Werkzeug-Framework beschrieben wird. Darüber hinaus kann ein allgemeines Schreibtisch-Konzept zur Verfügung gestellt werden. In dieses Schreibtisch-Konzept können dann Werkzeuge nach einem einfachen Schema eingehängt werden. Entwickler interaktiver Werkzeuge werden dadurch von einer lästigen und fehleranfälligen Routineanbindung befreit, ohne daß bereits fachliche Details des Anwendungsbereichs bekannt sein müssen. Konstruktiv erreichen wir dies, indem wir *Black-Box*- und *White-Box Frameworks* als Lösungen für ähnliche Entwicklungs- und Programmierprobleme in einem bestimmten Kontext vorgeben (siehe auch [Bir95]).

In der Modellarchitektur werden Frameworks über Verbindungsstücke miteinander und mit den neu zu schreibenden Teilen der Anwendungssoftware verbunden.

Mögliche Verbindungsstücke zwischen den Komponenten einer Modellarchitektur sind

- die Benutzt-Beziehung und
- die Vererbungsbeziehung.

Aber auch diese Verbindungen sind nicht beliebig wählbar. Um die Art und Weise der Verbindung für bestehende und neu zu entwickelnde Komponenten klarzustellen, verwendet man häufig Entwurfsmuster, etwa das Brücken-Muster (vgl. [Gam98]).

Eine weitere Strukturierungsmöglichkeit für Application-Frameworks ist ihre Organisation in Schichten. Für unsere WAM-Modellarchitektur unterscheiden wir zwei Arten von Schichten: protokollbasierte und objektorientierte (vgl. [Zül98]).

In einer protokollbasierten *Schichtenarchitektur* sind die Schichten hierarchisch angeordnet. Dabei liegt die Basismaschine als anwendungsfernste Schichten unten, die anwendungsnäheren Schichten liegen weiter oben. Jede Schicht stellt nach oben eine Schnittstelle zur Verfügung und kapselt dadurch die darunterliegende Implementierung. Ziel dieser Anordnung ist es, daß höhere Schichten ausschließlich die Schnittstelle ihrer nächst niederen verwenden, und daß darunterliegende Schichten ‚unsichtbar" sind. Ein bekanntes Beispiel einer protokollbasierten Schichtenarchitektur ist das 7-Schichten-OSI-Modell (vgl. [Ker89]).

Objektorientierte Schichten sind neben der Benutzt-Beziehung durch die Vererbungsbeziehung miteinander verbunden. Auch hier sind die höheren Schichten anwendungsnäher als die niederen Schichten, allerdings in dem Sinne, daß sie Konkretisierungen und Spezialisierungen der Konzepte in niederen Schichten realisieren. Vererbt wird nur innerhalb einer Schicht sowie von den niederen zu den höheren Schichten. Die Benutzt-Beziehung geht aber - wie bei der protokollbasierten Schicht - von oben nach unten. Meist werden objektorientierte Schichten auch nicht so strikt benutzt wie die protokollbasierten, d. h. es ist durchaus möglich, von einer höheren Schicht abstrakte Konzepte einer nicht unmittelbar darunterliegenden Schicht zu erben.

Abbildung 1 zeigt die Schichten der WAM-Modellarchitektur, deren Auswahl und Anordnung wir in diesem Abschnitt erläutern. Diese Schichtenarchitektur stellt eine logische Anordnung dar. Sie zeigt keine

Aufteilung in einzelne Prozesse oder auf verschiedene Maschinen. Damit unterscheidet sie sich von der vorherrschenden Interpretation der sogenannten *Drei-Schichten-Achitektur*. Diese logische Modellarchitektur soll dem Entwickler eine prinzipielle Orientierung über den Ort von Entwurfsentscheidungen geben.

Für die graphische Darstellung der Modellarchitektur haben wir eine U-Form - bestehend aus den Schichten Systembasis, Technologie sowie Handhabungs- und Präsentationsschicht - gewählt. Die U-Form soll verdeutlichen, daß mit diesen Schichten ein Rahmen für die fachliche Anwendungsentwicklung vorgegeben wird. In diesen Rahmen können die Teile eines konkreten Anwendungssystems sowie weitere Komponenten eines sich entwickelnden Application-Frameworks eingepaßt werden.

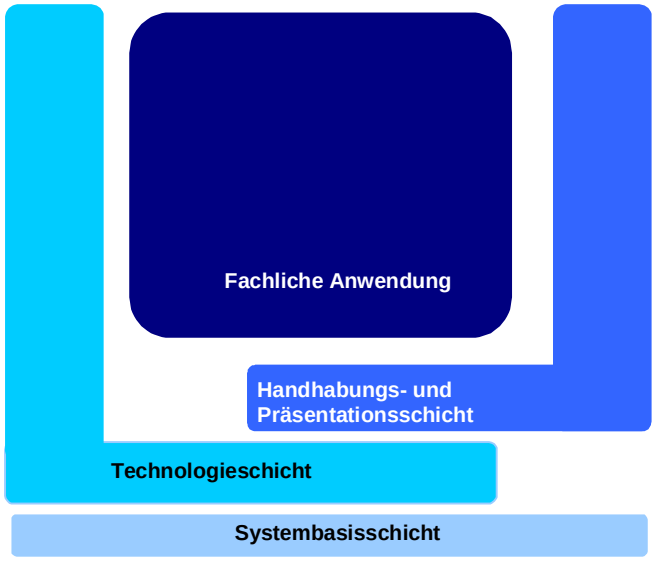

Abbildung 1: Die Schichten der WAM-Modellarchitektur (vereinfacht)

In den folgenden Abschnitten erläutern wir, wie die Schichten unserer Modellarchitektur prinzipiell aufgebaut sind:

Systembasisschicht

Die Systembasis kapselt als *protokollbasierte Schicht* alle technischen Komponenten, die für die Realisierung des Anwendungssystems und seine Anbindung an existierende Systeme notwendig sind:

- Betriebs- und Fenstersystem, eine eventuell eingesetzte Client/Server-Middleware oder persistente Datenspeicher, wie etwa relationale Datenbanken,
- allgemeine Komponenten zur objekt-orientierten Konstruktion, wie etwa Behälterklassen, einen Mechanismus zur Speicherbereinigung oder
- Serialisierungsmechanismen zur Transformation von Objektgeflechten in flache Strukturen und zu deren umgekehrter Wiederherstellung; aus technischen Gründen wird es oft erforderlich sein, hier ein *Metaobjektprotokoll* anzusiedeln.

Die Frameworks und Klassenbibliotheken dieser Schicht sind vollkommen unabhängig von einem speziellen Anwendungsbereich. Sie können daher zur Konstruktion verschiedenster interaktiver Anwendungssysteme eingesetzt werden. Durch die einzelnen Komponenten dieser Schicht werden meist Fremdprodukte und Standardsoftware gekapselt. Deshalb ist hier mit Änderungen zu rechnen (z. B. Wechsel der Fenstersystembibliothek von ‚AWT" nach ‚Swing"). Entsprechend muß diese Schicht besonders sorgfältig protokollbasiert realisiert werden, d. h. ein direkter Zugriff auf die gekapselten Komponenten ist zu vermeiden. Darüber hinaus sollten die Schnittstellen dieser Kapseln selbst von den darunterliegenden Implementierungen abstrahieren. Die Komponenten sind über Black-Box-Frameworks und Klassenbibliotheken im Sinne einer API gekapselt und damit für die darüberliegenden Schichten nicht sichtbar. Benutzt werden die Komponenten dieser Schicht insbesondere von der Technologieschicht.

Technologieschicht

Die Zweck der Technologieschicht kann an einem Beispiel erläutert werden: Wir nehmen an, daß die Systembasisschicht eine relationale Datenbank kapselt. Dann stellt sich die Frage, inwieweit dieser Umstand für anwendungsspezifische Schichten deutlich werden soll. Durch die Einführung der Technologieschicht treffen wir hier eine Entscheidung: Entweder wir stellen in der Technologieschicht ein allgemeines relationales Modell zur Verfügung, oder wir modellieren einen fachlich motivierten Persistenzdienst für heterogene Datenhaltungssysteme. Mit anderen Worten, implementieren wir in dieser Schicht ein anwendungsnäheres Modell über der zugrundeliegenden Basistechnik.

Die Technologieschicht enthält so Modelle der verwendeten Technik. Das Verhältnis zwischen dieser Schicht und der Systembasis wird durch das häufig verwendete Brücken-Muster charakterisiert. Während sich die verschiedenen Schnittstellen von Implementierungskomponenten in der Systembasisschicht befinden, werden hier die technologisch allgemeineren Konzepte versammelt. Die Technologie-schicht (objektorientierte Sicht) besteht größtenteils aus White-Box-Frameworks, die vereinzelt um erweiterbare Black-Box-Frameworks ergänzt werden, um bereits vorgefertigte Standardlösungen anzubieten. Innerhalb dieser Schicht finden sich beispielsweise:

- ein Framework zum Speichern und Laden von Objekten in und aus einem *Persistenzmedium*, d. h. ein allgemeines Datenhaltungskonzept (vgl. [Rie98]), und
- ein Framework zur Kommunikation mit anderen Umgebungen oder Prozessen, d. h. ein allgemeines Kommunikationskonzept.

Auch diese Schicht kann weitgehend anwendungsneutral verwendet werden. Allerdings sind hier schon bestimmte Verwendungszusammenhänge von technologischen Konzepten festgelegt, die den Charakter der zu entwickelnden Systeme prägen. So wird etwa mit dem Persistenzkonzept und der gewählten Kommunikationsart die Domäne der mit dem Application-Framework entwickelbaren Systeme weiter eingegrenzt. Aber auch die Art und Weise der Handhabung und Präsentation ist vorgeprägt. Unter Verwendung der bisher vorgestellten Systembasis- und Technologieschicht lassen sich interaktive und vernetzte Arbeitsplatzsysteme effizient entwickeln. Mit der im folgenden vorgestellten Handhabungs- und Präsentationsschicht grenzen wir diesen Bereich weiter ein: Dort stellen wir Frameworks bereit, die spezifisch auf den WAM-Ansatz zugeschnitten sind.

Handhabungs- und Präsentationsschicht

Das zentrale Merkmal der WAM-Modellarchitektur ist die Konstruktion interaktiver Anwendungssoftware unter Verwendung der *Entwurfsmetaphern* ‚Werkzeug", ‚Automat" und ‚Material". Dabei sind Werkzeuge und Automaten die Arbeitsmittel, mit denen am Arbeitsplatz umgegangen wird, um Materialien (Arbeitsgegenstände) zu bearbeiten. Für Entwickler bedeutet dies, daß sie mit Werkzeugen und Automaten die Art des Umgangs mit Materialien festlegen. Bei der Konstruktion von Werkzeugen unterscheiden wir zwei Anliegen:

- die Handhabung: Welche Funktionalität bietet das Werkzeug an, um das Material unterschiedlich zu bearbeiten?
- die Präsentation: Wie wird die Funktionalität des Werkzeugs an der Benutzungsoberfläche präsentiert?

Die Schicht ‚Handhabung und Präsentation" umfaßt Frameworks, die den wiederverwendbaren Anteil der Konstruktion interaktiver Anwendungssoftware implementieren. Als wiederverwendbar charakterisieren wir z. B. die Anteile eines Werkzeugs, die nicht die fachliche Funktionalität betreffen. Dazu gehört etwa die Implementierung des Kontrollflusses innerhalb eines Werkzeugs nach dem Beobachter-Muster (vgl. [Gam98]) oder ein ausimplementiertes Konzept zur Erzeugung von Werkzeugen und Sub-Werkzeugen.

Da die Frameworks dieser Schicht sowohl Arbeitsgegenstände (z. B. Ordner) als auch Konzepte im hier verwendeten Sinn (z. B. Kopplung zwischen Werkzeug und Material) definieren, muß diese Schicht eine objektorientierte Schicht sein.

Mit Bezug auf die WAM-Modellarchitektur gehören in diese Schicht Frameworks, die

- das jeweils gewählte Umgebungskonzept mit elektronischem Schreibtisch realisieren,
- den Umgang mit Strukturierungselementen (Behältern, Mappen und Stapeln) zur Organisation des Arbeitsplatzes unterstützen,

- Standardimplementierungen der Entwurfsmuster für Werkzeug, Material und Automaten bereithalten sowie
- generische Werkzeugkomponenten (wie Auflister) implementieren.

Die Handhabungs- und Präsentationsschicht greift auf die Technologieschicht und partiell auf die Systembasisschicht zu. Die hier beschriebene Schicht ist noch nicht an einem konkreten Anwendungsbereich orientiert und kann daher für alle Systeme verwendet werden, die dem gewählten Leitbild und den Entwurfsmetaphern entsprechen.

Das JWAM-Framework
Das JWAM-Rahmenwerk wird seit 1½ Jahren am Arbeitsbereich Softwaretechnik der Universität Hamburg von Studenten und wissenschaftlichen Mitarbeitern (weiter-)entwickelt. Die erste Version (JWAM 1.0) basierte auf dem JDK 1.1 und diente als Grundlage für Studien- und Diplomarbeiten. Im April 1998 haben wir die Version 1.1 freigegeben. Diese Version wurde und wird weiterhin in Studien- und Diplomarbeiten verwendet, zusätzlich jedoch auch in verschiedenen Lehrveranstaltungen. Die Version 1.2 wird auf dem JDK 1.2 basieren und ist für Anfang 1999 geplant. Das Framework wird auch in Kooperationsprojekten mit der Industrie eingesetzt. Das Framework enthält in der Version 1.1 40 Packages, 150 Klassen und 44 Interfaces.
Informationen über das JWAM-Rahmenwerk können über das Internet bezogen werden, unter der Adresse:
http://swt-www.informatik.uni-hamburg.de/Software/JWAM

Die JWAM-Architektur

Das JWAM-Framework realisiert die drei technischen Schichten Systembasisschicht, Technologieschicht sowie Handhabungs- und Präsentationsschicht der WAM-Modellarchitektur. Es kann zu einem domänenspezifischen Framework spezialisiert werden, indem die entsprechenden fachlichen Schichten auf die vorhandenen Schichten aufgesetzt werden.

Das JWAM-Framework unterstützt alle wichtigen nicht-anwendungsspezifischen Aspekte interaktiver Anwendungen. In Abbildung 2 ist die Schichtenarchitektur des JWAM-Frameworks mit den wichtigsten Sub-Frameworks visualisiert.

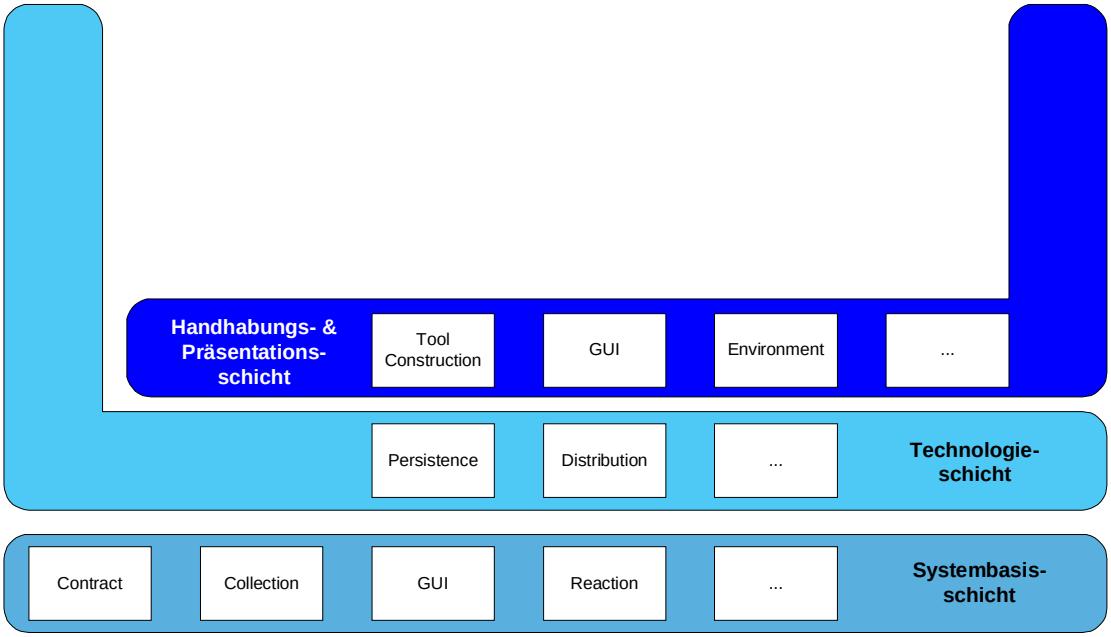

Abbildung 2: Die WAM-Schichtenarchtektur im JWAM-Framework

Die JWAM-Systembasisschicht enthält unter anderem die folgenden Sub-Frameworks:

- Contract: In der Systembasisschicht sind die Mechanismen des *Vertragsmodells* (siehe [Mey97]) für Java realisiert. Das Vertragsmodell wird durchgängig im gesamtem JWAM-Framework verwendet. Somit können Verletzungen von Vor- und Nachbedingungen im Framework sowie in den erstellten Anwendungen schnell entdeckt werden. Darüber hinaus ergänzen die spezifizierten Vor- und Nachbedingungen die Dokumentation von Operationen.

- Collection: Die Java-Standardbibliotheken unterstützen den Umgang mit Mengen gleichartiger Objekte in Behältern nur rudimentär. Neben den Arrays der Programmiersprache stehen lediglich zwei Behälterklassen (Vector, Dictionary) zur Verfügung. In der Systembasisschicht bietet eine Behälterklassenbib-liothek umfangreiche Möglichkeiten zur Verwaltung von Objektmengen.
- GUI: Die grundlegenden Mechanismen zur Präsentation an der Benutzungsoberfläche und zur Interaktion des Benutzers mit der Anwendung werden in der Systembasisschicht gekapselt. Obwohl Java eine plattformunabhängige Programmierung erlaubt, unterliegt die Anbindung der Benutzungsoberfläche ständigen Änderungen. So wurde das Ereignismodell beim Übergang vom JDK 1.0 auf JDK 1.1 komplett umgestellt, und mit dem JDK 1.2 wird ‚Swing" das ‚Abstract Window Toolkit" (AWT) ersetzen. Durch die Kapselung ist es möglich, die Auswirkungen dieser Änderungen auf das Anwendungssystem zu minimieren.
- Reaction: In interaktiven Softwaresystemen sind zur Laufzeit oft bidirektionale Beziehungen zwischen Objekten notwendig. Damit diese sich nicht negativ auf die Architektur des Systems auswirken, sind Mechanismen zur *losen Kopplung* notwendig (siehe [Roo96]). Gamma et al. beschreiben in [Gam98] dazu mehrere Entwurfsmuster (z. B. ‚Beobachter", ‚Befehl"). In der Systembasisschicht haben wir eine Reihe von Reaktionsmechanismen zur Anbindung der Benutzungsoberfläche sowie zum Bau von Werkzeugen vorgesehen.

Die JWAM-Technologieschicht enthält darauf aufbauend die Sub-Frameworks:
- Persistence: In den meisten kommerziellen Anwendungen müssen Objekte persistent gespeichert werden. Dabei werden objektorientierte Anwendungen in der Regel in ein heterogenes technisches Umfeld eingebettet. Daher sehen wir uns häufig mit der Anforderung konfrontiert, Objekte in relationalen Datenbanken zu speichern. In der Technologieschicht befindet sich ein Serialisierungsmechanismus, der das Speichern von Objekten in beliebigen Medien (z. B. relationalen Datenbanken) sowie deren spätere Rekonstruktion mit geringem Aufwand erlaubt.
- Distribution: Mit RMI (‚Remote Method Invocation") oder CORBA bietet Java Mechanismen zur Kommunikation in verteilten Systemen, die den Aufruf von Operationen an entfernten Objekten ermöglichen. Mit Hilfe des Nachrichtenvermittlers des JWAM-Frame- works können Prozesse Nachrichten an andere Prozesse verschicken. Der Nachrichtenvermittler eignet sich besonders, um in Client/Server-Systemen Klienten über Veränderungen am Server zu informieren. Der Nachrichtenvermittler nutzt wahlweise RMI oder CORBA. Schließlich finden sich in der Handhabungs- und Präsentationsschicht die Sub-Frameworks:
- Tool Construction: Nach dem WAM-Ansatz präsentieren sich dem Benutzer Anwendungssysteme als Sammlung von fachlich motivierten Gegenständen. Grundsätzliche Mechanismen zum Bau von Werkzeugen befinden sich in der Handhabungs- und Präsentationsschicht.
- GUI: Die Anbindung der Benutzungsoberfläche ist bei der Konstruktion interaktiver Anwendungen ein wichtiger Punkt. Daher bietet das JWAM-Framework in der Systembasis- sowie der Handhabungs- und Präsentationsschicht ausgefeilte Möglichkeiten, die Benutzungsoberfläche einer Anwendung zu konstruieren und anzubinden. Das GUI-Sub-Framework der Handhabungs- und Präsentationsschicht baut auf dem GUI-Sub-Framework der Systembasisschicht auf. Ziel dieser GUI-Anbindung ist es, beliebige Java-GUI-Builder zusammen mit dem JWAM-Framework zu verwenden, ohne das Framework oder die entwickelten Anwendungen ändern zu müssen.
- Environment: Die Umgebung stellt den Abschluß der Arbeitsumgebungen gegenüber den allgemein zugreifbaren Materialien und den Arbeitsumgebungen anderer Benutzer dar. Sie enthält den elektronischen Schreibtisch (Desktop). Er präsentiert die Werkzeuge, Materialien und die anderen fachlichen Komponenten. Der Anwender kann die angezeigten Piktogramme verschieben, in Behältern organisieren und Werkzeuge mit Materialien öffnen.

Die vorgestellten konzeptionellen Schichten und Sub-Frameworks müssen in geeigneter Form physikalisch organisiert und abgelegt werden. Dafür eignet sich das Package-Konzept von Java.

Klassen und Interfaces werden in Java in den sogenannten Packages organisiert. Jedes Interface und jede Klasse beginnt mit dem Namen des Packages , zu dem sie gehört. Packages können beliebig tief hierarchisch ineinander verschachtelt werden. Der Java-Compiler setzt die Package-Hierarchie in eine korrespondierende Verzeichnisstruktur um. Durch die Packages werden zum einen Namenskonflikte vermieden, da die Package-Namen als zusätzliche Identifikatoren beim Zugriff auf Interfaces und Klassen

verwendet werden können. Zum anderen werden über die Packages Teile der Zugriffsrechte geregelt. Klassen können wahlweise öffentlich oder nur innerhalb eines Packages sichtbar sein, genauso wie einzelne Operationen einer Klasse nur für Klassen desselben Packages zugreifbar sein können.

Korrespondierend zur WAM-Schichtenarchitekur ist die Package-Hierarchie des JWAM-Frameworks aufgebaut. Abbildung 3 zeigt ausschnittsweise die Package-Struktur des JWAM-Frameworks. Die erste Ebene bezeichnet das gesamte Framework mit dem Namen wam. Darunter befinden sich die Schichten aus der WAM-Modellarchitektur: Die Systembasisschicht befindet sich im Package system, die Technologieschicht im Package technology und die Handhabungs- und Präsentationsschicht im Package handling.

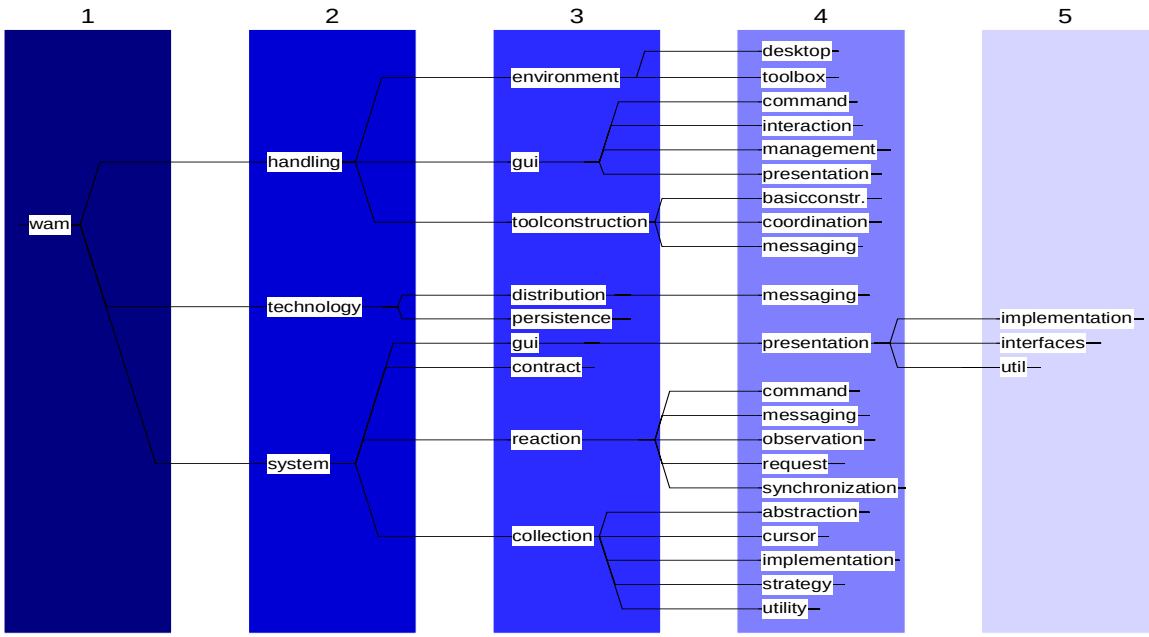

Abbildung 3: Die JWAM-Package-Struktur

Ab der dritten Package-Ebene finden sich die oben beschriebenen Sub-Frameworks mit ihren Interfaces und Klassen. Die weiteren Package-Ebenen dienen der internen Organisation des Frameworks und werden in den nachfolgenden Artikeln dieser Reihe näher erläutert.

Die Anwendungsentwickler können die hier beschriebene technische Funktionalität des JWAM-Frameworks nutzen und sich im wesentlichen auf die fachlichen Anteile ihrer Arbeit konzentrieren. Das JWAM-Framework stellt die Anbindung an die wichtigsten Betriebssystemdienste (Benutzungsoberfläche, Persistenz, Verteilung) sowie die wichtigsten Basisdienste (Vertragsmodell, Behälter) in konsistenter Weise zur Verfügung. Auf dieser Basis definiert das JWAM-Framework Sub-Frameworks für die Unterstützung des WAM-Ansatzes. Anwendungssysteme werden in die Schichten des JWAM-Frameworks eingebettet und nutzen die bereitgestellte Funktionalität. Sollte die bereitgestellte technische Funktionalität nicht ausreichen, so gibt die Schichtenarchitektur bereits einen Rahmen für die Erstellung und Integration weiterer Klassen und Sub-Frameworks vor. Nach unseren Erfahrungen sind Entwickler recht schnell in der Lage, den geeigneten Ort für eine neu entwickelte Klasse zu finden, wenn sie die Schichtenarchitektur einmal verstanden haben.

Ausblick

In diesem Artikel haben wir die grundlegenden Ideen und die Struktur des JWAM-Frameworks beschrieben. In den folgenden Artikeln dieser Reihe greifen wir einzelne Teile wieder auf und erläutern relevante Fragestellungen und Konstruktionsansätze für die Entwicklung von Frameworks. Im einzelnen werden wir die folgenden Themen behandeln:

- Werkzeugkonstruktion und Anbindung von Benutzungsoberflächen: Die Konstruktion interaktiver Anwendungskomponenten ist trotz maßgeschneiderter Entwicklungsumgebungen eine anspruchsvolle technische und fachliche Aufgabe, wenn wir über die simple - an Menüs und Masken orientierte -

Darstellung und Handhabung hinausgehen wollen. Wir diskutieren Entwurfs- und Konstruktionskriterien für den Werkzeugbau.

- Basiskonzepte (Vertragsmodell, Behälter): Vielen Anwendungsentwicklern ist die Bedeutung des Vertragsmodells oder der fachlichen Behälter unklar. Wir erläutern, weshalb diese Basiskonzepte wesentlich die Qualität des Entwurfs und der Konstruktion erhöhen.
- Zusammenspiel von Entwicklungsumgebungen (insbesondere GUI-Builder) und dem JWAM-Framework: Entwicklungsumgebungen beeinflussen die Art und Weise, wie Frameworks entwickelt und eingesetzt werden können.
- Persistenz: Ein nach wie vor spannendes Thema - wir gehen besonders auf heterogene Persistenzmedien und ihre systematische Anbindung ein.
- Verteilung und Kooperation: Noch immer wird Verteilung als ein vorrangig technisches Problem gesehen. Wir stellen ein aus dem Anwendungskontext abgeleitetes Verteilungskonzept vor, das die unterschiedlichen Kooperationsformen berücksichtigt.
- Vorgehensweise bei der Erstellung und Verwendung von Frameworks: Über die Besonderheiten des Framework-Prozesses ist bisher nur wenig geschrieben worden. Dabei zeigt sich schnell, daß hier andere Dinge zu berücksichtigen sind, als bei einem ‚normalen" Projekt zur Erstellung von Anwendungssoftware. Wir haben uns in den letzten Jahren auch mit diesem Thema befaßt und stellen Ergebnisse vor.

Danksagung

Die wesentlichen Konzepte der JWAM-Schichtenarchitektur wurden von Dirk Bäumer in seiner Dissertation ausgearbeitet. Sie beruhen auf seiner langjährigen Tätigkeit als Softwarearchitekt des Gebos-Systems der RWG.

Literatur

[Bäu95] D. Bäumer, R. Knoll, G. Gryczan, W. Strunk, H. Züllighoven, Objektorientierte Entwicklung anwendungsspezifischer Rahmenwerke, in: OBJEKTspektrum 6/95, S. 48-59

[Bäu97] D. Bäumer, G. Gryczan, R. Knoll, C. Lilienthal, D. Riehle, H. Züllighoven, Framework Development for Large Systems, in: Communications of the ACM, October 1997, Vol. 40, No. 10, S. 52-59

[Bäu98] D. Bäumer, Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme, Disser-ationsschrift am FB Informatik der Universität Hamburg, Januar 1998

[Bir95] A. Birrer, W.R. Bischofberger, T. Eggenschwiler, Wiederverwendung durch Framework-Technik - vom Mythos zur Realität, in: OBJEKTspektrum 5/95, S. 18-26

[Flo97] C. Floyd, H. Züllighoven, Softwaretechnik, in: P. Rechenberg, G. Pomberger (Hrsg.): Informatik Handbuch, Carl Hanser Verlag, 1997

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

[JWAM98] JWAM: Java Framework for the Tools and Materials Approach, Universität Hamburg, FB Informatik, Arbeitsbereich Softwaretechnik, 1998 (siehe http://swt-www.informatik.uni-hamburg.de/Software/JWAM)

[Ker98] H. Kerner (Hrsg.), Rechnernetze nach ISO-OSI, CCITT, Wolfsgraben, 1989

[Mey97] B. Meyer, Object-Oriented Software Construction, 2nd Ed., Prentice-Hall, 1997

[Rie95] D. Riehle, H. Züllighoven, A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor, in: J.O. Coplien, D.C. Schmidt (Hrsg.): Pattern Languages of Program Design, Addison-Wesley, 1995

[Rie98] D. Riehle, W. Siberski, D. Bäumer, D. Megert, H. Züllighoven, Serializer, in: R.C. Martin, D. Riehle, F. Buschmann (Hrsg.): Pattern Languages of Program Design 3., Addison-Wesley, 1998

[Roo96] S. Roock, H. Wolf: Konzeption und Implementierung eines »Reaktionsmusters« für objektorientierte Softwaresysteme. Studienarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Mai 1996.

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

Glossar

Application-Framework/Anwendungsrahmenwerk: Ein Anwendungsrahmenwerk (engl. *application framework*) ist eine spezielle Form eines Rahmenwerks, das die Grundstruktur und den Kontrollfluß einer vollständigen Anwendung enthält. Es liefert eine generische Lösung für eine Klasse von Anwendungsproblemen; eine Lösung, die aber in den meisten Fällen noch anwendungsspezifisch durch Ableitung spezialisiert oder durch Parametrisierung ergänzt werden muß. Darüber hinaus bieten Anwendungsrahmenwerke oft eine einheitliche Benutzungsschnittstelle sowohl in Form der Präsentation als auch der Handhabung (engl. *look and feel*). Verglichen mit Klassenbibliotheken kommt hier der Gesichtspunkt hinzu, daß die anwendungsspezifischen Komponenten mit passenden Schnittstellen in ein Anwendungsrahmenwerk eingehängt werden. Der Kontrollfluß der Anwendung ist insgesamt bereits durch das Anwendungsrahmenwerk definiert und wird anwendungsspezifisch nur noch angepaßt (nach [FZ97]).

Black-Box Framework/Black-Box Rahmenwerk: Bei der Entwicklung eines Anwendungssystems mit Hilfe eines Black-box-Frameworks werden von speziell dafür vorgesehenen Frameworkklassen Objekte erzeugt. Diese Objekte lassen sich häufig mit Hilfe von weiteren Objekten konfigurieren. Solche Konfigurationsobjekte werden ebenfalls von Klassen desselben oder eines anderen Black-box-Frameworks instanziiert.

Drei-Schichten-Architektur/Three-Tiers-Architecture: Die Drei-Schichtenarchitektur unterteilt ein Softwaresystem in Präsentationsschicht, funktionale Schicht und Datenschicht. Die Präsentationsschicht ist für die grafische Darstellung einer Anwendung verantwortlich. In der funktionalen Schicht wird die Anwendungslogik implementiert. Die unterste Datenschicht repräsentiert die Daten und bildet gleichzeitig die Schnittstelle zu einer eventuell eingesetzten Datenbank. Häufig werden die einzelnen Schichten als eigene Prozesse, teils auf separaten Computern, realisiert. Die Präsentationsschicht übernimmt die Rolle eines Clients, die funktionale Schicht spielt die Rolle eines Servers für die Benutzungsschnittstelle und die eines Clients in Verbindung mit den Datenobjekten. Die Datenschicht ist nur Server.

Entwurfsmetapher: Eine Entwurfsmetapher ist eine bildhafte, gegenständliche Vorstellung, die ein Leitbild fachlich und konstruktiv „ausstattet“, d.h. konkretisiert. Eine Entwurfsmetapher strukturiert die Wahrnehmung und trägt zur Begriffsbildung bei. Sie leitet die Vorstellung und Kommunikation über das, was fachlich analysiert, modelliert und technisch realisiert werden soll. Eine Entwurfsmetapher dient der Gestaltung von Softwaresystemen, indem sie Handhabung und Funktionalität für die Beteiligten verständlicher macht. Sie hat im WAM-Ansatz immer auch eine technische konstruktive Interpretation in Form von Konstruktionsanleitungen und Entwurfsmustern.

Framework/Rahmenwerk: Ein Framework ist eine Architektur aus Klassenhierarchien, die eine allgemeine generische Lösung für ähnliche Probleme in einem bestimmten Kontext vorgibt. Wiederverwendet werden dabei nicht einzelne Klassen, sondern die gesamte Konstruktion aus zusammenspielenden Komponenten. Ein Framework gibt so den Kontrollfluß für die Anwendung vor.

Leitbild: Allgemein ist ein Leitbild eine benennbare, grundsätzliche Sichtweise, anhand derer wir einen Ausschnitt von Realität wahrnehmen, verstehen und gestalten. Ein Leitbild repräsentiert immer auch eine Wertvorstellung. In der Softwareentwicklung gibt ein Leitbild im Entwicklungsprozeß und für den Einsatz einen gemeinsamen Orientierungsrahmen für die beteiligten Gruppen. Es unterstützt den Entwurf, die Verwendung und die Bewertung von Software und basiert auf Wertvorstellungen und Zielsetzungen. Ein Leitbild kann konstruktiv oder analytisch verwendet werden.

Lose Kopplung: Eine Komponente ist dann lose an eine zweite gekoppelt, wenn der Klient nicht die gesamte Schnittstelle des Anbieters kennt, sondern auf der Basis eines Typs nur den Ausschnitt der Schnittstelle (und ihr Verhalten), der die notwendigen Operationen für diese Koppelung beschreibt.

Metaobjektprotokoll: Als Metaobjektprotokoll wird die Menge der Schnittstellen von sogenannten Metaklassen bezeichnet, die es uns erlauben, Anfragen an Programmobjekte zu stellen. I. d. R. können über das Metaobjektprotokoll einerseits Anfragen an die Objekte gestellt werden, die das statische Anwendungsmodell repräsentieren (z.B. Welche Operationen bietet ein Objekt?); andererseits können

Objekte befragt werden, die das Laufzeitsystem repräsentieren (z.B.: Wie wird eine Operation ausgeführt? Wie greift ein Objekt auf seine Attribute zu?). Heute bieten die meisten objektorientierten Programmiersprachen (Smalltalk, Java, neuere C++ Versionen) in ihren Bibliotheken ein Metaobjektprotokoll an, sonst muß ein selbst entwickeltes Metaobjektprotokoll zu der Programmiersprache hinzugefügt werden.

Modellarchitektur: Eine Modellarchitektur beschreibt die allgemeinen Prinzipien hinter einer Softwarearchitektur. Sie umfaßt die grundlegenden Elemente, deren Verknüpfungen und die Regeln, die für eine Softwarearchitektur gelten. Eine Modellarchitektur gibt Anleitung bei der softwaretechnischen Realisierung eines Softwaresystems.

Objektorientierte Schicht: Objektorientierte Schichten sind im Gegensatz zu den protokollbasierten nach dem Prinzip von Generalisierung und Spezialisierung eher transparent im Sinne von »durchscheinend« aufgebaut. Hier sollen tiefere Schichten gerade für höhere Schichten sichtbar sein, damit der Entwickler entscheiden kann, ob er vorhandene Konzepte, soweit sie als instantiierbare Klassen vorliegen, benutzen kann oder wo eine Spezialisierung im Klassenbaum ansetzen muß (nach [Zül98]).

Persistenzmedium: Persistenzmedium ist ein Oberbegriff für Systeme, in denen Materialien persistent abgelegt werden können. Dies sind meist Datenbankmanagementsysteme, die sich in angebotenen Datenmodellen und Dienstleistungen unterscheiden können, und Dateisysteme.

Protokollbasierte Schicht: Eine protokollbasierte Schichtenarchitektur organisiert ein System hierarchisch. Eine Schicht stellt als Protokoll definierte Leistungen für die Elemente der eigenen oder der nächsthöheren Schicht zur Verfügung. Dabei abstrahiert jede Schicht von der darunterliegenden, in dem sie ein »höheres«, d.h. anwendungsnäheres Protokoll zur Verfügung stellt.

Die jeweiligen Schichten sind meist als Module realisiert. Durch die Trennung von Protokoll und Implementierung sollen Änderungen an einer Schicht möglichst lokale Auswirkungen haben, damit eine unabhängige Entwicklung der einzlnen Schichten gewährleistet werden kann (nach [Zül98]).

Schichtenarchitektur: Eine Schicht organisiert die softwaretechnischen Komponeten einer Modellarchitektur zu einer fachlich und technisch motivierten Entwurfs- und Kontruktionseinheit. Eine Schicht hat selbst keine Schnittstelle und keine Beziehung zu anderen Schichten; Schnittstellen und Beziehungen über Verbindungsstücke haben nur die enthaltenen Komponenten.

Die verschiedenen Schichten einer Modellarchitektur sind hierarchisch aufgebaut. Je nach Sichtbarkeit und Verwendungszusammenhang sprechen wir von einer protokollbasierten oder objektorientierten Schicht. Als Komponenten innerhalb einer Schicht verwenden wir Klassenbibliotheken und Rahmenwerke. Die Verbindungsstücke sind durch Konstruktionsmuster, Benutzt-Beziehung oder Vererbung realisiert (nach [Zül98]).

Strukturähnlichkeit: Strukturähnlichkeit bezieht sich im WAM-Ansatz auf das Verhältnis von Software und Anwendungsbereich: Die softwaretechnischen Komponenten eines Softwaresystems modellieren die relevanten Konzepte und Gegenstände des Anwendungsbereichs. Die Architektur des Anwendungssystems spiegelt die wesentlichen Bezüge zwischen den Konzepten und Gegenständen des Anwendungsbereichs wider (nach [Zül98]).

Vertragsmodell: Das Vertragsmodell betrachtet die Benutzt-Beziehung zwischen Klassen als ein Verhältnis von Leistungsanbietern und Klienten, das durch einen formalen Vertrag geregelt ist. Der Vertrag legt fest, welche Vorbedingungen ein Klient erbringen muß, damit der Anbieter seine Leistungen erfüllt. Verträge sind beim Anbieter beschrieben und bestehen aus Vor- und Nachbedingungen sowie Invarianten.

Werkzeug Automat Material-Ansatz/WAM-Ansatz: Der WAM-Ansatz strebt die Anwendungsorientierung in der Softwareentwicklung an. Anwendungsorientierte Software zeichnet sich durch hohe Gebrauchsqualität aus, d.h. die Funktionalität des Systems orientiert sich an den Aufgaben aus dem Anwendungsbereich, die Handhabung des Systems ist benutzergerecht und die im System festgelegten Abläufe und Schritte lassen sich je nach Anwendungssituation problemlos an die tatsächlichen Anforderungen anpassen. Auf der Grundlage objektorientierter Entwurfs- und Konstruktionstechniken vereinigt der WAM-Ansatz ein Leitmotiv mit Entwurfsmetaphern, anwendungsorientierte Dokumente und eine evolutionäre Vorgehensweise mit Prototyping.

White-Box Framework/White-Box Rahmenwerk: Mit Hilfe von White-box-Frameworks werden Anwendungen so entwickelt, daß ausgezeichnete Klassen des Frameworks in Unterklassen spezialisiert werden und dadurch den Anschluß an die übrigen Klassen des Anwendungssystems herstellen. Die Verwendung eines White-box-Frameworks ist nur mit Wissen über den internen Aufbau des Frameworks, insbesondere über das in den Klassen modellierte Zusammenspiel der Objekte, möglich.

Autoren

Dr. Guido Gryczan ist Akademischer Oberrat am Arbeitsbereich Softwaretechnik der Universität Hamburg. Sein Arbeitsschwerpunkt ist die Unterstützung von Kooperation und Koordination mit objektorientierten Softwaresystemen. Er liest email unter: gryczan@informatik.uni-hamburg.de

Carola Lilienthal ist wissenschaftliche Mitarbeiterin am Arbeitsbereich Softwaretechnik der Universität Hamburg. Sie arbeitet im Bereich Anwendungsentwicklung mit objektorientierten Rahmenwerken und hat in diesem Rahmen mehrere Artikel veröffentlicht sowie Beratungen und Schulungen durchgeführt. Sie liest email unter: lilienthal@informatik.uni-hamburg.de

Martin Lippert studiert Informatik mit Schwerpunkt Softwaretechnik an der Universität Hamburg. Sein

Hauptinteresse liegt im Entwurf und der Konstruktion von Frameworks sowie der Entwicklung mit Java. Er liest email unter: 4lippert@informatik.uni-hamburg.de

Stefan Roock und **Henning Wolf** sind wissenschaftlicher Mitarbeiter am Arbeitsbereich Softwaretechnik der Universität Hamburg im EU-Drittmittelprojekt REFORM. Ihre Arbeitsschwerpunkte sind Techniken des Managements objektorientierte Projekte und Konstruktion objektorientierter Frameworks. Sie lesen e-mail unter {roock, wolf}@informatik.uni-hamburg.de

Dr.-Ing. Heinz Züllighoven ist Professor am Arbeitsbereich Softwaretechnik der Universität Hamburg und Leiter des dortigen Softwaretechnik-Centers. Sein Arbeitsschwerpunkt sind Projekte und Publikationen zu den Themen Benutzer- Entwickler Kommunikation, Prototyping, Programmierumgebungen und objektorientierter Entwurf interaktiver Anwendungssoftware. Er liest email unter: zuelligh@informatik.uni-hamburg.de