Frameworkbasierte Softwareentwicklung

Walter R. Bischofberger

UBILAB, Union Bank of Switzerland
Bahnhofstr. 45, CH-8021 Zürich

e-mail: bischofberger@ubilab.ubs.ch

Einleitung[1]

Bei der Entwicklung der meisten Anwendungen wird heutzutage die gesamte gewünschte Funktionalität neu entworfen, implementiert und getestet, obwohl man immer wieder auf ähnliche Anforderungen und Lösungen stösst. Wissen über diese Ähnlichkeiten sammelt sich über Jahre hinweg in Form von Erfahrungen bei einzelnen Entwicklern an und erlaubt diesen, mit der Zeit immer schneller Lösungen für ihnen vertraute Anwendungsgebiete zu bauen. Dieses Wissen kann aber schlecht weitervermittelt werden und geht verloren, sobald der Entwickler die Firma verlässt.

Obwohl es unter den Anwendungen der meisten Anwendungsgebieten sehr starke Ähnlichkeiten gibt, hat man bis heute nur einen marginalen Grad von Wiederverwendbarkeit erreicht. Dies liegt meines Erachtens daran, dass beim Entwurf und der Implementierung meist nur ein Projekt, normalerweise sogar nur eine Anwendung, in Betracht gezogen wird. Daneben fehlt es immer an Zeit und Anreizen, um eine funktionierende Anwendung zu überarbeiten und wiederverwendbare Teile zu generalisieren.

Dies ist nicht weiter verwunderlich, wenn man in Betracht zieht, dass herkömmliche prozedurale und modulare Programmiersprachen einem Entwickler keine Mechanismen zur Verfügung stellen, um grössere Teile einer Anwendung zu generalisieren, wiederverwendbar zu gestalten und dann im konkreten Fall flexibel anzupassen. Dank Vererbung und Polymorphismus ist dies mit objektorientierten Programmiersprachen möglich.

Was ist ein Framework?

Im Gegensatz zu einer problemspezifischen Lösung ist ein Framework eine Standardlösung für eine Klasse verwandter Probleme. Ein Framework entsteht, indem man iterativ die generischen und spezifischen Aspekte einer Lösung voneinander trennt und die generischen Aspekte als Ganzes wiederverwendbar gestaltet.

Die Objekttechnologie bietet die nötigen Mechanismen, um ein System als eine Menge kooperierender Objekte zu modellieren. Anwendungsspezifische Details können dabei hinter Objektschnittstellen verborgen werden. Ein Framework besteht aus den Objekten, die die

---

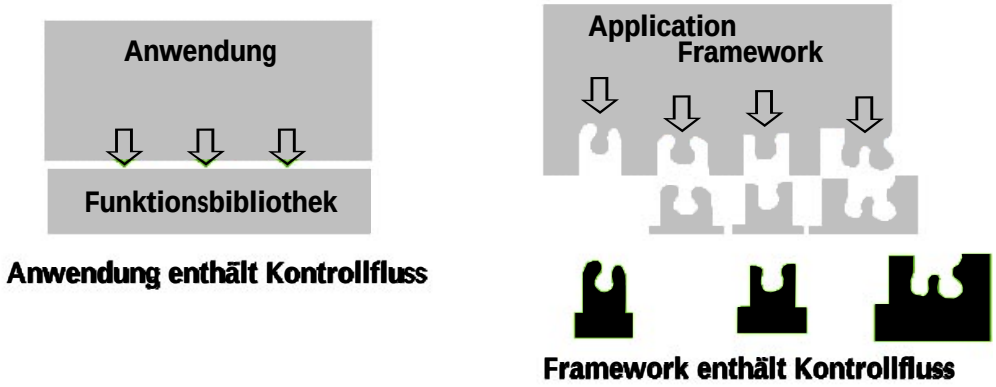
1 Veröffentlicht in Procs. OOP '95, München, 30. Januar - 3. Februar 1995

generischen Teile der Lösung realisieren, sowie aus den Schnittstellen der Objekte, die die anwendungsspezifischen Details implementieren. Ein Objekt, das sich hinter solch einer Schnittstellen verbirgt, kann später durch eine anderes, welches die anwendungsspezifische Funktionalität realisiert, ersetzt werden. Dieser Mechanismus ist in Figur 1 dargestellt. Die Flexibilität und Generizität eines Frameworks beruht auf der bewussten Trennung von generischen und spezifischen Teilen. Während die generischen Teile sehr konkret implementiert sind, wird bei der Einbindung der spezifischen Teile bewusst davon abstrahiert, wie etwas getan werden soll. Lediglich was getan werden soll, wird spezifiziert. Ein Framework ist folglich nicht nur eine Anzahl wiederverwendbarer Klassen. Es enthält auch eine generalisierte, wiederverwendbare Systemarchitektur für das entsprechende Anwendungsgebiet.
Ein gutes Beispiel für Frameworks sind Application Frameworks (Anwendungs–Rahmenwerke) wie MacApp [Ros86] oder ET++ [Wei94]. Beide abstrahieren die Gemeinsamkeiten von interaktiven, dokumentbasierten Werkzeugen. Dazu gehören z. B. Text- und Graphikeditoren. Sowohl MacApp als auch ET++ enthalten unter anderem die gesamte Logik für Oeffnen, Schliessen und Speichern von Dokumenten, für das Rückgängigmachen von Befehlen, für das Rollen und Splitten von Fenstern, sowie für das Verwalten von Menübalken.
Das Entwickeln eines Frameworks ist kein linearer Prozess. Es ist für den Entwickler vielmehr ein iterativer, zyklischer Lernprozess, bei dem man aufgrund der Implementierung konkreter Lösungen neue Ideen für wiederverwendbare Abstraktionen findet. Die Qualität dieser Abstraktionen lässt sich aufgrund der Einfachheit, mit der man sie für einen neuen Anwendungsfall wiederverwenden kann, messen.
Frameworks können auf verschiedenen Ebenen einer Anwendung verwendet werden, und ein Framework kann auf mehreren anderen Frameworks basieren. Application Frameworks, die die Beziehung der Objekte einer ganzen Anwendung modellieren, bauen zum Beispiel auf einer grossen Zahl kleinerer Frameworks auf. Frameworks sind nicht darauf beschränkt, das Zusammenspiel der Objekte innerhalb eines Prozesses zu modellieren. Die Objekte eines Frameworks können über mehrere Prozesse verteilt sein. Das ist zum Beispiel bei Frameworks zum Erstellen von OLE 2.0 oder OpenDoc Anwendungen der Fall.

Was ist der Unterschied zwischen dem Einsatz von Frameworks und herkömmlichen Funktionsbibliotheken?

Funktionsbibliotheken bieten gewöhnlich eine Menge Funktionen an, die gewisse Algorithmen für ein bestimmtes Anwendungsgebiet realisieren. Frameworks gehen viel weiter, indem sie die gesamte Architektur einer Standardlösung kapseln. Der Benutzer muss sich nicht mehr darum kümmern, wie das Framework gesteuert wird. Er passt es nur noch für seine spezifischen Bedürfnisse an. Während beim Einsatz einer Funktionsbibliothek die gesamte applikatorische Logik neu

implementiert werden muss, kann diese durch ein Application Framework zu einem grossen Teil wiederverwendet werden. Eine schematische Darstellung dazu liefert Figur 1.



Figur 1: Einsatz von Funktionsbibliotheken und Frameworks

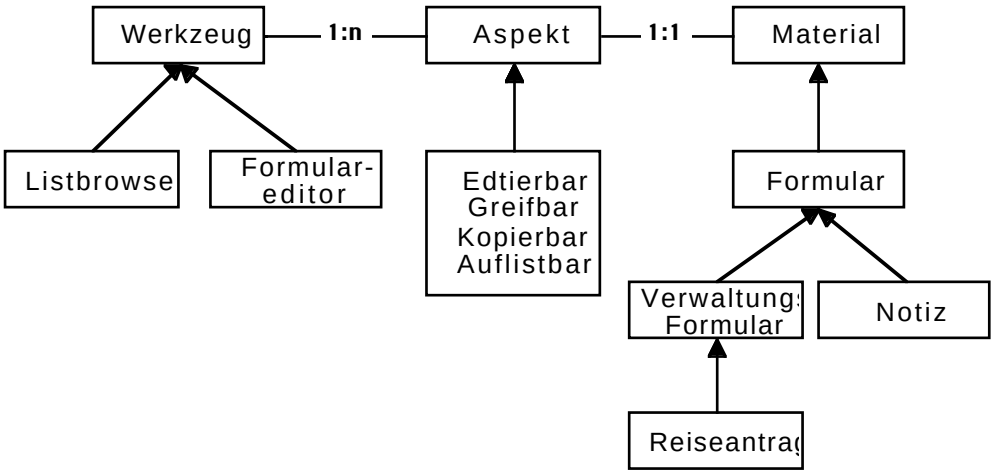
Was gewinnt man durch die Anwendung von Frameworks?

Mit guten Frameworks gelingt es häufig, einen Wiederverwendungsgrad um 70% und höher zu erzielen.
Das konsequente Anwenden von Frameworks erleichtert massiv das Entwickeln von Softwaresystemen, ohne dass die Mächtigkeit des verwendeten Implementierungsmechanismus einschränkt. wird Bei 4.Generationssystemen ist dies hingegen häufig der Fall.
Frameworks sind in Software gegossene Konzepte und Lösungen. Diese Konzepte helfen einem Entwickler beim Modellieren der Anwendungsdomäne und beim Erkennen der relevanten Beziehungen unter den Objekten. Der Vorteil erwächst daraus, dass die zentralen Objekte, Beziehungen und Schnittstellen im Framework bereits modelliert sind. Für einen mit der Anwendungsdomäne vertrauten Entwickler führt dies zu einem beschleunigten Projektfortschritt. Ein unerfahrener Entwickler ist in der Lage, Anwendungen zu schreiben, ohne die Anwendungsdomäne vollständig verstanden zu haben. Er findet einen grossen Teil des Grundwissens als integralen Bestandteil des Frameworks.
Das Erarbeiten des Wissens über ein spezielles Anwendungsgebiet ist mit erheblichen Aufwendungen verbunden. Frameworks können als Absicherung dieser grossen Investitionen verstanden werden. Während früher (unabhängig von der Menge Dokumentation) dieses Wissen zusammen mit den Entwicklern die Firma verliess, liegt es nun als abstrahierte, wiederverwendbare Softwarearchitektur vor.
Ein gutes Framework ermöglicht, Anwendungen zu entwickeln, ohne dass man alle Details der dazu benötigten Ressourcen verstehen muss. Die Details von Betriebssystem, Fenstersystem, Interprozesskommunikation, Datenbank, etc. sind typischerweise hinter domänenspezifischen Schnittstellen vor dem Entwickler verborgen. Das

Schreiben von OLE 2.0 fähigen Anwendungen auf Ebene der OLE 2.0 API's ist z.B. ein extrem komplexes Unterfangen. Die Verwendung von MFC (Microsoft Foundation Classes [MIC94]) erleichter die Entwicklung hingegen herheblich.
Das evolutionäre Entwickeln von Softwaresystemen wird durch Frameworks erleichtert, da der Entwickler auf einer ausgereiften Architektur aufbaut. Er macht damit im Bereich des Entwurfs in der Regel weniger Fehler als bei der herkömmlichen Entwicklung. Der Quelltext von Frameworks wurde bereits durch Verwendung in mehreren Applikationen getestet. Dies führt automatisch zu Produkten mit einer geringeren Fehlerrate.

Welche Gefahren gibt es bei der Anwendung von Frameworks?

Die Entwicklung von Frameworks ist teurer. Besteht nicht die Möglichkeit das Framework anschliessend einige Male einzusetzten, so ist es durchaus möglich, dass die Entwicklung, obwohl technisch erfolgreich, ein ökonomischer Fehlschlag wird.
Die Framework-Entwicklung erfordert viel Erfahrung auf dem Gebiet des objektorientierten Softwareentwurfs. Hat man keine erfahrenen Entwickler zur Verfügung, so darf man nicht zu schnell grossartige Ergebnisse erwarten.
Der Einsatz von Frameworks setzt voraus, dass man die grundlegenden Konzepte des Framework gut verstanden hat. Es kommt darauf an zu verstehen, wo man es auf welche Art erweitern und anpassen kann, um die eigenen spezifischen Bedürfnisse zu befriedigen. Dies bedeutet, dass je nach Framework und Fähigkeiten des Entwicklers eine mehr oder weniger lange Einarbeitungszeit notwendig ist. Diese schreckt vielfach ab, kann aber häufig schon beim erstmaligen erfolgreichen Wiederverwenden amortisiert werden.

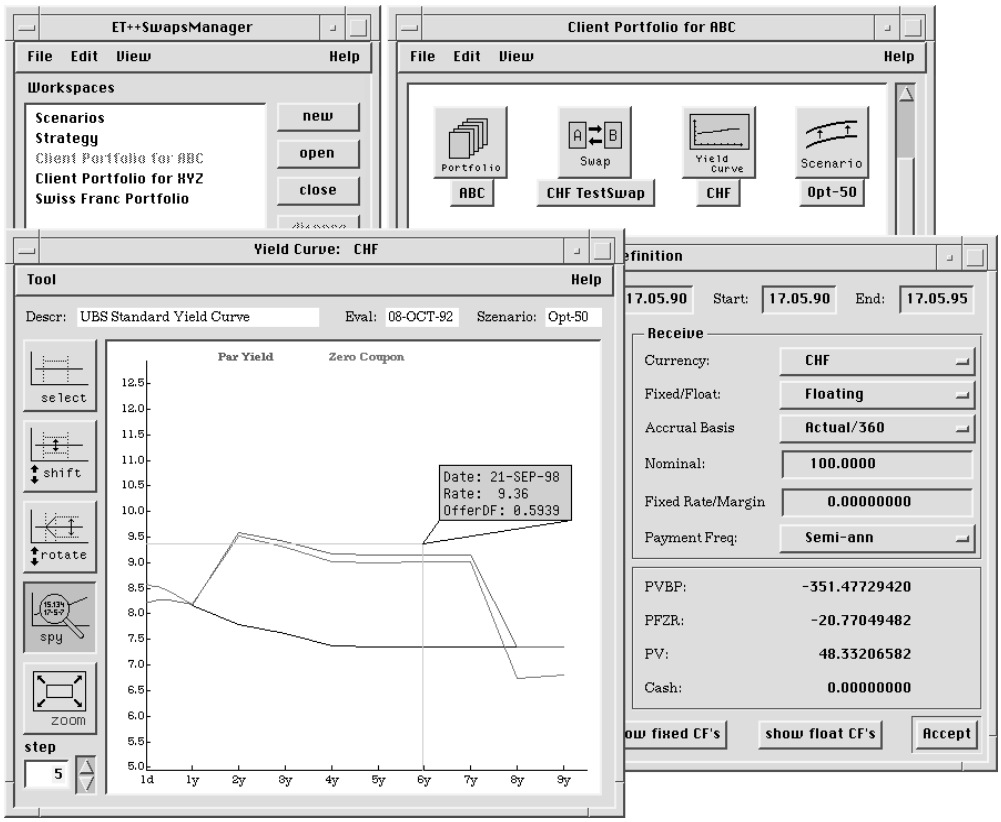


Figur 2: Vereinfachte Darstellung des Werkzeug-Aspekt-Material Frameworks [Kil94]

Was sind die etablierten Anwendungsgebiete für Frameworks?

Ersatz von 4.Generationssystemen
4. Generationssysteme (4GS) sind, abstrakt gesehen, nichts anderes als konfigurierbare Standardanwendungen für Informationssysteme. Ihr Vorteil ist, dass sie leicht erlernbar und einfach anzuwenden sind. Ihr Nachteil besteht darin, dass sie, wenn man einmal an ihre Grenzen stösst, nur sehr beschränkt erweiterbar sind.
Schreibt man für dasselbe Anwendungsgebiet ein Framework, so hat man gegenüber einem 4GS verschiedene Vorteile. Erstens kann man ein Paradigma wählen, das dem eigenen Anwendungsbereich besser angepasst ist als ein 4GS. Zweitens kann man das Framework fast beliebig erweitern und stösst damit an keine harten Grenzen. Drittens manipuliert man nicht nur Informationen mit Werkzeugen, sondern kann im Framework auch fachliche Abstraktionen modellieren, wie das Figur 2 zeigt.

Anspruchsvolle, graphische Benutzungsschnittstellen
Im Bereich der modernen graphischen Benutzungsschnittstellen ist man sich einig, dass sie ohne Objekttechnologie und mächtige Frameworks nicht vernüftig implementierbar sind. Figur 3 zeigt einen Bildschirmausschnitt des Swaps Manager, eines funktionalen Benutzungsschnittstellen-Prototyps für den interaktiven Handel mit Swaps. Er wurde von einem Studenten mit dem ET++ Application Framework [Wei94] in enger Kooperation mit Swaps Händlern in drei Personenmonaten entwickelt.

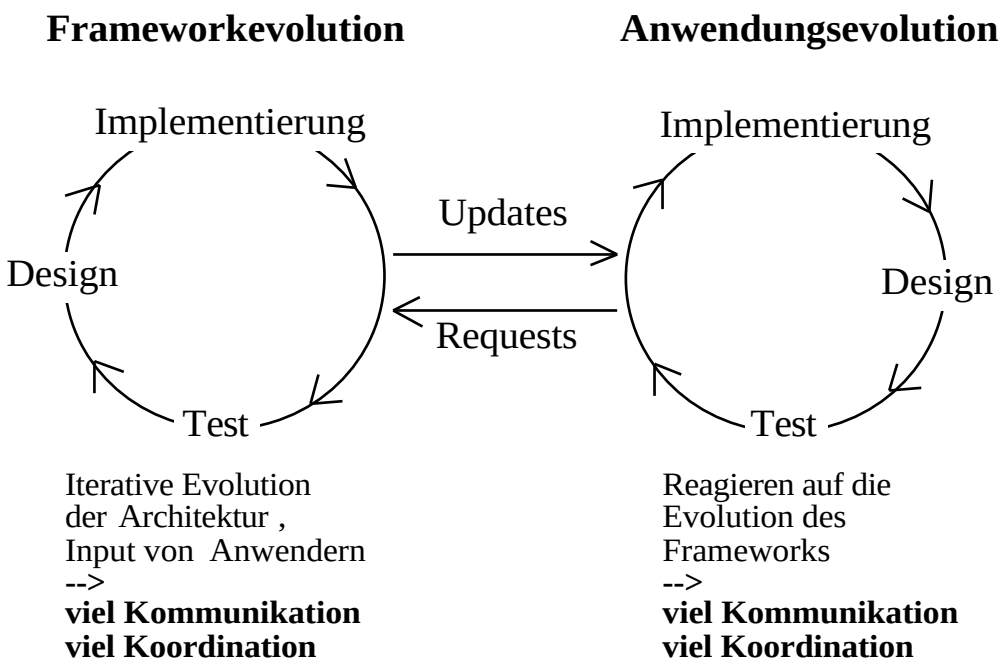


Figur 3: Swaps Manager Benutzungsschnittstelle (aus [Egg92])

Anspruchsvolle, generalisierbare Anwendungslogik
Frameworks für anspruchsvolle, generalisierbare Anwendungslogiken wurden bis heute leider wenig entwickelt, obwohl sie sehr vielversprechend wären. Ein gutes Beispiel für ein solches Framework ist das am UBILAB entwickelte Calculation Engine Framework. Es abstrahiert die Berechnung der Werte von derivativen Zinsprodukten [Bir93]. Der produktive Einsatz eines solchen Frameworks führt dazu, dass der Handel regelmässig neuer Produkte in kurzer Zeit mit guten Werkzeugen unterstützt werden kann. Die sehr komplexen Berechnungen brauchen nicht, wie heute üblich, mehrfach implementiert zu werden.

Wie verändert sich der Entwicklungsprozess bei der Anwendung von Frameworks?

Im Gegensatz zum linearen Softwareentwicklungsprozess, wie er z.B. von Boehm mit seinem Wasserfallmodell beschrieben wurde [Boe76], haben wir es bei der frameworkbasierten Softwareentwicklung mit mehreren, sich gegenseitig beeinflussenden, zyklischen Entwicklungsprozessen zu tun (Figur 4). Einerseits wird das Framework als solches iterativ entwickelt. Andererseits werden die auf dem Framework basierenden Anwendungen ebenfalls iterativ entwickelt. Dabei beeinflussen sich die Evolution des Frameworks und die der Anwendungen gegenseitig. Die Entwickler des Frameworks sind auf Anregungen und Kritik der Anwendungsentwickler angewiesen, um zum einen Schwachstellen im Framework zu identifizieren, und zum anderen weitere generalisierbare Aspekte zu finden. Die Anwendungsentwickler werden von den Frameworkentwicklern insofern beeinflusst, als dass sie von Zeit zu Zeit auf neue Versionen des Frameworks wechseln und von dessen Weiterentwicklung profitieren.
Diese Art von Entwicklung führt im Gegensatz zum konventionellen Vorgehen zu grösseren Kommunikations- und Koordinationsbedürfnissen und somit auch zu steigenden Anforderungen an die Entwicklungsumgebung. Ein starkes Umdenken im Verlgeich zur herkömmlichen Projektorganisation ist notwendig. Eine detaillierte Diskussion dieser Aspekte übersteigt den Rahmen dieses Aufsatzes. Die Problematik wird im Rahmen des Vortrags ausführlich behandelt.



Figur 4: Framework orientierter Entwicklungsprozess

Wie verändern sich die Anforderungen an die Entwicklungsumgebung?

Die objektorientierte Softwareentwicklung führt zu gesteigerten und neuen Anforderungen an die Entwicklungsumgebung, wie wir in [Bis94] dargelegt haben.

Die Entwicklung und Anwendung von Frameworks führt zu grösseren Kommunikations- und Koordinationsbedürfnissen und damit auch zu gesteigerten Anforderungen an die Entwicklungsumgebung. Dabei werden Konzepte und Werkzeuge für formelle und informelle Kooperation benötigt. Werkzeuge für formelle Kooperation erlauben es z.B. mehreren Entwicklern, sich für eine gewisse Zeit zu entkoppeln, um anschliessend ihre Resultate wieder kontrolliert und so effizient wie möglich zu mischen. Werkzeuge zur informellen Kooperation erlauben es, Information, die nicht sinnvoll in formalisierten Dokumenten festgehalten werden können, zu verwalten, zu suchen und zu kommunizieren. Weitere Informationen über formelle und informelle Kooperation sowie deren Unterstützung findet der Leser in [Bis95a] und [Bis95b].

Referenzen

[Bir93] Birrer A, Eggenschwiler T: Frameworks in the Financial Engineering Domain: An Experience Report. in ECOOP '93 Conference Proceedings, Springer Verlag, 1993

[Bis 94] Bischofberger WR, Kofler T, Schäffer B: Object-Oriented Programming Environments: Requirements and Approaches. In: Software - Concepts and Tools, Vol. 15 No. 2, Springer Verlag, 1994

[Bis95a] Bischofberger WR, Kofler T, Mätzel K-U, Schäffer B: Computer Supported Cooperative Software Engineering with Beyond-Sniff. To be published in Procs. of the 7th Conference on Software Engineering Environments, Noorwijkerhout, Netherlands, 5-7 April 1995

[Bis95b] Bischofberger WR, Mätzel K-U, Kleinferchner CF: Evolving a Programming Environment Into a Cooperative Software Engineering Environment. To be published in Procs. of CONSEG 95 (International Conference on Software Engineering Practices), New Delhi, February 1995

[Boe76] Boehm BW: Software Engineering. IEEE Transaction on Computers, Vol. 25, No. 12, Dec. 1976

[Egg92] Eggenschwiler T, Gamma E: ET++ SwapsManager: Using Object Technology in the Financial Engineering Domain. in Procs. of OOPSLA '92, ACM DIGPLAN Notices, Vol. 27, No. 10

[Kil94] Kilberth F, Gryczan G, Züllighoven H: Objektorientierte Anwendungsentwicklung. Vieweg Verlag, 1994

[MIC94] Microsoft Cooperation: Introducing Visual C++, Microsoft Visual C++, Development System for Windows and Windows NT Version 2.0. Microsoft Cooperation, 1994

[Ros86] Rosenstein L, Doyle K, Wallace S: Object-Oriented Programming for Macintosh Applications. in ACM Fall Joint Computer Science Conference, Dallas, Texas, 2-6 November 1986

[Wei 94] Weinand, A., Gamma, E.: ET++ - a Portable, Homogenous Class Library and Application Framework. Computer Science Research at UBILAB, Strategy and Projects; Proceedings of the UBILAB '94 Conference, Zurich, September 1994. Universitätsverlag Konstanz, Konstanz, 1994