Framelets als handliche Architekturbausteine

Wolfgang Pree, Egbert Althammer
Software Engineering Gruppe
Universität Konstanz
D-78457 Konstanz
Tel: +49-7531-88-44 33
Nachname@acm.org

Hermann Sikora
GRZ/RACON Software GmbH
Goethestr. 80
A-4020 Linz
Tel: +43-732-69 29-12 13
sikora@grz.at

Abstract: Obwohl objektorientierte Frameworks einen wichtigen Ansatz zur Wiederverwendung von Softwarearchitekturen darstellen, sind sie mit gravierenden Nachteilen behaftet. Das Design von Frameworks ist schwierig. Die Wiederverwendung solcher Artefakte ist oft aufwendig (siehe zB Lewis et al., 1995; Sparks et al., 1996; Fayad et al., 1998). Außerdem ist die interne Arbeitsweise verschiedener Frameworks meist nicht kompatibel, sodaß zwei oder mehrere Frameworks kaum kombiniert werden können. Als Alternative schlagen wir Framelets als kompakte Architekturbausteine vor, die wesentlich leichter verstanden, geändert und mit anderen Framelets kombiniert werden können. Der Beitrag behandelt zuerst die Framelets zugrundeliegenden Konzepte und skizziert in einer Fallstudie ein Framelet, das in Java implementiert wurde.

Stichwörter: Softwarearchitektur, objektorientierte Frameworks, Java, Softwarewiederverwendbarkeit, Design Patterns, Reflexion.

1 Gemeinsamkeiten zwischen Frameworks und Framelets

Frameworks und Framelets haben gemeinsam, daß sie flexible objektorientierte Softwarearchitekturen repräsentieren. Die Realisierung solcher Architekturen beruht auf Sprachelementen, die von objektorientierten Programmiersprachen bereitgestellt werden. Im folgenden verwenden wir die Bezeichnung (objektorientierte) Architektur als Oberbegriff für Framework und Framelet.

Im allgemeinen eignen sich objektorientierte Architekturen sehr gut für Bereiche, in denen zahlreiche ähnliche Anwendungen immer wieder entwickelt werden. Bestimmte Anwendungen werden durch Spezialisierung eines Anwendungsgerüsts (Framework) erstellt. Bass et al. (1998) bezeichnen solche Architekturen als *Product Lines*. Auf den Zusammenhang zwischen Frameworks und Komponenten wird beispielsweise in Szyperski (1998) sowie in Pomberger und Pree (1998) eingegangen.

Spezialisierung (= Anpassung) findet an bestimmten Stellen im Anwendungsgerüst statt, die wir *Hot Spots* nennen (Pree, 1995, 1997). Wir geben einer Architektur das Qualitätsattribut *gut entworfen*, wenn es die adäquaten Hot Spots für Anpassungen zur Verfügung stellt.

Hot-Spot-Anpassung durch Vererbung oder Schnittstellenimplementierung

Diese Hot Spots sind durch abstrakte Klassen oder Java-Schnittstellen realisiert. Klasse A in der Beispielklassenhierarchie (siehe Abbildung 1) veranschaulicht die sogenannte White-Box-Eigenschaft einer Architektur. Die abstrakte Methode von Klasse A, die in einer Unterklasse überschrieben werden muß, ist grau schraffiert. Die abstrakte Methode bildet in diesem Fall den Hot Spot.

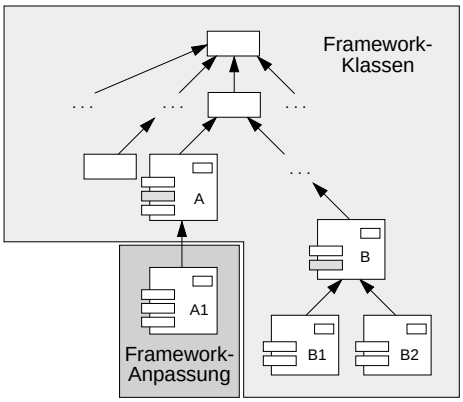

Abbildung 1 Beispiel einer Klassenhierarchie.

Entwickler können das Verhalten einer White-Box-Architektur ändern, indem sie im Fall von abstrakten Klassen die entsprechenden Methoden in Unterklassen überschreiben. Dabei müssen Design und Implementierung einer Architektur verstanden werden, zumindest zu einem bestimmten Detaillierungsgrad. Java-Schnittstellen entsprechen insoferne abstrakten Klassen, als sie nur aus abstrakten Methoden bestehen. Schnittstellen erlauben zusätzlich die Trennung von Klassen- und Typhierarchien. Auf diesen Aspekt wird hier nicht näher eingegangen.

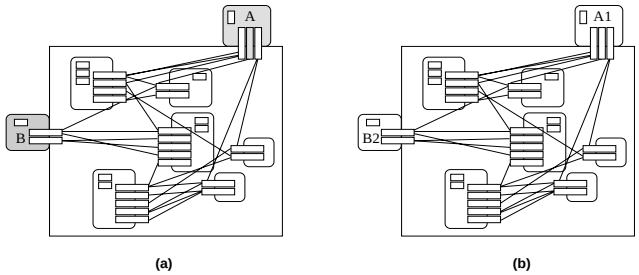

Abbildung 2 Architektur (a) vor und (b) nach der Adaptierung.

Hot-Spot-Anpassung durch Komposition

Black-Box-Architekturen bieten fertige Komponenten für Anpassungen an. Änderungen erfordern einfache Komposition, nicht das aufwendige Überschreiben oder Implementieren von Methoden. Die Hot Spots entsprechen den überschriebenen Methoden, wenn auch bei der Adaptierung nur mit den Komponenten als Ganzes gearbeitet wird. In der Klassenhierarchie in Abbildung 1 hat Klasse `B` bereits zwei Unterklassen `B1` und `B2`, die die abstrakte Methode von `B` implementieren. Abbildung 2(a) zeigt schematisch die Interaktion der Komponenten zur Laufzeit. Die Linien stellen Methodenaufrufe dar. Ein Entwickler paßt die Architektur an, indem er zB die Klassen `A1` und `B2` instantiiert und entsprechend "einsteckt" (siehe Abbildung 2(b)). Architekturen sind weder reine White-Box- noch reine Black-Box-Architekturen. Wird eine Architektur oft wiederverwendet, nimmt der Anteil der Black-Box-Komponenten zu, da bei wiederholter Wiederverwendung typische Anpassungen in Form von Black-Box-Komponenten entstehen.

2 Framelet versus Framework

Mit der Definition des Begriffes Framework ist keine Aussage über dessen Größe verbunden. Somit stellen auch sehr kleine, nach oben skizzierten Konstruktionsprinzipien definierte Architekturen, Frameworks dar. Probleme in Zusammenhang mit Frameworks resultieren aus unserer Sicht daraus, daß ein Framework typischerweise ein Architekturgerüst für einen *komplexen* Anwendungsbereich sein soll. Somit entstehen große, kompliziert verwobene Klassen- bzw. Schnittstellensammlungen mit versteckten logischen Abhängigkeiten, die grundlegende Modularisierungsprinzipien verletzen und schwierig mit anderen Frameworks zu kombinieren sind. Ein Framelet hingegen ist explizit ein kleiner, flexibler Architekturbaustein, welcher

- nicht den Hauptkontrollfluß einer Anwendung an sich zieht,
- klein (weniger als 10 Klassen) ist,
- eine klar definierte, einfache Schnittstelle hat.

Ansonsten weist ein Framelet die Eigenschaften eines Frameworks auf (siehe Abschnitt 1). Insbesondere beruht es auf dem sogenannten Hollywood-Prinzip (Weinand, 1989), auch *Call-Back*-Prinzip genannt, einer Whitebox-Architektur. In der Implementierung des Framelets werden abstrakte Methoden aufgerufen, die erst durch die konkret eingesteckten Objekte definiert werden.

Ein Framelet realisiert generisch einen Systemaspekt, der in diversen Anwendungen benötigt wird. Die Vision ist, größere Teile einer Anwendung durch teilweise automatisierte Black-Box-Komposition von Framelets zu realisieren, während Framelets selbst meist White-Box-Architekturen darstellen. Meta-Informationen sollten genutzt werden, um Framelets intern sowie untereinander zum Teil automatisch konfigurierbar zu machen.

Meta-Informationen als Basis für selbstkonfigurierende Framelets

Abbildung 2 zeigt auf, warum Frameworks und Framelets oft als Halbfertigfabrikate bezeichnet werden. Die weiß gezeichneten Objekte/Komponenten interagieren mit den als grau gefüllte, abstrakte Komponenten dargestellten Hot Spots. Durch Einsetzten konkreter Komponenten wird das Halbfertigfabrikat adaptiert. Wie oben erwähnt, werden meist abstrakte Klassen als Sprachkonstrukt zur Definition der abstrakten Komponenten verwendet. Javas Schnittstellenkonstrukt (*interface*) stellt eine Alternative zu abstrakten Klassen dar. Das Schnittstellen-Sprachkonstrukt ist gut einsetzbar, wenn die abstrakte Einheit entweder nicht an einer festen Stelle in der Klassenhierarchie positioniert werden kann oder die Einheit zu klein ist, um eine Klassendefinition zu rechtfertigen. Konzeptionell sind beide Konstrukte, die abstrakte Klasse und die Schnittstelle, aus Sicht der Framework-Entwicklung als gleichwertig anzusehen.

Abgesehen von diesen kanonischen Möglichkeiten zur Definition von Hot Spots, gibt es weitergehende Möglichkeiten, die wesentlich mehr Flexibilität, aber damit verbunden geringere Typsicherheit bieten. Beispielsweise können abstrakte Einheiten auf Basis von Meta-Informationen definiert werden: Ein Aspekt

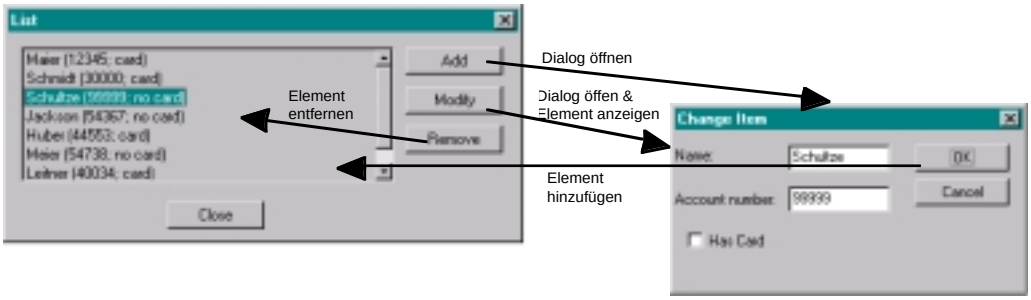

Abbildung 3 Eine Beispielspezialisierung (Kontoinformation) des List-Box-Framelets.

üblicher Meta-Informationsmodelle erlaubt zum Beispiel das Iterieren über die Instanzvariablen eines Objektes. Es können zur Laufzeit der Typ jeder Instanzvariable, der Name und der aktuelle Wert abgefragt werden. Nun kann etwa ein Framework definiert werden, das einen Hot Spot aufweist, der einzig und alleine diese Eigenschaft, nämlich das Iterieren über seine Instanzvariablen und dessen Manipulation, aufweist. Da das Meta-Informationsmodell diesen Service für alle Objekte anbietet, ist jedes Objekt zu dieser abstrakten Einheit kompatibel und kann zur Spezialisierung der Architektur eingesteckt werden. (Daraus sollte ersichtlich sein, was eingangs mit geringerer Typsicherheit gemeint war.)

Der Vorteil der Nutzung von Meta-Informationen in der Framelet-Entwicklung besteht darin, daß dadurch “intelligente” Framelets definierbar sind, die sich zum Teil automatisch konfigurieren, indem sie sich mit den für die Hot Spots eingesteckten Objekten generisch koppeln. Wie in Abschnitt 3 gezeigt wird, entfalten sich die Möglichkeiten von Meta-Informations-basierten Hot-Spots besonders gut im Kontext von Framelets.

Um die gewonnene Flexibilität sinnvoll nutzen zu können, muß für solche abstrakten Einheiten eine bestimmte Semantik, die angibt , was eine abstrakte Komponente zu leisten hat, andersweitig definiert werden. Das im folgenden Abschnitt präsentierte Framelet illustriert diesen Aspekt, indem es eine Namenskonvention, also ein sehr einfaches Mittel zur Semantikdefinition, benutzt. Es wird weiters aus der Fallstudie ersichtlich, daß der eingangs erwähnte Verlust von Typsicherheit durch eine geschickte Definition der Semantik, die auf Ebene des Anwendungsbereichs eines Framelets und somit auf einer anderen Ebene als der Programmiersprache erfolgt, wettgemacht wird.

3 Ein Framelet-Beipiel

Die Fallstudie ist aus dem Bereich der GUI-Programmierung, da Grundwissen über diese Domäne als bekannt vorausgesetzt werden kann und somit keine detaillierten Erläuterungen nötig sind. Man beachte, daß die Anwendbarkeit von Framelets nicht auf grafische Oberflächen beschränkt ist.

In der Benutzeroberfläche vieler Anwendungen finden sich wiederholt Listen von Einträgen, sogenannte List-Boxes. Zusätzlich werden Schaltflächen (Buttons) benötigt, um Elemente zur List-Box hinzuzufügen, beziehungsweise zum Ändern und Löschen von Einträgen. Diese GUI-Komponenten und die damit verbundenen Interaktionen müssen immer wieder neu implementiert werden. Ein solches kleines Programm-system läßt sich gut in ein Framelet verpacken. Wir nennen es List-Box-Framelet.

Spezialisierungen des List-Box-Framelets unter-scheiden sich nur im Dialog, der für das Bearbeiten eines Listenelements benutzt wird. Abbildung 3 veranschaulicht schematisch eine Spezialisierung des List-Box-Framelets. Die Pfeile zeigen die Interaktionen zwischen den sichtbaren Framework-Komponenten an: Wenn der Endbenutzer *Add* oder *Modify* drückt, öffnet das Framelet den spezifischen Dialog. Im Fall vom *Modify* werden die Inhalte des selektierten Elements der List-Box angezeigt. Durch Drücken des OK Buttons im Dialog wird ein Element der Liste hinzugefügt bzw. geändert. Durch Drücken auf den *Remove*-Button wird das selektierte Element von der Liste gelöscht. Das Framelet muß auch dafür sorgen, daß die *Modify*- und *Remove*-Buttons nur dann aktiviert werden können, wenn ein Element selektiert ist.

3.1 Überlegungen zum Design und zur Implementierung

Ein wichtiges Design-Ziel eines Framelets ist, die Wiederverwendung so einfach wie möglich zu gestalten. Das heißt im Fall des List-Box-Framelets, daß der Entwickler nur den Dialog zur Verfügung stellen muß, um die Daten eines Elements in der List-Box anzuzeigen. Außerdem muß das Framelet den Typ der angezeigten Elemente kennen. Idealerweise handhabt das Framelet automatisch die Übertragung von Daten zwischen den Instanzvariablen einer Elementinstanz und den GUI-Komponenten im Dialog. Wie kann das erreicht werden?

Die Grundidee zur Automatisierung der Daten-übertragung ist, eine einfache Namenskonvention zu

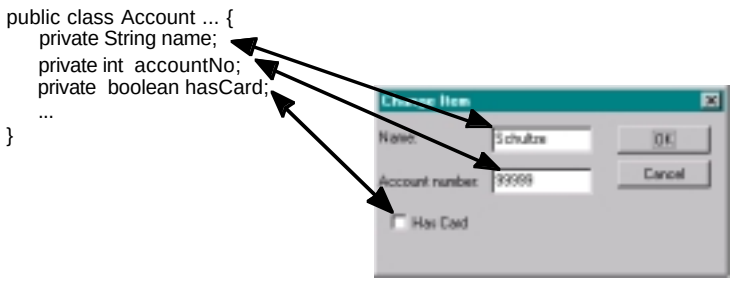

Abbildung 4 Automatisierter Datenaustausch zwischen Listenelement und Dialogfeld.

definieren: Die Namen der Instanzvariablen der Klasse, die ein Element repräsentiert, sind mit den Namen der entsprechenden GUI-Elemente im Dialog identisch. Eine Klasse Account könnte etwa dem Typ der angezeigten Listenelemente entsprechen. Eine der Instanzvariablen von Account ist beispielsweise accountNo. Daher hat auch das Bearbeitungsfeld im Elementdialog den Namen accountNo. (Dies ist in Abbildung 3 nicht sichtbar. Das GUI-Element mit dieser internen Bezeichnung zeigt dort die Eingabe "99999" an.)

Eine Komponente des List-Box-Framelets, ein sogenannter Matcher, sorgt für die Datenübertragung gemäß der Namenskonvention. Abbildung 4 veranschaulicht schematisch den Zweck eines Matchers. Um die Datenübertragung zu automatisieren, muß die Implementierung der Klasse Matcher auf Meta-Informationen beruhen: Wenn ein Matcher-Objekt Daten von einem Listenelement zum Dialog transferiert, iteriert der Matcher über alle Instanzvariablen der Elementklasse, erhält den Typ und den Wert und sucht jeweils nach dem entsprechenden GUI-Element im Dialog. (Gemäß der Namenskonvention sucht das Matcher-Objekt, sobald es den Namen einer Instanzvariable in der Elementklasse kennt, nach der gleichnamigen Instanzvariable im Dialog.) Die Übertragung in die andere Richtung, also vom Dialog hin zum Listenelement, funktioniert auf analoge Weise.

Das Meta-Informationsmodell in Java ist für das oben skizzierte Design ausreichend. Die Realisierung des List-Box-Framelets empfiehlt die Einhaltung der Konvention, die im Java-Beans-Standard (Sun, 1998) vorgeschlagen ist: Für jede Instanzvariable <name> sollten die beiden Manipulationsmethoden set<Name>(...) und get<Name>() definiert werden, um die Variable zu ändern. Das Meta-Informationsmodell in Java erlaubt zur Laufzeit sowohl die Abfrage der Instanzvariablen und Methoden, als auch deren Änderung bzw. deren Aufruf.

3.2 Eigenschaften und Wiederverwendung von Framelets

Zieht man die in Abschnitt 2 angeführten Eigenschaften eines Framelets in Betracht, stellt das List-Box-Framelet ein typisches Beispiel eines kompakten Architekturbausteines dar:

- Das List-Box-Framelet zieht nicht den Hauptkontrollfluß einer Anwendung an sich. Dennoch beruht es auf dem Hollywood-Prinzip von White-Box-Architekturen.
- Das List-Box-Framelet ist klein. Es besteht aus den Klassen Matcher, LBoxGroup und CollView. Die Klasse LBoxGroup entspricht der Gruppe der GUI-Elemente, die typischerweise mit einer List-Box verbunden sind, also die List-Box selbst sowie die Buttons *Add*, *Modify* und *Remove*. Die Klasse LBoxGroup implementiert auch den Steuerfluß zum Öffnen des Dialogs, wenn *Add* oder *Remove* gedrückt werden, und aktiviert bzw. deaktiviert Buttons entsprechend der Selektion in der List-Box. CollView ist eine Hilfsklasse für die Verwaltung und Anzeige der Listenelemente.
- Die Schnittstelle des List-Box-Framelets ist einfach und klar definiert. Ein Entwickler, der dieses Framelet wiederverwendet, übergibt dem Konstruktor von LBoxGroup nur zwei Parameter. Der erste ist der Name der Listenelementklasse als String. Aufgrund dieser Information können zur Laufzeit entsprechende Instanzen erzeugt werden. Der zweite Parameter ist eine Referenz auf den Dialog.
  Die Methode getItems() der Klasse LBoxGroup liefert die in der Liste enthaltenen Objekte.
  Es kann spezifiziert werden, ob eine einfache Listbox oder eine mit einer tabellarischen Rasterfeldansicht verwendet wird. Das stellt einen Black-Box-Aspekt des Framelets dar.

Das List-Box-Framelet repräsentiert lediglich ein sehr einfaches, aber anschauliches Beispiel für ein Framelet. Im Rahmen eines Projektes mit der RACON Software GmbH Linz, dem Softwarehaus der Raiffeisenbankengruppe Oberösterreich, wurden weitere Framelets im Rahmen der Umgestaltung der bei RACON eingesetzten Client-/Server-Architektur entwickelt. Beispielsweise wird durch ein RPC-Framelet der Aufruf der RACON-spezifischen *Remote Procedures* signifikant vereinfacht. Die RACON-RPC-Bibliothek besteht aus ca. 50 C-Funktionen, die zur Manipulation von Daten am Server und Host aufgerufen werden. Die

Implementierung eines einzelnen *Remote-Procedure*-Aufrufs erfordert jeweils einen Programmieraufwand von ca. 200–300 Zeilen Code. Beispielsweise müssen die Parameterwerte vor dem Aufruf einer *Remote Procedure* aus GUI-Elementen gelesen werden. Die Rückgabeparameter befinden sich in in einer C-Array-Struktur, aus der abhängig vom jeweiligen RPC die entsprechenden Parameter quasi herausgefiltert werden müssen. Die Werte müssen dann wiederum in entsprechenden GUI-Elementen angezeigt werden. Der eben skizzierte Programmcode im Umfeld eines RPC-Aufrufes ist immer ähnlich aufgebaut. Da der Programmcode dennoch zu unterschiedlich ist, um ihn in eine einfache Methode oder Funktion herauszufaktorisieren, wurde dafür ein Framelet mit selbstkonfigurierenden Eigenschaften entwickelt. Damit erfordert ein Remote Procedure Aufruf nur noch wenige Zeilen Code. Sowohl das oben skizzierte List-Box-Framelet als auch das RPC-Framelet stellen handliche Architekturkomponenten dar, die bei RACON eingesetzt werden.

4 Zusammenfassung und Perspektiven

Aufgrund erster Erfahrungen mit Framelets sehen wir in solchen kleinen flexiblen Architektur-Komponenten einen vielversprechenden Ansatz. Die skizzierte Kombination der White-Box- und Black-Box-Architekturaspekte in Form von Framelets kann zur Lösung der bekannten Probleme komplexer Frameworks beitragen. Das Beispiel des List-Box-Framelets zeigt, daß Framelets praktisch einsetzbare Bausteine für verschiedene Arten von Anwendungen sein können.

Obwohl Framework-zentrierte Entwurfsmuster (Gamma et al., 1995; Buschmann et al., 1996) Wissen über elegante Softwarearchitekturen darstellen, sind sie zu klein, um als wiederverwendbare Architekturkomponenten unmittelbar eingesetzt zu werden. Aufbauend auf den ersten Erfahrungen mit Framelets sehen wir Framelets als einen pragmatischen Kompromiß zwischen Entwurfsmustern und Anwendungsframeworks. Framelets können als die Kombination einiger Entwurfsmuster zu einem wiederverwendbaren Architekturbaustein betrachtet werden. Das List-Box-Framelet basiert zB auf einer Kombination des Musters *Observer* und des Musters *Bridge* (siehe Gamma et al., 1995). Die Aktualisierung der Listenelemente greift auf das *Observer*-Design zurück. Das *Bridge*-Design wird beispielsweise in der Matcher-Klasse angewendet.

Es gilt zu klären, bis zu welchem Grad eine Anwendung auf Framelets basieren kann. Das List-Box-Framelet und das RPC-Framelet lassen vermuten, daß für bestimmte Anwendungsbereiche Framelet-Familien entwickelt werden können, die lose gekoppelt werden. Somit können Framelets als mögliches Strukturierungsmittel von Frameworks betrachtet werden. Es kann in diversen Anwendungsbereichen sinnvoll sein, anstatt eines Anwendungsframeworks eine Familie von Framelets zu entwickeln.

Namenskonventionen stellen eine sehr einfache Art der Semantikdefinition dar. Deshalb werden sich künftige Arbeiten darauf konzentrieren, herauszufinden, wie Domänen-spezifische Semantik von abstrakten Einheiten praktikabel definiert werden kann.

Literatur

Bass L., Clements P. und Kazman R. (1998) *Software Architecture in Practice*. Reading, Massachusetts: Addison-Wesley

Buschmann F., Meunier R., Rohnert H., Sommerlad P. und Stal M. (1996) *Pattern-Oriented Software Architecture—A System of Patterns*. Wiley and Sons

Fayad M., Johnson R. und Schmidt D. (1998) *Object-Oriented Application Frameworks*. Wiley & Sons.

Gamma E., Helm R., Johnson R. and Vlissides J. (1995). *Design Patterns—Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison-Wesley

Lewis T., Rosenstein L., Pree W., Weinand A., Gamma E., Calder P., Andert G., Vlissides J., Schmucker K. (1995): *Object-Oriented Application Frameworks*. Manning Publications/Prentice Hall.

Pomberger G. und Pree W. (Eds.) (1998) *Special issue on componentware*. Software—Concepts & Tools, Springer-Verlag

Pree W. (1995). *Design Patterns for Object-Oriented Software Development*. Reading, MA: Addison-Wesley/ACM Press

Pree W. (1997). *Komponentenbasierte Softwareentwicklung mit Frameworks*. Heidelberg: dpunkt (Deutsche Übersetzung von: *Framework Patterns*. New York City: SIGS Books, 1996)

Sparks S., Benner K., Faris C.: *Managing Object-Oriented Framework Reuse*. IEEE Computer 29,9 (Sept 96), 52-62.

Sun Microsystems (1998): *JavaBeans specification 1.01*. http://splash.javasoft.com/beans/spec.html.

Szyperski C. (1998). *Component Software—Beyond Object-Oriented Programming.* Addison-Wesley

Weinand A., Gamma E. und Marty R. (1989). Design and implementation of ET++, a seamless object-oriented application framework. *Structured Programming*, 10(2), Springer Verlag