20 Framework-Entwurf

Ansgar Scherp, Susanne Boll

Die Wiederverwendung von Software-Architekturen ist einer der wichtigsten Faktoren im Software-Entwicklungsprozess, um Entwicklungsaufwand und -kosten zu reduzieren und gleichzeitig die Zuverlässigkeit und Qualität von Software zu steigern [BR91, BR90, SGM02].

Definition (Software-Framework) *Ein Software-Framework stellt typischerweise ein halbfertiges Architekturgerüst für einen (komplexen) Anwendungsbereich dar, das auf die Bedürfnisse und Anforderungen einer konkreten Anwendung aus diesem Anwendungsbereich angepasst werden kann.*

Die Motivation, die zur Konzeption und Erstellung eines Frameworks führt, ist oft ein ausgeprägtes Domänenwissen. Die langjährigen Erfahrungen bei der Entwicklung von Software-Anwendungen in einer bestimmten Domäne werden mit Hilfe eines Frameworks in ein einheitliches Architekturgerüst gegossen. Frameworks erweisen sich gerade dann als nützlich, wenn flexible und dennoch einfach wiederzuverwendende Software-Architekturen zu entwickeln sind [Pre97c], und stellen einen erprobten Ansatz zur Wiederverwendung von Software-Architekturen im Großen dar [DW99]. Der Aufwand zur Entwicklung von Frameworks ist signifikant höher als bei der konventionellen Anwendungsentwicklung. Daher stellen sie eine langfristige Investition dar, die sich erst dann auszahlt, wenn das Framework in mehreren Anwendungen eingesetzt wird [Pre97a].

In Abschnitt 20.1 werden zunächst die generellen Eigenschaften von Frameworks vorgestellt. Darauf basierend wird in Abschnitt 20.2 der Begriff des Frameworks geklärt. Dabei wird zwischen objektorientierten und komponentenbasierten Frameworks unterschieden. Zudem wird das Konzept des Frameworks gegen verwandte Konzepte abgegrenzt. Frameworks entstehen nicht von heute auf morgen und sind das Ergebnis eines längeren Entwicklungsprozesses. In Abschnitt 20.3 wird daher auf den Entwicklungsprozess von Frameworks eingegangen. In Abschnitt 20.4 wird der Entwurf von objektorientierten Frameworks beschrieben. Dabei wird

als Beispiel der Hotspot-getriebene Entwurf im Detail vorgestellt sowie kurz auf andere Entwurfsmethoden eingegangen. Abschließend wird in Abschnitt 20.5 der Entwurf von Komponenten-Frameworks betrachtet, bevor das Kapitel mit dem Fazit schließt.

20.1 Eigenschaften von Frameworks

Frameworks zeichnen sich typischerweise durch folgende drei Eigenschaften aus: die Umkehrung des Kontrollflusses, die Vorgabe einer konkreten Anwendungsarchitektur und ihre Anpassbarkeit durch vordefinierte Variationspunkte. Diese Eigenschaften werden im Folgenden kurz diskutiert.

20.1.1 Umkehrung des Kontrollflusses

Konventionell entwickelte Anwendungen nutzen typischerweise eine Reihe von Klassenbibliotheken, um die eigenen Funktionalitäten möglichst einfach zu realisieren. Der Kontrollfluss solcher Anwendungen, also die Reihenfolge, in der die Operationen ausgeführt werden [BD00], wird typischerweise von ihnen selbst gesteuert. Die Klassenbibliotheken werden lediglich von der sie nutzenden Anwendung aufgerufen, übernehmen jedoch niemals den Hauptkontrollfluss der Anwendung. Dieser Aufruf externer Klassenbibliotheken wird *Call-down-Prinzip* genannt und ist in Abbildung 20.1(a) dargestellt. Der obere Teil der Grafik symbolisiert die Anwendung und der untere Bereich die verschiedenen Funktionalitäten, die von externen Klassenbibliotheken zur Verfügung gestellt werden. Immer, wenn die Anwendung Funktionalitäten einer Klassenbibliothek benötigt, ruft sie die dort zur Verfügung gestellten Funktionalitäten auf. Klassenbibliotheken liefern keine Anwendungsstruktur mit, d. h., die Anwendungsentwickler müssen die Architektur ihrer Anwendung selbst entwerfen.

Frameworks invertieren den Kontrollfluss einer Anwendung [Has02]. Im Gegensatz zu konventionell entwickelten Anwendungen wird der Hauptkontrollfluss einer Anwendung die mit Hilfe eines Frameworks entwickelt wurde, vom Framework gesteuert. Dieser Aufrufmechanismus wird *Call-back-* oder auch *Hollywood-Prinzip* (»Don't call us, we'll call you!«) genannt und ist in Abbildung 20.1(b) dargestellt. Zentrale Bestandteile der Anwendungsarchitektur werden bereits vom Framework vorgegeben. Die anwendungsspezifischen Funktionalitäten einer Anwendung, die das Framework nutzt, werden aus dem Framework heraus aufgerufen. So hat das Framework den Hauptkontrollfluss über die Anwendung inne [JF88].

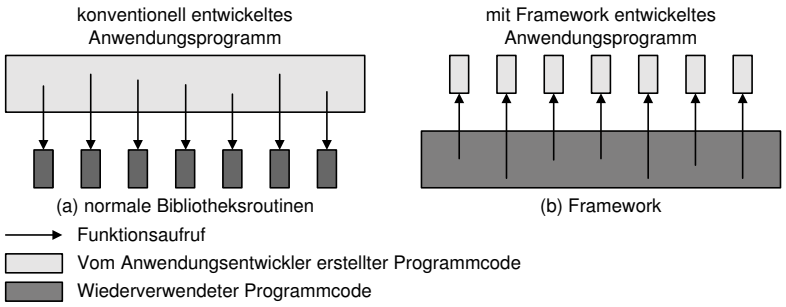

Abbildung 20.1: Darstellung des (a) Call-down-Prinzips und (b) Call-back-Prinzips (nach [Ack96, FPR02])

20.1.2 Vorgabe einer konkreten Anwendungsarchitektur

Gute Frameworks definieren bereits einen Großteil der Anwendungsarchitektur [Pre97c] und halten sie nur noch an bestimmten Punkten flexibel. Das sind die Stellen, an denen anwendungsspezifische Funktionalitäten integriert und aufgerufen werden (siehe nächsten Abschnitt 20.1.3 sowie [FPR02, Pre97c, Ack96]).

Die Invertierung des Kontrollflusses gibt Frameworks die Möglichkeit, als generische, erweiterbare, halbfertige Software-Architekturen (»Skelette«) zu dienen. Die Funktionalitäten, die durch die Anwendungsentwickler hinzugefügt werden, passen den generischen Algorithmus, der in einem Framework definiert und implementiert ist, an die Bedürfnisse und Anforderungen einer speziellen Anwendung an [JF88].

20.1.3 Anpassbarkeit durch Variationspunkte

Um für eine Vielzahl von Anwendungen in einer bestimmten Domäne eingesetzt werden zu können, stellen Frameworks flexible *Variationspunkte* (engl. *variation points*) zur Verfügung, an denen das Framework an die Anforderungen einer konkreten Anwendung angepasst und erweitert werden kann. Genauer gesagt handelt es sich hierbei um so genannte *offene Variationspunkte*, da die Anzahl der konkreten Ausprägungen der Variationspunkte des Frameworks nicht vorgegeben ist (siehe dazu die Diskussion um offene und geschlossene Variationspunkte im Zusammenhang mit Software-Produktlinien in Abschnitt 19.3.2). Offene Variationspunkte werden auch *Erweiterungspunkte* genannt. Sie sind die Stellen in der Framework-Architektur, an denen eine Entscheidung für eine bestimmte Funktionalität zwar bereits typisiert ist, die letztendliche Ausprägung

dieser Funktionalität aber offen gelassen und erst durch die jeweilige konkrete Anwendung bestimmt wird (vgl. [CBB+03]).

20.2 Arten von Frameworks

Neben der in der Einleitung angegebenen Definition, dass ein Framework ein halbfertiges Architekturgerüst für einen komplexen Anwendungsbereich darstellt, finden sich in der Literatur viele weitere und unterschiedliche Definitionen. Im Rahmen dieses Buchkapitels ist eine Differenzierung von Frameworks hinsichtlich der verwendeten Technologie zur Anpassung an die Anforderungen einer konkreten Anwendung von Bedeutung. Dazu sind historisch bedingt die objektorientierten Frameworks und die komponentenbasierten Frameworks zu unterscheiden: Im Falle von objektorientierten Frameworks geschieht die Anpassung durch Vererbung und (partielle) Überschreibung von abstrakten Klassen oder durch unterschiedliche Implementierungen von vordefinierten Schnittstellen. Bei komponentenbasierten Frameworks erfolgt die Anpassung ausschließlich durch Komposition von Komponenten, die die vordefinierten Schnittstellen des Komponenten-Frameworks implementieren. Neben diesen beiden Arten von Frameworks gibt es auch Mischformen von objektorientierten und komponentenbasierten Frameworks, die beide Technologien zur Anpassung verwenden.

Aus den Definitionen in der Literatur geht nicht immer eindeutig hervor, welche Art von Framework adressiert wird. Oftmals ist jedoch implizit eine der beiden Arten gemeint, wie beispielsweise in [GHJV04, Gam92, And04, HK03, Bal00, Pre95, Rog97]. Diese Framework-Definitionen wurden analysiert und in eine der beiden Framework-Arten einsortiert. Darauf basierend wird in den folgenden beiden Abschnitten 20.2.1 und 20.2.2 die in der Einleitung dieses Buchteils auf Seite 329 angegebene Definition objektorientierter und komponentenbasierter Frameworks hergeleitet. Zudem werden die Unterschiede zwischen objektorientierten und komponentenbasierten Frameworks dargestellt. Abschließend wird in Abschnitt 20.2.3 kurz auf Mischformen objektorientierter und komponentenbasierter Frameworks eingegangen.

20.2.1 Objektorientierte Frameworks

Seit der Einführung des Framework-Konzeptes Ende der 80er Jahre haben objektorientierte Frameworks große Aufmerksamkeit erhalten. Heute existieren objektorientierte Frameworks für eine Vielzahl von Anwendungsbereichen [Bos00]. Was die Definition objektorientierter Frameworks betrifft, stimmen die meisten Autoren darin überein, dass ein objektorientiertes

Framework eine wiederverwendbare Software-Architektur darstellt, die sowohl die Wiederverwendung des Entwurfs als auch des Programmcodes einschließt, wie beispielsweise [BMM+99, Ack96]. Von dieser Übereinstimmung abgesehen unterscheiden sich viele Definitionen. So werden zum Beispiel in [Bos00] objektorientierte Frameworks in kleinere Bestandteile zerlegt und in [Gov99] werden Unter- beziehungsweise Teilarten von objektorientierten Frameworks definiert. Die nach [BMM+99] wahrscheinlich am häufigsten zitierte und im Rahmen dieses Buchkapitels verwendete Definition ist die von Johnson und Foote [JF88], nach der ein objektorientiertes Framework aus einer Menge von Klassen besteht, die einen abstrakten Lösungsentwurf für eine Familie verwandter Probleme darstellt. Ein objektorientiertes Framework besteht aus einer Menge von konkreten und – insbesondere – abstrakten Klassen [Bal00, WJ90], die ein Software-System in Form einer generischen Anwendung für eine spezielle Domäne bereitstellen [Pre95]. Es handelt sich also um einen unvollständigen Entwurf und eine unvollständige Implementierung für Anwendungen in einer speziellen Domäne bzw. einem Problembereich [Bos00]. Die Variationspunkte objektorientierter Frameworks sind dabei die abstrakten Framework-Klassen. Zahlreiche Autoren folgen dieser Definition, wie zum Beispiel [SGM02, GHJV04, Pre95, BD00]. Das objektorientierte Framework definiert die Struktur der Klassen und Objekte und deren Verantwortlichkeiten. Es legt fest, wie Klassen und Objekte zusammenarbeiten (Kollaborationen) und wie der Kontrollfluss aussieht [Bal00, GHJV04, Pre95]. Ein objektorientiertes Framework gibt die Architektur einer Anwendung vor, damit sich die Anwendungsentwickler auf die Details der Anwendung konzentrieren können [Bal00]. Die Anwendungsentwickler erweitern das objektorientierte Framework um anwendungsspezifische Funktionalitäten durch Definieren konkreter Unterklassen der abstrakten Framework-Klassen [Bos00, Bal00]. Diese selbst definierten Unterklassen werden vom Framework nach dem Call-back-Prinzip aufgerufen (siehe Abschnitt 20.1.1).

Das Offen/Geschlossen-Prinzip Ein objektorientiertes Framework soll dem Offen/Geschlossen-Prinzip (engl. *open/close principle*) genügen [Zül05, Mey97]. Das heißt, es muss *offen* sein in Bezug auf (zukünftige) Erweiterungen und *geschlossen* hinsichtlich Modifikationen. Offen in Bezug auf (zukünftige) Erweiterungen bedeutet, dass das objektorientierte Framework um neue Funktionalitäten erweitert werden kann, auch um solche, die beim Entwurf des Frameworks noch nicht vorhersehbar waren. Geschlossen hinsichtlich Modifikation bedeutet, dass die abstrakte Anwendungslogik, die vom objektorientierten Framework den konkreten Anwendungen zur Verfügung gestellt wird, nicht durch Bildung konkreter Unterklassen modifiziert werden kann, um ein anderes Programmverhalten zu erzeugen, als von den Framework-Entwicklern vorgesehen war. Das bedeutet, dass

die konkreten Unterklassen ausschließlich die abstrakten Methoden der Oberklassen des objektorientierten Frameworks implementieren. Die im Framework bereits implementierten Klassen und Methoden dürfen nicht überschrieben und modifiziert werden.

In der Praxis wird das Offen/Geschlossen-Prinzip nicht immer eingehalten: So findet man in einigen Frameworks, wie zum Beispiel in .NET [Mic05b] von Microsoft und der grafischen Benutzungsoberfläche Swing [Sun05] von Sun Microsystems, eine Default-Implementierung vor, die von den Anwendungsentwicklern überschrieben wird.

Wiederverwendung von objektorientierten Frameworks Hinsichtlich der Wiederverwendung objektorientierter Frameworks wird zwischen der *White-Box-* und *Black-Box-Nutzung* unterschieden. Wird das Verhalten des objektorientierten Frameworks durch Vererbung von abstrakten Klassen aus der Klassenhierarchie des Frameworks angepasst, indem die abstrakten Methoden der Oberklassen des Frameworks in den konkreten Unterklassen der Anwendung überschrieben werden [JF88], so spricht man von einer White-Box-Nutzung des Frameworks bzw. von einem White-Box-Framework [BD00]. Diese Variante zur Nutzung objektorientierter Frameworks hat ihren Namen von der Eigenschaft, dass zur Umsetzung der anwendungsspezifischen Funktionalitäten durch Vererbung ein Einblick in die Implementierung des objektorientierten Frameworks möglich ist. Teile der Implementierung des White-Box-Frameworks liegen zur Inspektion offen, ohne aber eine Modifikation an der internen Implementierung zu erlauben (vgl. [SGM02], S. 555).

Im Gegensatz zu White-Box-Frameworks basiert das Anpassungskonzept von Black-Box-Frameworks nicht auf abstrakten Klassen, sondern auf einer Anzahl fertiger, d. h. implementierter Klassen. Änderungen am Verhalten des Frameworks werden durch Komposition dieser Klassen erreicht und nicht durch Vererbung [Pre97c]. Eine Black-Box-Wiederverwendung findet ausschließlich auf Basis von wohldefinierten Schnittstellen der Klassen und deren »vertraglichen« Spezifikationen statt [BD00]. Dazu implementieren die Anwendungsentwickler neue, anwendungsspezifische Funktionalitäten gegen die externen Framework-Schnittstellen. Folglich ist bei der Black-Box-Wiederverwendung die Qualität der Schnittstellenspezifikation von entscheidender Bedeutung [Pre97c].

Durch partielles Öffnen eines Black-Box-Frameworks zur White-Box-Wiederverwendung können beliebige »Graustufen« gebildet werden [SGM02]. Tatsächlich sind gängige objektorientierte Frameworks weder reine White-Box- noch reine Black-Box-Frameworks. Typischerweise entwickeln sich White-Box-Frameworks mit der Zeit, d. h. mit zunehmenden Reifegrad, zu Black-Box-Frameworks [RJ97, JF88]. Diese Entwicklung ergibt sich

vornehmlich durch häufiges Wiederverwenden des White-Box-Frameworks in konkreten Anwendungen.

Komposition von objektorientierten Frameworks Während ursprünglich angenommen wurde, dass zur Konstruktion einer Anwendung nur ein einzelnes objektorientiertes Framework verwendet wird, ist in den letzten Jahren zu beobachten, dass zunehmend mehrere Frameworks für die Konstruktion einer Anwendung zum Einsatz kommen [Bos00, SGM02]. Der Grund dafür liegt darin, dass heutige Anwendungen zunehmend mehrere Domänen abdecken (müssen), ein objektorientiertes Framework aber immer nur für genau einen Anwendungsbereich konzipiert ist. Werden mehrere unabhängig voneinander entwickelte objektorientierte Frameworks für die Konstruktion einer Anwendung kombiniert, so führt dies oft zu einer Fülle von Problemen. So beabsichtigt jedes objektorientierte Framework, bedingt durch seine Natur, den Hauptkontrollfluss der Anwendung zu übernehmen (siehe Abschnitt 20.1.1). Zudem kann es passieren, dass sich die verwendeten Frameworks in ihrer Funktionalität überlappen [BMM+99]. Bestimmte Teile der Anwendungsdomäne werden dann in mehreren Frameworks modelliert. Dies betrifft oftmals auch die Basisklassen der objektorientierten Frameworks. Versucht man ein objektorientiertes Framework durch Änderungen an den Basisklassen mit anderen Frameworks kompatibel zu machen, so kann das zahlreiche Änderungen an abgeleiteten Klassen zur Folge haben und grundlegende Entwurfsentscheidungen verwerfen lassen [Pre96b].

Die Interaktion zwischen verschiedenen objektorientierten Frameworks kann nur auf einer noch höheren Abstraktionsebene gelöst werden. Dazu bedarf es einer Ebene zur Nutzung objektorientierter Frameworks, die beschreibt, wo, wann und wie sich Teile von Frameworks überlappen oder interagieren [SGM02]. Wie ein normales objektorientiertes Framework die Kombination konkreter Klassen unterstützt, so könnten höherwertige »Teilsystem-Frameworks«, so genannte *Frameworks zweiter Ordnung*, dazu dienen, komplexere Teilsysteme zu kombinieren. Jedes dieser Teilsysteme kann als traditionelles objektorientiertes Framework (erster Ordnung) strukturiert und verstanden werden [SGM02]. Die Erarbeitung eines allgemeinen Ansatzes zur Bildung solcher Framework-Hierarchien ist Gegenstand aktueller Forschung [Pre97c]. Bestrebungen in diese Richtung sind Komponenten-Frameworks, die im folgenden Abschnitt vorgestellt werden.

20.2.2 Komponentenbasierte Frameworks

Während sich objektorientierte Frameworks über die Struktur ihrer Klassen und Objekte und deren Verantwortlichkeiten definieren, stehen bei den komponentenbasierten Frameworks bzw. Komponenten-Frameworks

die einzelnen Software-Komponenten und deren Schnittstellen im Vordergrund. Eine recht abstrakte Definition sieht Komponenten-Frameworks als Sammlung von verschiedenen einzelnen Komponenten mit vordefiniertem Kooperationsverhalten an, mit dem Zweck, Aufgaben in einem bestimmten Anwendungsbereich zu erfüllen [And04, Pre97a]. Szyperski et al. präzisieren das, indem sie Komponenten-Frameworks als Sammlung von Regeln und Schnittstellen definieren [SGM02], die die Interaktion von Komponenten eines Komponenten-Frameworks regeln (siehe auch [FSJ99b]). Diese Regeln und Schnittstellen stellen vetragsähnliche Vereinbarungen zwischen einem Anbieter eines »Dienstes« (die Software-Komponente) und dessen »Nutzer« (das Komponenten-Framework bzw. die konkrete Anwendung) dar, die neben funktionalen auch nicht-funktionale Aspekte des Dienstes beinhalten kann. Ein Komponenten-Framework erlaubt es Instanzen von Komponenten, die diesen Verträgen genügen, an bestimmten, durch diese Verträge typisierten Erweiterungspunkten in das Komponenten-Framework einzuhängen. Die Interaktion zwischen diesen Instanzen wird dabei vom Komponenten-Framework reguliert. Dies bedeutet, dass der Hauptkontrollfluss der Anwendung – wie bei den objektorientierten Frameworks – beim Komponenten-Framework liegt. Die einzelnen Komponenten eines Komponenten-Frameworks geben noch keine Anwendungsarchitektur vor, wohl aber das Komponenten-Framework selbst, also die Organisation und Strukturierung der einzelnen Komponenten, die sich aus den Anforderungen des betrachteten Anwendungsbereiches ergeben (vgl. [Sih01], S. 70). Im Gegensatz zu den objektorientierten Frameworks erfolgt die Anpassung des Verhaltens eines Komponenten-Frameworks an die Anforderungen einer konkreten Anwendung ausschließlich durch Komposition. Anstelle einzelner konkreter Klassen werden jedoch verschiedene Instanzen der Framework-Komponenten (und deren Klassen) komponiert. Folglich stellen Komponenten die Variationspunkte eines Komponenten-Frameworks dar.

Die einzelnen Komponenten eines Komponenten-Frameworks werden oftmals mit Hilfe objektorientierter Frameworks entwickelt [Bos00]. Dabei sollte ein objektorientiertes Framework immer nur zur Entwicklung von genau einer Komponente verwendet und nicht über mehrere Komponenten verteilt werden, um keine Vererbungsbeziehungen von Klassen verschiedener Komponenten zu erzeugen. Andernfalls ist eine der wichtigsten Eigenschaften von Software-Komponenten verletzt, nämlich die, dass eine Komponente *independently deployable* sein soll, d. h. als in sich geschlossene Einheit ausgeliefert (und eingesetzt) werden kann [SGM02].

Ein Komponenten-Framework kann alleine eingesetzt werden oder wieder mit anderen Komponenten oder Komponenten-Frameworks kooperieren. Im ersteren Fall wird die Anwendung durch genau ein Komponenten-Framework bestimmt. Um mit anderen Komponenten-Frameworks kooperieren zu können, sollten Komponenten-Frameworks ebenfalls als Kom-

ponenten modelliert werden. Das Konzept der Komponenten-Frameworks kann dann auch hierarchisch angewendet werden. Komponenten-Frameworks, die selbst wieder als Komponenten realisiert sind, sind im Gegensatz zu objektorientierten Frameworks gleich wieder auf das Zusammenspiel mit anderen Komponenten-Frameworks ausgelegt und lassen sich damit leichter integrieren [Sih01]. Komponenten-Frameworks werden als die derzeit höchste Stufe zur Wiederverwendung von Software-Architekturen angesehen [Bos00].

20.2.3 Mischformen von Frameworks

Neben objektorientierten und komponentenbasierten Frameworks sind auch Mischformen von Frameworks möglich. Diese hybriden Frameworks haben sowohl die Eigenschaften von objektorientierten als auch von komponentenbasierten Frameworks. In diesem Fall geschieht ein Teil der Anpassungen des Frameworks an die konkrete Anwendung über Vererbung, während ein anderer Teil über Implementierung von Schnittstellen realisiert wird. Durch Veränderung des Anteils, der durch Vererbung an die konkrete Anwendung angepasst wird, bzw. durch Veränderung des Anteils, der durch Implementierung von Schnittstellen realisiert wird, sind beliebige Varianten von Mischformen möglich.

20.2.4 Abgrenzung von Frameworks zu anderen Konzepten

In den vorangegangenen Abschnitten wurde der Begriff des Frameworks eingeführt. Es wurde eine Unterscheidung zwischen objektorientierten und komponentenbasierten Frameworks vorgenommen und in Beziehung zueinander gestellt. In diesem Abschnitt wird das Konzept des Frameworks gegen die verwandten Konzepte der Klassenbibliothek, Framelets, Entwurfsmuster, Software-Produktlinie und Referenzarchitektur abgegrenzt.

Eine Klassenbibliothek ist eine organisierte Sammlung von Klassen, aus denen der Entwickler nach Bedarf beliebige Einheiten verwenden kann. Im Gegensatz zu Frameworks erzwingen Klassenbibliotheken keine bestimmte Anwendungsarchitektur und übernehmen auch nicht den Hauptkontrollfluss einer Anwendung. Klassenbibliotheken, wie beispielsweise Qt [Tro05] und Microsoft Foundation Classes [Mic05a], werden immer von der konkreten Anwendung nach dem Call-down-Prinzip (siehe Abschnitt 20.1.1) aufgerufen und nicht umgekehrt, wie es bei den Frameworks der Fall ist.

Das Konzept der Software-Muster wird ausführlich in Kapitel 17 vorgestellt. Die Abgrenzung von Software-Muster gegen Frameworks ist in der Einleitung zu diesem Buchteil zu finden. Neben den in Kapitel 17 vorgestellten Software-Mustern gibt es auch noch so genannte Metamuster.

Metamuster können zur Beschreibung des strukturellen Teils der Entwurfslösungen von Entwurfsmustern verwendet werden und dienen zur Konstruktion objektorientierter Frameworks [JN99]. Sie werden ausführlich im Zusammenhang mit dem Entwurf objektorientierter Frameworks in Abschnitt 20.4.4 vorgestellt.

Framelets repräsentieren wie objektorientierte Frameworks flexible objektorientierte Software-Architekturen. Während objektorientierte Frameworks jedoch typischerweise abstrakte Architekturgerüste für komplexe Anwendungsbereiche darstellen, sind Framelets kleinerer Natur. Sie können als Kombination einiger Entwurfsmuster zu einem wiederverwendbaren Architekturbaustein betrachtet werden und realisieren generisch einen Systemaspekt, der in verschiedenen Anwendungen benötigt wird [PAS98]. Im Gegensatz zu konventionellen objektorientierten Frameworks stellen Framelets kleinere Architekturbausteine dar [PK00, PK99, PAS98], die aus weniger als zehn Klassen bestehen. Framelets ziehen auch nicht den Hauptkontrollfluss der Anwendung an sich.

Das Konzept der Software-Produktlinie wird ausführlich in Kapitel 19 und das Konzept der Referenzarchitektur in Kapitel 18 diskutiert. Eine Abgrenzung zu Frameworks ist wie bereits für die Software-Muster der Einleitung dieses Buchteils zu entnehmen.

20.3 Der Entwicklungsprozess von Frameworks

Die Entwicklung eines Frameworks, unabhängig davon, ob es sich um ein objektorientiertes oder komponentenbasiertes Framework handelt, ist eine anspruchsvolle Aufgabe und lohnt sich erst, wenn mehrere Anwendungen mit diesem Framework entwickelt werden. Erfahrungsgemäß sind mindestens drei Anwendungen notwendig, damit sich der höhere Entwicklungsaufwand eines Frameworks im Vergleich zur normalen Anwendungsentwicklung lohnt. Ein qualitativ hochwertiges Framework ist typischerweise das Ergebnis eines langen Entwicklungsprozesses und sollte mittels eines geeigneten Vorgehensmodells mit entsprechender Entwicklungsmethodik durchgeführt werden [CC02]. Aufgrund sich ändernder Anforderungen und der Komplexität von Frameworks ist der Entwicklungsprozess gekennzeichnet durch eine iterative und inkrementelle Vorgehensweise (vgl. [BMM+99]). Aus Beispielanwendungen und anderen existierenden Software-Produkten werden die relevanten Framework-Funktionalitäten für den Anwendungsbereich schrittweise abstrahiert. Dies schließt auch die Strukturierung des Frameworks hinsichtlich der unterstützten Anpassungsmöglichkeiten mit ein, d. h. die Bereitstellung von flexiblen Variationspunkten, mittels derer das Framework angepasst und erweitert werden kann. Diese Variationspunkte werden im iterativen Entwicklungsprozess

ermittelt und verfeinert [JN99]. Die Erfahrungen, die durch den Einsatz des Frameworks in Beispielanwendungen gewonnen werden, sind dabei die Eingabeparameter für die nächste Iteration des Framework-Entwicklungsprozesses. Diese iterative Entwicklung von Frameworks kann grob in die folgenden Phasen eingeteilt werden (siehe [BMM+99]): *Entwicklung*, *Nutzung*, *Komposition* sowie *Evolution und Wartung von Frameworks*. Die Eigenschaften und Herausforderungen dieser Phasen werden in den nächsten Abschnitten näher betrachtet.

20.3.1 Entwicklung von Frameworks

Die Entwicklung von Frameworks unterscheidet sich stark von der konventionellen Anwendungsentwicklung. Das liegt hauptsächlich daran, dass ein Framework alle Konzepte einer bestimmten Domäne betrachten und unterstützen soll, während bei der Entwicklung einer speziellen Anwendung nur die Konzepte von Interesse sind, die zur Umsetzung dieser einen Anwendung notwendig sind. Die Entwicklung eines Frameworks umfasst dabei die Aktivitäten, die typischerweise auch in Vorgehensmodellen für die klassische Software-Entwicklung zu finden sind (siehe [BMM+99]): Analyse der Anwendungsdomäne, Festlegen der Architektur, Entwurf, Implementierung, Testen und Dokumentation des Frameworks. Die Entwicklungsmethoden für die normale Anwendungsentwicklung unterstützen den Entwurf und die Entwicklung von Frameworks jedoch nicht in einem ausreichenden Maße. Der Entwurf von Frameworks erfordert einige neue Konzepte, die methodisch unterstützt werden müssen [BMM+99]. So muss zum Beispiel während der Analyse der Anwendungsdomäne berücksichtigt werden, dass sich im Gegensatz zur normalen Anwendungsentwicklung das Verhalten und die Konfiguration des Frameworks in verschiedenen Anwendungsfällen ändern können.

Bei der Entwicklung eines Frameworks ist die Festlegung der betrachteten Domäne (engl. *domain scope*) von entscheidender Bedeutung. Da ein Framework oft in vielen unterschiedlichen Szenarien einsetzbar sein soll, ist häufig eine Auseinandersetzung mit verwandten Themen erforderlich, die nicht zwangsläufig zu den Kernkompetenzen der Framework-Entwickler gehören [Sih01]. Wird die Domäne des Frameworks zu groß gewählt, so hat das Entwicklerteam nicht die notwendigen Erfahrungen und das Wissen in dieser Domäne, um dafür ein Framework zu entwickeln. Dies kann höhere Entwicklungskosten zur Folge haben sowie die Anwendbarkeit und den Nutzen des Frameworks beeinträchtigen. Die Entwicklung von kleinen Frameworks ist zwar finanziell weniger riskant und einfacher durchzuführen, dafür besteht die Gefahr, anfällig gegenüber Änderungen der Anwendungsdomäne zu sein [BMM+99]. So können Anwendungen, die ein solches Framework einsetzen, schnell über dessen Grenzen hin-

auswachsen [BMM+99] und nur aufwendig um neue Funktionalitäten erweitert werden. Aus diesem Grund muss die Domäne eines Frameworks sorgfältig ausgewählt und festgelegt werden. Das Entwicklerteam sollte in der Lage sein, gute Hypothesen über die zu erwartenden Variabilitäten des Frameworks zu erstellen, um das Risiko zu minimieren.

20.3.2 Nutzung von Frameworks

Unter der Nutzung eines Frameworks wird dessen Einsatz und Anpassung für eine konkrete Anwendung verstanden. Das Erlernen der Nutzung eines Frameworks kann unter Umständen lange dauern [BMM+99] und hängt direkt von der Größe und der Anzahl der zur Verfügung stehenden Variationspunkte ab (vgl. [Sih01]). Werden Frameworks externer Organisationen für die Entwicklung einer Anwendung eingesetzt, so ist zu prüfen, ob die jeweiligen Frameworks die Anforderungen der Anwendung erfüllen. Ein besonderes Augenmerk bei dieser Auswahl ist auf die Interoperabilität der Frameworks zu legen (siehe Abschnitt 20.3.3).

Eine wesentliche Voraussetzung für den erfolgreichen Einsatz von Frameworks ist eine gute Dokumentation. Eine Möglichkeit, die Software-Entwickler bei der Anwendung eines Frameworks und dessen Anpassung an eine konkrete Anwendungsdomäne zu unterstützen, sind so genannte Framework-Kochbücher (engl. *framework cookbooks*) [Pre97d]. Die »Rezepte« dieser »Kochbücher« beschreiben in Form von Prüflisten die Arbeitsschritte, die notwendig sind, um typische Aufgaben zur Anwendung und Anpassung des Frameworks durchzuführen. Eine andere Möglichkeit ist die Unterstützung durch einen Mentor, der idealerweise an der Entwicklung des Frameworks beteiligt war.

20.3.3 Komposition von Frameworks

Bei der Komposition von Frameworks werden mehrere Frameworks für die Entwicklung einer konkreten Anwendung eingesetzt. Die verwendeten Frameworks müssen dabei an die Architektur der Anwendung sowie der anderen Frameworks angepasst werden, damit diese zusammenarbeiten können. Die Komposition von Frameworks kann zu einer Fülle von Problemen führen, da diese naturgemäß bestrebt sind, die volle Kontrolle über die Anwendung innezuhaben (siehe dazu auch Abschnitt 20.2.1). Im Extremfall sind die Architekturstile, auf denen die Frameworks basieren, so unterschiedlich, dass es vielleicht unmöglich ist, sie miteinander zu komponieren. Diese Inkompatibilitäten von Architekturen werden als »architectural mismatch« bezeichnet. Ein erster Schritt zur Lösung dieses Problems scheint es zu sein, den Architekturstil, dem ein Framework unterliegt, explizit zu spezifizieren [BMM+99]. Werden zur Entwicklung einer Anwendung

mehrere Frameworks eingesetzt, so kann es auch passieren, dass sich die Frameworks in ihrer Funktionalität teilweise überlappen. Dann werden bestimmte Teile der Anwendung in mehreren Frameworks jeweils aus ihrem eigenen Blickwinkel modelliert und müssen zusammengeführt werden (siehe dazu [BMM+99], S. 73).

20.3.4 Evolution und Wartung von Frameworks

Die Entwicklung von Frameworks muss als eine langfristige Investition betrachtet werden. Als solche müssen Frameworks entsprechend weiterentwickelt und gewartet werden. Unter der Evolution beziehungsweise Wartung von Frameworks wird die Modifikation und Anpassung von Frameworks an veränderte Anforderungen der sie nutzenden Anwendungen und der betrachteten Domäne verstanden (nach [Som04]). Evolution und Wartung eines Frameworks beginnt bereits während der Entwicklung und nicht erst nach dem ersten Release. Schon früh in der Entwicklung eines Frameworks sind viele Iterationen bezüglich des Designs notwendig. Das liegt gerade am Ziel der Wiederverwendbarkeit von Frameworks. Die einzige Möglichkeit, dies zu prüfen, besteht darin das Framework in konkreten Beispielanwendungen wiederzuverwenden und die Schwächen am Entwurf zu identifizieren und zu verbessern (vgl. [BMM+99]).

20.4 Entwurf objektorientierter Frameworks

Ein objektorientiertes Framework zu entwerfen, erfordert sehr viel Erfahrung und Experimentieren [JF88]. Ein gut entworfenes objektorientiertes Framework ist typischerweise das Ergebnis von zahlreichen Entwurfsiterationen und sehr viel harter Arbeit [WJ90]. Das Streben nach Flexibilität um ihrer selbst willen liefert allerdings nicht automatisch ein erstklassiges Framework. Vielmehr gilt, dass Flexibilität wohl dosiert und in einer den Anforderungen des Anwendungsbereiches angemessenen Weise in ein Framework eingebaut werden sollte [Pre97c]. Unnötige Flexibilität erhöht dagegen sogar deutlich die Komplexität des Frameworks [FPR02]. Zur Entwicklung eines objektorientierten Frameworks sollte ein geeignetes Vorgehensmodell mit entsprechender Entwicklungsmethodik verwendet werden. Das Vorgehensmodell beschreibt, welche Aktivitäten zur Entwicklung des Frameworks wann durchzuführen sind. Die Entwicklungsmethodik unterstützt den Framework-Entwickler bei der Identifikation und Spezifikation der Flexibilitätsanforderungen an das Framework. In Abschnitt 20.4.1 bis Abschnitt 20.4.5 wird mit dem Hotspot-getriebenen Ansatz von Pree ein erprobtes Vorgehensmodell mit entsprechender Entwicklungsmethodik für objektorientierte Frameworks ausführlich vorgestellt. In Abschnitt 20.4.6

werden außerdem noch kurz andere Entwurfsmethoden für objektorientierte Frameworks genannt.

20.4.1 Der Hotspot-getriebene Entwurf von objektorientierten Frameworks

Der in Abbildung 20.2 gezeigte Hotspot-getriebene Ansatz stellt ein bekanntes und erprobtes Verfahren zur Entwicklung objektorientierter Frameworks dar und ist in zahlreichen Büchern und wissenschaftlichen Artikeln veröffentlicht worden [FPR02, Pre99, Pre97c, Pre97b, Pre96a, Pre95]. Da die Qualität eines objektorientierten Frameworks, wie in der Einleitung von Abschnitt 20.4 beschrieben, direkt von den Flexibilitätseigenschaften hinsichtlich der betrachteten Anwendungsdomäne abhängt, sieht der Hotspot-getriebene Entwurf eine explizite Aktivität zur Identifikation von geeigneten Variationspunkten vor [FPR02]. Eine sorgfältig durchgeführte Identifikation von Variationspunkten, den so genannten *Hotspots*, kann eine wertvolle Hilfe bei der Entwicklung eines guten Frameworks sein [Pre97c]. Im Folgenden werden die einzelnen Aktivitäten des Hotspot-getriebenen Entwicklungsprozesses beschrieben. Als Notation für die Klassendiagramme wird dabei die UML-F [FPR02, FPR00] verwendet, eine Erweiterung der Unified Modeling Language (UML) zur Beschreibung von Frameworks.

20.4.2 Definition eines speziellen Objektmodells

Der kritische erste Schritt bei der Entwicklung eines objektorientierten Frameworks ist die Identifikation der wichtigsten Abstraktionen, die so genannten *Schlüsselabstraktionen* (auch *Archetyp* [BB99, LBHB99]), des betrachteten Anwendungsbereiches [FPR02]. Die Objektmodellierung erfordert hauptsächlich anwendungsspezifisches Wissen. Die Software-Techniker unterstützen die Anwendungsexperten bei der Durchführung dieser Aufgabe. Die Unterscheidung zwischen diesen beiden Rollen ist allerdings nur eine hypothetische und soll ausdrücken, dass verschiedene Arten von Wissen zur Durchführung dieser Aktivität notwendig sind. So haben Software-Techniker in der Regel nur ein eingeschränktes Wissen über den zu modellierenden Anwendungsbereich.

Das initiale Objektmodell bildet die Basis des Hotspot-getriebenen Framework-Entwurfs. Zum Erstellen dieses Objektmodells können zum Beispiel CRC-Karten (*Class Responsibility Collaboration cards*) eingesetzt werden [FPR02]. CRC-Karten [BC89] eignen sich generell zum Auffinden von Klassen, Objekten und ihren Assoziationen [Pre97c]. Neben den CRC-Karten für das gesamte Objektmodell empfiehlt es sich, zudem einen separaten Satz an Karten zu erstellen, die nur die abstrakten Klassen des

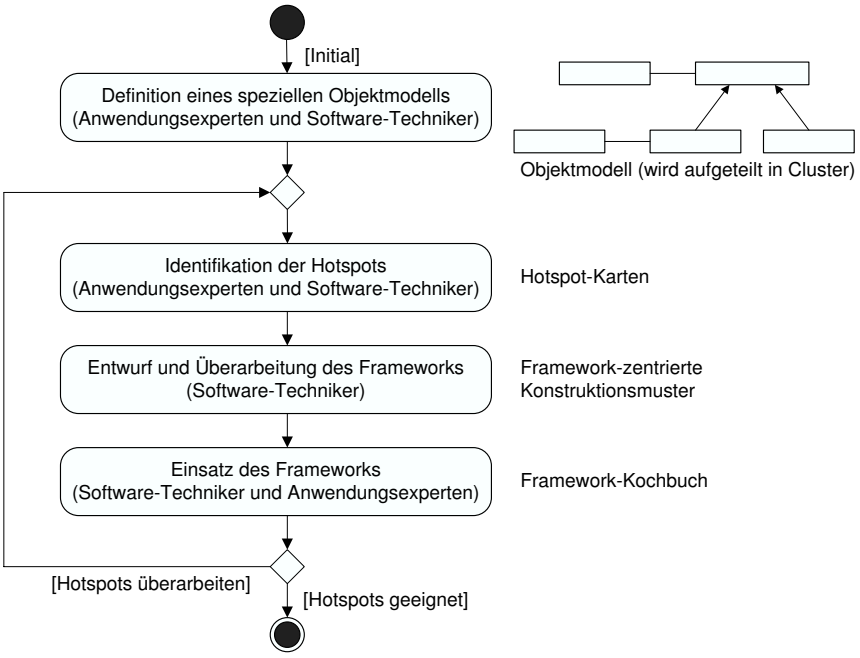

Abbildung 20.2: Der Hotspot-getriebene Entwurf objektorientierter Frameworks (nach [FPR02, Pre97c, Pre95])

zu entwickelnden objektorientierten Frameworks umfassen. Diese werden als aCiRC-Karten bezeichnet (*abstract Class/interface Responsibility Collaboration cards*). Das initiale Objektmodell wird auf Basis der identifizierten Schlüsselabstraktionen in einzelne Cluster gegliedert [Mey90]. Ein Cluster besteht aus einer Gruppe von Klassen und/oder Schnittstellen. Der Framework-Entwicklungsprozess wird den Clustern entsprechend in mehrere, sich zeitlich überlappende und unabhängige Software-Lebenszyklen aufgeteilt. Die einzelnen Cluster können sich also in verschiedenen Entwicklungsphasen befinden, die zudem unabhängig davon sind, in welcher Phase des Hotspot-getriebenen Entwurfs das zu entwickelnde Framework gerade ist. Diese Vorgehensweise spiegelt die Tatsache wider, dass der iterative Entwurf abstrakter Klassen und Schnittstellen erfordert, dass sich einige Cluster in der Implementierungsphase befinden. Wird während der Implementierung eines Clusters festgestellt, dass eine Abstraktion nicht passend ist, dann muss die Überarbeitung des Entwurfs dieser Abstraktion parallel mit der Implementierung dieses speziellen Clusters fortgeführt werden [FPR02].

Bevor nun mit dem Hauptzyklus des Framework-Entwicklungsprozes-

ses begonnen wird, ist es hilfreich, mehrere Objektmodelle von ähnlichen Anwendungen in der betrachteten Domäne zur Verfügung zu haben. Dadurch wird das Auffinden von Gemeinsamkeiten und deren spätere Abstraktion durch das Framework einfacher. Diese Objektmodelle stehen in der Regel jedoch nicht zur Verfügung, da das Erstellen einer anwendungsspezifischen Software-Lösung eine komplexe und iterative Aufgabe ist, in der Objektmodelle erstellt und verfeinert werden müssen, bis sie den Anforderungen genügen.

20.4.3 Identifikation der Hotspots und Erstellung der Hotspot-Karten

Das Hauptproblem bei der Identifikation der Hotspots ist, dass Anwendungsexperten nicht gewohnt sind, von einem konkreten Software-System zu abstrahieren und zu generalisieren. In der Regel wissen Anwendungsexperten nichts über objektorientierte Konzepte, wie Klassen, Objekte, Vererbung, Entwurfsmuster und Frameworks. Stattdessen können sie meistens gut in Software-Funktionalitäten denken und wissen typischerweise auch, durch welche grundlegenden Elemente sich ihr Anwendungsbereich auszeichnet. Zur Kommunikation zwischen den Software-Technikern und Anwendungsexperten muss daher eine gemeinsame Basis geschaffen werden, die auf den Funktionalitäten des Frameworks beruht und nicht angibt, in welchen Klassen und Schnittstellen welche Funktionalitäten platziert werden [FPR02, Pre97c]. Eine Möglichkeit, um die Framework-Funktionalitäten zu kommunizieren, sind Hotspot-Karten (engl. *variation point cards*).

Hotspot-Karten sind eine weitere Variante der CRC-Karten und stellen ein einfaches, aber effektives Hilfsmittel zur Dokumentation und Kommunikation von Flexibilitätsanforderungen zwischen den Anwendern und Framework-Entwicklern dar [Pre97c]. Sie dienen im ersten Schritt dazu, die Hotspots des zu entwickelnden Frameworks zu identifizieren, und helfen im zweiten Schritt beim konkreten Entwurf des Frameworks. Prinzipiell können sowohl Funktionen als auch Daten als Hotspots identifiziert werden. Die meisten Hotspots stellen aber wahrscheinlich Funktionen dar [Pre97c]. In Abhängigkeit vom Typ des identifizierten Hotspots wird entweder eine Funktionen- oder Daten-Hotspot-Karte erstellt. Ihr Aufbau ist in Abbildung 20.3 dargestellt. Funktionen-Hotspot-Karten dokumentieren die Funktionalitäten, die an dem objektorientierten Framework flexibel gehalten werden sollen. Auf Daten-Hotsport-Karten werden die Elemente der Anwendungsdomäne festgehalten, die im Zuge einer Framework-Implementierung verallgemeinert werden müssen. Für konkrete Beispiele von Funktionen- und Daten-Hotspot-Karten sei auf die Literatur verwiesen, wie zum Beispiel [Pre97c, Pre96a].

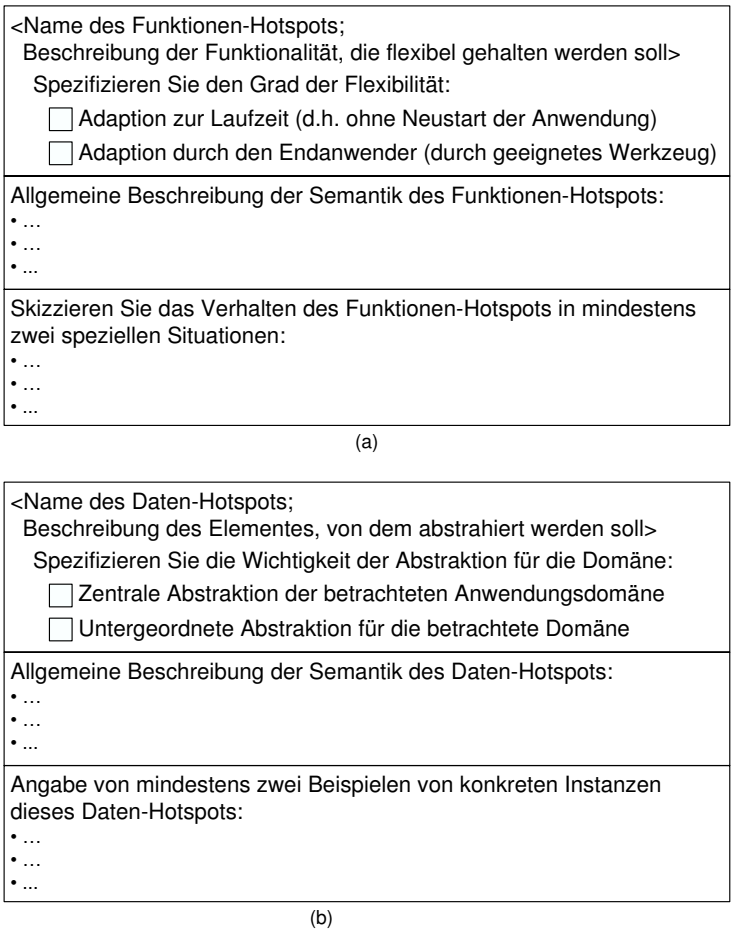

Abbildung 20.3: Aufbau einer (a) Funktionen-Hotspot-Karte und (b) Daten-Hotspot-Karte (nach [Pre97c, Pre96a])

Zur Realisierung einer Daten-Hotspot-Karte wird typischerweise eine abstrakte Klasse in das Objektmodell des zu entwickelnden Frameworks eingeführt. Daher kann die Dokumentation von Daten-Hotspots auf entsprechenden Karten auch weggelassen und das Objektmodell des Frameworks direkt manipuliert werden [Pre96a]. Im weiteren Verlauf dieses Kapitels konzentrieren wir uns daher auf die Umsetzung der Funktionen-Hotspots. Zudem vereinbaren wir, dass mit einem Hotspot bzw. einer Hotspot-Karte im Folgenden implizit ein Funktionen-Hotspot bzw. eine Funktionen-Hotspot-Karte gemeint ist.

Um mit Hilfe von Hotspot-Karten ein Objektmodell in ein objekt-

orientiertes Framework zu transformieren, sollte ihre Granularität *einer* Methode entsprechen. Diese Empfehlung weicht von der in der Literatur zum Hotspot-getriebenen Entwurf objektorientierter Frameworks [Pre97c, Pre95] angegebenen Empfehlung ab, die die Granularität einer Hotspot-Karte auf *in etwa einer* Methode festlegt. Die hier vorgenommene Festlegung der Granularität von Hotspot-Karten auf *eine* Methode begründet sich mit der Einführung so genannter Gruppen-Hotspot-Karten in Kapitel 21. Diese Gruppen-Hotspot-Karten werden dort verwendet, um die Komponenten des MM4U-Frameworks zu identifizieren und die Flexibilitätsanforderungen an diese Komponenten zu spezifizieren.

20.4.4 Entwurf und Überarbeitung des objektorientierten Frameworks mit Metamustern

Nachdem die initialen Hotspots identifiziert und auf Hotspot-Karten dokumentiert wurden, wird das Objektmodell hinsichtlich der in den Hotspot-Karten festgehaltenen Flexibilitätsanforderungen von den Software-Technikern modifiziert. Dazu definieren sie für jeden Hotspot, der auf den Hotspot-Karten beschrieben ist, einen »Haken« (engl. *hook*) im Entwurf des objektorientierten Frameworks, d. h., sie bestimmen, wo die zu den Hotspot-Karten gehörenden Methoden in der Klassenhierarchie des Frameworks platziert werden sollen [FHLS99]. Diese Methoden werden im Folgenden als *Einschubmethoden* (engl. *hook methods*) bezeichnet. Dabei unterstützen Framework-zentrierte Konstruktionsmuster, die so genannten *Metamuster* (engl. *meta pattern*), die Software-Techniker bei der konkreten Umsetzung der Hotspot-Karten [Pre95]. Metamuster stellen die essenziellen Konstruktionsprinzipien für objektorientierte Frameworks dar [Pre97c] und beschreiben, wie ein Framework zu entwerfen ist, unabhängig von einer bestimmten Domäne [Pre95]. Sie werden geformt aus *Schablonenmethoden* (engl. *template methods*) und den bereits genannten Einschubmethoden [Pre95]. Die Einschubmethoden stellen dabei abstrakte Platzhalter im objektorientierten Framework dar, die durch bereits implementierte Schablonenmethoden des Frameworks aufgerufen werden. Schablonenmethoden können abstraktes Verhalten, generischen Kontrollfluss oder auch das Interaktionsverhalten von Objekten des Frameworks beschreiben. Die prinzipielle Idee der Einschubmethoden ist, dass das Verhalten des objektorientierten Frameworks von den Anwendungsentwicklern durch Überschreiben dieser Einschubmethoden verändert wird [BD00]. Eine Spezialisierung, d. h. Anpassung, des objektorientierten Frameworks, findet »ausschließlich« über die Einschubmethoden statt [Pre97c, Pre95]. Dadurch wird die Einhaltung des Offen/Geschlossen-Prinzips sichergestellt (siehe Abschnitt 20.2.1). Schablonenmethoden beschreiben die abstrakte Anwendungslogik des objektorientierten Frameworks, die nicht mehr für Anpassungen mo-

difiziert werden kann. Schablonenmethoden werden daher auch *Frozen-Spot* genannt. Da neue Funktionalität nur über die Einschubmethoden in das objektorientierte Framework eingefügt werden kann, ist eine weite Voraussicht der Framework-Entwickler nötig, um die Einschubmethoden zu bestimmen [Pre97c]. Zur Modifizierung des Objektmodells reicht im Allgemeinen die Semantik eines Hotspots aus, um sowohl die Klasse als auch die Schablonenmethode zu finden, in die die Einschubmethode eingefügt werden soll [Pre97c].

Abbildung 20.4 stellt das Konzept der Schablonen- und Einschubmethoden beispielhaft dar. Die Schablonenmethode `t()` der Klasse `A` ruft, wie in Abbildung 20.4(a) gezeigt, die Einschubmethode `h()` auf. Die Klasse `B` in Abbildung 20.4(b) überschreibt die Einschubmethode `h()` durch Vererbung von Klasse `A`. Das dazugehörige UML-Klassendiagramm is in Abbildung 20.4(c) zu sehen. Die Menge an Metamustern, die für den Entwurf von objektorientierten Frameworks nötig sind, kann aus den Kombinationen von Schablonen- und Einschubklassen abgeleitet werden. Schablonen- und Einschubklassen können prinzipiell jede Komplexität annehmen. An der Anzahl der essenziellen Metamuster ändert das allerdings nichts, sondern nur an deren Größe.

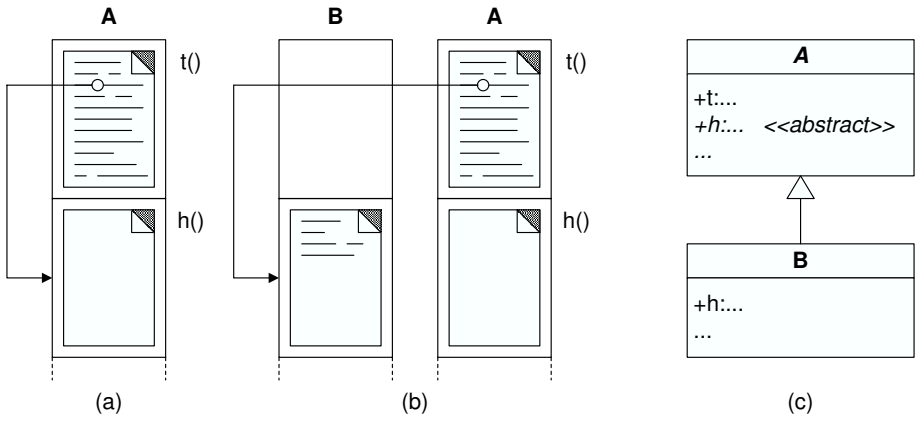

Abbildung 20.4: (a) Schablonen- und Einschubmethoden, (b) das Überschreiben von Einschubmethoden und (c) das dazugehörende UML-Klassendiagramm (nach [Pre97c])

Die Schablonen- und Einschubmethoden können, wie in Abbildung 20.4 dargestellt, entweder in einer gemeinsamen Klasse oder in zwei getrennten Klassen definiert werden. Eine Klasse, die Einschubmethoden enthält, wird als Einschubklasse (engl. *hook class*) bezeichnet und eine Klasse, die Schablonenmethoden enthält, wird Schablonenklasse (engl. *template class*) ge-

nannt. Die Einschubklasse parametrisiert dabei die Schablonenklasse. Da die Klasse A in Abbildung 20.4(c) sowohl die Schablonen- als auch die Einschubmethode enthält, ist sie sowohl Schablonen- als auch Einschubklasse und parametrisiert sich selbst. Um die Einschub- und Schablonenmethoden im Entwurf eines objektorientierten Frameworks explizit kenntlich zu machen, sieht die UML-F zwei Tags vor. Wie in Abbildung 20.5 dargestellt ist, werden Einschubmethoden im Klassendiagramm mit dem Tag «hook» und Schablonenmethoden mit «template» versehen. Diese Tags werden auch zum Kennzeichnen von Einschub- und Schablonenklassen verwendet.

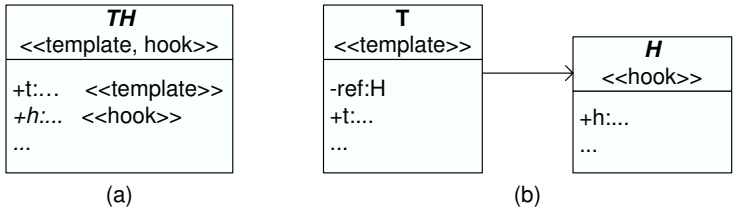

Abbildung 20.5: (a) Unifikation und (b) Separation von Schablonen- und Einschubmethoden (nach [Pre99])

Abhängig davon, ob eine Anpassung zur Laufzeit unterstützt werden soll oder nicht, werden die Einschub- und Schablonenmethoden in einer gemeinsamen Klasse oder in zwei getrennten Klassen definiert. Dazu wird entweder das *Unifikationsprinzip* (engl. *unification principle*) oder das *Separationsprinzip* (engl. *separation principle*) angewendet [Pre97c]. Beim Unifikationsprinzip werden, wie im Beispiel in Abbildung 20.5(a) dargestellt, die Schablonen- und Einschubmethode in der gemeinsamen Klasse TH definiert. Anpassungen sind hier durch Bildung entsprechender Unterklassen möglich und benötigen einen Neustart der Anwendung. Um eine Anpassung zur Laufzeit vornehmen zu können, werden die Einschub- und Schablonenmethoden nach dem Separationsprinzip in zwei Klassen getrennt, wie in Abbildung 20.5(b) dargestellt ist. Das Verhalten der Schablonenklasse T kann durch Komposition mit unterschiedlichen H-Objekten (also Instanzen von konkreten Klassen, die H implementieren) modifiziert werden. Die gerichtete Assoziation zwischen T und H drückt aus, dass ein T-Objekt auf ein H-Objekt verweist. Dies geschieht typischerweise über eine (Instanz-)Variable in der Schablonenklasse T, hier ref. Die H-Objekte können so zur Laufzeit ausgetauscht werden, ohne die Anwendung neu starten zu müssen. An dieser Stelle ist darauf hinzuweisen, dass die abstrakte Klasse H zum Beispiel in der Programmiersprache Java auch ein Interface sein kann. Prinzipiell ist die Realisierung durch eine abstrakte Klasse oder durch ein Interface als gleichwertig zu betrachten [PAS98].

Ist nicht sicher, ob eine Anpassung zur Laufzeit erforderlich ist oder nicht, so ist im Zweifelsfall die Realisierung über ein Interface vorzuziehen (vgl. [RJ97]). Ist nicht nur eine Anpassung der Funktionalität zur Laufzeit erforderlich, sondern soll die Anpassung auch durch den Endanwender vorgenommen werden können, so muss zusätzlich noch ein geeignetes Konfigurationswerkzeug entwickelt werden. Außerdem ist anzumerken, dass die Wahl zur Umsetzung der Hotspots mit dem Unifikations- oder Separationsprinzip nur für objektorientierte Frameworks existiert. Bei der komponentenbasierten Software-Entwicklung und damit auch bei den Komponenten-Frameworks (siehe Abschnitt 20.2.2) ist nur die Umsetzung nach dem Separationsprinzip möglich. Zudem müssen die Einschubmethoden in einem Interface definiert werden.

Neben den bisher vorgestellten Metamustern können die Schablonen- und Einschubklassen auch rekursiv miteinander kombiniert werden (rekursive Komposition). So kann, wie in Abbildung 20.6(a) dargestellt, die Schablonenklasse `T` ihrerseits eine Unterklasse der Einschubklasse `H` sein. Im Extremfall fallen Schablonen- und Einschubklasse zusammen und sind, wie in Abbildung 20.6(b) dargestellt, in einer Klasse vereinigt. Als Unterklasse von `H` kann die Schablonenklasse `T` nun eine weitere Instanz von sich selbst als Implementierung für die Einschubklasse `H` verwalten. Umgekehrt können aber Einschubklassen nicht als Unterklassen von Schablonenklassen definiert werden, da Einschubklassen nur aus den abstrakten Einschubmethoden bestehen. Dass die rekursive Verschachtelung nicht nur von theoretischem Interesse, sondern auch für die Praxis relevant ist, zeigen Beispiele in [Pre95, Pre96a]. Die Metamuster für objektorientierte Frameworks existieren darüber hinaus auch in einer Variante, bei der die Schablonenklasse nicht nur *eine* Instanz der Einschubklasse, sondern über eine Liste von Objektreferenzen eine *beliebige Menge* von Instanzen der Einschubklasse verwalten kann (siehe dazu [Pre95, Pre94]).

Der Entwurf der Klassenschnittstellen durchläuft im Allgemeinen sehr viele Iterationsschritte. Dies wird vornehmlich durch die verschiedenen Verwendungen des objektorientierten Frameworks in unterschiedlichen konkreten Anwendungen verursacht. So kann während einer dieser Iterationen deutlich werden, dass die Schablonenmethoden zu starr angelegt sind und das Framework dadurch unflexibel wird. Infolgedessen müssen weitere Einschubmethoden definiert und eingesetzt werden. Andererseits können sich aus der wiederholten Anwendung des objektorientierten Frameworks auch neue sinnvolle Schablonenmethoden ergeben. Implementieren zum Beispiel mehrere Anwendungsentwickler beim Anpassen des Frameworks einen ähnlichen Kontrollfluss, dann sollte dieser als Schablonenmethode in das Framework integriert werden.

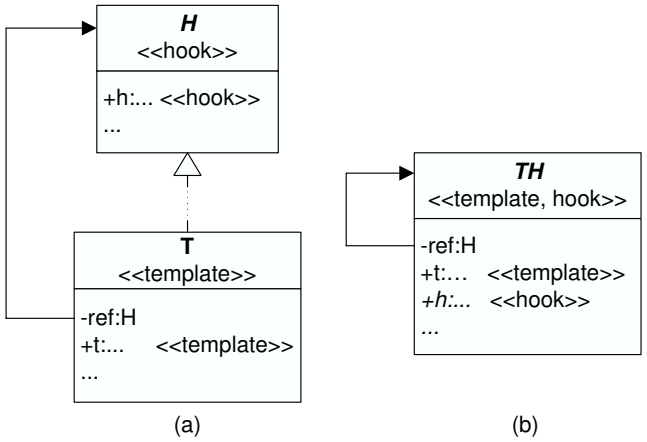

Abbildung 20.6: Rekursive Kombination von Schablonen- und Einschubklassen (aus [Pre99])

20.4.5 Einsatz des objektorientierten Frameworks

Ein objektorientiertes Framework muss mehrere Male wiederverwendet, d. h. in konkreten Anwendungen eingesetzt werden, um die Schwächen am Entwurf festzustellen (vgl. [BMM+99]). Die Schwächen am Entwurf eines Frameworks sind die Stellen, wo Einschubmethoden fehlen oder ungeeignet sind bzw. wo Schablonenmethoden zu wenig oder zu viel (abstraktes) Verhalten kapseln. Der in Abbildung 20.2 angegebene Zyklus stellt diese evolutionäre Vorgehensweise beim Entwurf objektorientierter Frameworks mit Hilfe von Hotspot-Karten dar. Die explizite Identifikation von Hotspots trägt hier zu einer signifikanten Reduzierung der Iterationen bei.

20.4.6 Andere Entwurfsmethoden für objektorientierte Frameworks

In den vorangegangenen Abschnitten 20.4.1 bis 20.4.5 wurde ausführlich der Hotspot-getriebene Entwicklungsprozess von objektorientierten Frameworks vorgestellt. Daneben gibt es auch noch einige andere Ansätze für die Entwicklung objektorientierter Frameworks: In [Sch99] wird der Entwurf von Frameworks durch systematische Generalisierung beschrieben, die Verwendung von Templates zur Spezifikation von Frameworks wird in [DW99] vorgestellt, in [Rie00] wird ein rollenbasierter Ansatz zur Entwicklung objektorientierter Frameworks vorgeschlagen und in [LN95] wird ein Vorgehensmodell für objektorientierte Frameworks vorgestellt,

das eine 71 Punkte umfassende Checkliste zur Framework-Entwicklung beinhaltet.

20.5 Entwurf von Komponenten-Frameworks

Zur komponentenbasierten Software-Entwicklung stehen entsprechende Vorgehensmodelle zur Verfügung, wie diejenigen zum Beispiel in [Som04, And04, Bos00, HHKS97]. Darüber hinaus existieren auch erprobte Verfahren zur Identifikation von Komponenten, wie zum Beispiel der in Kapitel 5 beschriebene Ansatz zur Bildung von Software-Kategorien [Sie04]. Im Gegensatz zu objektorientierten Frameworks, für die es mit dem Hotspot-getriebenen Entwicklungsprozess (siehe Abschnitt 20.4.1) bereits ein erprobtes Vorgehensmodell gibt, fehlt derzeit noch eine entsprechende Unterstützung zur Entwicklung von Komponenten-Frameworks. Folglich hängt die Qualität eines Komponenten-Frameworks letztendlich von der Erfahrung und dem Geschick der Framework-Entwickler ab [Sih01].

Eine Möglichkeit, die Entwicklung von komponentenbasierten Frameworks zu verbessern, ist, den Entwurf und die Entwicklung von Software-Komponenten gleich so auszurichten, dass flexible und anpassbare Komponenten entstehen. Die Komponenten werden also gleich so gebaut, dass daraus Komponenten-Frameworks gebildet werden können. Dadurch besteht die Hoffnung, durch Implementierung flexibler Komponenten zu einem geeigneten Komponenten-Framework zu gelangen. Die Anforderungen, die sich aus der Anwendungsdomäne eines potenziellen Komponenten-Frameworks ergeben, hat man dabei jedoch nicht vor Augen und damit auch nicht die Flexibilitätsanforderungen an die einzelnen Komponenten des potenziellen Frameworks. Dabei besteht die Gefahr, Flexibilität um ihrer selbst willen in das Framework zu integrieren, die dem Anwendungsbereich nicht angemessen ist (vgl. [Pre97c]) und damit die Komplexität des Frameworks unnötig erhöht (siehe [FPR02]). Stattdessen sollte Flexibilität, wie bei den objektorientierten Frameworks, mit Hilfe eines geeigneten Vorgehensmodells identifiziert und wohl dosiert in ein Komponenten-Framework implementiert werden (siehe auch Abschnitt 20.4).

Zur systematischen Unterstützung des Entwicklungsprozesses von Komponenten-Frameworks kann der in Abschnitt 21.4 vorgeschlagene Ansatz verwendet werden. Dieser stellt eine Modifikation des Hotspot-getriebenen Entwurfs objektorientierter Frameworks dar und integriert explizite Aktivitäten zur Identifikation der benötigten Framework-Komponenten und zur Spezifikation der Flexibilitätsanforderungen an diese Komponenten. Erprobt wurde dieser Entwicklungsprozess für Komponenten-Frameworks am Framework für personalisierte Multimedia-Anwendungen MM4U (Ab-

kürzung für »Multimedia for you«). Der Entwurf und die Implementierung dieses Frameworks wird im Kapitel 21 beschrieben.

20.6 Fazit

In diesem Kapitel wurden Frameworks als ein Konzept zur Wiederverwendung von Software-Architekturen eingeführt. Die Unterscheidung zwischen objektorientierten und komponentenbasierten Frameworks ist dabei von wesentlicher Bedeutung, da sich diese beiden Arten von Frameworks hinsichtlich der eingesetzten Technologie zur Erweiterung und Anpassung an die Anforderungen einer konkreten Anwendung unterscheiden. Da die Entwicklung eines Frameworks sehr aufwendig ist, stellt sie immer eine Zukunftsinvestition dar [Pre97c], die sich erst dann lohnt, wenn das Framework in mehreren Anwendungen genutzt und wiederverwendet wird [Pre97a]. Der höhere Entwicklungsaufwand eines Frameworks zahlt sich also erst auf lange Sicht aus. Im Gegenzug wird jedoch signifikant der Entwicklungsaufwand für die einzelnen konkreten Anwendungen reduziert.

Für die erfolgreiche Entwicklung von Frameworks sind neben dem Software-technischen Wissen auch organisatorische Aspekte wichtig (vgl. [RE99]). Die Entwicklung eines Frameworks in einem Projekt bedeutet immer auch eine Änderung der Organisationsstruktur des Entwicklerteams. So sollte das Entwicklerteam in Framework- und Anwendungsentwickler aufgeteilt werden, also in die, die das Framework entwickeln, und die, die es einsetzen und damit gegen die Framework-Anforderungen testen. Frameworks gehören heute leider immer noch nicht zur allgemeinen Projektkultur [Pre99], so dass wirklich gut entwickelte Frameworks nur schwer zu finden sind. Es ist jedoch festzustellen, dass als Konsequenz aus der zunehmenden Anzahl an anwendungsspezifischen Frameworks, sich Software-Architekten zukünftig genauso viel mit den Zwängen, die durch die Wahl und den Einsatz eines Frameworks entstehen, wie mit der Neuentwicklung von Anwendungen beschäftigen werden [BCK03]. Insgesamt ist daher eine weitere Zunahme der Bedeutung von Frameworks für die Software-Entwicklung zu erwarten.

Literaturverzeichnis

[Ack96] ACKERMANN, P.: *Developing Object-Oriented Multimedia Software – Based on MET++ Application Framework*. dpunkt.verlag, 1996

[And04] ANDRESEN, A.: *Komponentenbasierte Softwareentwicklung mit MDA, UML 2 und XML*, Bd. 2., neu bearbeitete Auflage. Hanser Verlag, 2004

[Bal00] BALZERT, H.: *Lehrbuch der Software-Technik, Bd. 1: Software-Entwicklung*. Spektrum Akademischer Verlag, 2. Aufl., 2000

[BB99] BENTSSON, P.-O.; BOSCH, J.: Haemo Dialysis Software Architecture Design Experiences. In: *Proceedings of the 21st International Conference on Software Engineering*, May 1999, S. 516–526

[BC89] BECK, K.; CUNNINGHAM, W.: A laboratory for teaching object oriented thinking. In: *OOPSLA '89: Conference proceedings on Object-oriented programming systems, languages and applications*, New York, NY, USA: ACM Press, 1989, S. 1–6, doi:http://doi.acm.org/10.1145/74877.74879

[BCK03] BASS, L.; CLEMENTS, P.; KAZMAN, R.: *Software Architecture in Practice*. SEI Series in Software Engineering, Addison-Wesley, 2. Aufl., 2003

[BD00] BRUEGGE, B.; DUTOIT, A.: *Object-Oriented Software Engineering – Conquering Complex and Changing Systems*. Prentice Hall, 2000

[BMM+99] BOSCH, J.; MOLIN, P.; MATTSSON, M.; BENGTSSON, P.; FAYAD, M. E.: Framework Problems and Experiences. In: FAYAD et al. [FSJ99a], S. 55–82

[Bos00] BOSCH, J.: *Design and Use of Software Architectures – Adopting and evolving a product-line approach*. Addison-Wesley, 2000

[BR90] BASILI, V. R.; ROMBACHF, H. D.: *Towards a comprehensive framework for reuse: model-based reuse characterization schemes*. Technischer Bericht, College Park, MD, USA, 1990

[BR91] BASILI, V. R.; ROMBACH, H. D.: Support for comprehensive reuse. In: *Software Engineering Journal* 6 (1991), Nr. 5, S. 303–316

[CBB+03] CLEMENTS, P.; BACHMANN, F.; BASS, L.; GARLAN, D.; IVERS, J.; LITTLE, R.; NORD, R.; STAFFORD, J.: *Documenting Software Architectures – Views and Beyond*. SEI Series in Software Engineering, Addison-Wesley, 2003

[CC02] CAREY, J.; CARLSON, B.: *Framework Process Patterns – Lessons Learned Developing Application Frameworks*. Addison-Wesley, 2002

[DW99] D'SOUZA, D. F.; WILLS, A. C.: *Objects, Components, and Frameworks with UML – The Catalysis Approach*. Addison-Wesley, 1999

[FHLS99] FROEHLICH, G.; HOOVER, H. J.; LIU, L.; SORENSON, P.: Reusing Hooks. In: FAYAD et al. [FSJ99a], S. 219–236

[FPR00] FONTOURA, M.; PREE, W.; RUMPE, B.: UML-F: A Modeling Language for Object-Oriented Frameworks. In: *ECOOP*, 2000, S. 63–82

[FPR02] FONTOURA, M.; PREE, W.; RUMPE, B.: *The UML Profile for Framework Architectures*. Object Technology Series, Addison-Wesley, 2002

[FSJ99a] FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. (Hrsg.): *Building Application Frameworks – Object-Oriented Foundations of Framework Design*. Wiley computer publishing, John Wiley & Sons, 1999

[FSJ99b] FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. (Hrsg.): *Implementing Application Frameworks: Object-Oriented Frameworks at Work*. Wiley computer publishing, John Wiley & Sons, 1999

[Gam92] GAMMA, E.: *Objektorientierte Software-Entwicklung am Beispiel von ET++ – Design-Muster, Klassenbibliothek, Werkzeuge*. Springer-Verlag, 1992

[GHJV04] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J.: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software*. Programmer's Choice, Addison-Wesley, Juli 2004

[Gov99] GOVONI, D.: *Java Application Frameworks*. John Wiley & Sons, 1999

[Has02] HASSELBRING, W.: Component-Based Software Engineering. In: CHANG, S. (Hrsg.), *Handbook of Software Engineering and Knowledge Engineering*, New Jersey: World Scientific Publishing, 2002, S. 289–305

[HHKS97] HEUER-HASENPLATT, H.; HOLLUNDER, B.; KITTLAUS, H.-B.; SCHUMACHER, N.: Bausteinorientierte Anwendungsentwicklung: Voraussetzungen, Anforderungen und Auswirkungen. In: *Objektspektrum*, SIGS-DATACOM GmbH, Nr. 3, 1997, S. 40–51

[HK03] HITZ, M.; KAPPEL, G.: *UML Work: Von der Analyse zur Realisierung*. dpunkt.verlag, 2. Aufl., 2003

[JF88] JOHNSON, R.; FOOTE, B.: Designing Reusable Classes. In: *Journal of Object-Oriented Programming* 1 (1988), Nr. 2, S. 22–35

[JN99] JACOBSON, E. E.; NOWACK, P.: Frameworks and Patterns – Architectural Abstractions. In: FAYAD et al. [FSJ99a], S. 29–54

[LBHB99] LUNDBERG, L.; BOSCH, J.; HÄGGANDER, D.; BENGTSSON, P.-O.: Quality Attributes in Software Architecture Design. In: *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, October 1999, S. 353–362

[LN95] LANDIN, N.; NIKLASSON, A.: *Development of Object-Oriented Frameworks*. Diplomarbeit, Department of Communication Systems, Lund Institute of Technology, Lund University, Lund, Sweden, 1995

[Mey90] MEYER, B.: Lessons from the design of the Eiffel libraries. In: *Communications of the ACM* 33 (1990), Nr. 9, S. 68–88, doi:10.1145/83880.84464

[Mey97] MEYER, B.: *Object-Oriented Software Construction*. Prentice Hall, 2. Aufl., 1997

[Mic05a] MICROSOFT CORPORATION, USA: About the Microsoft Foundation Classes. 2005, URL `http://msdn.microsoft.com/library/en-us/vcmfc98/html/_mfc_about_the_microsoft_foundation_classes.asp`

[Mic05b] MICROSOFT CORPORATION, USA: .NET Development. 2005, URL `http://msdn.microsoft.com/library/en-us/dnanchor/html/netdevanchor.asp`

[PAS98] PREE, W.; ALTHAMMER, E.; SIKORA, H.: Framelets als handliche Architekturbausteine. In: *Softwaretechnik 98*, Paderborn, September 1998

[PK99] PREE, W.; KOSKIMIES, K.: Framelets – Small is Beautiful. In: FAYAD et al. [FSJ99a], S. 411–413

[PK00] PREE, W.; KOSKIMIES, K.: Framelets – small and loosely coupled frameworks. In: *ACM Computing Surveys* 32 (2000), Nr. 1es, S. 6, doi:10.1145/351936.351942

[Pre94] PREE, W.: Meta Patterns — A Means for Capturing the Essentials of Reusable Object-Oriented Design. In: *Lecture Notes in Computer Science* 821 (1994)

[Pre95] PREE, W.: *Design Patterns for Object-Oriented Software Development*. ACM Press Books, Addison-Wesley, 1995

[Pre96a] PREE, W.: *Framework Patterns*. SIGS Books and Multimedia, 1996

[Pre96b] PREE, W.: Frameworks – Past, present, future. In: *Object magazine – improving software quality through object development & reuse* 6 (1996), Nr. 3

[Pre97a] PREE, W.: Component-Based Software Development – A New Paradigm in Software Engineering? In: *Software – Concepts and Tools* 18 (1997), Nr. 4, S. 169–174

[Pre97b] PREE, W.: Essential Framework Design Patterns. In: *Object magazine – improving software quality through object development & reuse* 7 (1997), Nr. 1

[Pre97c] PREE, W.: *Komponentenbasierte Softwareentwicklung mit Frameworks*. dpunkt.verlag, 1997

[Pre97d] PREE, W.: Object-Oriented Design Patterns and Hot Spot Cards. In: *IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS97)*, Como, Italy, September 1997

[Pre99] PREE, W.: Hot-Spot-Driven Framework Development. In: FAYAD et al. [FSJ99a], S. 379–393

[RE99] ROSEL, A.; ERNI, K.: Experiences with the Semantic Graphics Framework. In: FAYAD et al. [FSJ99b], Kap. 27, S. 629–657

[Rie00] RIEHLE, D.: *Framework Design: A Role Modeling Approach*. Dissertation, Swiss Federal Institute of Technology Zurich, 2000

[RJ97] ROBERTS, D.; JOHNSON, R.: Evolving Frameworks – A Pattern Language for Developing Object-Oriented Frameworks. In: *Pattern Languages of Program Design 3*, Illinois, USA: Addison-Wesley, 1997

[Rog97] ROGERS, G. F.: *Framework-Based Software Development in C++*. Prentice Hall Series on Programming Tools and Methodologies, Prentice Hall, 1997

[Sch99] SCHMID, H. A.: Framework Design by Systematic Generalization. In: FAYAD et al. [FSJ99a], Kap. 15, S. 353–378

[SGM02] SZYPERSKI, C.; GRUNTZ, D.; MURER, S.: *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley Component Software Series, Addison-Wesley, 2. Aufl., 2002

[Sie04] SIEDERSLEBEN, J.: *Moderne Softwarearchitektur – Umsichtig planen, robust bauen mit Quasar*. dpunkt.verlag, 2004

[Sih01] SIHLING, M.: *Methodische Entwicklung und rollenbasierte Integration von Komponentenframeworks*. Dissertation, Technische Universität München – Institut für Informatik, 2001

[Som04] SOMMERVILLE, I.: *Software Engineering*. Pearson and Addison Wesley, 7. Aufl., 2004

[Sun05] SUN MICROSYSTEMS: *Java Foundation Classes (JFC/Swing)*. Technischer Bericht, 1994–2005, URL `http://java.sun.com/products/jfc/`

[Tro05] TROLLTECH: *Trolltech - Qt Product Overview*. Technischer Bericht, 2005, URL `http://www.trolltech.com/products/qt/index.html`

[WJ90] WIRFS-BROCK, R. J.; JOHNSON, R. E.: Surveying current research in object-oriented design. In: *Communications of the ACM* 33 (1990), Nr. 9, S. 104–124

[Zül05] ZÜLLIGHOVEN, H. (Hrsg.): *Object-Oriented Construction Handbook – Developing Application-Oriented Software with the Tools and Materials Approach*. dpunkt.verlag, 2005