Ein objekt-orientiertes Framework für NFS-Dienste

Evelyn Mülthaler*
T-NOVA GmbH
Deutsche Telekom AG
Evelyn.Muelthaler@telekom.de

Gabor Herr
Technische Universität Darmstadt
Fachbereich Informatik
herr@iti.informatik.tu-darmstadt.de

Abstract. Das *Network File System (NFS)* ist ein verbreitetes Protokoll für den Dateizugriff auf nicht-lokale Dateisysteme (*remote file systems*). Seine Client/Server-Architektur mit dem einfach gehaltenen Server läßt die Realisierung dateiorientierter Dienste mit veränderter Semantik zu. Die Implementierung eines neuen NFS-basierten Dienstes ist aber noch zu aufwendig und die bereits existierenden NFS-basierten Dienste haben den Nachteil, schlecht erweiterbar und nicht portierbar zu sein. Dieser Artikel beschreibt, wie wir ausgehend vom NFS-Anwendungsgebiet ein *objekt-orientiertes Framework* für NFS-Dienste entworfen und in Java implementiert haben: einen generischen NFS-Server, der die notwendige Grundfunktionalität bereitstellt und die Realisierung neuer Dienste erleichtert. *Design Patterns* wurden eingesetzt, um die Variabilität der Anwendung zu modellieren und die Struktur des Frameworks einheitlich zu dokumentieren. Die Instantiierung des Frameworks wird anhand eines einfachen Beispiels gezeigt.

1 Motivation

Das *Network File System (NFS)* ist ein sehr verbreiteter Zugriffsdienst auf Ferndateisysteme (*remote file systems*). Seine Attraktivität ist in seiner einfachen Architektur wie auch in der Tatsache begründet, daß der NFS-Dienst mit simulierten Dateisystemen arbeiten kann. Die Semantik des Servers ist im NFS-Protokoll [Sun89, CPS95] einfach gehalten und wohldefiniert. Dazu kommt die wesentliche Eigenschaft von modernen Dateisystemen, daß es die Dateisystemobjekte uninterpretiert sieht. Deswegen bietet es sich an, durch die Neuimplementierung des Servers, Dateisysteme zu *simulieren* (virtuelle Dateisysteme) und damit Dienste mit unterschiedlicher Semantik über die NFS-Schnittstelle abzuwickeln (hier NFS-basierte Dienste genannt). Auf dieser Weise wurden bekannte

*Die in diesem Papier geäußerten Ansichten sind die des Autors und spiegeln nicht notwendigerweise die Ansichten der Deutschen Telekom wider.

NFS-basierte Dienste realisiert, z.B. das Crypto File System (CFS) [Bla93], der Automount Daemon (AMD) [Lei93], der Autocacher [Min93].

Um einen neuen Dienst zu realisieren, war es bisher notwendig den NFS-Server mit der veränderten Semantik von Grund auf neu zu implementieren. Dafür mußte man sich auch mit den dem NFS unterliegenden Protokollen gründlich auseinandersetzen. Da dies nicht trivial ist und auch viel Zeit in Anspruch nimmt, kommt die Implementierung eines NFS-Servers für viele nicht in Frage. Es wäre wünschenswert, einen neuen NFS-Server leichter implementieren zu können und einen bestehenden NFS-Server wiederverwenden zu können. Das Ziel dieser Arbeit ist, ausgehend von einem konkreten NFS-Server und dem Umfeld des NFS-Dienstes, einen wiederverwendbaren NFS-Server als Framework zu entwerfen und zu implementieren [Mue98].

Dieser Beitrag ist wie folgt aufgebaut: Abschnitt 2 gibt einen kurzen Einblick in die Terminologie von Frameworks und Design Patterns, Abschnitt 3 beschreibt zusammenfassend die Vorgehensweise in unserem Entwurf. Im Abschnitt 4 wird die Anwendung des Frameworks anhand eines einfachen Implementierungsbeispiel gezeigt und Abschnitt 5 beendet den Beitrag mit einer Zusammenfassung und einen Ausblick.

2 Frameworks und Design Patterns

Wiederverwendung von Software gewinnt in den letzten Jahren immer mehr an Bedeutung. Die verschiedenen Wiederverwendungstechniken ermöglichen Einsparung von Zeit und Kosten bei der Entwicklung von Anwendungen. Dadurch ist es erst möglich, komplexe Anwendungen so zu bauen, daß sie überschaubar und verständlich sind, da es eine gemeinsame Sprache für ihre Beschreibung gibt. Diese Einheitlichkeit wird stark von den Wiederverwendungstechniken unterstützt. Zu den Konzepten gehören auch Frameworks und Entwurfsmuster (*design patterns*).

Ein *Framework* ist eine generische Anwendung, aus der man unterschiedliche Anwendungen aus einem bestimmten Anwendungbereich schnell und kostengünstig erstellen kann (z.B, kunden- oder produktspezifische). Es beschreibt die *Architektur* eines objekt-orientierten Systems: die Art der Objekte, die das System beinhaltet, und die Art der Kommunikation zwischen den Objekten. Es beschreibt also die Struktur einer Anwendungskategorie. [Sch97, Joh97b]

Ein Framework setzt sich zusammen aus

- den *Klassen* - konkrete und abstrakte Klassen,
- der *Modellierung der Kommunikation* zwischen den Objekten
- und der *Dokumentation*.

Die abstrakten Klassen spezifizieren die Schnittstellen ihrer Unterklassen und stellen meistens auch Teile der Implementierung zur Verfügung. Veränderbare Aspekte der Anwendung werden in den abstrakten Klassen durch *abstrakte Methoden* (die Operation ist nicht implementiert) oder durch *Hook-Methoden*

(eine Default-Implementierung) realisiert. Eine konkrete Unterklasse *muß* die abstrakten Methoden seiner abstrakten Oberklasse implementieren und *darf* die Hook-Methode auswählen, die sie durch eine eigene Implementierung ersetzen will [Joh97a].

Mit einem Framework wird das Hauptprogramm einer Anwendung wiederverwendet. Das Framework bestimmt die Ablaufstruktur des Programms, und besitzt damit die Kontrolle über den Programmfluß. Der Anwendungsentwickler, der das Framework einsetzt, entscheidet, was in dem Framework eingebettet wird, und erweitert es vielleicht mit neuen Komponenten. Die Auswirkung ist, daß sein Code von dem Hauptprogramm, also vom Framework aus, aufgerufen wird.

Mit einem Framework wird sowohl die Domain-Analyse, der abstrakte Anwendungsentwurf, als auch die Implementierung wiederverwendet. Die *Analyse* besteht aus der Herausarbeitung des Kerns des Systems, der Heraushebung der wichtigen Objekte und dadurch die Bereitstellung einer Sprache zur Beschreibung des Problems. Zur Analyse gehört auch zu sehen, bezüglich welcher Aspekte sich mehrere Anwendungen eines Anwendungsbereichs ähneln, und bezüglich welcher Aspekte sich die Anwendungen unterscheiden. Der *Entwurfsteil* ist die Einteilung des Systems in Einheiten und die Festlegung der Vernetzung zwischen den Objekten, die diese Einheiten beschreiben. Dazu kommt die Entscheidung, welche Aspekte der Anwendungen festgelegt werden sollen und dadurch als nichtveränderbar definiert werden und welche Aspekte nicht festgelegt werden, also als veränderbar definiert werden. Der festgelegte Teil in einem Framework wird *Frozen Spot* bezeichnet und jeder als variabel gestalteter Aspekt wird *Hot Spot* genannt. Die Klassen, die das Framework enthält, bilden den *Implementierungsteil*. Die Klassen, die das Framework implementieren, definieren die Schnittstellen der Objekte, deren Interaktion die Anwendung realisieren wird, die mit Hilfe des Frameworks entwickelt werden soll.

Ein Nachteil von Frameworks ist, daß der Anwendungsentwurf nicht explizit und abstrakt wie in einem Entwurfsmuster beschrieben wird, sondern implizit in der Implementierung enthalten ist. Das führt zu einem großen Lernaufwand, der nur durch eine sehr gute Dokumentation verringert werden kann. Die Dokumentation soll beschreiben, was das Framework kann, welche Möglichkeiten es bietet, wie es zu erweitern und einzusetzen ist.

Ein Framework kann Entwurfsmuster zur Beschreibung seiner Architektur verwenden, was durch den Abstraktionsgrad und der Einheitlichkeit der Entwurfsmuster dem besseren Verständnis des Frameworks dient. Design Patterns (Entwurfsmuster) haben seit der Veröffentlichung des Werkes von Gamma, Helm, Johnson, Vlissides [GHJV95], das einen systematischen Katalog von anwendungsübergreifenden Entwurfsmustern darstellt, sehr viel an Bedeutung gewonnen und haben in Wissenschaft und Praxis eine weite Verbreitung gefunden.

Design Patterns sind abstrakte Muster, die ein Entwurfsproblem und seine Lösung beschreiben. Ein Entwurfsmuster gibt die Klassen, Objekte und ihre zur Problemlösung erforderliche Kooperation der betreffenden Dienste und Methoden an, inklusive Randbedingungen, Folgerungen und Implementierungsüberle-

gungen. Ein Entwurfsmuster benennt, abstrahiert und identifiziert die Schlüsselaspekte einer bekannten Entwurfsstruktur, was die Wiederverwendung des objekt-orientierten Entwurfes ermöglicht. Es identifiziert die teilnehmenden Klassen und Instanzen, ihre Rolle und Zusammenarbeit und die Verteilung der Verantwortungen. Jedes Entwurfsmuster konzentriert sich auf ein bestimmtes häufig auftretendes objekt-orientiertes Problem [GHJV95].

3 Framework-Entwurf

Ausgangspunkt für den Framework-Entwurf waren der konkrete NFS-Server und das Umfeld des NFS-Dienstes. Daraus haben wir durch mehrere Abstraktionsstufen einen generischen NFS-Server entworfen. Bei diesem Ansatz zum Entwurf leitet man die Architektur des Frameworks aus der Klassenstruktur der nichtflexiblen, spezialisierten Anwendung (das NFS) durch eine Folge von generalisierenden Transformationsschritten her [Sch97].

Mit den Erkenntnissen aus der Domain-Analyse der existierenden NFS-basierten Dienste wird die konkrete Anwendung, der NFS-Dienst, in Komponenten zerlegt, die jeweils eine Einheit bilden und ein wohldefiniertes Verhalten aufweisen. Diese Komponenten sind allen Applikationen des Anwendungsbereichs gemeinsam. Das Ergebnis dieser objekt-orientierten Zerlegung ist die *Anwendungsarchitektur*.

Es folgt die Feststellung, welche Aspekte des Verhaltens der Komponenten der Anwendungsarchitektur bei allen Applikationen des Anwendungsbereichs gleich sind, und welche sich anwendungsspezifisch verändern. Diese Erkenntnisse unterstützen die Durchführung der Verallgemeinerungsschritte augehend von der Anwendungsarchitektur. Bei jeder Komponente wird geprüft, ob es Anteile des Verhaltens gibt, die man verallgemeinern kann. Zu verallgemeinern heißt in diesem Kontext, zu entscheiden, welcher Teil des so entstehenden Frameworks austauschbar wird (die Hot Spots) und welcher Teil den festen Bestandteil des Frameworks bilden wird (die Frozen Spots).

Bei jedem Hot Spot wird die Art der Flexibilität analysiert und versucht ein Entwurfsmuster zu finden, welches die gewünschte Art der Flexibilität bietet. Das Ergebnis ist der *Feinentwurf* des Frameworks sein.

3.1 Anwendungsarchitektur

Die Komplexität der NFS-Anwendung wurde zunächst durch eine *objekt-orientierte Zerlegung* [Boo91] bewältigt. Wir betrachteten die NFS-Anwendung als eine Menge autonomer Bestandteile, die zusammenarbeiten, um eine bestimmte, übergeordnete Aufgabe zu erfüllen. Auf diese Weise verkörpert jede Komponente in unserem Ansatz ein eigenes Verhalten und repräsentiert ein Objekt aus der realen Anwendung. Aus dieser Perspektive ist eine Komponente eine greifbare Einheit, die ein wohldefiniertes Verhalten aufweist.

Während der Umfeldanalyse [Mue98] des NFS-Dienstes kristallisierten sich folgende Einheiten heraus:

- der *NFS-Server*, die Haupteinheit, stellt dem Client *remote procedures* zur Verfügung, die auf den Dateisystemobjekten operieren;
- der *Mount-Server* liefert dem Client den Einstiegspunkt auf das (virtuelle) Ferndateisystem, um dieses lokal ins Dateibaum einzubinden;
- *Clients* erteilen die Aufträge und nutzen das (virtuelle) Ferndateisystem;
- die *Filehandles* identifizieren jedes Dateisystemobjekt eindeutig;
- *Attribute* repräsentieren die Metadaten eines Dateisystemobjektes und beschreiben die Eigenschaften dieses Objekts;
- und schließlich die *RPC-Schicht*, die die Kommunikation zwischen Client und Server abwickelt [San90].

Die Einheiten und ihre Beziehungen zueinander sind in Abbildung 1 dargestellt. Die Trennlinie (der Schnitt) zeigt, welche Komponenten weiter für unser Framework-Entwurf relevant sind.

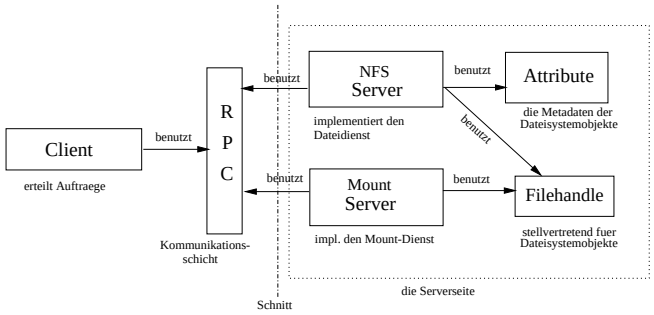


Abbildung 1. Zerlegung der NFS-Anwendung in Komponenten

3.2 Analyse der Variabilität

Ausgehend von der Anwendungsarchitektur, mit den Erkenntnissen der Umfeldanalyse, haben wir das Verhalten der Komponenten betrachtet. Wir wollten herausfinden, worin die Variabilität ihres Verhaltens lag, und welche Aspekte des NFS-Dienstes in ähnlicher Form auch in den anderen NFS-basierten Diensten zu finden waren.
Die Unterschiede zwischen verschiedenen NFS-Diensten liegen in

- der Funktionalität des File-Servers (die Semantik des Dateizugriffs)
- und in der Ausprägung der Objekte, die sich hinter dem virtuellen Dateisystem befinden.

Gemeinsam ist allen Diensten

- die Sicht auf die Objekte des virtuellen Dateisystems (die Attribute)
- und die NFS-Schnittstelle, also die Schnittstelle der File- und Mount-Server zu den NFS-Clients.

3.3 Die resultierende Framework-Architektur

Nach der Zuordnung von flexiblen und festen Anteilen in der Anwendungsarchitektur haben wir mit den Abstrahierungsschritten begonnen, also mit dem Entwurf der Framework-Architektur (s. Abbildung 2). Wir nahmen uns jede Komponente der Anwendungsarchitektur der Reihe nach vor. Da es den Rahmen dieses Berichtes sprengen würde, wird auf die Eigenschaft des NFS-Dienstes *weak cache consistency (wcc)* zu sichern, nicht eingegangen. Eine ausführliche Diskussion findet sich in [Mue98].

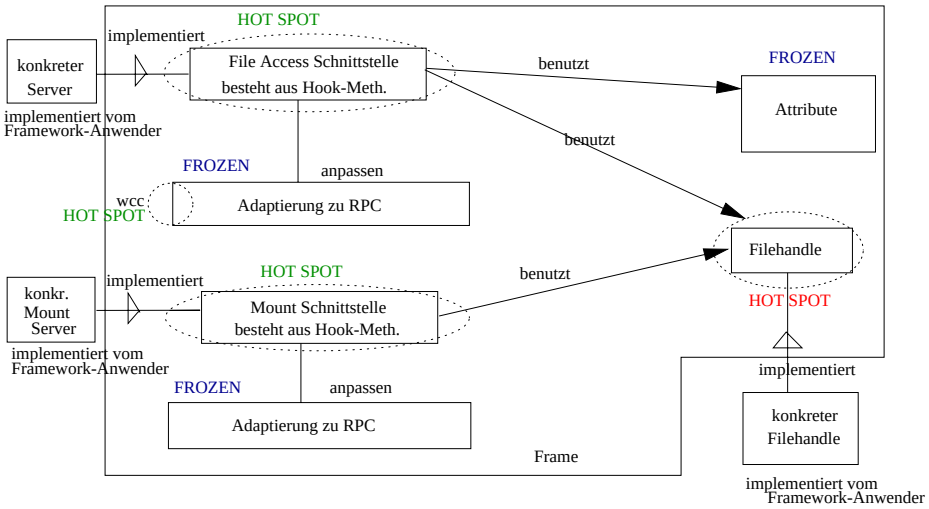


Abbildung 2. Die Framework-Architektur

3.3.1 Die Server

Der nächste logische Schritt war, die Server-Komponenten (den Mount- und den File-Server) in zwei Teile aufzuteilen: in einen flexiblen Teil mit einer abstrakten objekt-orientierten Schnittstelle, die für den Framework-Benutzer sichtbar sein wird, und einen festen Teil für die Adaptierung dieser Schnittstelle an den RPC-Kontext, also ihre Abbildung auf das NFS-Protokoll, die dem Framework-Benutzer verborgen bleibt. Die abstrakte Schnittstelle stellt die Verbindung zu der konkreten Implementierung der Server-spezifischen Funktionalität her. Der Adaptierungsteil ist zuständig für die Kommunikation mit den NFS-Clients.

In den weiteren Schritten haben wir die abstrakten Schnittstellen der beiden Server (File- und Mount-Server) so gestaltet, daß sie aus Methoden bestehen, die Platzhalter sind für Funktionalitäten, die nicht voraussehbar sind. Diese Funktionalitäten erreicht man mit Hook-Methoden, die dem Anwendungsimplementierer ermöglichen, seinen Code in dem Framework einzubinden. Dadurch braucht der Anwendungsimplementierer nur jene Methoden zu implementieren, die er auch wirklich benötigt. Damit bilden diese beiden Schnittstellen (NFS- und Mount-) je einen Hot Spot im Framework.

Neben der Abbildung der Anwendungsfunktionalität auf den RPC-Kontext hat die Adaptierungsschicht die weitere Aufgabe, die erheblichen Unterschiede zwischen den beiden NFS-Protokollversionen vor dem Anwendungsentwickler zu verbergen. Damit bildet die Adaptierungsschicht einen Frozen Spot.

3.3.2 Filehandles

Filehandles sind das Schlüsselelement im NFS (siehe [Mue98]). Sie repräsentieren die Objekte, die sich hinter dem virtuellen Dateisystem verbergen. Da die konkrete Ausprägung dieser Dateisystemobjekte aber nur der Anwendung bekannt sein muß, benötigt das Framework kein Wissen über die Syntax und Semantik der Filehandles. Damit ist die Komponente Filehandle als Ganzes veränderlich und anwendungsspezifisch.

Für das Verhalten, das Filehandles im Anwendungsbereich aufweisen, genügt es, nur die Existenz der Filehandles im Framework festzulegen und sie ansonsten als opak zu behandeln. Wird das Framework benutzt, um eine bestimmte Anwendung zu implementieren, muß der Framework-Anwender Gestalt und Verhalten von Filehandle implementieren.

4 Der Einsatz von Entwurfsmustern

Es bietet sich an, die generischen Anteile eines Frameworks mit Hilfe von Entwurfsmustern zu modellieren. Da Entwurfsmuster einen hohen Abstraktionswert besitzen und weitestgehend bekannt sind, dokumentiert man mit ihrer Verwendung gleichzeitig die Struktur des Frameworks.

Wir benutzen Entwurfsmuster, um die Struktur und Funktionsweise des Frameworks leichter verständlich und einheitlich zu gestalten. Dadurch werden auch die Anforderungen an die Dokumentation des Frameworks erfüllt, und der Anwender des Frameworks muß sich nicht mit Detailwissen belasten; es reicht, wenn er die Aufgabe versteht, die durch das Muster beschrieben wird [Joh92].

In diesem Abschnitt suchen wir zu jedem der identifizierten Hot Spots ein passendes Entwurfsmuster, welches die benötigte Variabilität bietet. Bei Bedarf zerlegen wir die Einheiten der Framework-Architektur in noch kleinere Teile und beschreiben sie schließlich durch Klassen.

4.1 Die Server-Funktionalität

Wir betrachten nun die Hot Spots des File- und Mount-Servers gemeinsam, da sie ein ähnliches Verhalten aufweisen. Die Flexibilität der abstrahierten Schnittstelle besteht in der anwendungsspezifischen Funktionalität der einzelnen Methoden. Bei verschiedenen Anwendungen kann dieselbe Methode eine andere Funktion haben.

Wir wollen also eine Menge konkret implementierter Anwendungen mit unterschiedlicher Funktionalität erreichen, die alle dieselbe abstrakte Schnittstelle mit ihren Hook-Methoden implementieren. Diese Anwendungen werden alle mit demselben gemeinsamen Kontext, der Adaptierungsschicht, zusammenarbeiten müssen, um den tatsächlichen NFS-Dienst zu implementieren. Der zwingende Kontext und die gemeinsame Schnittstelle mit unterschiedlicher Funktionalität führt uns zum *Strategy Pattern* [GHJV95].

Mit diesem Entwurfsmuster wurde eine Familie von Algorithmen definiert, deren Austauschbarkeit gewährleistet ist. Diese Algorithmen sind unterschiedlich, abhängig davon, welcher konkrete Server eingesetzt ist; jeder einzelne Algorithmus ist gekapselt und verhält sich nach außen (den NFS-Clients gegenüber) wie ein NFS-Server.

4.2 Die Erzeugung von Filehandles

Wie wir oben festgelegt haben, sind Filehandles aus der Sicht des Frameworks opak und das Wissen über die konkrete Ausprägung von Filehandles beschränkt sich auf deren Existenz. Das Problem dabei ist aber, daß in der Implementierung im Framework (in der Adaptierungsschicht) die konkreten Filehandle instantiiert werden müssen.

Die Art von Flexibilität, daß der konkrete Filehandle anwendungsspezifisch ist, und daß das Framework die konkrete Instantiierung nicht voraussehen kann, dafür aber weiß, wann sie ein Filehandle instantiieren muß, wird von einem *Factory Method Pattern* erzeugt [GHJV95]. Mit Hilfe des Entwurfsmusters Factory Method kapseln wir das Wissen, welches konkrete Filehandle realisiert werden soll und verlagern die Frage der konkreten Klasse nach außen. Obwohl die konkrete Klasse, die das Filehandle implementieren wird, nicht bekannt ist, ermöglicht dieses Entwurfsmuster ihre Instantiierung im Framework.

Der Feinentwurf des Frameworks ist in Abbildung 3 veranschaulicht. Auf dieser Entwurfsebene bilden die abgebildeten Komponenten Klassen. Anhand dieses Entwurfes wurde das Framework implementiert.

5 Implementierung

Die Implementierung des Frameworks erfolgte in Java, weil in dieser Sprache die objekt-orientierte Programmierung gut unterstützt wird. Außerdem eignet sich Java sehr gut für die Server-Entwicklung, da zum einen die Java-Plattform mit ihrer umfangreichen Systembibliothek einen guten objekt-orientierten Zugang auf die Netzebene des Rechners bietet. Zum anderen wird die Implementierung

Abbildung 3. Der Feinentwurf des Frameworks

von Multi-Threaded-Anwendungen auf einfacher Weise ermöglicht, was für die asynchrone Abarbeitung von konkurrierenden Anfragen der Clients wichtig ist.

Diese Vorteile werden mit dem Nachteil einer eingeschränkten Performanz erkauft. Dieser relativiert sich aber, wenn man in Betracht zieht, daß bei den hier angedachten Anwendungen die Flexibilität und nicht die Geschwindigkeit in Vordergrund steht. Durch moderne Java-Compiler und der Fortschritte in der Compiler-Technologie wird dieser Nachteil weiter verringert. Die vollständige Implementierung des NFS-Frameworks ist in [Mue98] zu finden.

5.1 Anwendungsbeispiel: Ein Read-Only Server

Um das Framework zu testen, haben wir zunächst eine einfache Beispielanwendung implementiert, die einen Read-Only Datei-Server realisiert [Mue98]. Der Server bietet die folgenden Dienste an:

- das Mounten eines Dateisystems,
- das Lesen einer Datei,
- das Lesen eines Verzeichnisses.

Wie in Abbildung 3 dargestellt, besteht eine Anwendung, die mit Hilfe eines Frameworks entwickelt wird, aus:

- den konkreten Klassen des Frameworks (vom Framework-Entwickler implementiert);
- die konkreten Ausprägungen der Hot Spots des Frameworks (vom Anwender implementiert);
- und aus dem Zusammenspiel der Klassen, die die Ausprägungen implementieren, mit den Klassen des Frameworks.

In Abbildung 4 wird die Beziehung der anwendungsspezifischen Klassen zu dem Framework veranschaulicht.

Im Java-Modell gibt es keine *I-nodes*, wie in Unix. Das heißt, wir mußten die bijektive Abbildung zwischen Dateien und Filehandles implementieren und auf die Konsistenz dieser Abbildung achten. Für die Verwaltung der bijektiven Abbildung implementierten wir den `FileHandleManager`. Seine Schnittstelle stellt zwei Methoden zur Verfügung. Eine Methode gibt bei Eingabe eines `SimpleFileHandle`-Objektes den dazugehörigen `File` aus (`getFileFor`), während die andere bei Eingabe eines Dateinames den dazugehörigen `SimpleFileHandle` liefert oder bei Bedarf neu berechnet.

Die Abbildung `SimpleFileHandle` – `File` modellierten wir über einen Vector in einer Richtung und über einer Hashtabelle in der anderen Richtung. Damit die Bijektivität beim parallelen Zugriff aus mehreren Threads konsistent bleibt, sind die Methoden, die darauf zugreifen, als `synchronized` definiert.

Ein Nachteil in der Anwendung des Frameworks ist, daß Java keine Schnittstelle zur Dateizugriffsebene des Betriebssystems bietet. Die Java-Plattform bietet nicht alle Aspekte eines Dateisystems, die für die NFS-Semantik wichtig sind.

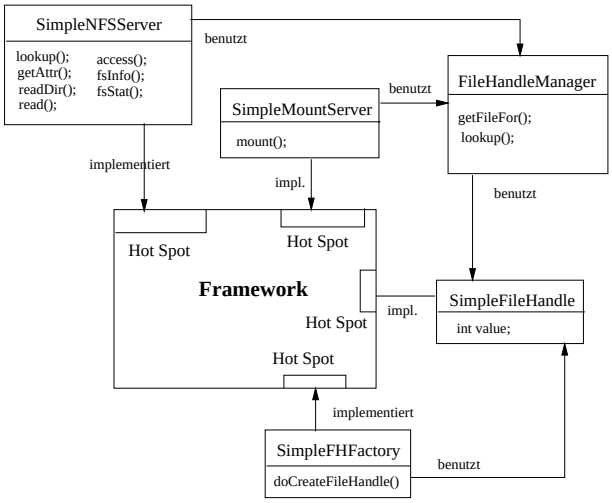


Abbildung 4. Interaktion zwischen Framework und Anwender-Klassen

Will man auf alle benötigten Attribute einer Datei zugreifen können, muß man `native`-Methoden benutzen. Bei der Verwendung von `native`-Methoden geht aber ein Teil der Portabilität verloren.

6 Zusammenfassung und Ausblick

In diesem Artikel wurde über den Entwurf eines objekt-orientiertes Framework für einen NFS-Server berichtet. Schwerpunkt dabei war das Entwurfsverfahren, anhand dessen ein objekt-orientiertes Framework aus einer konkreten Anwendung entstanden ist.
Um ausgehend von der konkreten Anwendung, das NFS, ein Framework zu realisieren, wurde folgende Vorgehensweise eingesetzt:

- Zuerst wurde eine Umfeldanalyse des NFS-Dienstes [Mue98] vorgenommen, um Objekte, Operationen und Beziehungen festzulegen, die für das NFS-Umfeld wichtig sind.
- Mit den gewonnenen Erkenntnissen wurde der konkrete NFS-Dienst objekt-orientiert zerlegt. Ergebnis war die Anwendungsarchitektur. Weiter wurde nur die Server-Seite in Betracht gezogen.
- Anhand der Analyse wurden auch die veränderlichen und festen Aspekte des NFS-Dienstes innerhalb seines Anwendungsumfeldes deutlich.
- Weiterhin wurden die Komponenten der Anwendungsarchitektur mehreren Abstrahierungsschritten unterzogen und, wo es möglich war, als generisch gestaltet. Das Ergebnis war die Framework-Architektur.

- Die generischen Anteile des Frameworks wurden mit Hilfe von Design Patterns modelliert, welche die gewünschte Art von Flexibilität erzeugen. Nach diesem Schritt hatten wir den Feinentwurf des Frameworks, anhand dessen es in Java implementiert wurde.

Das Framework wird für die Implementierung der Server-Seite eines NFS-Dateidienstes oder eines NFS-basierenden Dienstes eingesetzt. Das Anwendungsgebiet beinhaltet alle Anwendungen, die über die NFS-Schnittstelle abgewickelt werden. Ihre Gemeinsamkeit besteht darin, auf ein (virtuelles) Dateisystem zuzugreifen. Was sich hinter dem simulierten Dateisystem versteckt, hängt von der Semantik jeder Anwendung ab. All diese Anwendungen implementieren Dienste, die in der Zusammenarbeit mit NFS-Clients bestehen. Zwar haben diese Dienste verschiedene Funktionalitäten, trotzdem sehen die unterschiedlichen Server für die NFS-Clients gleich aus, und zwar wie NFS-Server. Die ganze Anwendungsbreite ist im Moment noch nicht überschaubar. Erst mit der Einsetzung des Frameworks werden sich die Grenzen der realisierbaren Anwendungen abzeichnen. Theoretisch ist es möglich, jeden Prozeß als Dateisystem zu tarnen und lokal zu mounten.

Mit jedem neu implementierten Dienst wird das Framework mächtiger, denn dadurch hat man immer mehr "Material" zum wiederverwenden. Wir sind dabei, auch andere NFS-basierende Dienste mit Verwendung des Frameworks zu implementieren und zum Wiederverwenden bereitzustellen. Die Dienste, an die wir gedacht haben, sind CFS und AMD, wie auch Dienste, wo es sich anbietet, einen Dateibaum zu simulieren (wie z.B. tar, ftp und http). Eine andere Möglichkeit wäre, Zugriffe auf Datenbanken über die NFS-Schnittstelle abzuwickeln.

Wie schon erwähnt, ist es mit Java nicht möglich, auf der Dateizugriffsebene des Betriebsystems zu arbeiten, weil die Java-Plattform keine Schnittstelle dazu bietet. Um die vollständige NFS-Semantik zu gewährleisten, wäre der Ausbau des Java-Interfaces eine nützliche Erweiterung, so daß darüber ein Zugriff auf Dateien erfolgen kann, wie es in betriebssystemspezifischen Programmiersprachen möglich ist (wie z.B C).

Literatur

[Bla93] Matt Blaze. A cryptographic file system for UNIX. In Victoria Ashby, editor, *1st ACM Conference on Computer and Communications Security*, pages 9–16, Fairfax, Virginia, November 1993. ACM Press.

[Boo91] G. Booch. *Object-oriented Design with Applications.* Benjamin/-Cummings Publishing Company, 1991.

[CPS95] B. Callaghan, B. Pawlowski, and P. Staubach. RFC 1813: NFS version 3 protocol specification, June 1995. See also RFC1094 [Sun89]. Status: Informational.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Joh92] Ralph E. Johnson. Documenting frameworks using patterns. *ACM SIGPLAN Notices*, 27(10):63–76, October 1992. *OOPSLA '92 Proceedings*, Andreas Paepcke (editor).

[Joh97a] Ralph E. Johnson. Components, frameworks, patterns. *ACM SIGSOFT Software Engineering Notes*, 22(3):10–17, May 1997.

[Joh97b] Ralph E. Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10):39–42, October 1997.

[Lei93] Marty Leisner. Tutorial: Using the AMD Automounter. *UNIX/world*, 10(1):70–??, January 1993.

[Min93] Ronald G. Minnich. The AutoCacher: A file cache which operates at the NFS level. In USENIX Association, editor, *Proceedings of the Winter 1993 USENIX Conference: January 25–29, 1993, San Diego, California, USA*, pages 77–83, Berkeley, CA, USA, Winter 1993. USENIX.

[Mue98] Evelyn Muelthaler. Ein objekt-orientiertes Framework fuer NFS-Server. Diplomarbeit, Technische Universität Darmstadt, FB Informatik, Fachgebiet Systemprogrammierung, Dezember 1998.

[San90] Michael Santifaller. *TCP/IP und NFS in Theorie und Praxis: UNIX in lokalen Netzen*. Addison-Wesley, 1990. ISBN 3-89319-246-8.

[Sch97] Hans Albrecht Schmid. Systematic framework design. *Communications of the ACM*, 40(10):48–51, October 1997.

[Sun89] Sun Microsystems, Inc. RFC 1094: NFS: Network File System Protocol specification, March 1989. See also RFC1813 [CPS95]. Status: Informational.