1 Einleitung

1.1 Warum Spring?

Ursprünglich ist Java als einfachere Alternative zu C++ gestartet. Bald standen aber Java und insbesondere Java EE, also die Java-Version für Geschäftsanwendungen, in dem Ruf, sehr kompliziert zu sein. Projekte müssen viel Rücksicht auf die Technologie nehmen und entwickeln Code, der mehr damit beschäftigt ist, die APIs zufriedenzustellen, als Geschäftslogik zu implementieren. Dies schlägt sich sowohl auf die Produktivität als auch auf die Motivation der Entwickler nieder. Außerdem ist die Anzahl der APIs und Bibliotheken inzwischen sehr groß, so dass die Einarbeitung aufwendig ist, zumal die Bibliotheken keinen durchgängigen Prinzipien entsprechen. Gleichzeitig sind die Bibliotheken eine große Stärke von Java, da sie viele Einsatzbereiche abdecken.

Häufig entstehen in Java-Projekten eigene Frameworks als Schicht über den verschiedenen Bibliotheken. Solche Frameworks müssen aber langfristig gewartet werden, obwohl sie rein technischer Natur sind und so nicht zu den eigentlichen fachlichen Aufgaben des Projekts beitragen. Außerdem ist eine wesentliche Voraussetzung für die Entwicklung eines guten Frameworks, dass man mit ihm in mehreren Projekten Erfahrungen sammelt und dadurch das Framework iterativ verbessert. Dies ist in einem einzigen Projekt oder Unternehmen kaum möglich, so dass die Qualität der Frameworks oft zu wünschen übrig lässt.

1.2 Was ist Spring?

Das Spring-Framework ist eine Möglichkeit, die Entwicklung mit Java deutlich zu vereinfachen. Es basiert auf drei Elementen:

1. Spring bietet eine vereinfachte und vereinheitlichte API-Schicht über viele Java-SE-APIs, Java-EE-APIs und Open-Source-Frameworks an. Der wesentliche Grund für die zunehmende Komplexität der Entwicklung mit Java ist nämlich nicht die Sprache selbst, sondern es sind die zahlreichen APIs, die Java anbietet. Diese APIs folgen unterschiedlichen Konzepten und sind oft unnötig schwer zu handhaben. Spring löst dieses Problem.
2. Eine wesentliche Herausforderung bei objektorientierten Systemen ist das Verwalten der Abhängigkeiten zwischen Objekten, und zwar insbesondere zwischen jenen, die Geschäftslogik als Services implementieren. Diese Dienste bauen aufeinander auf, so dass man Netze aus solchen Objekten erzeugen muss. Dieses Problem wird oft »irgendwie« im Code gelöst. Spring bietet mit Dependency Injection einen einheitlichen Weg, solche Objektnetze aufzubauen: Den Objekten werden abhängige Objekte anhand einer einfachen XML-Konfiguration zugewiesen (»injiziert«). Die Objekte suchen sich also nicht Referenzen zu anderen Objekten, sondern sind passiv. Dadurch sind sie von der Umgebung unabhängig, da sie diese nicht aktiv benutzen. Sie können flexibel in unterschiedlichen Umgebungen verwendet werden wie z.B. im Application-Server oder in einer »normalen« Java-SE-Umgebung. Außerdem ist durch Dependency Injection jede Spring-Anwendung konfigurierbar. Man kann beispielsweise recht leicht eine andere Datenbank nutzen. Vor allem das Testen wird erleichtert, da man das System leicht in Testumgebungen laufen lassen kann oder den Objekten spezielle Testobjekte mit reduzierter Funktionalität (z.B. Mocks) als abhängige Objekte zuweisen kann.
3. Schließlich bietet Spring eine Unterstützung für AOP (aspektorientierte Programmierung). Normalerweise sind Belange wie Transaktionen, Tracing oder Sicherheit im Code verstreut: Eine Methode eröffnet z.B. eine Transaktion, überprüft, ob der aktuelle Benutzer die nötigen Rechte hat, und implementiert dazwischen die Geschäftslogik. Dieses Vorgehen findet sich in recht vielen Methoden, so dass diese Belange in vielen unterschiedlichen Bereichen des Systems implementiert sind. AOP ermöglicht die zentralisierte und von der Geschäftslogik getrennte Implementierung solcher Belange. Außerdem können durch Aspekte die Annotationen recht einfach mit einer auszuführenden Logik verbunden werden.

Ein Ziel von Spring ist, dem Entwickler möglichst viele Freiheiten zu lassen. Man kann daher die einzelnen Teile des Frameworks unabhängig voneinander nutzen. Der Entwickler kann z.B. nur die Teile ver-

wenden, die in seinem Projekt einen besonders großen Vorteil bringen. Außerdem muss man mit Spring nicht eine bestimmte Lösung z.B. für Persistenz verwenden, sondern findet eine breite Palette von in Spring integrierten Frameworks vor. Zudem soll Spring wenig invasiv sein, d.h., der eigene Code hängt nur wenig vom Spring-Framework ab. Eine vollständige Unabhängigkeit lässt sich natürlich nicht immer erreichen.

Durch diese Ansätze richtet sich die Programmierung wieder mehr an Objektorientierung als an den verwendeten APIs aus. So werden Projekte in die Lage versetzt, Lösungen für die jeweiligen Projektziele zu erarbeiten, statt ohne Weiteres Nachdenken einem bestimmten Stil zu folgen, der durch die APIs vorgegeben wird. Gleichzeitig kann man mehr auf die eigentliche Logik fokussieren, statt eigene Abstraktionen über die verschiedenen APIs oder anderen Infrastrukturcode zu entwickeln.

Basierend auf Spring sind weitere Frameworks entstanden. Dazu zählen Spring Security [Spring Security] für Sicherheit oder Spring Web Services [SpringWebServices] für die Implementierung von Web Services sowie unterschiedliche Frameworks für Webanwendungen wie Spring MVC, Spring JavaScript [Spring JavaScrip], Spring Faces [SpringFaces] oder Spring Web Flow [SpringWebFlow]. Auch eine Portierung von Spring für das .NET Framework existiert [Spring.NET]. Und mit dem SpringSource dm Server steht mittlerweile auch ein speziell auf Spring abgestimmter Application-Server zur Verfügung, der zudem die Vorteile von OSGi sehr einfach nutzbar macht.

Diese Technologien zeigen auf, dass die Spring-Prinzipien auch in anderen Bereichen sinnvoll einsetzbar sind und Spring sich als umfassende Basis für die Entwicklung von Java-Systemen etabliert hat.

1.3 Spring und Java EE

Vorweg: Spring wird oft nur als eine Lösung für Enterprise-Java-Systeme angesehen. Das ist aber nicht der einzige Einsatzbereich: Sogar GUI-Anwendungen kann man mit Spring schreiben. Und es gibt mittlerweile auch eine Portierung von Spring auf Java ME. Spring ist also ein universeller Ansatz für die Entwicklung beliebiger Java-Anwendungen.

Dennoch stellt sich die Frage, warum man sich mit Spring beschäftigen sollte, wenn Java EE mittlerweile ähnliche Konzepte verfolgt und sich ebenfalls Vereinfachung zum Ziel gesetzt hat. Wenn man sich an den drei grundlegenden Elementen aus dem letzten Abschnitt orientiert, ist Spring in jedem Bereich dem Java-EE-Programmiermodell überlegen:

1. Java EE bietet zwar mittlerweile Vereinfachungen bei einigen APIs, aber im Gegensatz zu Spring keine zusätzliche Abstraktionsschicht für alle APIs und daher keine durchgängige Vereinfachung oder Vereinheitlichung. Außerdem sind viele bewährte Lösungen vor allem aus dem Open-Source-Bereich nicht Teil des Java-EE-Standards.
2. Dependency Injection ist mit Java EE nur für Java-EE-Elemente wie EJBs, Servlets usw. möglich, aber nicht für »ganz normale« Java-Objekte wie bei Spring. Dadurch kann dieses Konzept nicht durchgängig verwendet werden, was den Nutzen verringert.
3. Es gibt in Java EE keine auch nur annähernd mit Spring vergleichbare Unterstützung für AOP. Lediglich in EJB 3 gibt es ein einfaches Interceptor-Konzept.

Dennoch hat Java EE einen wichtigen Vorteil: Es ist ein Standard. Dadurch erreicht man Herstellerunabhängigkeit. Man kann die Java-EE-Implementierung austauschen und die Anwendung bleibt im Idealfall trotzdem ausführbar oder ist zumindest einfach portierbar. Dies ist jedoch eine Eigenschaft der Java-EE-*Umgebung*, nicht des *Programmiermodells*. Wenn man Spring als Abstraktion über einer Java-EE-Umgebung verwendet, profitiert man weiterhin von diesem Aspekt des Java-EE-Standards. Und man kann seine Anwendung sogar recht leicht auf Nicht-Java-EE-Umgebungen wie den SpringSource dm Server oder den Tomcat-Webserver portieren. Man gewinnt mit Spring also sogar im Bereich Portierbarkeit einiges hinzu.

Der andere Vorteil eines Standards wie Java EE ist, dass man viel Wissen am Markt über Java EE findet. Aber hier hat Spring keinen prinzipiellen Nachteil, sondern ist mittlerweile ein De-facto-Standard, zu dem es auch viel Wissen am Markt gibt – sei es als Consulting, Training, Bücher oder qualifizierte Mitarbeiter.

Neben der Standardisierung ist ein weiteres Argument für Java-EE-Umgebungen, dass wesentliche Eigenschaften von Enterprise Software wie Skalierbarkeit und Ausfallsicherheit, aber auch Deployment oder Monitoring durch sie abgedeckt werden. Auch hier gilt: Spring kann eine Abstraktion über Java EE anbieten, diese Eigenschaften bleiben dann unverändert. In einigen Bereichen wie Pooling oder Sicherheit bietet Spring eigene Lösungen an. Man kann in diesen Fällen die Spring-Lösung oder die Java-EE-Lösung mit dem Spring-Framework nutzen.

Oft wird Spring vor allem als technologische Alternative zu EJB gesehen. Gerade EJB stand sehr in der Kritik [TCLL03, Wol03], weil es sehr komplex ist, zu schwer testbaren Systemen führt und man viel zusätzlichen Code schreiben muss. Dennoch wird es oft als wesent-

liches Element von Java-EE-Architekturen verwendet, mit dem man die Geschäftslogik implementieren kann. Spring ist dafür eine sinnvolle Alternative, die die Schwächen von EJB nicht hat. Man kann Spring aber auch an vielen anderen Stellen nutzen. Es ist mit Spring sogar möglich, aus normalen Java-Objekten EJBs zu machen. Daher gibt es keinen echten Gegensatz zwischen Spring und EJB – man kann Spring auch als Framework für die Entwicklung von Java-EE- oder EJB-Anwendungen verwenden. Die Schwächen von EJB 2.1 haben aber zweifellos Spring zu einem guten Start verholfen. Im Vergleich zu EJB 3 bietet Spring als Option ein nahezu identisches Programmiermodell an, ohne dass man auf die zahlreichen fortgeschrittenen Features von Spring verzichten muss (siehe Abschnitt 6.5.3).

Mittlerweile haben aber nicht mehr Java-EE-Application-Server den größten Marktanteil, sondern einfache Webserver wie Tomcat. Sie sind wesentlich einfacher zu handhaben und benötigen weniger Systemressourcen als ein Java-EE-Server. Viele Features eines Java-EE-Servers werden in der Praxis auch gar nicht genutzt. Auf Servern wie Tomcat steht aber das Java-EE-Programmiermodell nicht vollständig zur Verfügung, insbesondere EJB fehlt. Gerade auf solchen Umgebungen kann Spring seine Stärken ausspielen, weil es sein Programmiermodell auch auf solchen einfachen Umgebungen anbieten kann. Daher ist Spring sicher auch ein wichtiger Grund für den Siegeszug von Tomcat als Ablaufumgebung für Enterprise-Java-Anwendungen.

1.4 Woher kommt Spring?

Rod Johnson hat die Grundlagen für das Spring-Framework in [Jo02] dargestellt. Es wurde aus der Praxis verschiedener Java-EE-Projekte geboren und ist somit von Anfang an in konkreten Projekten im Einsatz gewesen. Schließlich wurde Spring zu Open Source. Heute arbeiten kontinuierlich mehrere Committer an Spring, und es steht unter der Apache-2.0-Lizenz. Spring wird bis heute genau so weiterentwickelt, wie dies bei Frameworks gemacht werden sollte: Anhand des Feedbacks aus verschiedenen Projekten und Anwendungsfällen wird es laufend weiter verbessert.

Die Entwicklung des Spring-Frameworks und der darauf basierenden Technologien wie Spring Web Flow, Spring Web Services, Spring Modules, Spring OSGi oder Spring Security wird von der Firma SpringSource vorangetrieben, die zu Spring auch Training, Support und andere Dienstleistungen wie Consulting anbietet. SpringSource ist Arbeitgeber fast aller Spring-Committer. Außerdem beschäftigt SpringSource einige der Kern-Committer anderer Projekte wie Apache Tomcat, Apache HTTPD, Apache Active MQ, Hyperic oder auch Groovy und Grails.

1.5 Warum dieses Buch?

Dieses Buch gibt eine umfassende Einführung in Spring. Es fokussiert auf die Spring-Lösung eines Problems und stellt nicht zunächst vor, warum eine Lösung ohne Spring weniger gut ist – es wird nicht mit EJB oder Java EE »abgerechnet«, aber Integrationsmöglichkeiten werden natürlich aufgezeigt. Dadurch kann man das Buch auch ohne viel Vorwissen lesen, und es fokussiert auf das eigentliche Thema, nämlich Spring.

Ein wesentlicher Teil des Buchs beschäftigt sich mit der Unterstützung von Spring für verschiedene APIs. Dabei wird die Integration dieser APIs in Spring in den Mittelpunkt gestellt, aber man kann sich auch einen allgemeinen Überblick über die Möglichkeiten der APIs verschaffen. So kann die Einsetzbarkeit der APIs bewertet werden. Das vorliegende Buch vermittelt daher auch einen Eindruck der Möglichkeiten von Java vor allem im Enterprise-Bereich.

Zur Illustration wird im gesamten Buch ein durchgängiges Beispiel verwendet. So werden die Ansätze von Spring direkt am Code erläutert.

Ein weiterer wichtiger Aspekt des Buchs ist, dass es neben Spring selbst auch weitere auf Spring aufbauende Technologien erläutert:

- Das Spring-MVC-Framework bietet eine Alternative bei der Entwicklung von Webanwendungen.
- Spring Web Flow erlaubt die Abbildung von Abläufen auf einer Website und kann dabei nicht nur in das Spring-MVC-Framework, sondern z.B. auch in JSF integriert werden.
- Das Spring-Security-Sicherheitsframework erlaubt die Absicherung einer beliebigen Java-Webanwendung ohne Eingriff in den Code. Zusammen mit Spring kann auch auf Ebene der Geschäftslogik die Anwendung geschützt werden.
- Außerdem wird auf Spring Web Services und Spring Dynamic Module for OSGi Service Platforms und den SpringSource dm Server eingegangen.

Im Buch wird die Version 3.0 des Spring-Frameworks behandelt.

1.6 Patterns und dieses Buch

Spring basiert auf Prinzipien, die im vorliegenden Buch als Patterns dargestellt werden. Der Begriff des Patterns wurde in den 90er Jahren des vergangenen Jahrhunderts in den Bereich der Softwareentwicklung eingebracht [GHJV94, C2Wiki] und war damals einer der wichtigen neuen Begriffe. Mittlerweile sind Patterns eine wesentliche Grundlage

bei der Kommunikation zwischen Entwicklern und eine gute Möglichkeit, um Best Practices in der Softwareentwicklung weiter zu verbreiten.

Pattern-Definition

Obwohl Patterns sehr weit verbreitet sind, soll hier noch einmal kurz der Begriff erläutert werden:

> Jedes Pattern ist eine dreiteilige Regel, die eine Beziehung zwischen einem bestimmten Kontext, einem System von Kräften, die in diesem Kontext häufig auftreten, und einer Softwarekonfiguration ausdrückt, durch die sich die Kräfte aufheben. (nach [Hillside])

Patterns werden also unter den jeweils geschilderten Voraussetzungen häufig eingesetzt, oft ohne dass man dafür einen Namen hat. Daher ist es auch eher so, dass man Patterns in Softwaresystemen findet und benennt und nicht etwa erfindet.

Patterns haben viele Vorteile. Für dieses Buch ermöglichen sie eine explizite Benennung der Prinzipien von Spring. Dies erfolgt in einer einheitlichen Darstellungsform (Abschnitt 4.2.2), die auch außerhalb des vorliegenden Buchs verwendet wird und sich bewährt hat.

Die Patterns treten vor allem bei der Vereinfachung und Vereinheitlichung der verschiedenen APIs auf. Sie werden bei der Behandlung der einzelnen APIs schrittweise eingeführt. Gerade hier ist das Erläutern der Prinzipien wichtig, denn es gibt sehr viele in Spring integrierte APIs. Wenn man aber einmal die Prinzipien anhand der Patterns verstanden hat, ist leicht nachvollziehbar, wie die einzelnen Integrationen funktionieren.

1.7 Wie man das Buch lesen sollte

Die Kapitel 2 und 3 stellen Dependency Injection und aspektorientierte Programmierung dar. Da dies die Grundlagen von Spring sind, sollten diese Kapitel auf jeden Fall gelesen werden.

Wer vor allem im Middle-Tier entwickelt, sollte das Kapitel 4 über Transaktionen, das Kapitel 5 über die Unterstützung von Persistenz mit Spring und schließlich das Kapitel 6 über die Unterstützung von Technologien für die Implementierung verteilter Objekte lesen. Vor allem das Kapitel 8 über die Unterstützung der verschiedenen Enterprise APIs kann ebenfalls sehr nützlich sein.

Für Webentwickler ist vor allem das Kapitel 7 hilfreich, das die Ansätze von Spring in diesem Bereich beleuchtet. Abschnitt 8.2 befasst sich mit der Absicherung von Webanwendungen durch das Spring-Security-Sicherheitsframework. Schließlich stellt der Abschnitt 8.5 noch das Verschicken von E-Mails mit Spring dar.

Für Entwickler von GUI-Anwendungen kann abhängig von den Technologien, die unterhalb der GUI-Ebene verwendet werden, Kapitel 4 über Transaktionen, das Kapitel 5 über die Unterstützung von Persistenz mit Spring und das Kapitel 6 über verteilte Objekte mit Spring relevant sein. Auch einige der in Kapitel 8 vorgestellten Enterprise-APIs können von Interesse sein. Ebenfalls kann es sinnvoll sein, sich mit der OSGi-Unterstützung von Spring vertraut zu machen (Abschnitt 8.7), da OSGi auch die Basis von Eclipse RCP ist, das im Client-Bereich sehr weite Verbreitung hat.

Grundsätzlich sollte man auch einen Blick in Kapitel 9 werfen, um einen Einblick in die auf Spring aufbauenden Frameworks zu bekommen. Die möglichen Lesepfade werden in Abbildung 1–1 dargestellt.

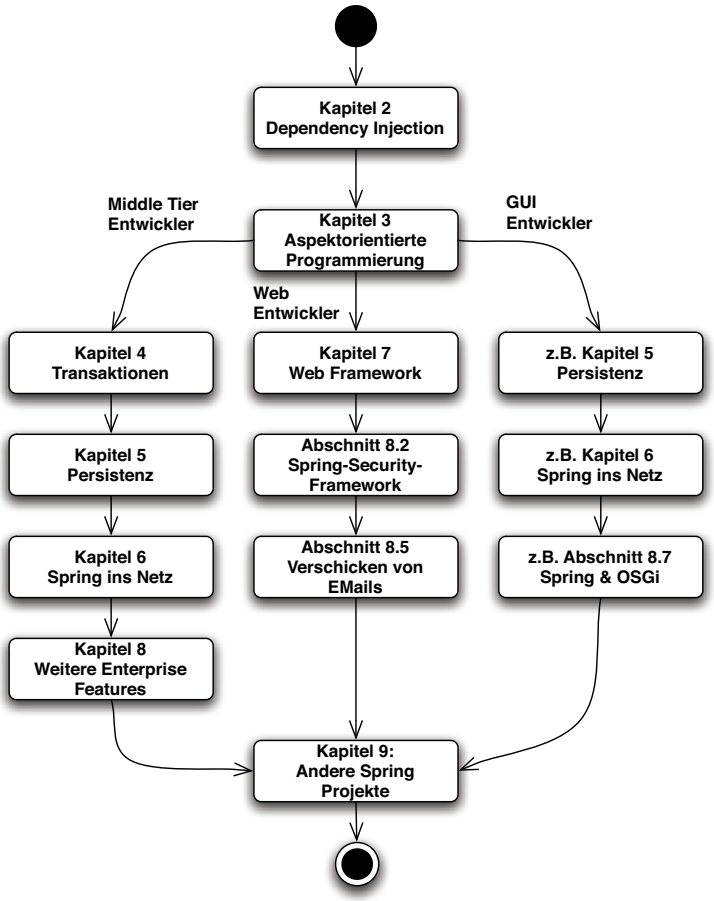

Abb. 1–1
Mögliche Pfade durch das Buch

Als Unterstützung zum Buch ist es auf jeden Fall sinnvoll, die Online-Dokumentation des Spring-Frameworks und der verschiedenen anderen

Frameworks zurate zu ziehen. Dort kann man ergänzend zu den grundlegenden Erläuterungen im Buch detaillierte Informationen finden. Viele Details finden sich in den Beispielen im Buch, deren Sourcecode man von der Website *http://www.spring-buch.de/* herunterladen kann.

Zu diesen Beispielen noch ein Hinweis: Sie können als Grundlage für die Implementierung einer eigenen Spring-Anwendung dienen, man muss dabei allerdings vorsichtig sein. Die Beispiele sollen nämlich vor allem zeigen, was man alles mit Spring machen kann. Daher implementieren sie oft dieselbe Funktionalität auf unterschiedlichen Wegen: Das würde man natürlich in einem realen Projekt gerade versuchen zu vermeiden, um zu einem einheitlichen Entwurf zu kommen. Eine Alternative stellt AppFuse [AppFuse] oder AppFuse Light [AppFuseLight] dar. Mit ROO [ROO] steht außerdem ein Generator bereit, der einem sehr schnell ein Spring-Projekt einschließlich eines automatischen Builds erzeugen kann.

1.8 Mit Spring entwickeln

Wenn man selbst Projekte mit Spring durchführen will, sollte man sich zunächst das Framework herunterladen [Spring]. Darüber hinaus gibt es einige Werkzeuge, die einen bei der Entwicklung mit Spring unterstützen können. Diese werden in Abschnitt 9.2 ausführlich vorgestellt. Gerade die Spring-IDE (Abschnitt 9.2.1) und die SpringSource Tool Suite (Abschnitt 9.2.2) sind wesentliche Vereinfachungen bei der Entwicklung.

1.9 Danksagung

Natürlich gilt mein Dank in erster Linie den Spring-Entwicklern, insbesondere Rod Johnson und Jürgen Höller, der Spring-Community und den Entwicklern der darauf aufbauenden Werkzeuge und Bibliotheken. Außerdem möchte ich mich bei den Reviewern bedanken, die mit ihren Kommentaren das Buch wesentlich beeinflusst haben: Berthold Daum, Sven Efftinge, Michael Hunger, Carsten Heyl, Rolf Katzenberger, Markus Kehle, Bernd Kolb, Johannes Link, Marcus Pant, Arjen Poutsma, Peter Rossbach, Arno Schmidmeier und Niko Schmuck.

Schließlich habe ich meinen Freunden, Eltern und Verwandten zu danken, die ich für das Buch oft vernachlässigt habe – insbesondere meiner Freundin. Und natürlich gilt mein Dank auch meinem Arbeitgeber SpringSource.

Last but not least möchte ich dem dpunkt.verlag und René Schönfeldt danken, der mich sehr professionell bei der Erstellung des Buchs unterstützt hat.