OPITZ CONSULTING

Dokumentation des AI Projektes

Allgemeine Informatik flexibel

Fachhochschule Köln

Gegenüberstellung zweier Frameworks in Bezug auf deren Tauglichkeit zur Entwicklung einer Bürgschaftsverwaltung

Author:

Mara Braun

Unternehmen:

OPITZ CONSULTING

Deutschland GmbH

Kirchstrasse 6 51647

Gummersbach www.opitz-consulting.com

Abstract

Die OPITZ CONSULTING Deutschland GmbH ist seit einigen Jahren mit der Wartung einer Webanwendung zur Verwaltung von Bürgschaften betraut. Da diese Anwendung jedoch schon vor einigen Jahren entwickelt wurde entspricht ihre Performanz nicht mehr den Anforderungen der heutigen Zeit. Daher soll die Anwendung neu entwickelt werden. Hierfür kommen zwei große Frameworks in Frage. Das erste Framework ist das Application Development Framework von Oracle, in dessen veralteter Version die zugrunde liegende Anwendung ursprünglich erstellt wurde. Das zweite Framework ist das zur Zeit verbreitete Framework Grails.

Im Folgenden Dokument sollen nun die beiden Frameworks miteinander verglichen werden. Dieser Vergleich soll die Vor und Nachteile der beiden Frameworks gegenüberstellen, um zu evaluieren, welches sich besser zur Neuentwicklung der Bürgschaftsverwaltung eignet.

Inhaltsverzeichnis

1 Einleitung **1**

2 Theoretische Grundlagen **2**

2.1 Entstehung 2

2.2 MVC Architektur allgemein 2

2.3 ADF 4

2.3.1 Grundlegendes 4

2.3.2 Architektur 4

2.4 Grails 6

2.4.1 Grundlegendes 6

2.4.2 Architektur 6

3 Gegenüberstellung der Frameworks **9**

3.1 Anforderungen an Web Application Frameworks 9

3.2 Vor- und Nachteile von ADF 12

3.3 Vor- und Nachteile von Grails 13

4 Projektspezifische Gegenüberstellung **16**

4.1 „AvaleNet“ 16

4.1.1 Projekthintergrund 16

4.1.2 Projektanforderungen 16

4.2 Vor- und Nachteile der Frameworks im Bezug auf „AvaleNet“ . 17

4.3 Aufwiegen der Vor- und Nachteile 19

5 Fazit **22**

Literaturverzeichnis **23**

Abbildungsverzeichnis

1 MVC-Architektur 3

2 MVC-Architektur von ADF 5

3 MVC-Architektur von Grails 7

4 Technologiestack von Grails 8
5 Zeitverlauf des Interesses an ADF und Grails weltweit (Quelle: http://goo.gl/Ma9kWJ) 13
6 Zeitverlauf des Interesses an ADF und Grails in Deutschland (Quelle: http://goo.gl/Ma9kWJ) 19

1 Einleitung

2 Theoretische Grundlagen

Dieses Kapitel soll für ein grundlegendes Verständnis der Architektur von ADF und Grails und dem Umfang dieser Frameworks sorgen, um im nachfolgenden Vergleich der beiden Frameworks die Vor und Nachteile nachvollziehen zu können.

2.1 Entstehung

Mit der Einführung von SOA (Service Oriented Architecture) in der Softwareentwicklung hat die Entwicklung von traditionellen Webanwendungen, in denen die Anwendung eine vollständige Lösung ist ein Ende gefunden. Moderne Anwendungen sind heute nicht mehr eine vollständige Lösung, sondern sie sind komponentenbasierte Benutzerschnittstellen, die lokale und remote Services für ihre Business Logik verwenden. (Oracle Fusion Developer Guide Introduction Seite xxi)

2.2 MVC Architektur allgemein

Dieses Kapitel soll dazu dienen, ein Verständnis für die den beiden Frameworks zu Grunde liegenden Architektur zu schaffen.

Die Model View Controller (MVC) Architektur wurde in den 1970er Jahren von Trygve Reenskaug für die Plattform Smalltalk entwickelt und spielt bis heute eine bedeutende Rolle in den meisten UI-Frameworks und dem UI-Design. Wie der Name schon vermuten lässt, besteht die MVC Architektur aus drei Rollen, dem Model, dem View und dem Controller.

- Das Model ist ein nicht sichtbares Objekt, dass einige Informationen der Domäne, wie z.B. alle Daten und Verhalten enthält. Diese Daten und Informationen müssen nicht denen die in der UI verwendet werden entsprechen.
- Die View dient dazu Informationen aus dem Model anzeigen zu können. Dies kann z.B. in Form einer HTML-Seite geschehen, in der die

gewünschten Informationen des Models dann angezeigt werde.

- Die letzte Rolle ist die des Controllers. Der Controller dient dazu Benutzereingaben anzunehmen, das Model zu manipulieren und das View Objekt zu aktualisieren.

Die wichtigste Trennung ist die Trennung von Model und View. Der Grundgedanke hierbei ist es, dass ein Entwickler wenn er eine View (Ansicht) entwickelt über andere Dinge nachdenkt, als wenn er das Model entwickelt. Beispielsweise denkt er bei der Entwicklung der View mehr über die Mechanisemen der UI nach und bei der Entwicklung des Models mehr über die Geschäftsstrategie nach. Zudem möchte ein User eventuell ein und die selbe Information aus dem Model in einer Anwendung auf unterschiedliche Weise dargestellt haben, was sich durch die Trennung von View und Model vereinfacht, da für jede Ansicht das selbe Model verwendet werden kann und sich nur die View ändert. Ein letzter Aspekt, der für die Trennung von Model und View spricht, ist das nicht sichtbare Objekte meist besser zu testen sind als sichtbare und so durch die MVC-Architektur die Modellogik getrennt von der Benutzeroberfläche (GUI) getestet werden kann.

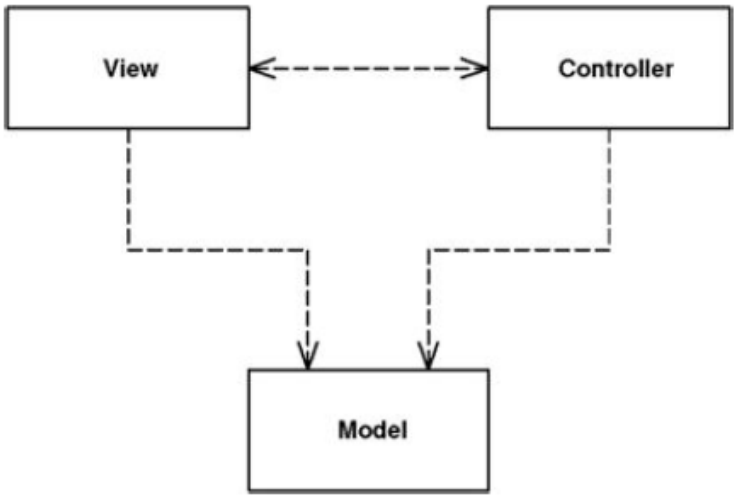

Abbildung 1: MVC-Architektur

Die Trennung von View und Controller dagegen ist weniger relevant und

nicht immer sinnvoll, weshalb in einigen Frameworks heute eine Kombination von View und Controller verwendet wird. Selbst in den meisten Versionen von Smalltalk, für die die MVC-Architektur ursprünglich entwickelt wurde, wurde meiste eine Kombination von View und Controller verwandt. Die Trennung von View und Controller ist vorallem dann sinnvoll, wenn er sich um Rich-Client-Systeme handelt oder der Controller von Web-Frontends separiert wird.
Wie auch in der Abbildung zu sehen, ist die Richtung der Abhängigkeiten für die MVC-Architektur entscheidend. Die Abbildung zeigt, dass sowohl View als auch Controller vom Model abhängig sind, jedoch keine Abhängigkeit des Models von View oder Controller besteht. Diese Unabhängigkeit des Models bedeutet, dass das Model auch bei Änderungen an View und Controller unverändert bleibt und damit Änderungen am View deutlich erleichtert. (Patterns of Enterprise Application Architecture - Martin Fowler)

2.3 ADF

Dieses Kapitel soll einen Einblick in die Möglichkeiten des Frameworks ADF bieten und dessen Aufbau und Bedienbarkeit erläutern.

2.3.1 Grundlegendes

Eine Lösung zur Entwicklung von modernen Anwendungen als komponentenbasierte Benutzerschnittstelle bietet ORACLE mit dem Application Development Framework (ADF). ADF ist ein Java EE (Java Enterprise Edition) Framework, das Anwendungsentwicklung mit Java, Java EE und SOA vereinfachen soll um ein breites Publikum von Geschäftsbereich und Technologie Experten anzusprechen, die zusammen arbeiten müssen, um langlebige Enterprise Softwarelösungen entwickeln zu können.(Oracle Fusion Developer Guide Introduction Seite xxiii)

2.3.2 Architektur

Das Application Developement Framework von Oracle basiert, wie auch einige andere Web Applicatin Frameworks heutzutage auf der im vorigen Kapitel

erklärten MVC-Architektur. Auch ADF hat Model, View und Controller, jedoch werden diese Schichten der Architektur noch durch die Ebenen Business Services und Data Services erweitert.

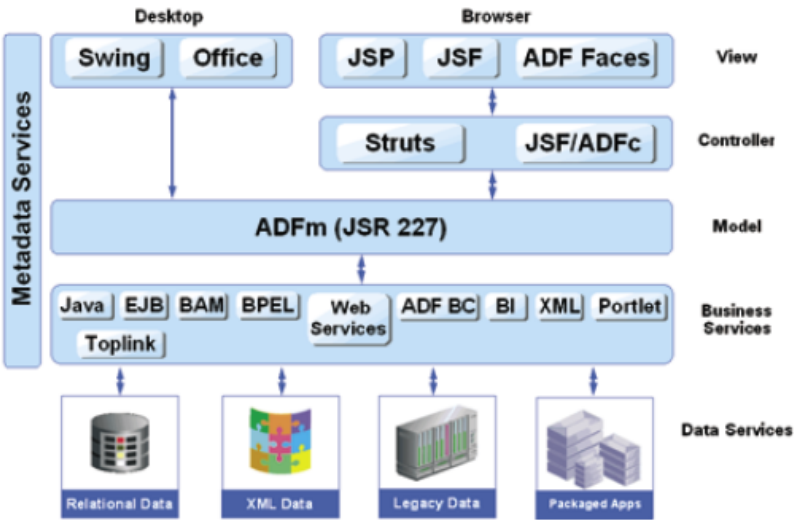

Abbildung 2: MVC-Architektur von ADF

Die View Ebene der ADF Architektur (siehe Abbildung) ist wie allgemein von der MVC Architektur vorgesehen die Anzeigeebene der Anwendung und unterteilt sich in die Kategorien Desktop und Browser, da mit ADF sowohl Java Swing Anwendungen (also Desktopanwendungen) als auch Web- oder auch Browser-Anwendungen erstellt werden können. Diese Unterteilung wirk sich auch auch auf die nächste Ebene der Architektur aus, denn eine Trennung von View und Controller wird für Desktopanwendungen nicht zwingend benötigt (siehe auch Kapitel MVC). Für Webanwendungen ist es jedoch sinnvoll einen Controller zu haben um Frontend und Controller getrennt zu behandeln. Der Controller von ADF Anwendungen hat die Aufgabe, Benutzer Aktionen zu interpretieren und zu entscheiden, welche Seite dem User in welcher Reihenfolge angezeigt werden.
Das Model ist hier die Abstraktionsschicht. Es besteht aus dem ADF Binding Layer und den Data Conrols. Die Data Controls bilden die Schnittstelle zu den Business Services und können daher verschiedene Geschäftslogikimplementierungen zur Verfügung stellen. Der Binding Layer verwendet wiederum

die jeweilige vom Data Control zur Verfügung gestellte Geschäftslogik, um mit dem Controller (wenn er vorhanden ist) interagieren zu können und entsprechende Daten im View anzuzeigen und Funktionen in der Geschäftslogik aufrufen zu können. Die beiden zusätzlichen Ebenen in der ADF-MVC-Architektur sind die Business Services und die Data Services. Mit Hilfe dieser Ebenen können in der Anwendung z.B. Web Services angebunden werden und es können Verbindungen zu Relationalen Datenbanken oder anderen Data Services hergestellt werden. (Quellen: Doag Artikel, ADF Developers Guide 11gR1,Oracle ADF Enterprise Application Development – Made Simple, ADF Buch)

2.4 Grails

Dieses Kapitel soll einen Einblick in den Aufbau des Frameworks Grails ermöglichen und damit die unterschiedlichen Möglichkeiten zur Verwendung dieses Frameworks grundlegend erläutern.

2.4.1 Grundlegendes

Auch Grails ist ein Framework, mit dem Web-Anwendungen erstellt werden können. Grails wurde 2005 unter anderem im Zuge der Beliebtheit von auf dynamische Sprachen basierenden Frameworks entwickelt, die angeführt wurde von Ruby on Rails(Glossareintrag). Grails basiert auf der 2003 entstandenen Programmiersprache Groovy, die wiederum auf Java basiert. Das wesentliche Ziel des Framework Grails wurde wie folgt beschrieben: „The goal of Grails is to create a platform with the essence of frameworks like Rails or Django or CakePHP, but one that embraces the mature environment of Java Enterprise Edition(Java EE) and its associated APIs.“ Grails soll also die guten Eigenschaften der bereits bestehenden Frameworks und Java EE verbinden um eine neues mächtiges Framework zu schaffen.

2.4.2 Architektur

Auch Grails verwendet, wie ADF und einige weiter Web Frameworks, die MVC-Architektur. Sie ist sehr ähnlich aufgebaut zu der im ersten Kapitel

beschriebenen MVC-Architektur. Es gibt Model, View und Controller die miteinander agieren, wobei der Controller im Fall von Grails nicht optional ist sondern immer benötigt wird um eine Interaktion zwischen Model und View zu ermöglichen.

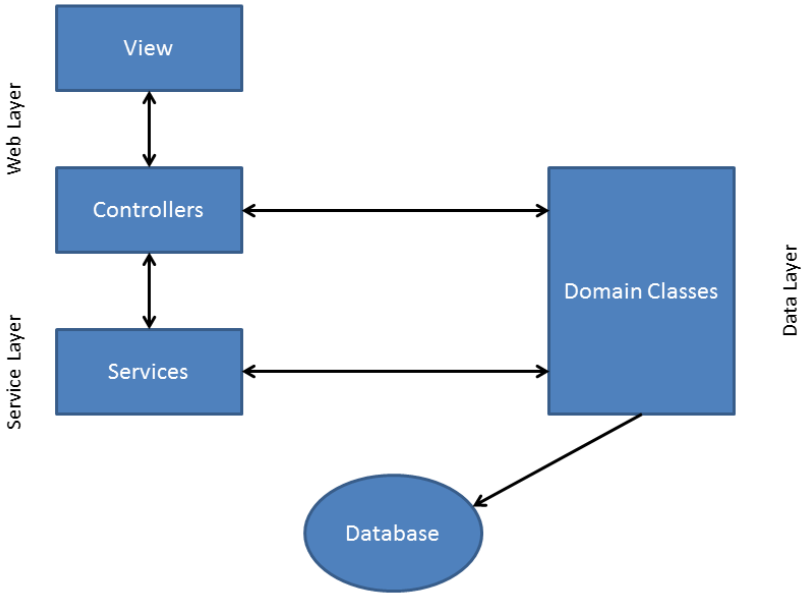

Abbildung 3: MVC-Architektur von Grails

In der Abbildung sowie auch im Framework selbst wird das Model durch Domain Klassen dargestellt. Diese Klassen sind persistent und werden von Grails automatisch einer jeweilig zugehörigen physischen Tabelle in einer Datenbank zugeordnet. Diese Zuordnung wird von der im Framework integrierten Technologie Hibernate übernommen. Des weiteren gibt es in Grails, wie auch in ADF die Möglichkeit Services wie zum Beispiel REST Web Services anzubinden. Über die Abbildung hinaus können zudem viele Funktionalitäten zusätzlich über Plugins hinzugefügt und verwendet werden.

In der folgenden Abbildung wird der von Grails verwendete Technologiestack dargestellt. Es ist dargestellt, dass Grails auf der Java Virtual Machine

(JVM) aufbaut und Grails durch die verwendete Programmiersprache Groovy, zudem die bekannte Java API und die entsprechenden zugehörigen Javadocs mit einem dynamischen Typsystem verbindet. Durch diese Verbindung ist damit sogar ein Mischen von statischen und dynamischen Typen möglich ist. In der Abbildung ist dies daran zu erkennen, dass sowohl Java als

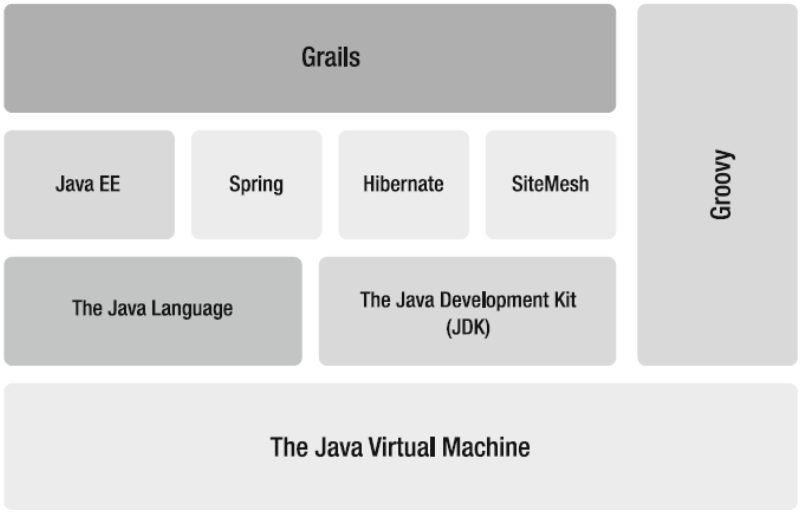

Abbildung 4: Technologiestack von Grails

Sprache, als auch das Java Development Kit (JDK) Teil des Grails Technologiestacks sind. Zusätzlich zu den allgemeinen Java Eigenschaften gehören zum Technologiestack aber auch:

- Hibernate: Als Standard für objektrelationales Mapping (ORM)
- Spring: das große und beliebte Open Source Inversion of Control Kontainer und Wrapper Framework für Java
- SiteMesh: ein robustes und stabiles Layout Rendering Framework
- Jetty: ein embeddable Servlet Kontainer
- HSQLDB: Ein reines Java relationales Datenbank Management System (RDBMS)

3 Gegenüberstellung der Frameworks

3.1 Anforderungen an Web Application Frameworks

Dieses Kapitel soll die wesentlichen Anforderungen an Web Application Frameworks zusammenfassen und eine Begründung enthalten, weshalb diese Anforderungen relavant sind. Es wird benötigt, um in den Folgenden Kapiteln die Vor und Nachteile der beiden Frameworks vergleichbar darstellen zu können.

Lizenz und Lizenzkosten

Da es sowohl Open Source als auch lizensierte Frameworks zur Entwicklung von Webanwendungen gibt, können sich die eventuell vorhandenen Lizenzkosten auf die Gesamtkosten des Projektes auswirken. Zudem müssen anfallende Lizenzkosten zum Betreiben der entwickelten Software bei der Auswahl eines Frameworks mit betrachtet werden. Ein weiterer zu beachtender Punkt bezüglich der Lizenzen ist die GNU General Public License (GNU GPL), denn sobald Teile des Quellcodes einer Software die unter diese Lizenz fällt in der eigenen Anwendung verwendet werden kann die Notwendigkeit bestehen, dass die eigene Anwendung ebenfalls unter diese Lizenz fallen muss. Daher ist insbesondere bei der Verwendung von GNU GPL Erweiterungen zu prüfen ob die Anwenwendung bei Verwendung ebenfalls unter die GNU GPL fällt.

Entwicklungsgeschwindigkeit

Die möglische Geschwindigkeit mit der mit einem Framework entwickelt werden kann ist sehr entscheident bei der Wahl des Frameworks, da die Entwicklungszeit und damit auch die Kosten für den Kunden möglichst gering gehalten werden sollen. Aus diesem Grund sollte auf diesen Aspekt geachtet werden und heirbei auch der folgende Punkt, die Lernkurve für ein eventuell neues Framework nicht außer acht gelassen werden.

Lernkurve

Da Web Application Frameworks nicht alle gleich aufgebaut sind oder ähnlich umfangreich sind, kann auch die Einarbeitungszeit für das entsprechende Framework unterschiedlich lang ausfallen. Da dies die Entwicklungszeit für die entsprechende Software verlängern oder verkürzen kann, gilt es die Lernkurve bei Auswahl des Frameworks zu beachten.

Community

Für die Entwicklung mit einem entsprechenden Framework kann die vorhandene Community ein essentieller Bestandteil sein, denn immer wieder können bei der Entwicklung Fehler oder Probleme auftreten, deren Lösung viel Zeit in Anspruch nehmen kann. In einer großen Community kann das Problem aber bereits von einem anderen Entwickler gelöst und festgehalten worden sein, was einiges an Zeit einsparen kann.

Testbarkeit

Die Testbarkeit von Software ist allgemein sehr wichtig, da sie zum einen komplex ist und sich zum anderen häufig ändern kann. Auf Grund dieser beiden Aspekte, ist es möglich, dass durch Änderungen Fehler entstehen können die in der komplexen Software vielleicht zunächst nicht auffallen. Eine gute Testabdeckung kann verhindern, dass solche Fehler übersehen werden.

Umfang und Qualität der Dokumentation

Eine Umfangreiche und verständliche Dokumentation ist sehr wichtig um mit dem Framework arbeiten zu können. Eine schlechte Dokumentation führt schnell zu Verwirrung und Frustration und verlangsamt damit auch die Entwicklungsgeschwindigkeit. Eine Umfangreiche und mit Beispielen versehene Dokumentation ist daher wichtig für die Auswahl des Frameworks.

Modularität

Da es sehr viele verschiedene Verwendungsmöglichkeiten für Web Anwendungen mit sehr unterschiedlichen Funktions Anforderungen gibt, ist es häufig sinnvoll nicht alle möglichen Funktionalitäten im Framework zu integrieren, sondern diese durch Erweiterungen (im nächsten Punkte näher erläutert) verfügbar zu machen. Ein leichtgewichtigeres Framework ist für kleine Webanwendungen daher besser geeignet, da weniger nicht benötigte Funktionen enthalten sind und den Umfang des Frameworks nicht unnötig vergrößern.

Erweiterbarkeit

Bei der Entwicklung einer Web Anwendung ist es gut Möglich, dass Funktionen verwendet werden sollen, die für sehr viele Web Anwendungen die selbe ist (z.B. das Login). Daher ist die Erweiterbarkeit des Frameworks und die Möglichkeit der Verwendung von Plugins für Web Application Frameworks ein wesentlicher Aspekt.

Versionierbarkeit

Da Software und damit auch Webanwendungen häufig im Team entwickelt werden, muss eine Möglichkeit zur Versionierung der zu entwickelnden Anwendung gegeben sein. Hierbei ist es wünschenswert, dass beim Zusammenführen der verschiedenen Entwicklungsstände möglichst wenig Konflikte auftreten.

Langlebigkeit

Die Langlebigkeit eines Frameworks ist insbesondere im Bezug auf den zugehörigen Support relevant, da dieser mit dem Aussterben des Frameworks ebenfalls weg fällt. Es ist also für eine Anwendung die für eine lange Dauer zum Einsatz kommen soll zu beachten, dass auch das Framework für diese Dauer warscheinlich weiterhin unterstützt wird.

Performanz

Die Performanz von Web Application Frameworks und den daraus resultierenden Anwendungen ist ebenfalls ein wesentlicher Punkt bei der Auswahl eines Frameworks, jedoch wird dieser hier nicht genauer betrachtet, da es schwer ist sinnvoll vergleichbare Werte für die Performanz zweier Frameworks zu finden.

3.2 Vor- und Nachteile von ADF

Dieses Kapitel soll die Vor und Nachteile von ADF anhand des zuvor aufgestellten Anforderungskatalogs erläutern und herhorheben, inwiefern sich ADF von anderen Web Application Frameworks abhebt.

Da ADF eine proprietäre kommerzielle Software ist fallen bei Einsatz der, mit diesem Framework entwickelten Software Lizenzkosten an. Jedoch hat die Verwendung eines proprietären Frameworks den Vorteil, das der Quellcode in keinem Fall offen gelegt werden muss, was für einige Kunden essentiell ist. Der Umfang des Frameworks ADF lässt jedoch nicht als modular bezeichnen, da ADF ein kaum erweiterbares Framework ist und entsprechend den gesamten Umfang der möglichen Funktionen schon zu Beginn enthält. Dieser große Umfang des Frameworks wirkt sich wiederum auf die Lernkurve des Frameworks aus und bewirkt, dass das Framework insbesondere für Entwickler mit wenig Erfahrung im prozeduralen und 4GL-Hintergrund schwerer zu erlernen ist. Einen Überblick über das Framework ADF zu bekommen kann daher einiges an Zeit erfordern und wirkt sich damit ebenfalls auf die Entwicklungsgeschwindigkeit aus. Einem Entwickler, der bereits Erfahrung mit ADF gesammelt hat und damit vertraut ist, ist as sogenannte „Rapid Application Development“ möglich, das ADF durch seinen baukastenartigen Aufbau und die vielen Wizards ermöglicht. Dieses schnelle Entwicklungstempo wird zudem durch die Umfangreiche Dokumentation und den Support von Oracle selbst durch zahlreiche Tutorials für die unterschiedlichsten Problemszenarien unterstützt. Die zu ADF gehörige Community ist weltweit deutlich kleiner als die zum Framework Grails gehörige. Dies lässt sich anhand der Folgenden

Abbildung erkennen. Die Grafik zeigt den zeitlichen Verlauf des Interesses an

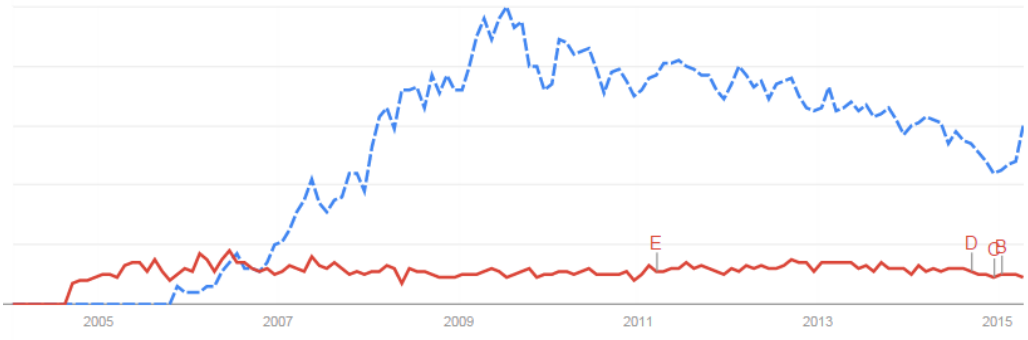

Abbildung 5: Zeitverlauf des Interesses an ADF und Grails weltweit (Quelle: `http://goo.gl/Ma9kWJ`)

den beiden Frameworks ADF (in rot) und Grails (in blau) weltweit anhand der Häufigkeit von Suchanfragen über Google. Das Suchinteresse wird hierbei relativ zum Höchstwert dargestellt. Diese Grafik lässt den Schluss zu, dass ADF zwar eine kleinere Community hat aber sich durchaus als Langlebig erwiesen hat, das das Interesse an ADF mehr oder weniger konstant bleibt. Die Langlebigkeit des Frameworks, die auch in Zukunft zu erwarten ist bietet den Vorteil, dass auch der Support weiterhin besteht und damit auch spätere Wartungsaufgaben erfüllbar sind. Auch die im Anforderungskatalog aufgeführten Punkte Versionierbarkeit und Testbarkeit erfüllt ADF, da es sich zum einen mit den bekannten Versionierungstools GIT und SVN versionieren lässt und damit auch die Arbeit in größeren Teams zulässt, zum anderen aber auch beispielsweise Unit Tests möglich sind um z.B. selbst erstellte Java Klassen und Methoden testen zu können. Bei der Versionierbarkeit gilt es jedoch zu beachten, dass nicht jeder Schritt für andere Entwickler Nachvollziehbar bleiben, da ADF viele Wizards zur Erstellung von Funktionen, Klassen und Dateien bietet.

3.3 Vor- und Nachteile von Grails

Dieses Kapitel soll die Vor und Nachteile von Grails anhand des zuvor aufgestellten Anforderungskatalogs erläutern und herhorheben, inwiefern sich

Grails von anderen Web Application Frameworks abhebt.

Grails ist ein modulares Open Source Framework, dass unter die Apache License in der Version 2.0 fällt. Diese Lizenz hat zunächst keine Auswirkungen auf die zu entwickelnde Anwendung, da im allgemeinen diese Lizenz nur eine Auswirkung hätte, wenn das Framework Teile des eigenen Source Codes in das Programm kopieren würde. Dies kann aber bei Grails höchstens bei Plugins (den Zahlreichen erweiterungsmöglichkeiten von Grails) vorkommen, die wiederum alle ihre eigenen Lizenzen haben. Dies hat wiederum den Nachteil, dass man für jede der vielen möglichen Erweiterungen des Frameworks, die man verwenden möchte zunächst überprüfen muss ob die Lizenz Auswirkungen auf die zu entwickelnde Software hat. Diese Überprüfungen können wenn sie notwendig sind, negative Auswirkungen auf die Entwicklungsgeschwindigkeit haben. Ohne diese Überprüfungen sind Entwicklungsgeschwindigkeit und Lernkurve für Entwickler mit Java Vorkenntnissen sehr gut, da diese Grails schnell lernen können und entsprechend schnell entwickeln können. Grails bietet allerdings weniger Wizards und ist nicht wir ADF nach dem Baukasten Prinzip für Rapid Application Development ausgelegt. Es muss also der meiste Code selbst geschrieben werden. Durch sehr viel selbst geschriebenen Code wird zum einen eine umfangreiche Testabdeckung benötigt, zum anderen aber auch eine gute Doku und Commmunity. Diese Punkte sind gegeben, da Grails die Java basierte Programmmiersprache Groovy verwendet und damit zum einen die gleichen Testmöglichkeiten bestehen, zum anderen aber auch die Javadocs als Dokumentation gültig sind, welche sehr umfangreich sind. Zusätzlich zu der Java Doku hat Grails aber auch eine eigene umfangreiche Dokumentation und eine große Community, die in entsprechenden Foren Frage beantwortet und Beispiele für verschiedene Problemstellungen zu Verfügung stellt. Wie auch für ADF lässt sich dir Größe der Community etwa am Zeitverlauf des Interesses an den beiden Frameworks und an den Einträgen in dem wohl größten genutzten Portal „Stackoverflow“[1] ablesen. Was die Langlebigkeit von Grails betrifft so ist das Interesse an Grails weniger konstant, jedoch ist es hoch genug um davon aus-

[1] http://stackoverflow.com/questions/tagged/grails

gehen zu können, dass Grails noch einige Zeit bestehen wird. Ein Weiterer für Webapplication Frameworks relevanter Punkt ist die Versionierbarkeit. Das Versionieren der zu entwickelnden Software via z.B. SVN oder GIT ist problemlos möglich. Hierbei hat Grails jedoch den Vorteil, dass durch den vielen eigenen Code und weniger Wizards die ausgeführten Schritte später leichter nachzuvollziehen sind, was für Entwicklung in große Teams ein großer Vorteil ist.

4 Projektspezifische Gegenüberstellung

4.1 „AvaleNet“

Dieses Kapitel soll die zu erstellende Anwendung „AvaleNet“ grundlegend beschreiben um einen Einblick zu erhalten, welche Aspekte der beiden Frameworks ADF und Grails für die Anwendung relevant sein können.

4.1.1 Projekthintergrund

Die zu migrierende Anwendung „AvaleNet“ ist die Web Anwendung eines Bankhauses zur Verwaltung Bürgschaften. Dies beinhaltet unter anderem die Verwaltung von Kunden-, Konto- und Avaldaten. Die Anwendung „AvaleNet“ wurde vor sechs Jahren in einer heute veralteten Version (Version 10) des Frameworks ADF entwickelt. Da dies eine heute schlechte Performanz der Anwendung und eine erschwerte Wartbarkeit zur Folge hat, soll die Anwendung nun migriert werden. Hierfür stehen die beiden Frameworks ADF und Grails zur Auswahl.

4.1.2 Projektanforderungen

Die Anwendung „AvaleNet“ dient zur Verwaltung von Bürgschaften, hierbei müssen Kundendaten mit zugehörigen Konten sowie Daten zu den Bürgschaften selbst gespeichert werden. Diese Daten werden in einer relationalen Datenbank abgelegt, auf die die Anwendung „AvaleNet“ sowohl lesend als auch schreibend zugreifen kann. Die Avale, sowie die Konten der Kunden sollen über Listen anzeigbar sein, wobei ein Wechsel zu einer Detailansicht eines Kontos oder eines Avals möglich ist. Außerdem ermöglicht es die Anwendung einem Mitarbeiter des Bankhauses bereits bestehende Daten von Kunden, Konten oder Avalen zubearbeiten und es stehen ihm einige Export-Funktionen zur Verfügung. Er kann beispielsweise eine Liste aller, zu einem Kunden gehörigen Avale in ein PDF-Dokument exportieren, oder die Einzeldaten pro Aval auf diese Weise auflisten lassen. Zudem sollen alle Änderungen in der Anwendung historisiert werden um auch über bereits getilgte Avale

Bescheid zu wissen und nachhalten zu können wer zuletzt welche Änderung vorgenommen hat.

4.2 Vor- und Nachteile der Frameworks im Bezug auf „AvaleNet“

Dieses Kapitel soll die im vorigen Kapitel erläuterten Vor- und Nachteile der beiden Frameworks Grails und ADF im Bezug auf die zu entwickelnde Anwendung „AvaleNet“ erläutern.

Der zunächst wichtigste Aspekt bei der Auswahl eines Web Applikation Frameworks ist, dass das entsprechende Framework alle Funktionalitäten enthalten muss, die für die Umsetzung des geplanten Projektes notwendig sind. Dabei müssen die Funktionalitäten nicht von vornherein im Grundumfang des Frameworks vorhanden sein, sondern sie können auch durch Erweiterungen des Frameworks später hinzuzufügen sein. Für „AvaleNet“ erfüllen beide Frameworks diesen Punkt, da die Anwendung mit ihren Funktionen nur wenig über die Standard Funktionen einer Web Anwendung hinaus geht. ADF beinhaltet diese Funktionen schon im Grundumfang des Frameworks und für Grails lassen sich die zusätzlich benötigten Funktionen über Erweiterungen (Plugins) hinzufügen. Zudem hat ADF bezüglich der zu entwickelnden An-

Verwendetes Plugin	Lizenz
Spring Security Core Plugin	Apache License 2.0
Hibernate Grails Plugin	Apache License 2.0
Jquery Grails Plugin	Apache License 2.0
Rendering Grails Plugin	Apache License 2.0
Tomcat Grails Plugin	Apache License 2.0

Tabelle 1: Verwendete Plugins und zugehörige Lizenzen (Quelle: `https://grails.org/plugins/`)

wendung „AvaleNet“ den Vorteil, dass es sich um ein proprietäres Framework handelt und damit keinesfalls der Quellcode der Anwendung offengelegt werden muss. Dies ist gerade bei einer Anwendung für ein Bankhaus relevant,

die mit sensiblen Kundendaten agiert. Allerdings ist auch die Verwendung von Grails hier nicht kritisch, da kaum zusätzliche Plugins verwendet werden und diese keine kritischen Lizenzen enthalten (siehe Tabelle 1), die den Entwickler zwingen die zu entwickelnde Software ebenfalls zu einer Open Source Software zu machen. Einzig das Rendering Plugin, das selbst auch unter die Apache Lizense 2.0[2] fällt verwendet Bibilotheken, die unter die GNU Lesser General Public License[3] (LGPL) fallen, welche ebenfalls kein starkes Copyleft beinhaltet und den Nutzer so ebenfalls nicht zwingt den Quellcode offen zu legen. Der Grund für die geringe Notwendigkeit von Plugins ist, dass „AvaleNet“ eine kleine Anwendung ist, die keine umfangreiche Funktionalität erfordert. Dies hat ebenfalls zur Folge, dass die Entwicklung der neuen Anwendung kein großes Team erfordert, sondern von einer Person mit geringem Aufwand (weniger als ein halbes Jahr) umgesetzt werden kann. Da die mit der Entwicklung betraute Person zudem hauptsächlich Kenntnisse um Java Umfeld und keine Erfahrung mit ADF hat spielt die Lernkurve hier für die gesamte Entwicklungsdauer inklusive Einarbeitungszeit eine große Rolle. Hier hat Grails den Vorteil, dass es mit Java Grundkenntnissen schneller zu erlernen ist als ADF ohne Vorkenntnisse. Ein weiterer Nachteil von ADF bezüglich „AvaleNet“ ist nicht vorhandene Modularität und der damit verbundene Überfluss an Funktionen, die das Framework bereitstellt, welche aber für eine solche kleine Anwendung nicht benötigt werden und es dem Entwickler erschweren das Framework schneller zu erlernen. Unabhängig von dem im vorigen Kapitel aufgestellten Anforderungskatalog hat ADF jedoch den Vorteil, dass die zu migrierende Anwendung bereits in ADF geschrieben wurde und ein erfahrener Entwickler die Anwendung daher mithilfe der alten Anwendung deutlich schneller in der neuen Version von ADF (12.0.3) nachbauen könnte. Zudem ist ein Großteil der benötigten Lizenzen für die Verwendung einer ADF Anwendung zusammen mit einer Oracle Datenbak bereits vorhanden und es würde bei Verwendung der neuen Version von ADF damit keine großen Mehrkosten anfallen. Allerdings gibt es in dem mit der Wartung und Migration betrauten Unternehmen (OPITZ CONSUL-

[2] http://www.apache.org/licenses/LICENSE-2.0.html
[3] https://www.gnu.org/licenses/lgpl.html

Abbildung 6: Zeitverlauf des Interesses an ADF und Grails in Deutschland (Quelle: `http://goo.gl/Ma9kWJ`)

TING Deutschland GmbH) nur wenige Mitarbeiter mit umfangreichen ADF Kenntnissen dafür aber einige mit guten Java und Grailskenntnissen, weshalb Grails für diese Situation von Vorteil wäre. Diese Verteilung des Interesses ist nicht unüblich, wie die Grafik zum Interesse an ADF und Grails weltweit bereits gezeigt hat und auch für Deutschland zeigt diese Verteilung, dass das Interesse an ADF deutlich geringer ist als das an Grails. Dieses stärkere Interesse an dem Framework Grails hat den Vorteil, dass in einem Forum, wie dem viel genutzten Forum „Stackoverflow“ deutlich mehr Einträge zu Grails zu finden sind als zu ADF (Grails: 21183, ADF: 1024)[4].

4.3 Aufwiegen der Vor- und Nachteile

Dieses Kapitel gewichtet die Vor und Nachteile der beiden Frameworks und stellt sie vergleichend gegenüber, um letztendlich zu entscheiden, welches der beiden Frameworks sich besser zur Entwicklung der Anwendung „AvaleNet“ eignet.

Von den in den vorigen Kapiteln aufgeführten Kriterien zur Auswahl eines Web Application Frameworks sind insbesondere die Kriterien Lizenz und Lizenzkosten, Entwicklungsgeschwindigkeit, Lernkurve, Umfang und Qualität der Dokumentation und Modularität von großer Bedeutung für die Anwen-

[4] http://stackoverflow.com/tags

dung „AvaleNet“. Zu begründen ist dies damit, dass die Kriterien insbesondere die Kosten für das Projekt und den Arbeitsaufwand gering halten, denn ein modulares Framework verkürzt die Einarbeitungszeit und damit die Entwicklungsdauer. Dies wird ebenfalls durch eine gute Dokumentation gestützt. Hier bietet Grails im Gegensatz zu ADF den Vorteil, dass es keine Lizenzkosten hat und auch den Vorteil von ADF, ein proprietäres Framework zu sein und damit die Offenlegung des Quellcodes nicht zu erzwingen, wiegt Grails damit auf, dass die Lizenzen der verwendeten Plugins keinerlei Probleme für die Geheimhaltung des Quellcodes von „AvaleNet“ bringen. Zudem ist die Lernkurve von Grails deutlich besser als die von ADF, da Grails deutlich modularer aufgebaut ist. Die Entwicklungsgeschwindigkeit ist der einzige Punkt in dieser Auflistung, in dem ADF besser geeignet ist als Grails, da ADF das sogenannte „Rapid Application Development“ ermöglicht. Jedoch ist die steht die geförderte Entwicklungsgeschwindigkeit in keinem Verhältnis zur benötigten Einarbeitungszeit, weshalb dies für ein so kurzes Projekt kaum eine spürbare Auswirkung hätte. Für größere Projekte kann die gesteigerte Entwicklungsgeschwindigkeit allerding durchaus ein Grund sein ADF Grails vorzuziehen.

Weniger wichtig aber ebenfalls von Bedeutung sind die Kriterien Community, Testbarkeit, Erweiterbarkeit und Fähigkeiten im Unternehmen, weil sie ein möglichst reibungsfreies Etwickeln ermöglichen und sicherstellen, dass die Anwendung später möglichst fehlerfrei funktionieren kann. Auch hier bietet Grails mehr Vorteile als ADF, da Grails zum einen eine deutlich größere Community und damit schnellere Hilfe bei auftretenden Problemen bietet und zum anderen eine große Anzahl an Erweiterungen zur Verfügung stellt und es ermöglicht eigene Erweiterungen zu schreiben. Diese Anzahl möglicher Erweiterungen kann jedoch für andere Projekte, die weniger Standard Plugins verwenden zum Nachteil werden, da es Zeit kostet die richtige Erweiterung mit einer zu den Anforderungen des Projektes passenden Lizenz zu finden und es sich teilweise nicht von vornherein sagen lässt, ob sich ein Plugin tatsächlich für ein Vorhaben eignet.

Für „AvaleNet“ eher unwichtig sind Versionierbarkeit und Langlebigkeit, da für eine einzelne Person Versionierung zwar sinnvoll ist um immer einen

lauffähigen Stand der Anwendung zu haben und widerherstellen zu können. Dies lässt sich jedoch auch über regelmäßige Sicherungen des Projektes realisieren und erfordert nicht zwingend ein Versionierungstool. Die Langlebigkeit des Frameworks wird hier als weniger wichtig eingestuft, da das Framework primär zur Entwicklung der Anwendung bewertet wird und eine weitere Unterstützung des Frameworks hauptsächlich Auswirkungen auf den späteren Support bei Wartung der Anwendung hat. Im Bezug auf die Langlebigkeit hat das proprietäre Framework ADF allerdings den klaren Vorteil, dass ein Framework, dass Lizenzkosten erfordert auch für eine gewisse Dauer den Support und das bestehen des Frameworks und der zugehörigen Komponenten garantiert, wohingegen das bestehen von Grails sehr start von der Popularität des Frameworks abhängt.

5 Fazit

Dieses Kapitel fasst noch einmal zusammen, weshalb das Framework gewählt wurde und welches nach der Gegenüberstellung besser zur Entwicklung geeignet ist.

Literatur

[Fitzgerald u. a.] Fitzgerald, B. ; Russo, N. ; DeGross, J.