entwickler.press

Flash Builder 4 und Java

Kickstart

Florian Müller

Florian Müller

Flash Builder 4 und Java

Kickstart

entwickler.press

Florian Müller
Flash Builder 4 und Java

ISBN: 978-3-86802-047-2

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Ihr Kontakt zum Verlag und Lektorat:
Software & Support Verlag GmbH
entwickler.press
Geleitsstraße 14
60599 Frankfurt am Main
Tel: +49(0) 69 63 00 89 - 0
Fax: +49(0) 69 63 00 89 - 89
lektorat@entwickler-press.de
http://www.entwickler-press.de

Projektleitung: Sebastian Burkart
Korrektorat: Ella Klassen, Katharina Klassen
Layout: SatzWERK, Siegen (www.satz-werk.com)
Umschlaggestaltung: Maria Rudi
Belichtung, Druck & Bindung:
M.P. Media-Print Informationstechnologie GmbH, Paderborn

Inhaltsverzeichnis

V	**Vorwort**	**9**
D	**Danksagung**	**11**
1	**Einleitung**	**13**
1.1	User-Interface-Technologien	13
1.1.1	Geschichtsunterricht: Benutzeroberflächen	16
1.1.2	Java und GUI-Technologien	21
1.2	Flex und Flash Builder 4	23
1.2.1	Begriffsdschungel: Flex, Flash, Flash Player, Flex Builder...	24
1.2.2	Hintergrundinformationen BlazeDS	26
1.2.3	BlazeDS technisch	29
1.2.4	Flash-Builder-4-Konzepte	32
1.2.5	Flex Builder 3 versus Flash Builder 4	36
1.2.6	Kosten: Flash Builder, Flash Catalyst, BlazeDS	38
1.2.7	Adobe-Designer-Werkzeuge	40
1.3	„User Guide“ für das Buch	41
1.3.1	Vorgehensweise	42
1.3.2	Fahrplan	42
2	**Installation**	**45**
2.1	Eclipse-Installation	46
2.2	Apache Tomcat installieren	49
2.3	Flash-Builder-Plug-in installieren	52
2.4	Flash Catalyst installieren	55
2.5	Flash-Player-Installation	57
2.6	Wrap Up	58
3	**Flash Builder 4 – Ramp up!**	**59**
3.1	Neues Projekt erstellen	60
3.1.1	Exkurs: Flex Halo und Flex Spark	62

3.2 Oberflächenerstellung 70
3.3 Chefkoch-Clientlogik 75
3.4 Pimp my Style! 80
3.5 Wrap up und Ausblick 84

4 Server Binding 85
4.1 BlazeDS-Setup 85
4.2 Chefkoch Deluxe – jetzt mit Server Binding! 87
4.3 Hallo Server 91
4.4 Hello Client! 96
4.5 Serverrezepte 100
4.5.1 Exkurs zum Affenhaus: Monkey-Skripte 104
4.6 Von Java nach ActionScript 106
4.6.1 Debugging 110
4.7 Wrap up 113

5 Photoshop-Basics 115
5.1 Amadeus-Projektbeschreibung 116
5.2 Photoshop-Basics 118
5.3 Photoshop: Layer und Gruppierungen 119
5.4 Heavy Metal mit Amadeus 123
5.4.1 Blending Options 126
5.5 Oberflächenstruktur 128
5.6 Tabellenerstellung 129
5.6.1 Layer als Grundlage für Hover-Effekte 132
5.7 Felder und Buttons 132
5.8 Vorbereitungen für States 135
5.9 Wrap up 138

6 Flash Catalyst 139
6.1 Los geht's! 140
6.2 Felder zuweisen 142
6.3 Buttontransformation 145

6.4	Tabellen und Listen	150
	6.4.1 Tabellen-Hover-Effekt	154
	6.4.2 Transitions: Darf's ein bisschen weicher sein?	156
6.5	Navigation via States	157
6.6	Wrap up	161
7	**Java-Integration**	**163**
7.1	Projekt-Setup	164
7.2	Mitarbeiterverwaltung-Backend	165
7.3	Java-Flex-Konfiguration	171
7.4	Java- und Flex-Hochzeit	173
7.5	Feinschliff	187
	7.5.1 Mitarbeiterauswahl und Änderungen	190
	7.5.2 Mitarbeiter hinzufügen	192
	7.5.3 Bilder verwenden	193
	7.5.4 Mitarbeiter-Tasks	198
7.6	Let's network	203
7.7	Wrap up	204
8	**360-Review: Flash Builder 4**	**205**
8.1	Sackgasse!	205
8.2	„Größenwahnsinn"	208
8.3	Grafik, Grafik, Grafik...	209
8.4	Codelesbarkeit	211
	8.4.1 Exkurs: Kommentare	213
8.5	Modularität	216
8.6	Wrap up	218
9	**Unit Testing**	**219**
9.1	Taschenrechnerlogik	221
9.2	Logikanbindung	231
9.3	Wrap up	234
10	**Und wie geht es weiter?**	**235**

11	**Gute Frage!**	**239**
11.1	Was ist das Flex SDK?	239
11.2	Flash-Builder-Komplett-IDE oder Eclipse-Plug-in?	241
11.3	Flash, Silverlight, Java oder doch HTML5?	241
11.4	Flash? Flex? Gibt es einen Unterschied?	243
11.5	Kosten für Flash Builder und Catalyst	244
11.6	Wird ein besonderer Server für Flex-Java-Projekte benötigt?	244
11.7	Ist der Einsatz von BlazeDS auch in kommerziellen Projekten kostenlos?	244
11.8	Müssen beim Einsatz von BlazeDS bestimmte Server-Ports geöffnet werden?	245
11.9	Was benötigt ein Anwender, um eine Flex-Anwendung auszuführen?	246
11.10	Wie hoch ist die Verbreitung des Flash-Player-Plug-ins?	246
11.11	Was benötigt ein Entwickler, um eine Flex-Anwendung zu erstellen?	248
11.12	Was ist LCDS?	248
11.13	Was ist ColdFusion?	249
11.14	Was ist der Unterschied zwischen .swf-und .swc-Dateien?	250
11.15	Flex und MVC?	250
11.16	Welche Skills sind in einem Team für ein großes Flex-Java-Projekt erforderlich?	252
11.17	Was ist das Endprodukt einer Flex-Java-Anwendung, was „landet“ auf dem Server?	253
11.18	Wann wird zur „Ein-Projekt-Strategie“ gegriffen, wann sollten zwei Projekte angelegt werden?	253
11.19	Stateful oder Stateless Flex?	254
11.20	Wird Mehrsprachigkeit in Flex-Anwendungen unterstützt?	255
11.21	Gibt es Foren bzw. eine Community rund um Flex?	255
	Stichwortverzeichnis	**257**

Vorwort

Man kann niemanden überholen, wenn man in seine Fußstapfen tritt.

(François Truffaut, franz. Regisseur und Schauspieler)

Das Schreiben von Büchern ist wie das Bestreiten eines Triathlons. Überquert man bei einer solchen Veranstaltung die Ziellinie, beschließt man häufig, nie wieder an einer derartigen Quälerei teilzunehmen und die Wochenenden zukünftig lieber gemütlicher anzugehen, statt freiwillig eine solche Tortur auf sich zu nehmen. Dann dauert es meistens genau zwei Wochen und es „juckt" wieder; man schwingt sich auf das Rad, um ein paar lockere Runden zu fahren, und schon ist man mitten in der Vorbereitung für den nächsten Wettkampf.

Als ich vor zwei Jahren das Manuskript für das letzte Flex-Java-Buch abgegeben habe, stand ein ähnlicher Vorsatz im Raum: Word sollte die nächsten fünf Jahre nur noch für Artikel und Rechnungen genutzt werden, das Thema Buch war zum damaligen Zeitpunkt ausgereizt. Allerdings geriet dieser Vorsatz ins Wanken, als die erste Betaversion von Flash Builder 4 veröffentlicht wurde. Die darin umgesetzten Konzepte ließen schnell erkennen, dass es sich um mehr als „nur" eine neue Version handelt.

Schon zu Flex-3-Zeiten scheiterte die Akzeptanz von Flex im Java-Umfeld häufig daran, dass es schlicht und einfach an guter Dokumentation für Java-Entwickler mangelte. Es ist verhältnismäßig einfach, im Internet eine Quelle zu finden, die einen netten Flex-Java-Showcase präsentiert, der jedoch im realen Projekteinsatz schnell an seine Grenzen stößt. Entsprechend resultierte daraus die Motivation, ein Werk zu erstellen, dass Flex für einen Java-Entwickler realistisch darstellt, mit sämtlichen Höhen und Tiefen des Frameworks.

Exakt die gleiche Motivation liegt dem aktuellen Werk zugrunde. Einerseits werden technische Konzepte und HowTos vermittelt, andererseits wird vom Standard-Showcase-Abstand genommen und realistisch gezeigt, was geht und was nicht geht. Flash Builder 4 bietet Java-Entwicklern die Möglichkeit, mit einem revolutionären Ansatz in die Entwicklung von Rich

Internet Applications einzusteigen, dabei ist es jedoch wichtig, die Marketingkonzepte und Showcases, die im Internet kursieren, links liegen zu lassen und eigenständig zu entwickeln. Entsprechend passt das einleitende Zitat „Man kann niemanden überholen, wenn man in seine Fußstapfen tritt...“ besonders gut, auch für Sie heißt es nach dem Durcharbeiten des Buchs, eigenständige Konzepte und Architekturen zu entwickeln, nur so differenzieren Sie sich von der breiten Masse der 0815-Flex-Anwendungen mit Performanceengpässen und monolythischen Skriptgrundlagen.

Der Focus des Buchs liegt primär auf der Vermittlung von Konzepten, entsprechend kann leider nicht jedes Listing „auseinandergepflückt“ und detailliert erläutert werden – deshalb sind Sie an dieser Stelle gefordert, indem Sie Flex-/Java-Wissenslücken selbstständig schließen. Ich versichere Ihnen jedoch: Falls Sie einmal an einer Stelle hängen sollten, da Ihnen die eingesetzte Flex-Komponente o. Ä. nicht bekannt ist, kann diese problemlos in Eigenrecherche erarbeitet werden – eine Erklärung bzw. Dokumentation einer Flex-Komponente ist im Web in Kürze gefunden, eine kompakte realistische Darstellung von Flash-Builder-4-Konzepten und -Beispielen kann leider nicht so leicht gefunden werden, aber genau dafür halten Sie dieses Buch in den Händen!

Ich wünsche Ihnen viel Erfolg auf der Überholspur, den Blinker haben Sie durch den Kauf des Buchs bereits gesetzt, Ihr Part ist es jetzt, ordentlich Gas zu geben! Ich wünsche Ihnen nicht nur viel Erfolg, sondern auch viel Spaß auf der Flash-Builder-4-Überholspur, Sie werden sehen: Mit dem Flash-Builder-4-Cabrio macht das Überholen richtig viel Spaß, eine Geschwindigkeitsbegrenzung gibt es nicht, es liegt also ganz bei Ihnen, wie viel Sie aus Flash Builder 4 für sich und Ihr Projekt herausholen!

Florian Müller

Danksagung

Momentan decken sich die Adobe-Flex-Releasezyklen mehr oder weniger mit den Geburtsterminen unserer Kinder, da ich solche Geburtstermine gerne als terminliche Meilensteine verwende, bedanke ich mich bei unserer noch ungeborenen zweiten Tochter Hannah für die Definition eines solchen Meilensteins!

Auch möchte ich mich bei meiner Frau Kathrin dafür bedanken, dass mir während des gesamten Projekts der Rücken frei gehalten wurde, und versichern, dass ich nicht anstrebe für jeden weiteren Flex-Release einen zugehörigen familiären Meilenstein zu definieren!

Ein großes Dankeschön geht auch in den hohen Norden nach Oldenburg an Lars Röwekamp, der das Buckprojekt als Lektor bzw. „Betaleser" begleitet hat, ich freue mich, in Lars einen verlässlichen UI-Verbündeten und Partner gefunden zu haben, auf den man bei gemeinsamen Workshops, Codecamps etc. ganz einfach zählen kann!

Des Weiteren möchte ich dem Software & Support Verlag meinen Dank aussprechen, angefangen bei einem kleinen Artikel vor sechs Jahren über das damals aktuelle Casabac Framework hat sich aus diesem Artikel in den vergangenen Jahren eine tolle Partnerschaft entwickelt, durch die es richability möglich wurde, im UI-Umfeld die heutige Marktpräsenz zu erreichen. Die Arbeit mit dem Software & Support Verlag ist stets geprägt von Fairness, Professionalität und vor allen Dingen Spaß – dafür herzlichen Dank nach Frankfurt!

1 Einleitung

1.1 User-Interface-Technologien

Jeder Entwickler und Teamleiter kennt das Problem: Wenn es darum geht, eine Anwendung zu „pimpen“, indem das Backend mit Spring erweitert wird, so findet sich schnell ein Entwickler aus dem Team, der nach dieser Aufgabe lechzt – wenn es jedoch darum geht, die bestehende Ajax-Oberfläche durch eine neue Swing-Oberfläche zu ersetzen/erweitern, herrscht meistens betroffenes Schweigen, und jeder hofft, dass dieser Kelch an ihm vorüberzieht. Der einzige Entwickler, der mehr oder weniger Spaß (und vor allen Dingen Erfolg) mit dieser Knacknuss haben könnte, ist meistens derjenige, der die Oberfläche in der Vergangenheit selbst erstellt hat – und das ist ziemlich dünn!

Das Thema User-Interface-Technologien ist bei der Entwicklung von Software in der Regel immer ein unbeliebtes Thema. Dies liegt darin begründet, dass bereits der Auswahlprozess der Benutzeroberflächentechnologie nicht ganz trivial ist und es zwar gewisse Standards gibt (z. B. JSF), die Standards jedoch häufig nicht zu den Anforderungen passen. Wenn es beispielsweise darum geht, eine einfache Benutzeroberfläche für einen Webshop zu erstellen (natürlich mit einer Prise Web 2.0 Look and Feel), könnte JSF ein wenig oversized für diese Art von Anforderungen sein, ganz zu schweigen, dass die Anzahl verwendbarer Bibliotheken durch den Zusatz „Web 2.0 Look and Feel“ ganz ordentlich schrumpfen wird. Wurde dann endlich eine Technologie ausgewählt, muss das Binding zum Server möglichst einfach sein, sodass im Idealfall die Benutzeroberfläche mehr oder weniger autark entwickelt werden kann. In der Theorie eine nette Vision, in der Praxis meistens jedoch kaum vorhanden. In vielen Projekten ist das GUI stattdessen von Anfang an in die Anwendung eingebunden, an dieser Stelle gilt die alte Bauernweisheit:

„*Ist das GUI früh im Projekt drinnen, gibts für Festverdrahtung kein Entrinnen...*“
(Altgermanischer Bauernkalender oder so ähnlich...)

Unabhängig davon, dass diese Bauernregel natürlich frei erfunden ist, beinhaltet diese jedoch ein wahre Aussage: Benutzeroberflächen werden in der Regel extrem stark mit der Anwendungslogik verknüpft, die Vision von der Trennung zwischen Frontend- und Backend-Entwicklung beschränkt sich oftmals auf die Erstellung von Styles durch eine Agentur, an der Oberfläche selbst schraubt jedoch das Entwicklerteam ordentlich mit. Sätze wie „...ich habe mir mal einen Button auf dem GUI angelegt, um meinen Service zu testen und nehm' den nachher wieder runter..." dürften dann keine Seltenheit sein, und es kommt zu einem Verhaltensmuster ähnlich des altbekannten „System.out.println-Debugger-Entwickler". In der Theorie wird natürlich vehement bestritten, dass jemals ein Entwickler aus dem Team die *System.out.println*-Anweisung verwendet hat, in der Praxis können Sie jedoch sicher sein, den einen oder anderen Sittenstrolch auf frischer Tat zu ertappen und noch Spuren im Code zu finden.

Die eigentliche Entwicklung der Benutzeroberfläche stellt Entwickler dann häufig vor eine weitere Hürde: Nicht jeder Entwickler wurde mit den nötigen künstlerischen Genen ausgestattet (ich übrigens auch nicht), um eine gut aussehende, intuitive Benutzeroberfläche zu erstellen: Viele Entwickler kennen die Situation, man hat eine Nachtschicht eingelegt, um dem Chef am nächsten Tag eine perfekte Anwendungsoberfläche zu erstellen, und persönlich findet man, in der letzten Nacht ein absolutes Killer-GUI erstellt zu haben, das eigentlich schon den Stempel *Web 4.0 Certified* tragen sollte. Die Begeisterung verfliegt dann jedoch schnell im Rahmen der Vorstellung, und Worte wie „nicht intuitiv, grausam überladen, zu bunt, nicht zumutbar..." prägen die Vorstellung der nächtlichen Resultate. Fakt ist an dieser Stelle, dass der Einsatz eines UI-Frameworks noch kein Garant für eine gute Oberfläche ist – und je mehr ein Entwickler unterstützt wird, z. B. durch GUI Builder oder aber Komponenten, die Out of the box schon etwas hermachen, desto größer ist die Wahrscheinlichkeit, dass Situationen, wie die hier beschriebene, umschifft werden können. Wird dennoch ein Framework eingesetzt, dass den bekannten Google-Style (= no Style) unterstützt, muss sich der Entwickler die Frage stellen, ob es wirklich seine Kernaufgabe ist, GUIs aufzuhübschen oder ob vielleicht eine falsche Entscheidung hinsichtlich der GUI-Technologie gefällt wurde.

Wenn dann schließlich sämtliche, hier beschriebene Hürden genommen wurden, geht es an das Testen der Anwendung. Leider, so scheint es zumindest manchmal, hat man in der Euphorie der Web-2.0-Welle wichtige Bestandteile der Anwendungsentwicklung sträflich vernachlässigt, so z. B. das Thema User-Interface-Tests. Eine Technologie, wie beispielsweise JUnit, die sich im Backend zum De-facto-Standard etabliert hat, gibt es im GUI-Bereich nicht. Das Spektrum an Technologien, angefangen beim Thin Client bis hin zum Fat Client, ist zu breit, um diese mit einer einzigen Technologie unter einen Hut bringen zu können. Entsprechend zeigt sich an dieser Stelle dann oftmals, mit wie viel (oder eben wenig) Weitsicht die ursprüngliche UI-Technologieauswahl getroffen wurde: Im Idealfall findet man ein Framework, dass mit geringem Initialaufwand eingesetzt werden kann, um beispielsweise Unit-Tests in der Benutzeroberfläche ausführen zu können. Manchmal ist das jedoch nicht der Fall und nach zwei Wochen Setup des Testing Frameworks stellt man fest, dass dieses jetzt zwar irgendwie läuft – das Drücken des EXECUTE-Buttons ist jedoch immer mit einem extremen Nervenkitzel verbunden, da man nicht weiß, ob es vielleicht wieder frameworkbedingt „knallt", weil ein Eintrag in der 600-zeiligen XML-Konfiguration des Testing Frameworks geändert wurde.

Ganz zum Schluss kommt dann irgendwann noch das Thema Wartung zur Sprache, und auch hier finden sich gängige Verhaltensmuster aus der Praxis, die dem einen oder anderen Entwickler bekannt vorkommen dürften: Wenn nach einem Jahr Softwareentwicklung dann der Rollout der Anwendung in fröhlicher Runde gefeiert wird, hofft jeder Entwickler heimlich, dass diese Veranstaltung gleichzeitig der letzte Auftritt bei Kunde XYZ ist und es bereits nächste Woche mit einem neuen und viel spannenderen Projekt weitergeht. Nur der Entwickler, der die Oberfläche erstellt hat, wird das dumpfe Gefühl nicht los, dass er auf weiter Flur der Einzige ist, der in den kommenden Monaten die sicherlich nicht ausbleibenden Frontend-Änderungen umsetzen kann – und bereits einige Tage später hat genau dieser Entwickler eine E-Mail mit dem Betreff „Dringende Änderungen im Spesenmodul GUI..." in seiner Posteingangsbox.

Zugegebenermaßen: Das hier beschriebene Worst-Case-Szenario tritt in der Praxis (zum Glück) eher selten auf dass die einzelnen Punkte jedoch im Rahmen eines Projekts auftauchen, ist keine Seltenheit, und ich bin sicher, dass auch Sie bereits Opfer des einen oder anderen hier aufgeführten Klassikers geworden sind. Ziel des Buchs ist, Ihnen im Kontext der Adobe-Flex-Technologie zu demonstrieren, wie Sie die hier genannten Pain Points zukünftig vermeiden können, ohne dabei den Anspruch zu erheben, dass

Adobe Flex der heilige Gral der User-Interface-Technologien ist. Im Rahmen des ersten Kapitels wird eine kurze GUI-Standortpeilung durchgeführt à la „wo stehen wir momentan, welche Technologien gibt es etc....“. Es ist wichtig zu verstehen, was sich momentan auf dem Markt tut, nur so kann abgeschätzt werden, ob Adobe Flex auch für Sie eine Option ist oder eher „technisches Privatvergnügen“.

Flex, Flash, Flash Builder, Flex Builder... – innerhalb dieses Kapitels werden Sie bereits auf diese Begriffe stoßen, für Leser ohne Flex-/Flash-Hintergrundwissen wird empfohlen, Kapitel Begriffsdschungel: Flex, Flash, Flash Player, Flex Builder... an dieser Stelle vorzuziehen. Sollten Sie an dieser Stelle der Auffassung sein, dass Flex und Flash sowieso das Gleiche sind, empfehlen wir gleichfalls einen kurzen Abstecher zum Kapitel Begriffsdschungel: Flex, Flash, Flash Player, Flex Builder....

Darüber hinaus werden Sie konkret erfahren, was sich hinter Adobe Flex verbirgt, welche Toolpalette dahintersteht und vor allen Dingen warum Flash Builder 4 oftmals als revolutionäre GUI-Technologie bezeichnet wird. Nach dem ersten Kapitel werden Sie eine Vorstellung haben, was hinter Flex, Flash, Flash Builder und Co. steckt, damit Sie dann im zweiten Kapitel nach einer kurzen Installation richtig loslegen können. In diesem Sinne müssen Sie sich noch kurz gedulden, bevor wir richtig in die Tasten hauen, es ist jedoch wichtig, das „Big Picture“ im Kontext von GUI-Technologien zu kennen, nur so können solide und langfristig Technologieentscheidungen gefällt werden.

1.1.1 Geschichtsunterricht: Benutzeroberflächen

Betrachtet man das Thema GUI-Technologien als einen Baum, so war die Auswahl eines Pfads der für die eigenen bzw. projektspezifischen Anforderungen passt, zu Zeiten des Ajax-Hypes und auch zuvor vergleichsweise einfach. Bevor die Ajax-Lawine ins Rollen gebracht wurde, war die Frage ganz einfach, Thin Client à la „HTML-Darstellung mit wenig Interaktionsmöglichkeiten für den Benutzer“ oder Fat Client à la „hohe Interaktivität, muss aber lokal installiert werden“? Die Trennung war einfach, und wenn der Benutzer mit der Anwendung wirklich arbeiten sollte (klassisches Beispiel: Maske für einen Versicherungssachbearbeiter zur Erfassung von Scha-

densfällen), fiel die Wahl fast immer auf den Fat Client. Der Thin Client wurde lediglich eingesetzt, um Informationen darzustellen, z. B. um eine Übersicht der Schadensfälle des vergangenen Jahres anzuzeigen, ohne dass der Manager dabei tief in der Anwendung wühlen muss. Der zugehörige Entscheidungsbaum ist in Abbildung 1.1 dargestellt.

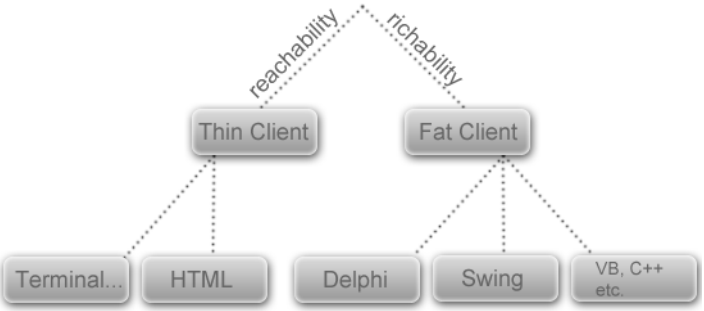

Abbildung 1.1: GUI-Baum vor der Ajax-Welle

Dank Ajax wurde dann jedoch im Browser schlagartig mehr möglich, und grundsätzlich konnte auf einmal die ganze Palette an GUI-Interaktionen, die man bisher lediglich aus der Fat-Client-Welt kannte, im Browser nachbilden. Entsprechend wurde der Entscheidungsbaum um einen weiteren Pfad bereichert, allerdings darf man im Nachhinein sagen, dass dieser Pfad geringfügig überbewertet wurde oder besser gesagt: Der Fat-Client-Pfad wurde in dieser Zeit sträflich vernachlässigt! Ursache für die stiefmütterliche Behandlung des Fat-Client-Pfads war der Denkansatz, dass man rein optisch betrachtet im Browser alles machen konnte, was im Fat Client auch möglich war – mit dem netten Vorteil, dass jeder auf die Anwendung zugreifen konnte und das Thema Rollout komplett von der Bildfläche verschwand. Der Fehler in diesem Denkansatz liegt darin, dass lediglich visuell gedacht wurde, der Entwicklungsprozess jedoch nicht in die Überlegungen einbezogen wurden. Dass es aus rein technischer Sicht möglich ist, im Browser all das zu machen, was der Benutzer aus der Fat-Client-Welt kennt, ist durchaus korrekt. Nur dass der Aufwand für eine selbstentwickelte Drag-and-Drop-Routine in JavaScript ungefähr um den Faktor zehn größer ist als die Verwendung einer Out-of-the-box-Swing-Routine muss eben gleichfalls in Betracht gezogen werden, alles andere resultiert in einer „Milchmädchenrechnung“. Ganz klar muss erwähnt werden: Genau aus diesem Grund,

dass die Drag-and-Drop-Routine nicht selbstentwickelt werden muss, haben zu dieser Zeit sämtliche Ajax Frameworks einen Hype erleben können, da bereits fertige Komponenten zur Verfügung gestellt wurden, um dem Entwickler ein möglichst hohes Abstraktionslevel für JavaScript und HTML zur Verfügung zu stellen. Allerdings steckte leider nicht hinter jedem Ajax Framework ein Projekt mit ausreichend Manpower, um diese Abstraktion konsequent und browserübergreifend umzusetzen, häufig stellte sich die Qual der Wahl „wenige Komponenten und dafür 95 % Cross-Browser-Support“ oder eben „umfassende Komponentenbibliothek und dafür 100 % Firefox- und 75 % Internet-Explorer-Unterstützung...“. An dieser Stelle muss jedoch auch erwähnt werden, dass die Komponentenbibliotheken für viele Zwecke durchaus ausreichend sind, und um einen schönen Webshop mit Ajax-Features zu erstellen, stehen nach wie vor etliche solide Frameworks zur Verfügung. Nur wenn es darum geht, dem Versicherungssachbearbeiter eine Maske für die Schadensfälle zu implementieren, darf heute und hätte zum damaligen Zeitpunkt durchaus der Fat-Client-Pfad in die Entscheidung mit einbezogen werden dürfen. Der Ajax-Entscheidungsbaum (ca. 2001 – 2008) ist in Abbildung 1.2 dargestellt.

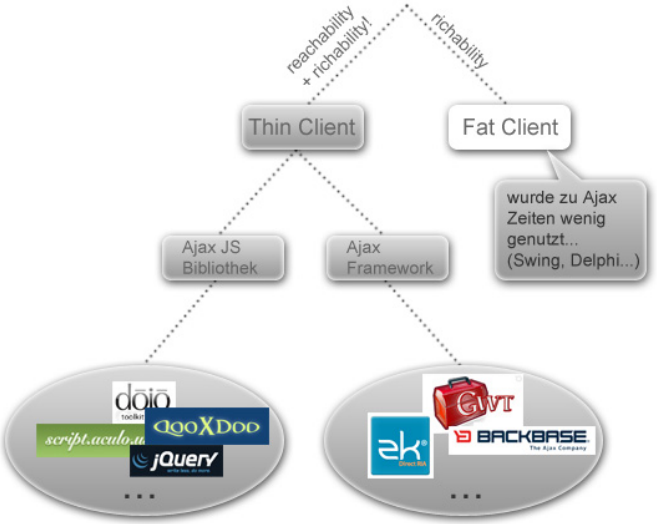

Abbildung 1.2: Alles Ajax oder was?

Folglich musste man feststellen, dass man dem heiligen Gral der ultimativen Benutzeroberflächentechnologie zwar ein ordentliches Stück nähergekommen war, die Suche musste jedoch fortgesetzt werden, denn der Entwicklungsprozess von Ajax-Anwendungen war ganz und gar nicht „Gral-like". Genau an dieser Stelle kommen Rich Internet Applications (RIAs) ins Spiel, überzeugende GUI-Resultate die man bereits aus der Ajax-Ära kannte, diesmal jedoch gepaart mit einem einfachen Entwicklungsprozess. Und warum können RIAs einfacher entwickelt werden als Ajax-Anwendungen? Prinzipiell hat man an dieser Stelle aus den Fehlern der Vergangenheit gelernt und festgestellt, dass eine früher häufig getroffene Annahme auf den Benutzer in dieser Form nicht mehr zutrifft: Der Benutzer ist nämlich durchaus bereit Plug-ins zu installieren, sofern die darauf basierende Anwendung einen Mehrwert liefert. Ajax-Anwendungen basieren auf dem Denkansatz „ich passe die Anwendung dem Benutzer an und unterstütze alle Browser, damit der Benutzer nichts installieren muss und die Anwendung auf 100 % aller Clients ausgeführt werden kann...". RIAs hingegen akzeptieren die Grenzen dessen, was im Browser möglich ist und was nicht, für alles was darüber hinausgeht, muss der Benutzer seinen Teil beitragen und ein Plug-in installieren. Innerhalb des Plug-ins herrschen dann eigene Gesetze, angefangen bei der Programmiersprache/dem Programmiermodell bis hin zur Toolpalette. Die für die Entwicklung zur Verfügung gestellt wird. Dies bedeutet zwar meistens auch eine neue Sprache im Client, im Gegenzug wird dem Entwickler jedoch garantiert: Kümmer' Dich nur darum, dass Du die Sprache im Griff hast, wir sorgen dafür, dass die Oberfläche sich überall gleich verhält und die Funktionen, die Du implementierst, einwandfrei ausgeführt werden können. Im RIA-Kessel brodelt es nicht ganz so heftig wie im Ajax-Kessel. Das liegt daran, dass die Entwicklung einer Laufzeitumgebung (Plug-in) und einer Entwicklungsumgebung den Softwareriesen (Microsoft, Google, Adobe, Sun...) vorbehalten ist, entsprechend übersichtlich gestaltet sich der aktuelle Rich-Client-Ast (Abbildung 1.3) – Thin Client und Fat Client stehen als Technologiepfade nach wie vor zur Verfügung und dürfen nicht (wie zu Ajax-Zeiten...) vernachlässigt werden.

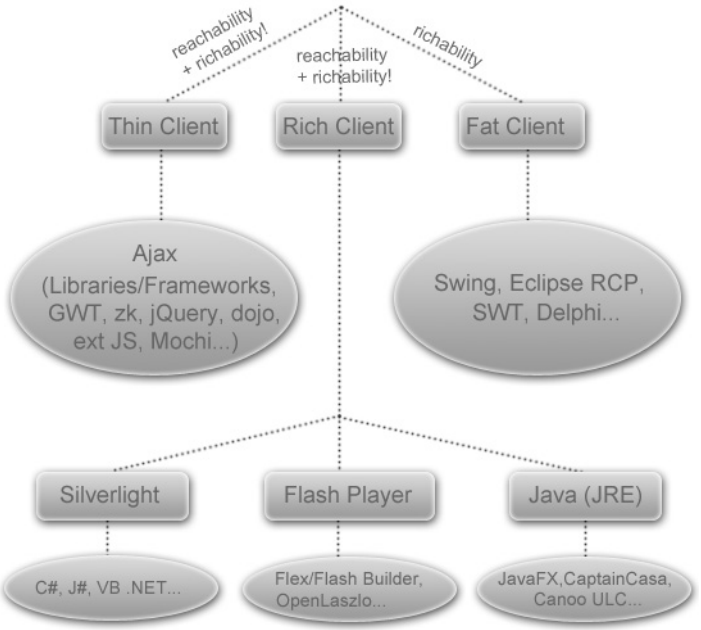

Abbildung 1.3: Ausgewachsener GU(mm)I-Baum

Zusammenfassend kann gesagt werden: Im Bereich User-Interface-Technologien tut sich Einiges. Wichtig dabei ist, ganz einfach immer im Hinterkopf zu behalten: Es gibt nicht die „richtige“ oder „falsche“ bzw. „beste“ Technologie. Ziel des Buchs ist es auch nicht, Ihnen Adobe Flex als diese vorzustellen, viel mehr wird realistisch gezeigt, was Flex kann und wo die Grenzen liegen, wann Sie also besser auf den Fat- oder Thin-Client-Pfad ausweichen sollten. Die hier abgebildeten Bäume erheben übrigens keinen Anspruch auf Vollständigkeit, sondern dienen ganz einfach als grafische Stütze, um das „Big Picture“ zu vermitteln – und genau das sollten Sie bereits jetzt erfasst haben, um im nächsten Kapitel gezielt Java und GUI-Technologien unter die Lupe nehmen zu können.

1.1.2 Java und GUI-Technologien

Adobe Flex kann nicht nur eingesetzt werden, um Oberflächen für Java Backends zu erstellen, der Focus des Buchs liegt jedoch genau auf dieser Kombination. Warum Java ausgerechnet mit Adobe Flex kombiniert wird, erläutert das vorliegende Kapitel. Dabei ist es wichtig, die Historie (schon wieder Geschichtsunterricht) von Java und GUI-Technologien kurz unter die Lupe zu nehmen.

Ursprünglich gab es im Bereich Java GUIs etwas namens „AWT". Die Abkürzung steht für **A**bstract **W**indow **T**oolkit und ermöglichte die Erstellung von plattformunabhängigen Oberflächen basierend auf Java – böse Zungen sprachen übrigens bereits damals vom **A**nnoying **W**indow **T**oolkit, Motivation für die zuletzt genannte Abkürzung war die Tatsache, dass komplexe GUI-Elemente, die unter Umständen nicht auf allen Betriebssystemen vorhanden sind, ganz einfach weggelassen wurden.

Der zweite Vorstoß von Sun in den Oberflächenbereich war dann schon wesentlich ausgereifter, seit 1998 zählt die Swing-Grafikbibliothek und Programmierschnittstelle zum festen Bestandteil der Java Runtime. Kinderkrankheiten (der grausigen Lila-Look-and-Feel-Style der ersten Releases fällt ebenfalls in diese Kategorie...) wurden Schritt für Schritt beseitigt, und mittlerweile ist die Entwicklung von Swing-Oberflächen eine der solidesten Möglichkeiten, um eine Oberfläche auf eine Java-Anwendung zu flanschen. Darüber hinaus existieren GUI Builder (z. B. Netbeans IDE[1]), die das Jonglieren mit Swing-Elementen auf Editorbasis ein für alle mal beenden, stattdessen können Swing-Oberflächen ganz einfach per Drag and Drop erstellt werden. Dank Java Web Start (siehe Kompakttext „Applet und Web Start") wird durch Swing nicht nur der Bereich Fat Client abgedeckt, sondern gleichzeitig ein Rich-Client-Kandidat bereitgestellt – eine Swing-Anwendung kann grundsätzlich als Fat Client oder via Web gestartet werden.

Basierend auf den bisherigen Überlegungen wäre die konsequente Schlussfolgerung: Wir brauchen Flex als Frontend-Technologie doch eigentlich gar nicht, da mit Swing Hybrid Clients (Fat und Rich) erstellt werden können, was will man also mehr? Diese These ist durchaus vertretbar, allerdings wird dabei ein Aspekt nicht berücksichtigt: Dank Web 2.0 und Ajax reicht es meistens nicht mehr aus, dass eine Oberfläche „nur" funktioniert, die Oberfläche muss auch Spaß machen.

1 http://netbeans.org/

Web Start ist die konsequente Weiterentwicklung der Java-Applet-Technologie: Der Hauptunterschied besteht darin, dass Applets immer über den Browser gestartet werden müssen. Web Start ermöglicht das Starten einer entfernten Java-Anwendung auch ohne Browser über die Ausführung einer so genannten JNLP-Datei. Auch die aus der Applet-Welt bekannten JRE-Kompatibilitätsprobleme (z. B. Anwendung erstellt mit JDK 1.5, Benutzer hat nur JRE 1.4 installiert...) werden durch Web Start gelöst, da das benötigte Runtime Environment vom Entwickler selbst bestimmt und mitgeliefert werden kann. Java-Anwendungen, die über Web Start gestartet werden, werden genau wie Applets in einer Sandbox ausgeführt.

Genau das wurde auch von Sun erkannt, und in einer Nacht- und Nebelaktion wurde das JavaFX-Projekt ins Leben gerufen. Zielsetzung des Projekts: Mit klobigen Swing-Oberflächen sah man sich im Rich-Client-Segment chancenlos gegenüber der Konkurrenz Silverlight und Adobe Flash, entsprechend musste ein Gegenpol geschaffen werden, um auch mit Java tolle GUI-Resultate erzeugen zu können. Als Programmiersprache für den Client kommt dabei JavaFX Script zum Einsatz, die Anwendung selbst kann über Java Web Start ausgeführt werden. Leider ist die JavaFX-Komponentenbibliothek eher mager, zwar existiert ein GUI Builder (JavaFX Composer), dieser ist jedoch eher Lippenbekenntnis zur visuellen Erstellung von Oberflächen und in puncto Reifegrad nicht vergleichbar mit Flex/Flash Builder. Entsprechend häufig schlagen viele Projekte (noch) einen Bogen um JavaFX.

So gesehen steht ein Entwickler, der eine Rich-Client-Lösung erstellen möchte, die nicht nur funktioniert, sondern beispielsweise auch eine Prise Web 2.0 enthält, vor einem Problem: JavaFX könnte zwar eingesetzt werden, die Entwicklungsdauer einer Anwendung kann jedoch dadurch auch explodieren – es fehlen an dieser Stelle nach wie vor Referenzprojekte bzw. Erfahrungswerte.

Genau hier kommt Adobe Flex ins Spiel, denn Adobe Flex kann genau wie JavaFX als Frontend-Ergänzung an eine Java-Anwendung gekoppelt werden. Mit dem kleinen Unterschied, dass Adobe Flex ein wesentlich breiteres Komponentenspektrum umfasst und somit garantiert ist, dass Ihre Anwendung an der Stelle nicht seitens des Frameworks limitiert wird. Darüber hin-

aus existiert Flex nicht erst seit gestern, entsprechend liegen Erfahrungswerte hinsichtlich Entwicklungsdauer vor, die es Ihnen ermöglichen, zumindest eine grobe Aufwandabschätzung vorzunehmen.

Denken wir an den zuvor erwähnten Java-Entwickler und die Ausgangslage, so stellt Flex für diesen eine Lösungsmöglichkeit dar, um sein Java Backend mit einem zeitgemäßen Frontend zu ergänzen. Die Kombination aus bewährter Technologie auf dem Server (Java) und einer ansprechenden, gleichfalls bewährten Frontend-Technologie (Flex), ist einer der Ursachen dafür, dass sich genau diese Kombination steigender Beliebtheit erfreut, und Sie werden innerhalb des Buchs sehen: Die Oberflächen machen nicht nur dem Anwender Spaß, sondern auch Ihnen bereits zur Entwicklungszeit.

Nachdem Adobe Flex bisher als Blackbox betrachtet wurde, ist es langsam Zeit, einen konkreten Blick in diese Box zu werfen. In den nachfolgenden Kapiteln werden wir Flex etwas greifbarer machen und uns genauer anschauen, was technologisch dahintersteckt, und wie Flex grundsätzlich überhaupt mit Java verknüpft werden kann. Und wir werden im Begriffs Chaos aufräumen, denn Flex, Flash, Flash Builder, Flex Builder, Flash Player sind definitiv zu viele Begriffe für eine Technologie...auch wenn die Marketingstrategen von Adobe diese Ansicht nicht teilen.

1.2 Flex und Flash Builder 4

Bisher wurde auf abstrakter Ebene davon gesprochen, dass Flex eingesetzt werden kann, um Oberflächen für Java-Anwendungen zu erstellen. Wie das aber funktioniert, welche Schritte dafür notwendig sind, und was am Ende dabei rauskommt, wurde bisher noch nicht betrachtet, entsprechend ist es Zeit, technologisch konkret zu werden – genau das ist das Ziel der nachfolgenden Kapitel, wir werden gemeinsam eine Verständnisgrundlage schaffen, damit Sie wissen, was überhaupt entwickelt wird. Darüber hinaus werden die neuen Konzepte von Flash Builder 4 vorgestellt, denn Bücher rund um Flex Builder 3 und Flex gibt es mittlerweile wie Sand am Meer, so gesehen muss es einen triftigen Grund dafür geben, dass sich jemand die Mühe macht, an dieser Stelle ein neues Werk zu verfassen. Ich versichere Ihnen an dieser Stelle: Wir werden nicht nur eine weitere Schippe Sand auf den Flex-3-Strand werfen, stattdessen werden wir einen komplett neuen Strand aufschütten (Dubai lässt grüßen...), und jeder, der dachte, mit Flex Builder 3 bereits gut bedient zu sein, wird erstaunt sein, was mit Flash Builder 4 möglich ist.

1.2.1 Begriffsdschungel: Flex, Flash, Flash Player, Flex Builder...

„...Flex, das ist doch Flash, oder...?“, sicherlich eine der am häufigsten gestellten Fragen, wenn Sie mit einem (Java-)Entwickler über Flex sprechen. Die Frage ist nicht unberechtigt, denn was bei einer Flex-Anwendung am Ende im Client, also Flash Player, „läuft“, ist eine Flash-Datei. Wo liegt also der Unterschied zwischen einer Flash- und einer Flex-Anwendung? Um diese Frage zu beantworten, muss der Entwicklungsprozess für Flex-Anwendungen in die Überlegungen einbezogen werden. Abbildung 1.4 zeigt schematisch die Entwicklung von Flex-nwendungen.

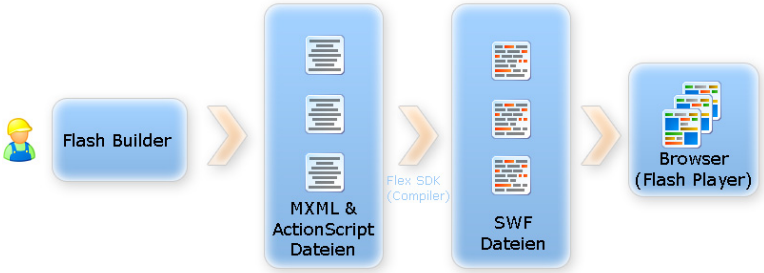

Abbildung 1.4: Entwicklung einer Flex-Anwendung

Die Figur mit dem Helm repräsentiert dabei Ihren Job als Entwickler, und Ihre Arbeit beginnt fast immer im Flash Builder. Flash Builder ist eine Entwicklungsumgebung, die Ihnen die einfache Erstellung von Oberflächen ermöglicht. Das Ergebnis Ihrer Drag-and-Drop-Oberflächengestaltung resultiert in einer MXML-Datei, die Kernfunktion von Flash Builder ist die Transformation der von Ihnen visuell erstellten Oberfläche in MXML-Code. Flash Builder ist an dieser Stelle jedoch kein Muss, MXML-Code kann genauso gut mit einem Editor von Hand erstellt werden, entsprechende MXML-Kenntnisse vorausgesetzt. Die GUI-Logik einer Flex-Anwendung erstellen Sie übrigens mit ActionScript, einer Adobe-eigenen Sprache, die es Ihnen ermöglicht, clientseitige Logik zu implementieren.

MXML- bzw. ActionScript-Code kann vom Flash Player nicht direkt interpretiert werden, es handelt sich um ein abstraktes Zwischenformat, das dann in einem weiteren Schritt durch das Flex Software Development Kit (SDK) in eine *.swf*-(ShockWave-Flash-)Datei übersetzt wird und im Flash Player ausgeführt werden kann.

Betrachtet man die Entwicklung einer Flash-Anwendung (Abbildung 1.5), so kann man feststellen dass die letzten beiden Schritte, unabhängig davon, ob Flex oder Flash entwickelt wird, identisch sind.

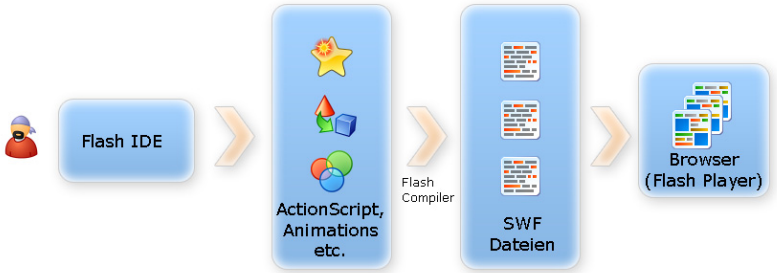

Abbildung 1.5: Entwicklung einer Flash-Anwendung

(Darüber hinaus kann basierend auf dem Entwicklericon ganz links festgestellt werden, dass Flash-Entwickler scheinbar immer wesentlich cooler auftreten als die seriösen Flex-Entwickler mit dem Helm...das ist jedoch nur ein kleines Klischee am Rande und nicht der Hauptunterschied)

Der eigentliche Unterschied besteht in den eingesetzten Entwicklungsumgebungen: Flash-Entwickler verwenden eine Flash IDE (Adobe Flash CS4, SWiSH Max[2]), die IDE ist ausgerichtet auf die einfache Erstellung von Animationen, Dreh- und Angelpunkt der IDE: die Zeitleiste, mit deren Hilfe der Ablauf von Animationen gesteuert werden kann. Darüber hinaus verwenden Flash-Entwickler keine abstrakte Sprache wie MXML für die Definition von Oberflächen, stattdessen verwenden Flash-Entwickler eine breite Palette von grafischen Werkzeugen (ähnlich der Photoshop-Werkzeug-Palette), um die Oberfläche bzw. Animationsinhalte der Anwendung zu erstellen. Für die Übersetzung wird kein dediziertes Flex SDK eingesetzt,

2 http://www.swishzone.com/index.php

ein „nackter“ Flash-Compiler übernimmt die Überführung von *.fla*-Inhalten (Speicherformat für Flash-Seiten) in eine ausführbare *.swf*-Datei.

Flash- und Flex-Anwendungen werden folglich mit unterschiedlichen Werkzeugkästen erstellt: Zielsetzung der Flash IDE ist es dabei, animierte Inhalte via Web zur Verfügung zu stellen, die Flex IDE hingegen hat die Erstellung komponentenbasierter Oberflächen zur Zielsetzung.

Soweit, so gut – gerne würde ich an dieser Stelle direkt zum nächsten Kapitel springen, denn eigentlich sollte jetzt glasklar sein, was der Unterschied zwischen Flex und Flash ist und wie die einzelnen Anwendungsgattungen implementiert werden können. Wäre da nicht noch ein Knieschuss von Adobe, den es zu erwähnen gilt: Das Buch, das Sie in den Händen halten, heißt „Flash Builder 4 und Java“, allerdings ist die Zielsetzung (zumindest nach Inhaltsangabe auf dem Buchrücken) die Erstellung von Oberflächen und nicht die Erstellung von Klicki-Bunti-Animationen, warum also Flash Builder? Leider wurde seitens Adobe die Entscheidung gefällt, im Rahmen von Flex-Release-4 ein kleines Renaming vorzunehmen, es wurde entschieden *Flex* Builder in *Flash* Builder umzubenennen. Die tieferen Beweggründe für das Renaming sind möglicherweise nur den Marketinghäuptlingen von Adobe bekannt, zur besseren Differenzierung zwischen Flex und Flash hat dieser Schritt jedoch leider nicht beigetragen. Wenn Sie also einem Kollegen den Unterschied zwischen Flex und Flash erklären wollen, dürfen Sie sich darauf gefasst machen, einen Schlag weiter auszuholen, um die Unterschiede klarzustellen, statt mit der einfachen Erklärung „Flex=Komponentenoberflächen, Flash=Animationen“ davonzukommen.

Innerhalb des Buchs wird übrigens stets die Rede von Flex-Anwendungen sein, um einen weiten Bogen, die Flex-/Flash-Irrungen und -Wirrungen zu schlagen...

1.2.2 Hintergrundinformationen BlazeDS

Bisher wurde lediglich betrachtet, wie eine Oberfläche mit Flash Builder erstellt werden kann – das ist jedoch noch nicht einmal die halbe Miete, denn richtig spannend wird es erst, wenn Java ins Spiel kommt. Das Bindeglied zwischen Flex- und Java-Welt heißt „BlazeDS“, grundsätzlich handelt es sich bei BlazeDS „nur“ um eine einfache Sammlung von Java-Klassen die mittels XML-Dateien konfiguriert werden können. Bevor BlazeDS aus der technischen Perspektive betrachtet wird, soll ein kurzer Blick auf die BlazeDS-Historie geworfen werden.