

RIA: Eine Idee, deren Zeit gekommen ist



Rich Internet Applikationen (RIA) sind die nächste Generation der Webtechnologie. Sie verbessern die Benutzerschnittstellen und erweitern den Anwendungsbereich von Webapplikationen entscheidend.

So offensichtlich das heute ist, so lange ist der Weg, der zu dieser Erkenntnis geführt hat: Vor etwa zehn Jahren beginnen sich die Entwickler mit webbasierter Technologie zu beschäftigen und erkennen schnell, dass komplexe Anwendungen mit HTML nicht benutzerfreundlich zu gestalten sind. Kein Wunder: Der Stand der Technik sind Client/Server-Applikationen, die mit umfangreichen GUI-Bibliotheken entwickelt werden. Benutzerzentriertes Design ist ein wichtiges Thema.

Alle warten gespannt auf die Weiterentwicklung der Web-Standards. Einige, die nicht warten können, beginnen mit der Entwicklung von dem, was wir heute RIA-Technologie nennen. Dann kommt der Siegeszug des Internets. Die IT-Mana-

ger sehen diesen Hammer und plötzlich wird alles zum Nagel: Fast alle Anwendungen werden mit HTML geschrieben – bis zum Exzess. Eine ganze Generation von Entwicklern wird im Glauben ausgebildet, seitenorientierte Interaktion und Browser-Schnittstellen seien benutzerzentriertes Design.

Die Web-Standards entwickeln sich langsam, und kaum jemand kümmert sich um die Anforderungen der Benutzerinteraktion. In diesem Zustand dämmert der Mainstream jahrelang dahin. Die erwähnten RIA-Technologien fristen ein Nischendasein.

Vor etwa drei Jahren dann ein leiser Weckruf: Das populäre Eclipse-Projekt erhöht mit seiner Rich-Client-Plattform das Bewusstsein in der Java-Community, dass man außerhalb des Browsers schönere und oft auch nützlichere GUIs bauen kann. Der Begriff des Rich Client etabliert sich als Synonym zur klassischen Client/Server-Applikation der Achtziger- und Neunzigerjahre.

Anbieter der erwähnten RIA- bzw. webbasierten Rich-Client-Technologien reagieren darauf, indem sie den Begriff

der Rich Thin Clients einführen, um sich von der klassischen Fat-Client Eclipse-Plattform zu differenzieren. Etwa gleichzeitig lanciert Macromedia mit Flex 1.0 den Begriff Rich Internet Application, der sich schließlich durchsetzt und Rich Thin Client ersetzt.

Im Januar 2005 schreibt Jesse James Garrett einen Artikel über JavaScript-basierte Internet-Technologie. Eigentlich ist das ein alter Hut, aber Garrett erfindet einen tollen Namen: AJAX. Und dann ist es so weit: Das Thema wird in Tageszeitungen wie der New York Times aufgegriffen und wird zum Allgemeingut. In der nächsten Jahresfrist gibt es internationale Konferenzen und ein Dutzend Bücher zu AJAX und RIA. Das benutzerzentrierte Design wird wiederentdeckt und als geschäftsfördernd eingestuft. Ein Glücksfall, denn unterdessen hat sich die Technologie im Schatten des Mainstreams weit über den AJAX-Ansatz hinaus entwickelt und kann eine breite Palette von Bedürfnissen abdecken. Die Benutzer werden's danken.

Marc Domenig, Gründer und CEO von Canoo

canoo

Canoo Engineering AG

Kirschgartenstrasse 7

CH-4051 Basel

Tel: ++41 61 228 94 44

Fax: ++41 61 228 94 49

→ www.canoo.com

Rich Internet Applications und AJAX

Entscheidungshilfe



Es gibt Hunderte von Kriterien für die Evaluation von RIA- und AJAX-Produkten; so viele, dass man leicht die Prioritäten aus dem Auge verliert. Dieser Artikel postuliert einen Evaluationsprozess mithilfe eines Entscheidungsbaums. Er fragt in hierarchischer Sequenz nach den wichtigsten Anforderungen und Produktmerkmalen mit dem Ziel, bei der Wahl J2EE-basierter RIA-Produkte bedürfnisgerecht zu helfen.

von Marc Domenig

Die Fragen fokussieren auf die Anforderungen, die RIA-Produkte erfüllen können und sollen. Spezifische Produkte werden nicht diskutiert, da deren individuelle Eigenschaften von den fundamentalen Themen ablenken.

RIA und AJAX werden manchmal synonym verwendet. Das ist nicht ganz richtig, denn AJAX steht für „Asynchronous JavaScript and XML“ und bezieht sich nur auf JavaScript-basierte RIA-Lösungen. Ich verwende diese Definition, bin aber mit anderen Autoren einverstanden, dass AJAX-Konzepte auch von anderen RIA-Technologien verwendet werden [1].

Einfache Benutzerschnittstelle?

Die erste und wichtigste Frage lautet: Ist die Benutzerschnittstelle der Anwendung so einfach, dass sie HTML-fähig ist? Wenn

ja, ist HTML die beste Lösung, denn es ermöglicht unbeschränkte Erreichbarkeit über Browser.

Eine Schnittstelle ist einfach genug für HTML, wenn sie nur geringe Anforderungen an die Interaktivität stellt. Wenn einer der folgenden Punkte die Schnittstelle verbessert, sollte RIA-Technologie erwogen werden:

- Partielle Bildschirm-Updates
- Asynchrone Kommunikation
- Server-Push
- Direkte Manipulation unterstützende Komponenten
- Mehrfach koordinierte Fenster
- Modale Dialoge
- Menüs
- Tastaturnavigation

RIA-Technologie ermöglicht Rich-Client-Eigenschaften in einer Webinfra-

struktur. Ziel ist es, die Vorteile von Desktop- und Webapplikationen miteinander zu kombinieren. Drei clientseitige Basistechnologien stehen dafür zur Verfügung: JavaScript, Java und Flash.

Ubiquität, Industriequalität oder Multimedia?

Bei der Wahl einer RIA-Technologie lautet die entscheidende Frage, ob unbeschränkter Endbenutzerzugriff über Browser (Ubiquität), Industriequalität oder Multimediafunktionen am wichtigsten sind. Heißt die Antwort Ubiquität, ist JavaScript/AJAX die beste Lösung, da JavaScript in allen populären Browsern integriert ist, wenn auch in unterschiedlichen Varianten. Für Industriequalität empfiehlt sich eine Lösung mit Java, denn es übertrifft in diesem Bereich die Optionen mit Scripting-Sprachen. Sind Multimedia-Funktionen die

Hauptanforderung, ist Flash die attraktivste Lösung.

Ubiquität: JavaScript/AJAX

AJAX ist kein Produkt und keine neue Technologie, sondern ein neuer Name für eine RIA-Technologie, die auf JavaScript, XML und anderen Technologien basiert. Da die meisten JavaScript-Produkte die AJAX-Terminologie verwenden, werden die beiden zusammen behandelt. JavaScript/AJAX-Produkte eignen sich für Applikationen, von denen Benutzer erwarten, dass sie ohne weiteres in jedem beliebigen Browser laufen, ohne dass ein Plug-in installiert werden muss.

Obwohl JavaScript/AJAX in vielen heutigen Internetanwendungen von Nutzen sein könnte und die Technologie seit Jahren vorhanden ist, gibt es nur wenige überzeugende Implementierungen wie Google Suggest und Google Maps, die J.J. Garret in seinem Artikel über AJAX [2] als Schlüsselbeispiele anführt.

JavaScript/AJAX stellt höchste Ansprüche an die Programmierung. Es ist kein Zufall, dass die Expertise und die Finanzkraft von Google hinter den besten Anwendungen stehen. Internetriesen wie Amazon und eBay nutzen das Potenzial nicht, obwohl ihre Websites zum Beispiel mit Scrolllisten, partiellen Bildschirmupdates oder durchsuchbaren Kategoriebäumen wesentlich verbessert würden.

Die folgenden Fragen helfen bei der Entscheidung, ob ein JavaScript/AJAX-basiertes Produkt den gegebenen Anforderungen genügt.

Unterstützte Browser und Betriebssysteme

JavaScript-Implementierungen variieren signifikant in verschiedenen Browsern und Betriebssystemen. Ein gutes AJAX-Programm muss also für verschiedenste Kombinationen von Browsern und Betriebssystemen konzipiert sein und entsprechend unterschiedliche Codes ausführen – der Alptraum jedes Programmierers. Das wichtigste Merkmal eines JavaScript/AJAX-Produktes ist deshalb, wie gut es die Handhabung dieser Konstellationen unterstützt und deren Verwaltung in Anwendungscode vermeidet. Die

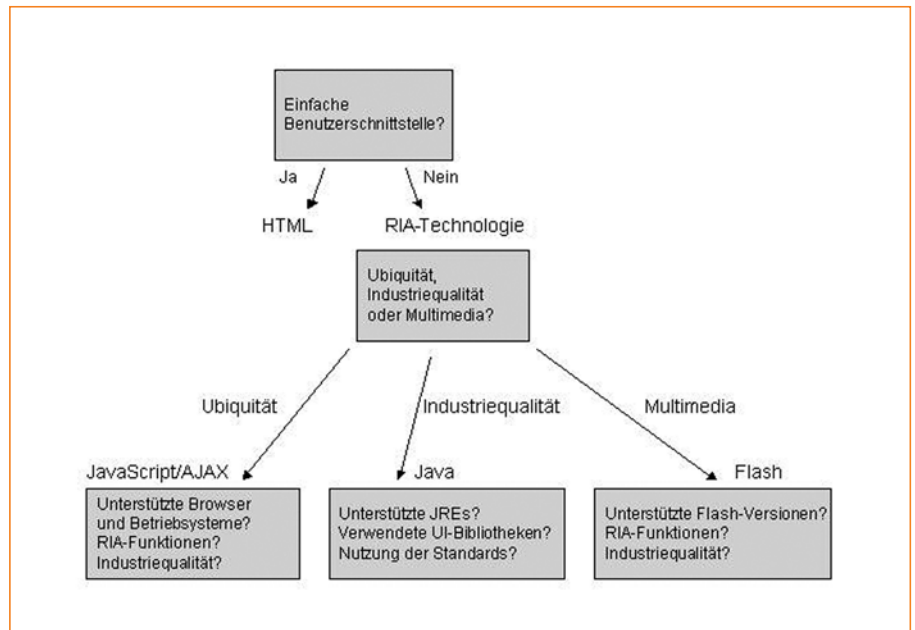


Abb. 1: Entscheidungsbaum

von Google Maps und Google Suggest unterstützten Kombinationen können als Vergleichswert für die Evaluation dienen:

- IE 6.0+ auf Windows
- Firefox 0.8+ auf Windows, Mac, Linux
- Safari 1.2.4+ auf Mac (Google Suggest 1.2.2 oder später)
- Netscape 7.1+ auf Windows, Mac, Linux
- Mozilla 1.4+ auf Windows, Mac, Linux
- Opera 8+ auf Windows, Mac, Linux (Google Suggest 7.5.4+)

RIA-Funktionen

JavaScript stellt nur einfache Rich-Client-Komponenten zur Verfügung. Ein JavaScript/AJAX-Produkt muss deshalb höher stehende Komponenten proprietär implementieren und wird nur einen Bruchteil der RIA-Funktionen einer Komponentenbibliothek für Desktopanwendungen zur Verfügung stellen. Wichtige Fragen sind u.a.:

- Wie vergleichen sich die Komponenten mit solchen für Desktop-Bibliotheken wie Java Swing?
- Ist das Look-and-Feel änderbar?
- Gibt es ein API für die Entwicklung zusätzlicher Komponenten?
- Können fremde Komponenten integriert werden?

- Gibt es einen Markt für fremde Komponenten?
- Gibt es Accessibility-Unterstützung?

Industriequalität

Industriequalität einer Technologie drückt sich in Eigenschaften wie Wartbarkeit, Stabilität, Verfügbarkeit, Skalierbarkeit, Performanz und Sicherheit aus. Diese Merkmale sind mit JavaScript/AJAX-Produkten nicht einfach realisierbar, da diese auf einer Scripting-Sprache aufbauen und in unterschiedlichen Laufzeitumgebungen funktionieren müssen. Man darf deshalb nicht dasselbe Qualitätsniveau wie bei homogenen Java-Produkten erwarten. Außerdem ist die Bandbreite der Lösungen aus architektonischer Sicht sehr breit: Alle in Abbildung 4 gezeigten Architekturen sind in heutigen JavaScript/AJAX-Lösungen zu finden.

Wie die Option ganz rechts zeigt, kann der JavaScript-Code auf eine anwendungsunabhängige Präsentationsengine beschränkt werden. Dieser reine Thin-Client-Ansatz hat mehrere Vorteile: Anwendungen können vollständig in Java geschrieben werden; das serverseitige Programmiermodell ist homogen; der Anwendungscode muss weder auf Client und Server noch auf verschiedene Programmiersprachen verteilt werden und er läuft

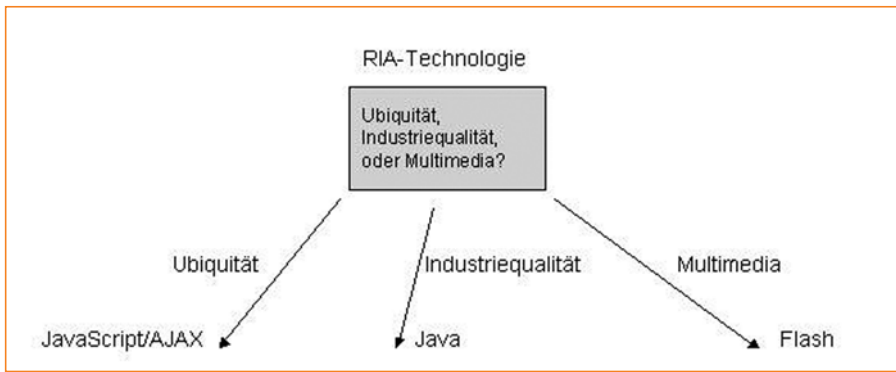


Abb. 2: Entscheidungsstufe RIA-Technologie

hauptsächlich in der robusten serverseitigen Umgebung.

Reine Thin-Client-Architektur kommt bei JavaScript-basierten RIA-Produkten selten vor. Die drei Optionen links in Abbildung 4 zeigen, wie die Funktionalität normalerweise für jede Anwendung individuell auf Client und Server verteilt wird. Die Option ganz links ist eine reine Fat-Client-Architektur, bei der der gesamte Code in JavaScript geschrieben ist. Fat-Client- und hybride Ansätze können für gewisse Szenarien geeignet sein, aber im Hinblick auf Industriequalität ist eine reine Thin-Client-Architektur vorzuziehen. Entscheidende Fragen sind deshalb:

- Was für eine Architektur wird unterstützt bzw. verlangt: thin, fat oder hybrid?
- Ist das Programmiermodell serverseitig oder aufgeteilt?
- Wird der Anwendungscode rein in Java oder in einer heterogenen Mischung mit JavaScript, Java und proprietären XML-Sprachen geschrieben?
- Wie viel des Produktcodes ist JavaScript?
- Wie wird das Sicherheitsproblem des clientseitigen Scriptings gelöst?

Industriequalität: Java

Rein auf Java basierte RIA-Produkte sind die beste Grundlage für Industriequalität. Mit seiner soliden Standardbasis, der homogenen Technologie, der großen Auswahl an Entwicklungswerkzeugen und dem hohen Weiterentwicklungspotenzial garantiert Java, dass ein gut konzipiertes Produkt langfristig und kosteneffizient verwendet werden kann. Wartbarkeit, Stabilität, Verfügbarkeit, Skalierbarkeit, Performanz und Sicherheit sind auf vielen

verschiedenen Plattformen gewährleistet.

Da die Java-GUI-Technologie für vollwertige Desktopanwendungen entwickelt wurde, lassen ihre Rich-Client-Funktionen nichts zu wünschen übrig. Es gibt umfassende, individuell anpassbare Standardkomponenten sowie einen Markt mit zahlreichen Zusatzkomponenten. Im Gegensatz zu vielen JavaScript-Erzeugnissen haben Java-RIA-Produkte normalerweise einen Thin-Client-Ansatz wie ganz rechts in Abbildung 4. Der Client ist eine anwendungsunabhängige Präsentationsengine, die als Applet im Browser, als Java-Programm auf dem Desktop oder auch auf einem PDA laufen kann. Bei der Evaluation von Java-basierten RIA-Produkten sind die nun folgenden Fragen wichtig.

Unterstützte JREs (Java Runtime Environments)

JRE-Implementierungen sind besser standardisiert als JavaScript-Laufzeitsysteme. Mit einem bestimmten JDK (Java Development Kit) geschriebene Programme laufen normalerweise im entsprechenden JRE, unabhängig davon, ob das JRE auf einem Browser, einem Desktopbetriebssystem oder einem PDA läuft. Das Versionsproblem entsteht vor allem beim JRE und den das JRE unterstützenden Zielumgebungen.

Die meisten RIA-Produkte unterstützen mehrere JRE-Versionen. Einige basieren auf der AWT-Widget-Bibliothek von JDK 1.1 und laufen auf JRE 1.1 oder später. Andere nutzen Swing und laufen auf JRE 1.2 oder später. Die

Vor- und Nachteile, die mit der Unterstützung mehrerer bzw. alter und neuer Versionen verbunden sind, müssen genau geprüft werden, da die älteste unterstützte Version den „kleinsten gemeinsamen Nenner“ bestimmt. Ein auf dem systemnahen AWT basiertes Produkt mit tiefem „Nenner“ wird meist auf vielen JREs laufen, da spätere Java-Versionen gut nach unten kompatibel sind. Ein auf Swing basiertes Produkt mit hohem „Nenner“ wird von den Vorteilen neuerer Java-Versionen profitieren. Die wichtigsten Fragen lauten:

- Welche Apparate/Umgebungen für die Endbenutzer sind vorgesehen?
- Welche verfügbaren JREs können einfach darauf installiert werden?
- Auf welchen JREs läuft das Produkt?
- Welches JRE ist der „kleinste gemeinsame Nenner“ des Produkts?
- Nutzt das Produkt die Verbesserungen neuerer JDK/JRE-Versionen?

Verwendete UI-Bibliotheken

Die clientseitige Java-Technologie hat sich stark entwickelt. Die systemnahe AWT-Bibliothek wurde mit der funktional höherwertigen, im Laufe der Jahre wesentlich verbesserten Swing-Bibliothek ergänzt. Das Eclipse-Projekt bietet das systemnahe SWT und das höherwertige JFace. RIA-Produkte haben die Funktionalität von hochwertigen Bibliotheken. Einige von ihnen implementieren die höheren Funktionen auf der Basis der systemnahen Bibliotheken selber. Der Vorteil dieses Ansatzes ist, dass mehrere UI-Technologien unterstützt werden können, so lange die Funktionalität auf den „kleinsten gemeinsamen Nenner“ reduziert wird. Der Nachteil ist,

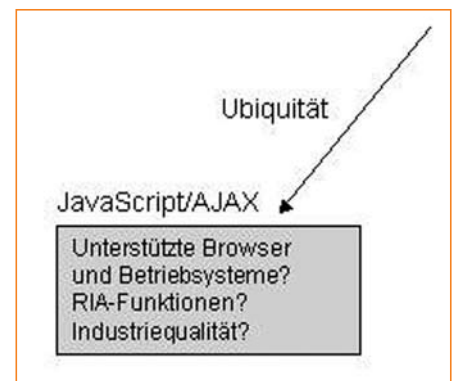


Abb. 3: Entscheidungsstufe JavaScript/AJAX

dass die höheren Funktionen proprietär sind. Wird dagegen eine höherwertige Bibliothek wie Swing als Basis für das RIA-Produkt verwendet, verringert sich die proprietäre Fertigungstiefe. Sogar das API kann aus der Standardbibliothek übernommen werden [3]. Ein solches Produkt erlaubt natürlich auch die bessere Nutzung der Weiterentwicklung des höherwertigen Standards. Wichtige Fragen sind hier:

- Basiert das Produkt auf systemnahen (AWT, SWT) oder höheren Bibliotheken (Swing, JFace)?
 - *systemnah*: Unterstützt es mehrere UI-Bibliotheken auf dem Client?
 - *höher*: Ist das API proprietär oder ein Server-Side-Proxy-Ansatz [3]?
- Wie groß ist die proprietäre Fertigungstiefe der Client-Technologie?
- Gibt es ein Erweiterungs-API für die Entwicklung neuer Komponenten?
- Können fremde Komponenten integriert werden?
- Kann das Look-and-Feel angepasst werden?
- Ist Accessibility-Unterstützung vorhanden?

Nutzung der Standards

Java-Standards sind wichtig, wenn Software möglichst wenig proprietär sein soll. Wir haben gesehen, dass die proprietäre Fertigungstiefe der Client-Technologie sehr unterschiedlich sein kann. Dasselbe trifft für die Serverseite und die Client/Server-Kommunikation zu. J2EE definiert Standards, die die Client/Server-Kommunikation z.B. auf Frage-Antwort-Protokolle und bestimmte Transportprotokolle wie HTTP(S) oder RMI/IIOP beschränken. Darüber hinaus erlauben EJBs kein Multithreading, das nicht vom Container verwaltet wird. Dadurch wird gewährleistet, dass Skalierungen wie das Clustering von der J2EE-Infrastruktur gehandhabt werden können. RIA-Produkte, die eigene Server/Proxies haben oder bidirektionale Kommunikation erlauben, verstoßen gegen diese Standards und benötigen proprietäre Software, die die entsprechenden Funktionen der Standardinfrastruktur ersetzt. Im Allgemeinen sollte ein Produkt, das die Java-Standards effizient einsetzt, keine eigenen Funktionen haben, die auch

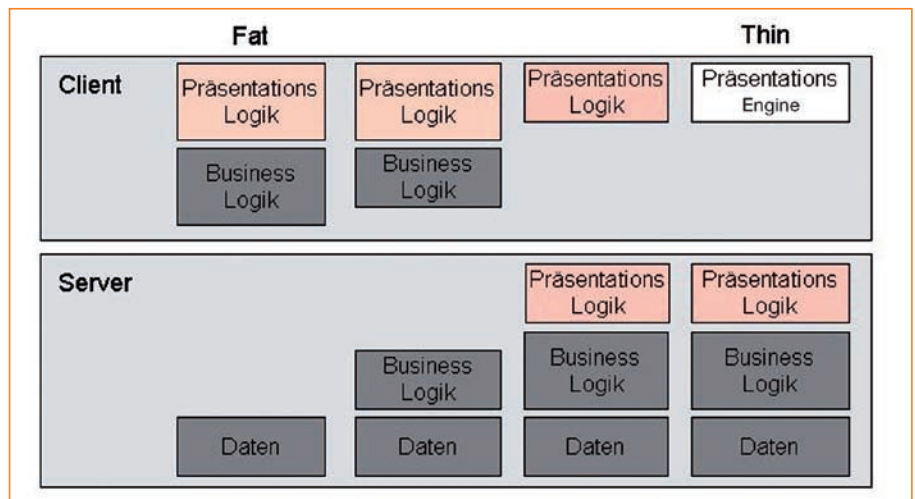


Abb. 4: RIA-Architektur-Optionen

an die Standardinfrastruktur delegiert werden könnten. Die folgenden Fragen sind hier wichtig:

- Wird das Session-Handling an den J2EE-Container delegiert oder hat das Produkt einen proprietären Server bzw. Proxy?
- Wird Clustering unterstützt und an die J2EE-Plattform delegiert?
- Wird die Kommunikation durch J2EE-konforme Protokolle geregelt?
- Ist die Laufzeitinstallation konfigurierbar und sowohl in einem Servlet- als auch einem EJB-Container möglich?
- Ist die Laufzeitinstallation als Portlet nach dem JSR-168-Standard möglich?
- Werden die Java-VM-Standards genutzt, um Stand-alone-Einsatz zu ermöglichen, indem Client- und Serverseite in einem einzelnen Prozess installierbar sind?

Multimedia: Flash

Flash wurde für Multimedia-Anwendungen entwickelt. Die Flash-Kerntechnologie umfasst eine Ausführungsumgebung (den Flash-Player), das binäre Fileformat für Filme, SWF, und die Scripting-Sprache ActionScript. Komfortable Tools, die SWF generieren und ActionScript-Programmierung unterstützen, helfen bei der Gestaltung von Multimedia-Anwendungen und Webseiten. Da Flash-basierte RIA-Produkte für die Anwendungsentwicklung relativ neu sind, sind die Technologie und die Tools noch

voll in der Entwicklung. Deshalb muss in allen Bereichen unbedingt auf die Versionen geachtet werden.

Die Architekturen von Flash-basierten und JavaScript/AJAX-Ansätzen sind ähnlich. Die wichtigen Fragen sind hier deshalb fast die gleichen wie bei JavaScript/AJAX.

Unterstützte Flash-Versionen

Die Flash-Kerntechnologie ist proprietär und wird vom Anbieter straff kontrolliert. Programme, die für eine bestimmte Version geschrieben werden, laufen auf den davon unterstützten Browsern und Betriebssystemen. Darum muss beim Flash-Player, ActionScript und dem durch ein Tool erzeugten SWF-Format auf die Versionen geachtet werden. Die wichtigen Evaluationsfragen entsprechen denjenigen für Java-Produkte:

- Welche Apparate/Umgebungen für die Endbenutzer sind vorgesehen?
- Welche verfügbaren Flash-Player können einfach darauf installiert werden?
- Auf welchen Playern läuft das Produkt?
- Welcher Player ist der „kleinste gemeinsame Nenner“ des Produkts?
- Nutzt das Produkt die Verbesserungen neuerer ActionScript/SWF/Player-Versionen?

RIA-Funktionen

Da ActionScript, SWF und der Flash-Player eigentlich nicht für die Rich/Cli-

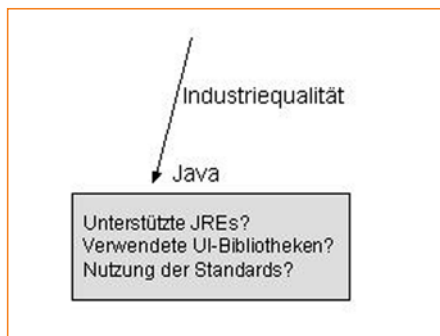


Abb. 5: Entscheidungsstufe Java

ent-Entwicklung konzipiert wurden, müssen die gleichen Fragen wie bei den JavaScript/AJAX-Produkten gestellt werden:

- Wie vergleichen sich die Komponenten mit solchen für Desktop-Bibliotheken wie Java Swing?
- Ist das Look-and-Feel änderbar?
- Gibt es ein API für die Entwicklung zusätzlicher Komponenten?
- Können fremde Komponenten integriert werden?
- Gibt es einen Markt für fremde Komponenten?
- Gibt es Accessibility-Unterstützung?

Industriequalität

Da sie wie JavaScript/AJAX-Ansätze auf einer Scripting-Sprache aufbauen, gibt es bei Flash-basierten Produkten ähnliche Möglichkeiten für die Ausführung von Code in der Client-Umgebung. ActionScript kann auf eine anwendungsunabhängige Präsentationsengine beschränkt werden, wodurch eine reine Thin-Client-Architektur wie ganz rechts in Abbildung

4 entsteht. Es ist aber auch möglich, mit ActionScript beliebigen Anwendungscode für hybride oder Fat-Client-Architekturen zu schreiben. Wie bereits erwähnt, führt eine Thin-Client-Architektur im Allgemeinen zu besserer Qualität, da der Anwendungscode weder auf Client und Server noch auf verschiedene Programmiersprachen verteilt werden muss und hauptsächlich in der robusten serverseitigen Umgebung läuft.

Gängige Flash-basierte RIA-Produkte unterstützen keine reine Thin-Client-Architektur. Sie haben ein heterogenes Programmiermodell und benötigen unterschiedliche Tools für die verschiedenen zu kombinierenden Programmiersprachen. Dies kann akzeptabel sein, wenn sich die proprietären Tools im Wesentlichen darauf beschränken, Multimediafunktionen zu erzeugen und der eigentliche Anwendungscode in der serverseitigen Java-Umgebung läuft. Nützliche Fragen sind:

- Was für eine Architektur wird unterstützt/verlangt: thin, fat oder hybrid?
- Ist das Programmiermodell serverseitig oder aufgeteilt?
- Wird der Anwendungscode rein in Java oder in einer heterogenen Mischung mit ActionScript, Java und proprietären XML-Sprachen geschrieben?
- Wie heterogen, leistungsstark und proprietär ist das benötigte Toolset?
- Ist die Verwendung von ActionScript und proprietären Tools auf die Multimedia-Funktionalität der Anwendung beschränkt?
- Wie wird das Sicherheitsproblem des clientseitigen Scriptings gelöst?

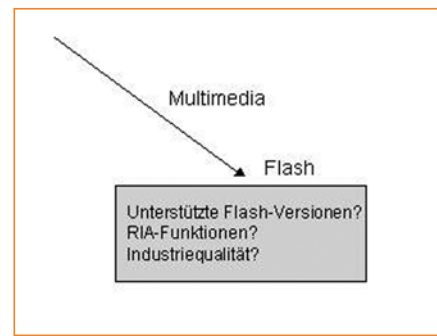


Abb. 6: Entscheidungsstufe Flash

Schlussfolgerungen

RIA und AJAX sind neue Namen für eine Technologie, die es schon seit Jahren gibt. Durch den Hype um diese Bezeichnungen ist es zurzeit schwierig, die verschiedenen Produkte zu bewerten. Sie unterscheiden sich wesentlich durch ihre technischen Grundlagen und ihre Tauglichkeit für spezifische Anforderungen. Der hier vorgestellte Entscheidungsbaum hilft dabei, die richtigen Fragen zu stellen, und führt so durch die wichtigsten Entscheidungen.

Links & Literatur

[1] Coach K. Wei: AJAX: Asynchronous Java + XML?: www.developer.com/design/article.php/3526681

[2] Jesse James Garrett: AJAX: A New Approach to Web Applications: www.adaptivepath.com/publications/essays/archives/000385.php

[3] Bernhard Wagner: Server-Side Swing for Rich Internet Applications: javadesktop.org/articles/canoo/