Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung Seminar Ringvorlesung

Leif Hartmann

Rich Internet Applications

Fakultät Technik und Informatik
Studiendepartment Informatik

Inhaltsverzeichnis

1 Einleitung

Web-Anwendungen werden schon seit einiger Zeit unterstützend oder als Alternative zu Desktop-Anwendungen eingesetzt. Durch die Verwendung von Web-Anwendungen, wird Benutzern der Zugriff auf Anwendungen ermöglicht, ohne Installationsaufwand für etwaige Client-Software betreiben zu müssen. Ein Beispiel für solche unterstützenden Anwendungen sind Groupware-Anwendungen, wie etwa Kalender und Email-Clients.

Da diese klassischen Web-Anwendungen einige Nachteile im Funktionsumfang gegenüber Desktop-Anwendungen haben, können sie ihre Desktop-Pendants nicht vollständig ersetzen. Rich Internet Applications stellen einen weiteren Schritt in der Entwicklung von Anwendungen dar. Sie erweitern klassische Web-Anwendungen um Funktionalitäten, die an Desktop-Anwendungen angelehnt sind. [Farrell und Nezlek (25–28 June 2007)].

> Macromedia defines RIAs as combining the best user inter-face functionality of desktop software applications with the broad reach and low-cost deployment of Web applications and the best of interactive, multimedia communication. The end result: an application providing a more intuitive, responsive, and effective user experience. [Duhl (2003)]

Diese Definition von Macromedia beinhaltet einen wichtigen Aspekt von Rich Internet Applications: Die Vorteile von klassischen Web-Anwendungen und Desktop-Anwendungen werden hierbei vereint. In Abschnitt 2.2 wird genauer auf diese Vorteile eingegangen.

1.1 Zielsetzung

Ziel dieser Ausarbeitung ist es Problemstellungen aufzuzeigen, die sich im Allgemeinen bei der Verwendung von Rich Internet Applications (RIA) ergeben. Daraus werden Anforderungen an ein Entwicklungsprozess aufgestellt, anhand derer im Rahmen einer Masterarbeit eine spezielle RIA-Technologie untersucht werden soll.

2 Rich Internet Applications

In der Einleitung wurde bereits der Begriff „klassische Web-Anwendung“ verwendet. Dieser Begriff bezeichnet im Rahmen dieser Ausarbeitung eine Anwendung, die als Client einen Browser verwendet, der ausschließlich für die Benutzerschnittstelle zuständig ist. Dabei kommt nur sehr wenig Logik zum Einsatz. Diese beschränkt sich auf kleine Javascript-Hilfen, um beispielsweise Formularwerte zu überprüfen. Insbesondere wird durch diese Javascript-Logik keine Kommunikation mit dem Server durchgeführt.

Ein wesentlicher Unterschied zwischen klassischen Web-Anwendungen und RIAs ist die Kompetenzverteilung der Daten und der Logik (siehe Abbildung 1). In einer RIA können die Kompetenzen unterschiedlich verteilt werden. Je nach RIA-Technologie liegt der Schwerpunkt der Logik auf Client- oder auf Serverseite. Letztendlich bleibt es dem Entwickler überlassen, wie die Kompetenzen verteilt werden.

2.1 Technologien

Diese Ausarbeitung beschäftigt sich mit RIAs im Allgemeinen, weitestgehend auf Technologieunabhängiger Ebene. Jedoch ist es nötig, für einige Aspekte, eine grobe Unterscheidung von RIA-Technologien vorzunehmen:

Rein Browser-basierte Anwendungen Hierbei handelt es sich um RIAs, die als Client ausschließlich einen Browser verwenden. Unter diese Kategorie fallen alle AJAX-Anwendungen[1], die keine weiteren Browser-Plugins benötigen.

Plugin-basierte und Stand-Alone-Anwendungen Einige RIA-Technologien verwenden zusätzliche Laufzeitumgebungen auf Clientseite. Hierfür wird beispielsweise Flash oder Java verwendet.

[1] Für eine Einführung in die AJAX-Technologie, siehe [Garrett (2005); Paulson (Oct. 2005)]

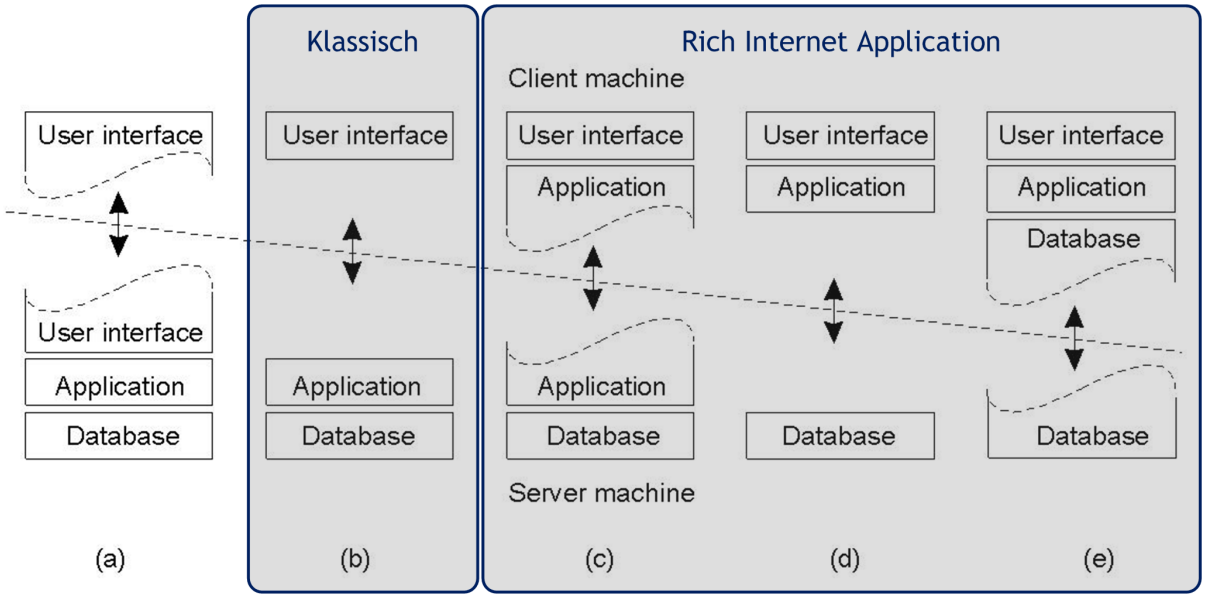

Abbildung 1: Kompetenzverteilung (angelehnt an [Tanenbaum und van Steen (2003)])

2.2 Vorteile von Rich Internet Applications

Im den folgenden Abschnitten werden die Vorteile von RIAs gegenüber klassischen Web-Anwendungen und Desktop-Anwendungen aufgeführt. Eine genauere Betrachtung einzelner Aspekte folgt in Abschnitt 2.3. Ein Überblick über die Unterschiede ist Abbildung 2 zu entnehmen.

2.2.1 Vorteile gegenüber klassischen Web-Anwendungen

Flüssigere Bedienung Die Antwort auf eine Benutzerinteraktion besteht bei klassischen Web-Anwendungen aus kompletten HTML-Seiten. Dadurch entsteht bei den meisten Anfragen sehr viel überflüssiger Overhead. In RIAs werden in der Regel nur kleine Teile der GUI verändert, wodurch sich der Overhead verringert und die Antwortzeiten auf Anfragen des Benutzers kürzer werden. Näheres hierzu, siehe Abschnitt 2.3.4.

Mehr Daten und Logik auf Clientseite Da mehr Daten und Logik auf Clientseite verarbeitet werden können, muss nicht ständig auf den Server zugegriffen werden, wodurch die Bedienung ebenfalls flüssiger wird. Näheres hierzu, siehe Abschnitte 2.3.1 und 2.3.2.

Weniger Netzwerklast Aus den beiden obigen Punkten ergibt sich insgesamt eine Reduzierung der Netzwerklast.

Bekannte Benutzerschnittstelle Die Benutzerschnittstelle von RIAs orientiert sich in der Regel an Desktop-Anwendungen, wodurch die Umgewöhnung für den Benutzer vereinfacht wird. Näheres hierzu, siehe Abschnitt 2.3.3.

Feature	C/S, Desktop	Web	RIA
Universal client (browser)	YES	YES	YES
Client installation	Complex	Simple	Simple
Interaction capabilities	Rich	Limited	Rich
Server-side business logic	YES	YES	YES
Client-side business logic	YES	Limited	YES
Full page refresh required	NO	YES	NO
Frequent server round-trips	NO	YES	NO
Server-to-client communication	YES	NO	YES
Disconnected functioning	YES	NO	YES

Abbildung 2: Vergleich von Desktop-Anwendungen, klassischen Web-Anwendungen und RIAs [Bozzon u. a. (2006b)]

Offline-Funktionalität Einige RIA-Technologien bieten die Möglichkeit mit ihnen entwickelte Anwendungen teilweise oder komplett ohne Serverzugriff zu verwenden (nach dem ersten Zugriff). Näheres hierzu, siehe Abschnitt 2.3.5.

2.2.2 Vorteile gegenüber Desktop-Anwendungen

Geringer Administrationsaufwand Der größte Vorteil von RIAs gegenüber „normalen“ Desktop-Anwendungen ist der geringe Administrationsaufwand. Die Anwendung wird dabei nicht mehr in starren Release-Zyklen ausgeliefert, sondern den Benutzern als Service angeboten [O'Reilly (2005)]. Auf Clientseite ist nur die Installation einer Laufzeitumgebung notwendig. Dabei kann es sich um einen Browser handeln, ein Browser-Plugin (z. B. Flash oder das Java-Plugin) oder aber eine Stand-Alone-Laufzeitumgebung. In jedem Fall ist es nicht nötig, die in der Laufzeitumgebung ausgeführte Anwendung, selbst zu installieren oder manuell Updates einzuspielen.

Einfacher Zugriff Sofern die Laufzeitumgebung auf Clientseite bereits installiert ist – insbesondere Browser sind in der Regel bereits verfügbar – kann von jedem Client, der Zugang zum Server hat, auf eine RIA zugegriffen werden.

2.3 Problemstellungen

Durch die erweiterten Möglichkeiten von RIAs (insbesondere in der Kompetenzverteilung) wächst auch der Grad an Komplexität. Im Folgenden werden einige Problemstellungen, die sich aus der größeren Komplexität ergeben, aufgezeigt und mit klassischen Web-Anwendungen verglichen.

2.3.1 Daten

Wie bereits in Abbildung1 zu sehen, sind persistente Daten in klassischen Web-Anwendungen ausschließlich auf Serverseite verfügbar. Eine Ausnahme bilden hierbei Cookies, die allerdings nicht mit Sicherheit über längere Zeit auf dem Client verweilen. Außerdem bieten Cookies nur sehr eingeschränkte Zugriffsmöglichkeiten (beispielsweise im Gegensatz zu einer Datenbank) [Preciado u. a. (5–6 Oct. 2007)].

Auch temporäre Daten werden primär auf Serverseite gehalten. Beispielsweise wird der Inhalt eines Warenkorbes für einen Benutzer, für die Dauer der Session, komplett auf dem Server zwischengespeichert. Um die verschiedenen Warenkörbe den einzelnen Benutzern zuzuordnen, wird üblicherweise eine Session-ID verwendet, die bei jeder Anfrage an den Server gesendet wird.

RIAs können die Daten auf unterschiedliche Persistenzebenen (temporär und persistent) und Architekturschichten (Serverseite und Clientseite) verteilen. Je nach Technologie, können beide Persistenzebenen in beiden Architekturschichten hinterlegt werden [Bozzon u. a. (2006b)]. Es ist beispielsweise möglich einen Warenkorb im Speicher des Clients zu hinterlegen, sodass der Benutzer schnelleren Zugriff auf dessen Inhalt erhält.

Clientpersistenz ist ein wichtiger Aspekt für Offline-Funktionalität (siehe Abschnitt 2.3.5). Rein Browser-basierte RIA-Technologien können dies ohne zusätzliche Plugins jedoch nicht gewährleisten. Die Clientlogik – in diesem Fall ausgeführt von der Javascript-Engine des jeweiligen Browsers – kann nicht auf persistente Speichermedien des Clients zugreifen.

Aufgrund der erweiterten Möglichkeiten Daten auf dem Client zu verwenden, sollte eine RIA-Technologie Mechanismen zur Verfügung stellen, die die Client/Server-Kommunikation erleichtern. Das Marshalling und Unmarshalling[2] sind wichtige Aufgaben, die nach Möglichkeit von einer Middleware soweit vereinfacht werden sollten, dass sie für Entwickler transparent geschehen.

2.3.2 Anwendungslogik

In [Preciado u. a. (5–6 Oct. 2007)] werden drei Ausführungsarten der Logik bezüglich ihrer Verteilung unterschieden:

Serverseitig Die Ausführung der Logik für eine bestimmte Aufgabe findet ausschließlich auf Serverseite statt. Klassische Web-Anwendungen verwenden üblicherweise ausschließlich diese Ausführungsart, da der Client wenig Funktionalität hierfür anbietet oder komplexe Logik nur umständlich realisiert werden kann.

Clientseitig Die Ausführung der Logik für eine bestimmte Aufgabe findet ausschließlich auf Clientseite statt. Wenn für die Ausführung einer bestimmten Aufgabe nicht auf Serverdaten zugegriffen werden muss, ist es oftmals sinnvoll die Clientkapazitäten auszunutzen, um überflüssige Netzwerklast zu vermeiden.

Gemischt Hierbei wird die Ausführung so aufgeteilt, dass ein Teil auf Server- und ein Teil auf Clientseite erfolgt. Dies macht z. B. dann Sinn, wenn die Clientlogik auf wenige Daten zu-

[2] In diesem Fall das Umwandeln zwischen den Datenformaten von Client und Server

greifen muss, die aus einem großen Datenstamm auf Serverseite ausgewählt werden. Die Auswahl der Daten geschieht auf Serverseite, während die Verarbeitung auf Clientseite ausgeführt wird, um die Netzwerklast zu verringern.

2.3.3 Präsentation

Die GUI-Elemente von klassischen Web-Anwendungen sind auf die Formular-Elemente von HTML beschränkt. Auch wenn diese für viele Aufgabenbereiche ausreichend sind, gibt es diverse Steuerelemente, die in Desktop-Anwendungen häufige Verwendung finden und nicht mit HTML-Formular-Elementen abgedeckt werden können. Ein Beispiel hierfür sind dynamische Tabellen, die in vielen Datenintensiven Desktop-Anwendungen verwendet werden. Für Entwickler von klassischen Web-Anwendungen bleibt in diesem Fall die Möglichkeit eine ähnliche Funktionalität mit den vorhanden Elementen umzusetzen. Dies führt allerdings in der Regel zu einer umständlichen Bedienung. Alternativ können Entwickler versuchen mittels Javascript und HTML komplexere GUI-Elemente nachzubauen, was jedoch mit viel Aufwand und schlecht wartbarem Code einhergeht. Rein Browser-basierte RIA-Technologien arbeiten auf diese Weise und nehmen dem Entwickler den Aufwand ab.

Neben den HTML-Steuerelementen selbst, ist die Interaktionen mit ihnen relativ eingeschränkt. So sind für Desktop-Anwendungen übliche Methodiken, wie Drag-and-Drop und das Verschieben von Dialogen und Fenstern nicht ohne umständliche Umwege möglich.

Ein wesentlicher Vorteil von RIAs ist die Erweiterung der Benutzerschnittstelle bezüglich der genannten Punkte. Ziel der meisten RIA-Technologien ist es unter anderem im Bereich Benutzerschnittstelle einen vollständigen Ersatz für Desktop-Anwendungen zu bieten. Außerdem soll die Einbindung und Verwendung von Multimedia-Elementen, wie Videos und Bilder, vereinfacht werden.

Einige RIA-Technologien versprechen die Einbindung von mobilen Geräten, wie PDAs oder Mobiltelefonen als Client-Geräte. Je nach verwendeter Technologie (z. B. Flash, HTML mit Javascript, Java), bieten die Laufzeitumgebungen auf mobilen Geräten nur eingeschränkte Funktionalität. Entwickler müssen darauf achten, die Benutzerschnittstelle so zu strukturieren, dass sie auch mit kleinen Bildschirmen noch gut zu bedienen ist.

Diese Schwierigkeit trifft auch auf klassische Web-Anwendungen zu, die auf mobilen Geräten eingesetzt werden, jedoch ist die Anpassung der Benutzerschnittstelle hierbei nicht so aufwendig, da sie insgesamt einfacher gestaltet ist.

2.3.4 Kommunikation

Die Client/Server-Kommunikation in klassischen Web-Anwendungen ist stets synchron. Der Client stellt eine HTTP-Anfrage (ggf. mit Parametern z. B. aus Formularen) und erhält als Antwort eine komplette HTML-Seite. Dadurch entsteht viel überflüssige Netzwerklast, insbesondere wenn nur sehr kleine Teile der GUI (z. B. eine einzelne Zahl) verändert werden.

RIAs verwenden andere Kommunikationsformen, wodurch die GUI flüssiger reagiert und weniger Netzwerklast entsteht:

Kleinere Servernachrichten Der Client einer RIA erhält als Antwort auf Anfragen an den Server, nur selten komplette HTML-Seiten, sondern kleine Nachrichten, die nur die geforderten Daten enthalten. Dies kann z. B. eine Beschreibung der zu ändernden GUI-Elemente sein.

Asynchrone Kommunikation RIAs verwenden in den meisten Situationen asynchrone Kommunikation. Der Client stellt hierbei eine Anfrage und wartet nicht auf die Antwort des Servers. Sobald die Antwort eintrifft, wird eine Callback-Funktion ausgeführt, die die Antwort verarbeitet und damit z. B. ein GUI-Element ändert. Insbesondere im Falle von rein Browserbasierten Clients ist dies sinnvoll, da Javascript im Browser single-threaded läuft. Würde hierbei synchrone Kommunikation verwendet werden, würde die Benutzerschnittstelle der Anwendung nach einer Anfrage nicht reagieren, bis die Antwort des Servers eingetroffen ist.

Server-Push Viele RIA-Technologien bieten die Möglichkeit vom Server Nachrichten an den Client zu schicken, ohne dass dieser sie explizit anfordert. In Anwendungen in denen Live-Daten eine Rolle spielen, wie z. B. Chat-Anwendungen, ist dies sinnvoll, um die Netzwerklast zu verringern. Ein alternatives, periodisches Pollen erzeugt viel Overhead, da ständig Anfragen an den Server geschickt werden, auch wenn keine neuen Informationen für den Client vorliegen.

Da die meisten Technologien HTTP als Kommunikationsprotokoll verwenden, ist ein „echtes“ Server-Push nicht möglich. Stattdessen kommt ein Mechnismus namen „deferred reply“ [Puder (2007)] zum Einsatz. Dabei wird vom Client eine Anfrage an den Server gestellt, dessen Antwort verzögert wird. Erst wenn der Server eine Nachricht an den Client schicken will, geschieht dies in Form einer Antwort auf die wartende Anfrage. Um einen Timeout der Anfrage zu vermeiden antwortet der Server periodisch (aber selten) mit einer leeren Nachricht. Sobald der Client diese Antwort erhält, stellt er eine erneute Anfrage.

Ein wesentlicher Nachteil dieses Verfahrens ist, dass es, bei steigender Anzahl von Clients, nicht gut skaliert. Für jeden Client muss so, über die gesamte Verbindungsdauer hinweg, eine zusätzliche Verbindung offen gehalten werden.

2.3.5 Offline-Funktionalität

Ein wesentlicher Nachteil von klassischen Web-Anwendungen ist die ständige Kommunikation mit dem Server. Sobald keine Verbindung zum Server besteht, kann die Anwendung nicht mehr bedient werden.

Einige RIA-Technologien bieten für diesen Fall Offline-Funktionalitäten an, durch die der Client auf lokale Datenreplikate zurückgreifen kann, wenn keine Verbindung zum Server besteht. In solchen Fällen sind Mechanismen nötig, die die Synchronisation der Datenbestände zwischen Client und Server übernehmen, wenn eine erneute Verbindung hergestellt wird. Insbesondere bei Anwendungen mit vielen Benutzern, die auf den gleichen Serverdaten arbeiten, ist dies selten ein trivialer Vorgang, der nicht unbedingt transparent für den Entwickler von der RIA-Plattform abgenommen werden kann. In [Leff und Rayfleld (May–June 2006)] werden verschiendene Kom-

munikationsmodelle für diese Problemstellung erläutert, die hier nur kurz vorgestellt werden. Auf die Vor- und Nachteile dieser Modelle wird hier ebenfalls nicht näher eingegangen.

Message-Based Model Hierbei wird explizit (also nicht transparent für den Entwickler) zwischen Logik, die direkt auf dem Client (im Offline-Modus) ausgeführt wird und Logik, die auf dem Server ausgeführt werden soll unterschieden. Der Client erstellt eine Liste von Anwendungs-spezifischen Nachrichten, die die ausgeführte Logik beschreiben. Bei einer erneuten Verbindung werden diese an den Server geschickt. Auf Serverseite sorgt Anwendungs-spezifische Logik dafür die vorgenommenen Änderungen an den Datenbestand des Servers zu propagieren.

State Replication Im Gegensatz zum Message-Based Model wird hierbei (transparent für den Entwickler) direkt auf einer lokalen Client-Datenbank gearbeitet. Die veränderten Datensätze werden von einer Middleware protokolliert und bei einem Wiederaufbau der Verbindung an den Server geschickt.

Method Replay In diesem Modell geschieht die Replikation ebenfalls transparent für den Entwickler durch eine Middleware. Im Gegensatz zu State Replication werden hierbei allerdings nicht die veränderten Datensätze protokolliert, sondern (wie beim Message-Based Model) die Logik, die auf Serverseite ausgeführt werden soll.

2.3.6 Sicherheit

Sicherheitsmechanismen für RIAs unterscheiden sich grundsätzlich nicht von Sicherheitsmechanismen für klassische Web-Anwendungen. Daher können bekannte Sicherheitskonzepte übernommen werden: Da auch in RIAs üblicherweise HTTP für die Client/Server-Kommunikation zum Einsatz kommt, kann die Absicherung auf Transport-Ebene durch SSL (bzw. HTTPS) erfolgen. Für die Authentifizierung können ebenfalls übliche Mechanismen verwendet werden, wie z. B. einfache HTTP-Authentifizierung oder Formular-basierte Authentifizierung.

Ein erhöhtes Risiko bilden allerdings Anwendungen, bei denen viele Daten über längere Zeit auf dem Client verwendet werden. In klassischen Web-Anwendungen ist es üblich, den Benutzer anhand einer Session-ID zu identifizieren. Sobald sich der Benutzer ausloggt oder ein Session-Timeout auftritt, sind auf Serverseite alle der Session zugeordneten Daten nicht mehr zugänglich.

Ein mögliches – wenn auch nicht sehr häufiges – Angriffsszenario besteht darin, dass ein Angreifer nach dem eigentlichen Benutzer dessen Client verwendet (also physisch anwesend ist oder anderweitig Zugriff auf den Clientrechner hat). Denkbar ist beispielsweise, dass der Benutzer in einem Internet-Café seine Mails über einen Webmail-Zugang liest. Vergisst der Benutzer sich abzumelden, hat ein potentieller Angreifer (nach Ablauf des Session-Timeouts) keinen Zugriff auf die Serverdaten. Es ist ihm lediglich möglich die gerade angezeigten Daten zu lesen.

Auch für RIAs ist die Verwendung von Sessions mit Timouts üblich. Jedoch können hierbei deutlich mehr Daten aktuell auf dem Client verfügbar sein (bis hin zu persistenten Daten). Daher ist es notwendig nicht nur auf Serverseite, sondern auch auf Clientseite für geeignete „Aufräum-Mechanismen“ zu sorgen, die nach dem Ausloggen des Benutzers oder nach Ablauf des Session-Timeouts alle flüchtigen Daten löscht.

Unabhängig von dem erläuterten Szenario ist es sinnvoll, Sicherheits-kritische Daten nur dann vom Client verarbeiten zu lassen, wenn es nicht anders möglich ist.

2.3.7 Testen

Automatisierte Tests sind ein wichtiger Bestandteil in Software-Entwicklungsprozessen. Für die meisten Sprachen, sowohl für klassische Web-Anwendungen als auch für RIAs, sind Unit-Test-Blibliotheken verfügbar. Für das Testen der Serverseite sind diese eine gute Hilfe, um automatisierte Tests durchzuführen.

Das Testen des Clients von klassischen Web-Anwendungen ist ebenfalls verhältnismäßig einfach möglich. Dafür können Test-Frameworks, wie HttpUnit[3], HtmlUnit[4] oder WebTest[5] verwendet werden, die die relevanten Funktionen eines Browsers emulieren und die Ergebnisse von Anfragen auf Schlüsselelemente untersuchen. Da RIAs wesentlich mehr Logik auf Clientseite ausführen und die GUI komplexer ist, wird das automatisierte Testen deutlich schwieriger.

Es hängt von der verwendeten Technologie ab, ob vorhandene Test-Frameworks weiterhin verwendet werden können. Für rein Browser-basierte RIAs, kann beispielsweise WebTest verwendet werden [Guillemot und König (2006)]. Die Integration der Testumgebung kann jedoch umständlich sein, da Entwickler oftmals keinen direkten Einfluss auf den generierten HTML-Code haben und WebTest darauf ausgelegt ist, den HTML-Code zu analysieren.

Auch das Testen der Client/Server-Kommunikation ist schwieriger, wenn sie asynchron verläuft. Da der Client nicht auf die Serverantwort wartet, kann nur anhand eines Timeouts festgelegt werden, wann eine Anfrage als verloren und damit nicht beantwortet gilt.

2.4 Anforderungen

Im Folgenden werden einige Anforderungen an eine RIA-Technologie und an einen damit erfolgenden Software-Entwicklungsprozess gestellt. Dabei stehen zum einen die verfügbaren Funktionen der verwendenten Technologie im Vordergrund und zum anderen allgemeine Aspekte der Software-Entwicklung, wie z. B. Wartbarkeit (insbesondere für größere Anwendungen).

2.4.1 Anforderungen an die Technologie

Robuste Laufzeitumgebung Die Laufzeitumgebung des Clients sollte keine Inkompatibilitätsprobleme zwischen verschiedenen Plattformen haben. Bei rein Browser-basierten Technologien ist dies aufgrund der Browser-Vielfalt schwierig.

Hohe Verbreitung Wenn eine RIA nicht nur als Intranet-Anwendung mit stark begrenztem Benutzerumfeld eingesetzt wird, sondern als öffentlich zugängliche Internet-Anwendung, ist es nötig, dass die Laufzeitumgebung auf möglichst vielen Clients bereits verfügbar ist.

[3] http://httpunit.sourceforge.net/ – Verifiziert am 14.02.2008
[4] http://htmlunit.sourceforge.net/ – Verifiziert am 14.02.2008
[5] http://webtest.canoo.com/ – Verifiziert am 14.02.2008

Einfache Clientinstallation Da einer der größten Vorteile von RIAs der geringe Installationsaufwand auf Clientseite ist, sollte die Installation einer ggf. nötigen Laufzeitumgebung möglichst einfach erfolgen.

Offline-Funktionalität Die Technologie sollte Mechanismen bereitstellen, durch die verhindert werden kann, dass Verbindungsabbrüche die Anwendung unbenutzbar machen. Außerdem ist für Anwendungen, die speziell als Desktop-Ersatz entwickelt werden, Clientpersistenz von großer Bedeutung.

2.4.2 Anforderungen an den Entwicklungsprozess

GUI Unabgängig davon ob ein grafischer GUI-Designer bei der Erstellung der Oberfläche hilft, sollte die Sprache in der die GUI beschrieben wird geeignet gewählt sein. Insbesondere wenn Anwendungen für stark variierende Bildschirmauflösungen entwickelt werden, ist die Entwicklung der Oberfläche nicht immer einfach.

Automatische Tests Es sollte ohne große Umstände möglich sein, Testfälle für Client- und Serverseite zu entwickeln.

IDE Um produktiv entwickeln zu können, sollte eine brauchbare IDE (Integrierte Entwicklungsumgebung) verfügbar sein, die neben üblichen Funktionen wie Code-Completition und Syntax-Highlighting auch Unterstützung für Code-Generierung[6] und Refactoring mitbringt.

3 Fazit und Ausblick

In Kapitel 2 wurde gezeigt, dass die erweiterten Möglichkeiten von Rich Internet Applications eine höhere Komplexität von solchen Anwendungen, im Vergleich zu klassischen Web-Anwendungen, mit sich bringen. Die vorgestellten Problemstellung und die daraus aufgestellten Anforderungen können als Grundlage für eine Untersuchung von ausgewählten Technologien herangezogen werden. Hierbei wird weniger wert nur auf die angebotenen Funktionen der RIA-Technologien gelegt, als auf den Entwicklungsprozess. Die Technologien auf denen viele RIA-Frameworks aufbauen, sind nicht neu, und es existieren bereits einige Anwendungen, die die erweiterten Möglichkeiten von RIAs zeigen. RIA-Frameworks sollen aber vor allem die Möglichkeit bieten, gut wartbare Anwendungen zu entwickeln und Hilfestellung bei der Architektur der Anwendungen leisten. In der Ausarbeitung für das Fach Anwendungen 2 [Hartmann (2008)], werden einige ausgewählte RIA-Frameworks und -Technologien vorgestellt.

3.1 Ausblick auf Masterarbeit

Der nächste Schritt in Richtung Masterarbeit ist die Auswahl einer geeigneten Technologie und eines Untersuchungsszenarios. Bei dem Szenario soll es sich vorraussichtlich um eine konkrete,

[6] In diesem Fall ist Code-Generierung für kleine Code-Teile gemeint, wie z. B. das automatische Erstellen von Getter- und Setter-Methoden

zu entwickelnde Anwendung, anhand der die Problemstellungen untersucht werden. Je nachdem wie der Schwerpunkt des Szenarios gewählt wird, kann dies Auswirkungen auf die Wahl der Technologie haben. Einige Beispiele:

- Handelt es sich um eine Intranet-Anwendung mit einem begrenzten Benutzerkreis, ist die Installation eines bestimmten Browsers, eines Browser-Plugins oder einer Stand-Alone-Runtime von geringerer Bedeutung.
- Soll die Anwendung im Internet verfügbar sein und ein möglichst großer Benutzerkreis darauf zugreifen, ist es wichtiger, dass möglichst viele Benutzer die Anwendung ohne Installation nutzen können.
- Ist die Anwendung als Ersatz für eine Desktop-Anwendung gedacht, die lokale Dateien bearbeitet, ist Clientpersistenz von größerer Wichtigkeit.
- Ist die Anwendung grafisch anspruchsvoll, kommen rein Browser-basierte Ansätze nicht unbedingt in Frage [Carfagno (2007)].

Eine denkbare Anwendung für eine Untersuchung ist eine „typische“, dateninstensive Intranet-Anwendung. In einem direkten Vergleich zu bestehenden Web-basierten Intranet-Anwendungen sollten allerdings weitere – in dieser Ausarbeitung nicht untersuchte – Aspekte betrachtet werden, um die Vorteile gegenüber klassischen Web-Anwendung zu verdeutlichen. Hierbei spielt die erweiterte GUI eine wesentliche Rolle. Eine Veränderung der GUI macht nur dann Sinn, wenn dies die Bedienbarkeit fördert. Dafür ist eine Untersuchung der Usability denkbar, die sich an den Maßstäben einer Desktop-Anwendung orientiert. Außerdem ist Barrierefreiheit ein weiterer Aspekt, der durch eine komplexere GUI im Allgemeinen, und durch AJAX-Anwendungen im Speziellen, nicht zwangsläufig gefördert wird [Stringer u. a. (2007)].

Neben der Untersuchung einer konkreten Technologie, ist ein Vergleich einiger ausgewählter Technologien bezüglich ausgewählter Schwerpunkte denkbar.

Literatur

[Bozzon u. a. 2006a] BOZZON, Alessandro ; COMAI, Sara ; FRATERNALI, Piero ; CARUGHI, Giovanni T.: Capturing RIA concepts in a web modeling language. In: *WWW '06: Proceedings of the 15th international conference on World Wide Web*. New York, NY, USA : ACM, 2006, S. 907–908. – ISBN 1-59593-323-9

[Bozzon u. a. 2006b] BOZZON, Alessandro ; COMAI, Sara ; FRATERNALI, Piero ; CARUGHI, Giovanni T.: Conceptual modeling and code generation for rich internet applications. In: *ICWE '06: Proceedings of the 6th international conference on Web engineering.* New York, NY, USA : ACM, 2006, S. 353–360. – ISBN 1-59593-352-2

[Carfagno 2007] CARFAGNO, Virginio: *Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung*, HAW Hamburg, Diplomarbeit, 2007

[Duhl 2003] DUHL, Joshua: *White paper: Rich Internet Applications*. IDC. 2003. – URL `http://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf`

[Farrell und Nezlek 25–28 June 2007] FARRELL, J. ; NEZLEK, G.S.: Rich Internet Applications The Next Stage of Application Development. In: *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on* (25-28 June 2007), S. 413–418. – ISSN 1330-1012

[Garrett 2005] GARRETT, Jesse J.: *Ajax: A New Approach to Web Applications*. Verifiziert am 22.02.2008. 2005. – URL `http://adaptivepath.com/ideas/essays/archives/000385.php`

[Guillemot und König 2006] GUILLEMOT, Marc ; KÖNIG, Dierk: Web testing made easy. In: *OOPSLA '06: Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. New York, NY, USA : ACM, 2006, S. 692–693. – ISBN 1-59593-491-X

[Hartmann 2008] HARTMANN, Leif: *Rich Internet Applications - Technologien*, HAW Hamburg, Seminarausarbeitung, 2008

[Leff und Rayfleld May–June 2006] LEFF, A. ; RAYFLELD, J.: Programming model alternatives for disconnected business applications. In: *Internet Computing, IEEE* 10 (May-June 2006), Nr. 3, S. 50–57. – ISSN 1089-7801

[O'Reilly 2005] O'REILLY, Tim: *What Is Web 2.0 – Design Patterns and Business Models for the Next Generation of Software*. Verifiziert am 11.02.2008. 2005. – URL `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`

[Paulson Oct. 2005] PAULSON, L.D.: Building rich web applications with Ajax. In: *Computer* 38 (Oct. 2005), Nr. 10, S. 14–17. – ISSN 0018-9162

[Preciado u. a. 5–6 Oct. 2007] PRECIADO, J.C. ; LINAJE, M. ; COMAI, S. ; SANCHEZ-FIGUEROA, F.: Designing Rich Internet Applications with Web Engineering Methodologies. In: *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on* (5-6 Oct. 2007), S. 23–30

[Preciado u. a. 26 Sept. 2005] PRECIADO, J.C. ; LINAJE, M. ; SANCHEZ, F. ; COMAI, S.: Necessity of methodologies to model rich Internet applications. In: *Web Site Evolution, 2005. (WSE 2005). Seventh IEEE International Symposium on* (26 Sept. 2005), S. 7–13. – ISSN 1550-4441

[Puder 2007] PUDER, Arno: A cross-language framework for developing AJAX applications. In: *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*. New York, NY, USA : ACM, 2007, S. 105–112. – ISBN 978-1-59593-672-1

[Stringer u. a. 2007] STRINGER, Elizabeth C. ; YESILADA, Yeliz ; HARPER, Simon: Experiments towards web 2.0 accessibility. In: *HT '07: Proceedings of the 18th conference on Hypertext and hypermedia*. New York, NY, USA : ACM, 2007, S. 33–34. – ISBN 978-1-59593-820-6

[Tanenbaum und van Steen 2003] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme*. Pearson Studium, 2003. – ISBN 3-8273-7057-4