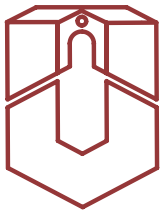


UNIVERSITÄT
OSNABRÜCK

INSTITUT FÜR KOGNITIONSWISSENSCHAFTEN

In Kooperation mit dem Fraunhoferinstitut für Arbeitswirtschaft und Organisation (IAO)
und dem Competence-Center für Mensch-Maschine-Interaktion in Stuttgart

Bachelorthesis

Eine Rich Internet Application zur Evaluation von natürlichsprachlichen Dialogprototypen in einem Wizard-of-Oz Szenario

Matthias Peissner (Fraunhofer IAO - CC HCI)
Prof. Dr. Oliver Vornberger (Universität Osnabrück)

Oktober 2008

Manuel Fittko
Matr.-Nr.: 912723
manuel@fittko.de

Abstract

Ausgangspunkt der Arbeit ist die Tatsache, dass bestehende Systeme zur Konzeption und zur Evaluation von Voice-User-Interfaces (VUIs), dem sprachlichen Pendant der grafischen Benutzersysteme (GUIs), dem Nutzer nur eine sehr beschränkte Unterstützung bieten und insbesondere nicht im ausreichenden Ausmaß auf die Bedürfnisse von Dialog-Designern, Linguisten und Usability-Experten eingehen. Aufgrund von fehlenden oder unzulänglichen grafischen Benutzerinterfaces zum Prototyping und zum Testen von Sprachdialogsystemen kann die Umsetzung von Dialogdesigns zumeist nur von Informatikern oder entsprechend geschulten Experten durchgeführt werden. Ziel ist es folglich eine internetbasierte Applikation (Rich Internet Application) zum Entwerfen und Testen von Dialogprototypen umfassend zu konzipieren und die Funktionalitäten die zur Durchführung eines Benutzertests notwendig sind, in einem vertikalen Prototypen zu implementieren.

Basis der Implementation, ist die Entwicklungsumgebung Adobe Flex Builder 3 sowie die ECMAScript- und E4X-konforme, objektorientierte Skriptsprache ActionScript 3, in Kombination mit dem Java-basierten Wowza Media Server.

Danksagungen

Ich möchte allen, die mich bei der Realisierung dieser Arbeit und der Fertigstellung meines Studiums unterstützt haben ein besonderen Dank aussprechen.

Ein ganz besonderer Dank gilt meiner Familie und Sibylle für ihre Geduld ihren liebevollen Rückhalt.

Für die konzeptionelle Unterstützung, ihre Hilfe bei der Umsetzung und Fertigstellung sowie ihr Interesse an meiner Arbeit danke ich: Matthias Peissner, Prof. Oliver Vornberger, Prof. Kai-Christoph Hamborg, Carla Umbach, Wolfgang Beinhauer, Brigitte Ringbauer, Simon Thiel, Marian Gunkel, Marco Santi, Cornelia Hipp, Fabian Hermann und dem CC HCI.

Für die gewissenhafte Korrektur und die vielen hilfreichen Ratschläge danke ich: Sibylle, Eric, Daniel, Sebastian, Micha, Nazim, Philipp, Cornelia, Carsten, Martin, Simon, Timo.

Für die finanzielle Unterstützung danke ich meinen Eltern, der Fraunhofer Gesellschaft, und der Nolte&Lauth GmbH.

Ich danke auch Adobe Systems, Wowza Media Systems, der Eclipse Foundation und sämtlichen Software-Herstellern und Entwicklern, die mir durch die kostenfreie Bereitstellung ihrer Software die Realisierung dieser Arbeit erst ermöglicht haben.

Außerdem danke ich allen haupt- und ehrenamtlichen Flex-Bloggern, dem Wowza Media Support und allen, die sonst in irgendeiner Form zum Gelingen dieser Arbeit beigetragen haben.

Inhalt

1. Einleitung und Hintergrund dieser Arbeit

Motivation für diese Arbeit ist meine immer wiederkehrende Erfahrung, dass im Entwicklungsprozess von User-Interfaces aller Art, insbesondere von Webseiten und grafischen Benutzerschnittstellen, als auch von Voice User Interfaces, die Konzeption und die tatsächliche Durchführung aus einer Hand geschehen. Zum einen scheint vielen Entwicklern die Vorgehensweise eines *iterativen Design- und Entwicklungsprozesses*[1] unbekannt zu sein. Zum anderen fehlt es oftmals an der Bereitschaft, interdisziplinär an die Entwicklung von Mensch-Maschine-Schnittstellen heranzugehen. So sind bei größeren Projekten zwar meist Interface Designer mit dem Design von User-Interfaces beschäftigt, die Umsetzung erfolgt jedoch meist unmittelbar auf Grundlage grafischer Mock-ups, ein umfassendes User-Testing wird nicht selten erst während oder nach einer sogenannten *public beta* Phase durchgeführt, also mit dem fast fertigen Produkt. Sicherlich lässt sich aufgrund vielfältiger Erfahrungen der Konzepter, etablierter Standards bei User-Interface Komponenten[2] oder bereits existierender Anwendungen, welche die Grundlage für eine Neuentwicklung darstellen, viel über den tatsächlichen Nutzwert der zu entwickelnden Applikation schließen. Dennoch ist ein wie auch immer geartetes Usability-Testing, also eine Überprüfung des Nutzwertes, unabdingbar für eine größtmögliche Akzeptanz des Endproduktes. Geschieht eine solche Überprüfung bereits zu einem frühen Zeitpunkt im Software Engineering Prozess, also anhand vertikaler- oder horizontaler Prototypen[3] – funktionstüchtigen Ausschnitten der in der Entwicklung befindlichen Software – so lassen sich viele Konzeptionsfehler bereits frühzeitig auffinden. Dies führt nicht nur im Allgemeinen zur Qualitätssicherung, sondern es bieten sich, wenn auch in durch die Anforderungen und etwaige Pflichtenhefte begrenztem Maße, Anpassungen am (Interaktions-)konzept an, welche wiederum erneut die Einbindung von Designern und Konzeptern sowie von psychologisch geschulten Usability Professionals erfordern.

1.1 Arten des Usability-Testings

Die Überprüfung der Nutzbarkeit der entwickelten Applikation kann auf mannigfaltige Art und Weise geschehen, wobei sich der Umfang und die Art des Testings nach dem Zeitpunkt im Entwicklungsprozess richten. Je früher im Entwicklungsprozess ein solches Testing stattfindet, desto weniger Ressourcen müssen dementsprechend dafür aufgewendet werden. Natürlich sollte dabei der Zeitpunkt des Testings und eines möglichen *Redesigns* der Applikation gut gewählt werden, um den dafür erforderlichen Aufwand wiederum zu minimieren. Ge-

1 Pawar, S. A. (2004)
2 z.B. Menüs oder Tab-Reiter
3 vgl. Nielsen (1993)

schieht ein umfangreiches Testing zu früh, so lassen sich unter Umständen kaum Rückschlüsse auf die tatsächliche Qualität des überprüften Konzeptes schließen, da eventuell wichtige Funktionen noch nicht implementiert wurden, bereits implementierte Funktionen noch nicht ausreichend stabil sind oder die Performanz noch nicht dem endgültigen System entspricht. Wird zu spät getestet, so stellt sich oftmals das Dilemma, einen Kompromiss zwischen einer verbesserten Konzeption und scheinbar vergeblicher Entwicklungszeit zu finden. Handelt es sich um schwere Konzeptionsfehler, so ist prinzipiell auch zu diesem Zeitpunkt eine Korrektur unabdingbar. Befindet sich das Produkt jedoch bereits in einer *Public Alpha-* oder *Beta-Phase*, so sind größere Korrekturen zumeist erst im nächsten Entwicklungszyklus zu realisieren. Das traditionelle Usability Testing hat sich die Tatsache, dass nutzerorientierten Interaktionskonzepten im Software-Engineering lange Zeit eine untergeordnete Rolle zugestanden wurde und dass entweder am Prototyping oder am begleitenden Testing gespart wurde, zu Nutzen gemacht. Leider nimmt die Überprüfung der Nutzbarkeit bei vielen Softwareentwickelnden Unternehmen noch immer den Status eines *notwendigen Übels*[4] ein. Oftmals steht eine Zertifizierung, bzw. ein für gut befinden der entwickelten Software im Vordergrund, wobei sogar manchmal essentielle Kritikpunkte aufgrund scheinbar zu aufwendiger Behebung im Sande verlaufen oder sogar unerwünscht sind.

1.2 Rapid Prototyping im Software-Entwicklungsprozess

Diesem Dilemma lässt sich nur mit einer frühstmöglichen Überprüfung von Interaktionskonzepten begegnen, wobei sich aufgrund der bereits geschilderten Problematik unausgereifter oder „verbugter“ Prototypen, insbesondere ein sogenanntes *Rapid Prototyping*[5] anbietet. Der Begriff des Rapid Prototyping stammt ursprünglich aus dem Maschinenbau und beschreibt eine Technik, in der Musterbauteile aus bereits bestehenden Konstruktionszeichnungen oder CAD-Daten in leicht zu bearbeitenden Materialien gefertigt werden, um beispielsweise Eigenschaften wie Aerodynamik oder Ästhetik eines Bauteils schon zu einem frühen Zeitpunkt, in einer mit dem späteren Endprodukt vergleichbaren Art und Weise, beurteilen zu können. Analog dazu kann im Bereich des Software Engineering und dabei gerade im Bereich Mensch-Maschine-Interaktion, auf Grundlage leicht verfügbarer Komponenten, mit dem Endprodukt bereits in mancherlei Hinsicht nahe kommenden Eigenschaften, eine Art des Prototyping eingesetzt werden, welche nur relativ geringe bis überhaupt keine Kenntnisse im Bezug auf die tatsächliche Umsetzung und Implementierung des Softwarekonzeptes voraussetzt. Im Bereich des Software Engineering und vorwiegend grafischer Benutzerschnittstellen wird jedoch üb-

4 Ansorge, P. & Haupt, U. (1997)
5 vgl. http://www.usabilitynet.org/tools/rapid.htm

licherweise nicht von *Rapid*-, sondern von *low-fidelity*[6]*-Prototyping* gesprochen. Die bis heute gängigste und einfachste Methode, solche low-fidelity Prototypen zu realisieren, ist bis heute das papierbasierte Prototyping[7]. Hiermit lassen sich im Bereich der grafischen Benutzerschnittstellen insbesondere Features wie Navigationsstruktur, Wortwahl, Farbgebung und Layout testen und beurteilen, ohne diese auch nur ansatzweise implementieren zu müssen. Das Verhalten des Systems wird hier gänzlich von einem *Wizard* simuliert, wobei Benutzereingaben von der Versuchsleitung *„ausgeführt“* und so entsprechende Folgezustände einer Applikation erreicht werden. Die Performanz und die Realitätsnähe des auf diese Weise simulierten Systems hängt leider stark von einigen begrenzenden Faktoren ab:

- Komplexitätsstufe des zu überprüfenden Prototyps
- Vorbereitung des Wizards und Organisation der Papierbögen und Elemente
- Auswahl der Testpersonen und deren „Tagesform“, bzw. Motivation

Zwar lassen sich durch papierbasierte Prototypen auf einfachste und kostengünstigste Art und Weise bereits eine Vielzahl an Konzeptionsfehlern und Mängeln im Design frühzeitig erkennen und beseitigen. Ab einer gewissen Komplexitätsstufe des Konzepts sowie der verwendeten Interaktionstechniken lässt sich ein solches Vorgehen jedoch nur noch schwer realisieren. Beispiele für eine nur schwer zu simulierende Nutzereingabe sind die in Zeiten von *Multi-Touch*[8] Eingabegeräten immer häufiger verwendeten Interaktionsformen des *Drag&Drop* oder des *Vergrößerns* und *Verkleinerns* visueller Elemente.

1.3 Wizard-of-Oz-Testing

Auch im Bereich der Sprachdialogsysteme, also Benutzerschnittstellen bei denen in erster Linie per Sprache mit der Computer kommuniziert wird, ist ein solches *Wizard-of-Oz*[9] Testing ein häufig verwendetes Mittel, um Interaktionskonzepte effektiv und frühzeitig zu testen und entsprechend zu optimieren. Die Rolle des Sprachdialogsystems übernimmt hierbei ein speziell geschulter *Wizard*, der das System anhand einer bereits definierten Dialogstruktur, festgelegter oder bereits aufgenommener Systemausgaben simuliert. Ein besonderer Fokus liegt hierbei nicht nur auf *Wortwahl*, *Satzbau* und *Dialogstruktur*, sondern besonders auf dem Behandeln von fehlender oder falscher Nutzereingaben und möglicher Systemfehler wie einer falschen Erkennung und Weiterleitung von Nutzereingaben.

Um einigen der angesprochenen Probleme bei rein papierbasierten Tests entgegenzuwirken

6 vgl. Virzi et al. 1995
7 vgl. Nielsen 1990, S. 315-320
8 vgl. http://cs.nyu.edu/~jhan/ftirtouch/
9 vgl. http://www.cc.gatech.edu/~steven/files/AEL-IEEEPervasive05.pdf

sind mittlerweile gerade im Bereich der Konzeption von Websites, also grafischen Benutzerschnittstellen, mehr oder weniger ausgereifte Produkte erhältlich, mit denen sich Designelemente oder fast komplette Konzepte bereits ohne größeren Aufwand in eine (semi-)interaktive Form überführen lassen[10]. Der Wizard (bzw. die Versuchsleitung) übernimmt in einem solchen Szenario viel weniger die Simulation des Systemverhaltens, als die Aufgabe das Verhalten des Nutzers zu beobachten und gegebenenfalls auf Eingaben zu reagieren. Eine solche softwarebasierte Simulation von frühen low-fidelity Prototypen bietet neben dem Vorteil, den Wizard zu entlasten und dementsprechend weniger Anfällig auf etwaige Fehler des Wizards zu sein den entscheidenden Vorteil, dass das Medium in dem getestet wird, dem tatsächlichen Nutzungskontext bereits sehr nahe kommen kann. Dies ist bei der Verwendung rein papierbasierter Prototypen nicht ansatzweise gegeben. Hinzu kommt, dass einige verfügbare Programme, vor allem im Bereich Web-Design und Website-Konzeption (s.o.) bereits den ersten Schritt in Richtung Implementation darstellen können, ein HTML-Code Export ohne größeren Aufwand realisierbar ist und sich Anpassungen sehr leicht vornehmen lasssen.

Bei Sprach-Benutzerschnittstellen, also Interfaces bei denen die Sprache den hauptsächlichen Kommunikationskanal zwischen Mensch und Maschine darstellt, gestaltet sich diese Form der Entwicklung und der Evaluation früher Prototypen zwar insgesamt schwieriger, es existieren jedoch bereits einige Programme mit deren Hilfe sich low-fidelity Prototypen erstellen und testen lassen. Exemplarisch für sprachbasierte Interfaces habe ich mich dazu entschieden, entsprechend meiner bisherigen Tätigkeit als studentische Hilfskraft beim Fraunhoferinstitut IAO[11] in Stuttgart, speziell dem EU Projekt ASK-IT[12], und der Beteiligung am ESPA Projekt[13] an der Universität Osnabrück, mich auch in dieser Arbeit auf den Entwicklungsprozess natürlichsprachlicher Dialogsysteme[14] zu fokussieren. Das Anwendungsgebiet des zu testenden begrenzten Szenarios entspringt somit auch dem ASK-IT Projekt, in dem es meine Aufgabe war, ein Sprachdialogsystem zur Reiseplanung zu konzipieren, welches speziell auf die Bedürfnisse sehbehinderter Personen eingeht. Bereits im Rahmen dieses Projektes ist in mir die Idee zu einem leicht zu bedienenden System zur Erstellung und zum Testen von low-fidelity Prototypen im Bereich natürlichsprachlicher Dialogsysteme gereift. Die Durchführung des Dialogdesigns für den ASK-IT Prototypen erfolgte dann mittels des frei verfügbaren Online Tools *Voxeo Designer*[15], mit dem sich Dialoge in einer hierarchischen Struktur erstellen lassen. Voxeo bietet sogar die Option, die Systemausgaben per *Text-To-Speech* von einem Voice Ser-

10 z.B. http://www.axure.com/
11 http://www.hci.iao.fraunhofer.de/de/
12 vgl. http://ask-it.org
13 vgl. http://www.espa.uos.de/
14 Boyce, S. J. (2000)
15 vgl. http://designer.voxeo.com/

ver generieren zu lassen. Allerdings wird zur Zeit leider nur die englische Sprache unterstützt. Des weiteren fehlt es diesem Ansatz noch an Möglichkeiten im Bereich Usability-Testing und Evaluation, welche sich jedoch dank des webbasierten Ansatzes leicht integrieren ließen.

Da sich Sprachdialogsysteme erst seit einem knappen Jahrzehnt im kommerziellen Einsatz befinden und die Technik bis heute ein limitierender Faktor bei der Konzeption natürlich-sprachlicher Dialogsysteme[16] ist, besteht jedoch grundsätzlich ein erhöhter Bedarf nach einer Optimierung und Standardisierung der Komponenten, die in einem Sprachdialogsystem benötigt werden. Wenn die Spracherkennung eines Dialogsystems nicht wunschgemäß funktioniert, kann die Arbeit in ein ausgeklügeltes Dialog-Design verpuffen. Dennoch lassen sich durch ein qualitiv hochwertiges Design, mit gut durchdachten Fehlerbehandlungsmechanismen[17], viele Fehlerquellen minimieren und die oftmals frühzeitig eintretende Frustration bei der Nutzung eines – vom technischen Aspekt aus gesehen – mangelhaften Systems verhindern. Ein Beispiel für ein solches Programm, das speziell für die Simulation eines Sprachdialogsystems und die Evaluation früher Prototypen entwickelt wird, ist das frei verfügbare Wizard-of-Oz Tool von Richard Breuer[18], welches auf TCL/TK[19] basiert. Ursprünglich als Kommandozeilentool entwickelt, bietet *Rick's Wizard-of-Oz Tool* jedoch leider nur eine äußerst rudimentäre grafische Schnittstelle. Bietet der verwendete VoiceXML-Standard[20] den Vorteil, dass sich Prototypen bei entsprechenden Kenntnissen ohne großen Aufwand als Startpunkt für die tatsächliche Entwicklung nehmen lassen, so sind eben diese vorausgesetzten Kenntnisse der VoiceXML-Metasprache ein Hindernis bei der Entwicklung von low-fidelity Prototypen durch Konzepter und Dialogdesigner mit begrenztem technischen Hintergrund.

1.4 Zielsetzung

Diese Arbeit setzt sich, unabhängig von der erfolgten Konzeption und Implementation einer Rich Internet Application zum Testen von frühen VUI-Prototypen, zum Ziel, die Wichtigkeit eines iterativen Designprozesses von User-Interfaces aller Art zu unterstreichen. Wie im Folgenden erläutert, bietet sich mit der Verwendung internetbasierter Technologien, wie Adobe Flex/AIR auf der Client-Seite sowie serverbasierte Software wie dem Wowza Media Server (WMS) zum Streaming von Multimediadaten und zur Synchronisation verbundener Clients, ein ideales Werkzeug, welches sich zudem zur Zeit in einer rasanten Entwicklung befindet und sich zunehmender Akzeptanz erfreut.

16 vgl. Smith et al. 1995
17 vgl. Peissner et al. 2004, S. 45 ff, Cohen et al. (2004), S. 67 ff
18 vgl. http://www.softdoc.de/
19 http://www.tcl.tk/
20 http://www.w3.org/TR/voicexml20/

Gerade der Bereich des Testings von Sprachapplikationen ist ein, meiner Meinung nach, ideales Anwendungsgebiet für die Demonstration bereits angesprochener zukunftsweisender Technologien sowie insbesondere dem Live Streaming von Sprache (*Voice-Over-IP*). Im Bezug auf das Wizard-of-Oz-Testing von Sprachdialogsystemen bieten sich also, neben den bereits vorhandenen ausgereiften und weit verfügbaren grafischen Komponenten, die in Adobe Flash (oder in den verwendeten und auf Adobe Flash aufbauenden Produkten Adobe Flex/ AIR) bereits integriert sind, die Kommunikationskanäle, und ein somit besonders einfach durchzuführendes *Remote-Testing*. Blieb bei herkömmlichen papierbasierten Usability-Tests von low-fidelity Prototypen zumeist gar keine andere Möglichkeit, als mit Wizard und Benutzer ortsgebunden oder sogar im selben Raum zu testen, so erschließt sich mit Hilfe des internetbasierten Testings, der Integration von Multimediainhalten und Techniken, wie dem *Live-Tagging* von Audio-/Video Streams,[21] eine Fülle an Möglichkeiten. Die Kombination aus Flex/WMS bot mir hier, wie im Folgenden näher erläutert, eine sehr gute Basis für die Entwicklung einer Rich Internet Application zum Remote-Testing von Voice-User-Interface-Prototypen.

21 http://livedocs.adobe.com/flash/9.0_de/main/00001037.html

2. Rich Internet Applications – Eine Einführung

Unter einer Rich Internet Application versteht man eine beliebig komplexe Software, die ausschließlich über das Internet verfügbar ist und über einen handelsüblichen Browser gestartet werden kann. Charakteristisch für eine solche Anwendung ist die Integration verschiedenster multimedialer Inhalte auf einer Plattform, wobei viele bislang nur von herkömmlichen Desktopanwendungen bekannte Features abgedeckt werden können. Neben den schon länger bekannten Web-Technologien, *JavaScript*, *HTML* und *Flash*, existieren mittlerweile viele unter anderem auf diesen Standards aufbauende Technologien, die bereits als sehr ausgereift bezeichnet werden können. Der grundsätzliche Trend im Bereich Webdesign und bei der Entwicklung von Internetservices, geht ohnehin immer mehr in die Richtung, dass zunehmend komplexer werdende Komponenten in sogenannte *Content Management Systeme* integriert werden, wobei vor allem JavaScript und die damit einhergehende Verwendung dynamischer Technologien wie *DHTML*[22] oder *AJAX*[23], eine immer größere Bedeutung zukommt. Mit den Technologien *Adobe Flex*, einer vor allem für Softwareentwickler interessanten Weiterentwicklung des bereits etablierten und weit verfügbaren *Adobe Flash* sowie neuerdings *Microsoft's Silverlight*[24], sind mittlerweile ausgereifte Produkte führender und konkurrierender Softwareunternehmen verfügbar, die den zu erwartenden Umbruch im Softwarebereich weiter vorantreiben dürften. Mit *Adobe AIR*[25] und *Google Gears*[26], sind Produkte verfügbar, die die Nutzung von Rich Internet Applications zumindest temporär auch offline und Browser unabhängig ermöglichen. Dennoch sind diese Produkte weiterhin plattformunabhängig, da sie auf gängigen plattform-unabhängigen Technologien aufbauen. Im Falle von *Adobe Flex*[27] ist zudem mittlerweile eine ausgreifte Entwicklungsumgebung auf Basis des frei verfügbaren Industriestandards, der *Eclipse Plattform*[28], erhältlich, welches die Entwicklung komplexester internetbasierter Applikationen bestens unterstützt.

2.1 Vorteile von Rich Internet Applications

Bei Services, die auf den angesprochenen Technologien aufbauen, beispielhaft seien hier einmal die bereits einem breiten Publikum bekannten webbasierten Services der Firma Google,

22 vgl. http://de.selfhtml.org/dhtml/intro.htm
23 vgl. u.a. http://www.adaptivepath.com/ideas/essays/archives/000385.php
24 http://silverlight.net/
25 http://www.adobe.com/de/products/air/
26 http://gears.google.com/
27 http://www.adobe.com/de/products/flex/overview/
28 http://www.eclipse.org

wie das *Google Mail*[29] Interface oder die *Google Spreadsheets*[30] genannt, wird im Allgemeinen nicht wie bei herkömmlichen Web-Services jede Anfrage und Interaktion des Nutzers zunächst auf dem Server des Anbieters bearbeitet, sondern geschieht vorwiegend clientseitig, also üblicherweise im Webbrowser des Nutzers. Im Falle von Adobe Flash/Flex, befinden sich die Inhalte im sogenannten *Adobe Flash-Player*[31], der wiederum in herkömmliche Internet-Seiten eingebettet werden kann[32]. Sind neue Daten erforderlich, so können diese per *AJAX* (*Asynchronous JavaScript and XML*) oder mit vergleichbaren Technologien jederzeit *asynchron* im Hintergrund hinzugeladen werden. Dies bietet neben dem Vorteil, dass Ladezeiten minimiert werden können, bzw. seltener komplette Inhalte nachgeladen werden müssen, die Eigenschaft, dass Programme nicht erst auf den Client, also den heimischen PC, oder den Firmenrechner geladen und installiert werden müssen. Gerade in einem sich ständig in Bewegung befindlichen Softwaremarkt bietet dies den zusätzlichen Vorteil, dass fortlaufend Anpassungen an einem Programm vorgenommen werden können, ohne dass dies dem Nutzer angekündigt werden muss. Insgesamt kann den Rich Internet Applications assistiert werden, dass sie in der Lage sind, die Entwicklung im Bereich des Software Engineerings, nicht nur im Bezug auf die Vertriebsform über das Internet, in völlig neue Bahnen zu lenken. Sind bis heute Betriebssysteme wie *Microsoft Windows*, Textverarbeitungsprogramme wie *Microsoft Word* oder Bildverarbeitungsprogramme wie *Adobe Photoshop* der Standard, so könnten dies in Zukunft komplett- oder zumindest überwiegend internetbasierte Applikationen sein[33]. Denkbar sind so völlig neuartige Geschäftsmodelle, bei denen beispielsweise nicht mehr pro Installation ein größerer Betrag für die Software gezahlt werden muss, sondern kleinere Beträge für die Nutzung der Software, wie es zum Teil schon im Bereich der Musik- oder Video-/TV-Vermarktung üblich ist[34]. Insgesamt lässt sich also festhalten, dass RIAs nicht nur durch die weitgehende Plattformunabhängigkeit, den vereinfachten Update- und Upgradeprozess, die leichtere Verfügbarkeit über das Internet sowie der vereinfachten Einbindung multimedialer Inhalte auftrumpfen, sondern allgemein von einer potentiell höheren Dynamik bei den Entwicklung und Vermarktung von internetbasierten Services profitieren können.

29 http://mail.google.com/
30 http://docs.google.com/
31 http://www.adobe.com/de/products/flashplayer/productinfo/features/
32 http://code.google.com/p/swfobject/
33 https://www.photoshop.com/express/
34 vgl. Rifkin, J. (2007)

2.2 Der Sicherheitsaspekt in RIAs

Natürlich ist der Sicherheitsaspekt ein ernst zu nehmender Punkt, wenn es darum geht, sich auf weitgehend internetbasierte Services verlassen zu müssen. So ließen sich bei bestehenden Sicherheitslücken durch etwaige „boshafte“ Veränderungen der RIAs Schadcode auf dem Rechner des Nutzers installieren, oder sensitive Daten ausspionieren. Um diesem Problem entgegenzutreten, sind zumindest die bereits angesprochenen, am weitesten verbreitesten Technologien mit einer sogenannten *Security-Sandbox*[35] ausgestattet, welche den Computer auf dem die Rich Internet Application läuft, effektiv vor unerwünschtem Zugriff schützt. Dies birgt wiederum den Nachteil eines beschränkten Zugriffes auf für herkömmliche Programme verfügbare Systemressourcen. So ist es zum Beispiel nicht möglich, aus dem Adobe Flash-Player auf Daten zuzugreifen, die auf dem Rechner des Clients abgelegt sind. Ebenso können keinerlei Daten, so sie denn nicht für das Caching verwendet werden, auf dem Client Rechner abgelegt werden. Es bietet sich im Falle des Adobe Flash Players lediglich die Option, die *Cookie*-Funktionen des Browsers zu verwenden, in den die Applikation eingebettet ist. Genau hier setzt Adobe‘s AIR an, das auf dem Client-Rechner installiert, Flex-basierten Applikationen den Zugriff auf das Dateisystem ermöglicht.

2.3 Nachteile aktueller RIAs

Neben der schon angesprochenen Security-Sandbox Problematik und dem damit zusammenhängenden limitierten Zugriff auf für herkömmliche Applikationen verfügbare Systemressourcen existieren noch eine Reihe weiterer Probleme und Hindernisse für die Verbreitung von Rich Internet Applications. Ein vielfach genanntes Problem ist die Einschränkung durch allzu limitierte Programmbibliotheken und lange vernachlässigte Skriptsprachen, wie insbesondere JavaScript. Nach einer durchgehenden Unterstützung von *Objekt Orientierter Programmierung* (OOP)[36], wie es bei Desktopapplikationen auf Basis von *Java* oder *C++* schon seit geraumer Zeit etablierter Standard ist, suchte man im Bereich der Internetbasierten Skriptsprachen lange Zeit vergebens. Erst mit des aktuell in der Entwicklung befindlichen ECMA Standards der 4. Generation[37] sowie Adobe‘s ActionScript 2 und 3, scheint sich dies zu ändern. Somit wird es Entwicklern ermöglicht, immer komplexere und leistungsfähigere Programme für das Web zu entwickeln, welche zukünftig sicher auch auf Features wie *Multi-Threading*[38], der Unterstützung mehrerer Prozessoren sowie die Verwendung der Grafik-Hardware auf der jeweiligen Plattform unterstützen werden.

35 vgl. http://livedocs.adobe.com/flex/201/html/05B_Security_176_01.html
36 vgl. http://livedocs.adobe.com/flex/2/docs/00001837.html
37 vgl. http://www.ecmascript.org/es4/spec/overview.pdf
38 vgl. http://blog.fitc.ca/post.cfm/multi-threading-is-coming-sort-of-to-flash-player-10

2.3.1 Gestiegene Anforderungen

Mit den immer komplexer werdenden Inhalten geht allerdings unvermeidbar ein ständig größer werdender Anteil an Code einher, der zunächst auf den Client geladen werden muss, bevor der Service funktionieren kann. Durch asynchrones Nachladen bestimmter Inhalte lässt sich dies zwar weitestgehend minimieren, es bleibt jedoch dabei, dass eine Fülle an Komponenten bereits in der Basisapplikation integriert werden müssen, wodurch die Ladezeit und die Anforderungen an den verwendeten Client unvermeidbar steigen.

Ein weiterer bedeutender Nachteil von Rich Internet Applications ist neben der beschränkten Indizierfähigkeit des kompilierten Binär-Codes durch Suchmaschinen, die mangelnde Unterstützung auf mobilen Clients. So unterstützen viele mobile Clients, wie beispielsweise das *Apple iPhone* oder auf *Windows Mobile* basierende Smartphones anderer Anbieter, Flash oder JavaScript nicht oder nur äußerst eingeschränkt[39]. Allerdings wird auch hier ein Umbruch erwartet, da sowohl die Software- als auch die Hardwareentwicklung in diesem Bereich unaufhaltbar voranschreitet und auch das – mittlerweile auf dem Massenmarkt angekommene – mobile Internet zunehmend performanter wird.

39 vgl. http://www.engadget.com/2008/03/19/adobe-flash-for-iphone-might-be-a-little-harder-than-we-thought/

2.4 Vorzüge einer Rich Internet Application zum Testing von VUI-Prototypen

Warum habe ich mich also dafür entschieden, die Entwicklung einer Rich Internet Applikation zum Testen von low-fidelity Dialogprototypen mittels Adobe Flex voranzutreiben?

Abgesehen von den bereits weiter oben beschriebenen Nachteilen des RIA-Ansatzes bietet dieser Ansatz vor allem in Bezug auf mein Konzept einer Prototyping- und Testing-Anwendung von (Voice-) User-Interface Prototypen eine Vielzahl an Möglichkeiten, die mir herkömmliche Desktop-Anwendungen nicht, oder nur eingeschränkt und mit erhöhtem Aufwand verbunden, bieten. Ich möchte hier insbesondere noch einmal die Möglichkeit des *verteilten* Testings über das ebenfalls schon beschriebene *Client-Server* Szenario nennen. Diese Herangehensweise der Kombination flashbasierter Clients auf der einen und dem (Wowza) Media-Server auf der anderen Seite bietet mir die Möglichkeit, sowohl Nutzertests orts- und plattformunabhängig durchzuführen als auch die Entwicklung von VUI-Prototypen auf diese Art und Weise zu unterstützen und zu vereinfachen. So könnten mehrere VUI-Designer beispielsweise gleichzeitig an einem Projekt arbeiten, wobei ein Designer lediglich die Dialogstruktur festlegt und der andere in einem Audio-Recording Studio mit geschulten Sprechern für die Aufnahmen sorgt. Dank des Internetbasierten Ansatzes sind der Ortsunabhängigkeit tatsächlich keine Grenzen gesetzt. So könnte beispielsweise an einem Tag das deutschsprachige Dialogdesign in Stuttgart am Fraunhoferinstitut durchgeführt werden und am nächsten Tag in San Francisco das deutschsprachige Dialogdesign in eine englische Variante überführt und die Systemausgaben mit *native speakers* aufgenommen werden. Ebenfalls denkbar ist die zusätzliche Implementation und Bereitstellung eines *Spectator*-Clients, mit dessen Hilfe Wizard-of-Oz-Tests an jedem Ort der Welt angeschaut werden und gegebenenfalls Auffälligkeiten protokolliert werden können.

Es existieren zwar auf Desktop-Ebene etliche professionelle Produkte, die das Usability Testing vor allem von Internetseiten und desktopbasierten Programmen auf ähnliche Art unterstützen. Diese Produkte sind jedoch zumeist, wie *Techsmith's Morae*[40], weder plattformunabhängig noch kostenlos, bzw. überall verfügbar. Das in dieser Arbeit vorgestellte Konzept sowie der implementierte Prototyp sollen also – unabhängig von dem beschriebenen Anwendungsfall des Wizard-of-Oz-Testings von VUI-Prototypen – auch auf diesem Gebiet Alternativen aufzeigen. So ließe sich eine *screen-capturing* Komponente, wie sie beispielsweise in *Morae* verwendet wird, auch in Flash/Flex mittels *VNC-Protokoll*[41] bequem umsetzen und in mein Konzept integrieren.

40 http://www.techsmith.de/morae.asp
41 Projekt FlashLight VNC (http://www.wizhelp.com/flashlight-vnc/)

Ein weiterer Vorteil des Client-Server Ansatzes sind im übrigen die gleichbleibenden und verhältnismäßig geringen Anforderungen an die Clients, wobei es in Bezug auf die Performanz des Gesamtsystems vor allem auf die Internetanbindung der Clients und die Performanz des verwendeten Servers ankommt. Vor allem während eines Wizard-of-Oz-Tests ist es wichtig, eine geringstmögliche Verzögerung zu gewährleisten. Im lokalen Netzwerk und über Wireless Lan stellte diese Anforderung, zumindest was das Live-Streaming von Audioinhalten anbelangt, kein Problem dar. Die Latenz bei der Kommunikation von Wizard und Benutzer über den Media-Server liegt im lokalen Anwendungsszenario deutlich unter 0,2 Sekunden, wobei nicht nur von der Qualität der Netzwerkverbindung abhängt, sondern auch von der verwendeten Puffergröße auf Seiten der Clients. Es kann allerdings nicht ausgeschlossen werden, dass die Latenz bei einer Internetverbindung deutlich ansteigt. Wer schon einmal mit Skype oder sonstigen Voice-Over-IP-Anbietern telefoniert hat, wird die Problematik einer verzögerten Übermittlung von Sprachpaketen nachvollziehen können.

2.5 Vorzüge von Adobe Flex

Zunächst einmal bietet mir Adobe Flex als Entwickler, verglichen mit anderen internetbasierten *Frameworks*, den entscheidenden Vorteil, dass die in Flex Verwendung findende Skriptsprache *ActionScript 3* (*AS3*), im Vergleich zu den JavaScript-Engines aktuell führender Webbrowser,[42] die wesentlich mächtigere und ausgereiftere ist. Zwar existieren mittlerweile etliche Erweiterungen für JavaScript, die dem Entwickler vieles erleichtern sollen, wie beispielsweise die JavaScript-Bibliotheken *Prototype*[43], *MooTools*[44] oder *jQuery*[45]. Ganz unabhängig davon wie ausgereift diese Bibliotheken auch sind, so können sie über die Unzulänglichkeiten der aktuellen verwendeten JavaScript-Versionen im Vergleich zu AS3 nicht hinwegtäuschen. Vor allem in Bezug auf die Umsetzung objektorientierter Programmierung, welche im Desktopbereich bereits seit über 10 Jahren der Standard ist und insbesondere in den vorherrschenden Programmiersprachen *Java*, *C++* und C# Anwendung findet. JavaScript bietet, wie in älteren Versionen von Adobe‘s (vormals Macromedia‘s) ActionScript, lediglich die Unterstützung einer auf sogenannten *Prototypen*[46] basierenden *(Pseudo-)*OOP-Technik. Während AS3, ganz wie aus Java oder C++ bekannt, auf Klassenhierarchien aufbaut, kann mit den aktuell verbreiteten *JavaScript-Dialekten* und *-Bibliotheken* eine objektorientierte Programmierung nur ansatzweise realisiert werden. Ab Version 2.0 soll allerdings auch JavaScript die vierte Edition

42 u.a. Internet Explorer 6/7, Mozilla Firefox 2/3, Opera 9.6, Apple Safari 3
43 http://www.prototypejs.org/
44 http://mootools.net/
45 http://jquery.com/
46 vgl. Noble et al. (1995)

des *ECMAScript-Standards* (*ES4*)[47] implementieren und dementsprechend, wie heute schon durch AS3 realisiert, eine klassenbasierte OOP ermöglichen.

2.5.1 Abwärtskompatibilität

Bei solchen einschneidenden Änderungen an einer Programmiersprache stellt sich selbstverständlich immer die Frage nach der Abwärtskompatibilität, also ob Implementationen, die den neuesten Standard einer Programmiersprache verwenden, auch auf älteren Plattformen lauffähig sind. Da die Interpretation der JavaScript-Programme im Allgemeinen durch den verwendeten Web Browser geschieht, muss bei der Entwicklung eines JavaScript-Programms darauf geachtet werden, dass die JavaScript-Applikation auf den am weitesten verbreiteten Browsern lauffähig ist. Ein großes Problem hierbei ist, dass heute noch gut ein Viertel der Nutzer[48] einen Browser verwenden, der sowohl technisch nicht mehr auf dem neuesten Stand ist, als auch viele anerkannte Standards der W3C (u.a. CSS, DHTML) noch nie zu 100% unterstützt hat und zudem nicht mehr weiter entwickelt wird: *Microsoft's Internet Explorer 6*. Allerdings gibt es nicht nur aufgrund der weiten Verbreitung veralteter Browser Probleme. Vielmehr unterscheiden sich die Implementationen der verschiedenen Browser in einigen Details, wodurch man als Entwickler unweigerlich auf vielerlei unliebsame Überraschungen trifft.[49] Zudem verzichten viele Nutzer gänzlich auf JavaScript, wodurch die meisten JavaScript basierten *Web 2.0*-Anwendungen nicht – oder nur äußerst eingeschränkt – verwendet werden können.

Auch hier spielt Adobe Flash mit AS3 seine Vorteile voll aus. Alleine schon aufgrund der Tatsache, dass der aktuellste Flash-Player (Version 9), in dem auch die per Adobe Flex entwickelten Programme laufen, unabhängig vom verwendeten Browser laut Statistik[50] weltweit bei über 95% aller Anwender installiert ist, können solche Kompatibilitätsprobleme weitgehend ausgeschlossen werden. Dementsprechend kann viel Zeit, die gewöhlich für das Debugging einer JavaScript-basierten Rich Internet Application unter verschiedensten Plattformen aufgebracht werden muss, eingespart und für andere Dinge verwendet werden. In Bezug auf die Performanz der RIA lässt sich zwar konstatieren, dass der Flash-Player in verschieden Browsern unterschiedlich schnell läuft, vor allem wenn mit (halb-)transparenten Flashobjekten innerhalb einer Website gearbeitet wird[51]. Es handelt sich aber hier bei weitem nicht um die Performanceunterschiede die bei den JavaScript-Engines der verschiedensten Browser anzutreffen sind[52], ganz von der nicht immer kompatiblen JavaScript Implementation und der

47 vgl. http://www.ecmascript.org/es4/spec/overview.pdf
48 vgl. http://marketshare.hitslink.com/report.aspx?qprid=2
49 vgl. http://www.quirksmode.org/dom/compatibility.html
50 vgl. http://www.adobe.com/products/player_census/flashplayer/version_penetration.html
51 vgl. http://renaun.com/blog/2006/06/14/40/
52 vgl. http://www.heise.de/ct/08/19/182/

nicht immer vollständig gegebenen Standardkonformität nach ECMAScript abgesehen.

Allerdings liegt auch in der Proprietarität und vor allem der Tatsache, dass es sich bei Adobe um einen kommerziellen Anbieter handelt, wiederum ein oft genannter Nachteil des Adobe Flash Players. Obwohl Adobe mittlerweile Teile des Quelltextes des Flash Players freigegeben und per Spende an die Mozilla Foundation übergeben hat[53] sowie schon seit längerem alternative Player für kompilierte *.swf*-Dateien erhältlich sind (s.u.), handelt es sich weiterhin um eine *closed-source* Software[54]. Im Gegensatz zu auf JavaScript basierenden Programmen wird im Flashplayer zudem lediglich der bereits kompilierte Binärcode geladen, wobei auf den Quelltext der Flash-, bzw. .swf-Datei nicht mehr ohne weiteres zugegriffen werden kann. Trotz der Tatsache, dass es sich bei Adobe‘s Flash-Player um ein proprietäres Produkt handelt, gibt es mittlerweile eine Vielzahl von Open Source Projekten rund um das Thema Adobe Flash[55,56].

Die im Adobe Flex SDK enthaltenen Komponenten, welche bei Adobe Flex bereits enthalten sind und für die Erstellung von Flex-Applikationen verwendet werden können, sind im übrigen quelloffen und können offiziell und ohne weitere Erlaubnis beliebig angepasst und erweitert werden.

2.5.2 Debugging

Im Bezug auf das Debugging von JavaScript-Code gibt es zwar mittlerweile für die meisten aktuell erhältlichen Browser mehr oder weniger ausgereifte Tools,[57] welche die Arbeit ungemein erleichtern können. Verglichen mit den Möglichkeiten in Adobe Flex, speziell dem enthaltenen Debugger der Eclipse-basierten Adobe Flex IDE[58], bleiben jedoch auch hier zumeist viele Wünsche offen. Es kann allerdings als Nachteil angesehen werden, dass zum Debugging eines Flex-Projekts jedes Mal das komplette Projekt kompiliert werden muss, was unter Umständen einige Zeit in Anspruch nehmen kann.

2.5.3 Multimedia

Da Adobe Flash/Flex allgemein vor allem unter dem Stichwort Multimedia bekannt geworden ist[59], ist die Unterstützung für alle Arten von Multimediainhalten ausgezeichnet. So können nicht nur ohne Probleme Audio- oder Videosequenzen eingebettet werden, sondern es besteht

53 http://www.mozilla.org/projects/tamarin/
54 vgl. http://www.theregister.co.uk/2008/09/26/adobe_google_mozilla_tensions/
55 http://osflash.org/projects
56 vgl. http://www.infoq.com/news/2007/12/top-10-flex-misconceptions
57 besonders hervorzuheben sei hier das Projekt *FireBug* für den Mozilla Firefox Browser
58 *Integrated Development Environment* (integrierte Entwicklungsumgebung)
59 vgl. u.a. http://www.youtube.com

auch die Möglichkeit beliebige Vektorgrafiken im *Adobe-Illustrator-* (*.ai*) oder *SVG-Format* (*.svg*) zu integrieren. Dies erfordert bei JavaScript-Applikationen in jedem Fall zusätzliche Module[60], wobei beispielsweise die Anzeige von SVG-Dateien oder das Abspielen von Audio- und Videostreams von den wenigsten Browsern nativ unterstützt wird. Gerade die Kombination von *Adobe Flex* mit einem *Flash Media-Server* bietet hier eine Vielzahl an Möglichkeiten, welche ich im Folgenden noch detailliert beschreiben werde.

60 u.a. Real Player, Apple Quicktime Player, Windows Media Player

2.5.4 Fazit

Abgesehen von alternativen Produkten im Bereich der Rich Internet Applications wie Microsoft‘s Silverlight, welche ich aus verschiedenen Gründen nicht in Erwägung gezogen habe, bin ich zu dem Schluss gekommen, dass sich mir mit der für Studenten kostenlos erhältlichen[61] Adobe Flex Entwicklungsumgebung in Kombination mit dem ebenfalls für meine Zwecke kostenlosen Wowza Media Server[62] ein Paket anbietet, das sich in idealer Weise für die Umsetzung meines Konzeptes einer RIA zum Prototyping und Testing von VUI-Konzepten eignet. Zudem ist es für mich persönlich von Vorteil, dass ich bereits eingehende Erfahrung mit Adobe- bzw. Macromedia-Flash gesammelt habe sowie auch was die Backend-Implementation mit der Java-Basierten Wowza IDE[63] anbelangt, einige Vorerfahrungen mitbringen konnte. Über die Vorzüge eines internetbasierten Ansatzes einer Rich Internet Application habe ich ebenfalls entsprechende Argumente vorgebracht und ich gehe davon aus, dass ich mit dem von mir implementierten Prototypen, dem Endergebnis dieser Bachelorthesis, einige der genannten Vorzüge des RIA Ansatzes demonstrieren kann.

61 https://freeriatools.adobe.com/flex/
62 http://www.wowzamedia.com/store.html
63 http://www.wowzamedia.com/labs.html#wowzaide

3. Allgemeine Einführung zum Thema Sprachdialogsysteme

3.1 Sprachdialogsysteme

Um einen kurzen Überblick in das Anwendungsgebiet des für diese Arbeit angefertigten Prototypens einer Rich Internet Application zu geben, möchte ich zunächst einmal eine allgemeine Einführung in das Themengebiet der Sprachdialogsysteme vornehmen.

Sprachdialogsysteme gewinnen in den letzten Jahren zunehmend an Bedeutung und sind schon seit einiger Zeit auf dem Markt vertreten. Insbesondere wenn es darum geht, relativ einfach strukturierte Anfragen auf einer Datenbank zu bearbeiten, bieten sich Sprachdialogsysteme an, um diese Art von Aufgaben von menschlichen Operateuren in Call-Centern zu übernehmen. Es ist also unter anderem auf die Sprachdialogsysteme zurückzuführen, dass in diesem Bereich überhaupt erst eine überwiegende Automatisierung von Kommunikationsprozessen einsetzen konnte. Meistens kommen jedoch auch weiterhin herkömmliche Call-Center zum Einsatz, um entweder für den Notfall eine Möglichkeit vorzusehen, die Kommunikation weiterzuführen oder um anspruchsvollere Themengebiete abzudecken, welche bislang nur unzureichend durch den Einsatz von Sprachdialogsystemen bedient werden können.

Vor allem bei den großen Telekommunikationsunternehmen sowie beispielsweise der telefonischen *Reiseauskunft* der *Deutschen Bahn*, sind diese Systeme bereits seit längerem, auch dank immer besser werdender Hard- und Softwarekomponenten, mit zunehmendem Erfolg im Einsatz und einer breiten Masse der Bevölkerung bekannt. In diesem Fall handelt es sich um serverbasierte Sprachdialogsysteme, welche über herkömmliche Telefonie-Schnittstellen angesprochen werden können. Durch immer leistungsfähigere und schrumpfende Hardwarekomponenten, sind mittlerweile sogar ausschließlich clientseitige Systeme, wie beispielsweise in neueren *Automobilen der Mittelklasse*[64], im kommerziellen Einsatz. Hierbei muss klar unterschieden werden zwischen Geräten, die lediglich eine Sprachausgabekomponente beinhalten[65] und Systemen, die eine, wie auch immer geartete, Kommunikation mit dem Nutzer ermöglichen.

Ein Sprachdialogsystem (SDS) besteht aus mehreren Komponenten, oder Subsystemen, wobei man es sowohl auf der Hardware-Ebene als auch auf der Software-, bzw. funktionalen Ebene beschreiben kann.

64 vgl. http://www.openpr.de/news/156919/Tele-Atlas-ermoeglicht-sprachgesteuerte-Navigation-in-Mercedes-Fahrzeugen.html

65 z.B. die meisten mobilen Navigationssysteme

3.2 Systemarchitektur

Von der Hardware-Ebene aus betrachtet besteht ein nach dem klassischen Client-Server-Prinzip arbeitendes SDS aus verschiedenen miteinander verbundenen, aber autonomen Komponenten. Üblicherweise handelt es sich clientseitig um ein einfaches Telefon, welches die Möglichkeit bietet, ein SDS, genauer – analog zum allseits bekannten Internetbrowser – einen *Voice-Browser* über eine Telefonnummer anzurufen. Das Telefon sollte neben der Eingabemöglichkeit über das Mikrofon und der Ausgabemöglichkeit über den Lautsprecher eine Tastatur besitzen, mit deren Hilfe auch während des Gesprächs Ton-, bzw. genauer DTMF-Signale übermittelt werden können. Die Eingaben, also Sprach- oder Tonsignale, werden vom Voice-Browser am anderen Ende der Leitung, also genau genommen auf der Serverseite, an eine Spracherkennungskomponente weitergeleitet. Sobald die Nutzereingaben vom Server entsprechend verarbeitet und eine Reaktion ausgewählt worden ist, werden die Systemausgaben in ein für den Voice-Browser lesbares Format (üblicherweise VoiceXML) umgewandelt und an den Browser gesendet. Dieser interpretiert die Daten wie ein Web-Browser und übermittelt die Ausgaben[66] als Audiosignal an das Telefon des Anrufers. Die XML-Metasprache VoiceXML (s.o.) ist ein vom Word-Wide-Web-Consortium (W3C)[67] ausgegebener Standard, der für die in diesem Kontext relevanten, im Voice-Browser zu verarbeitenden Aktionen spezifiziert wurde.

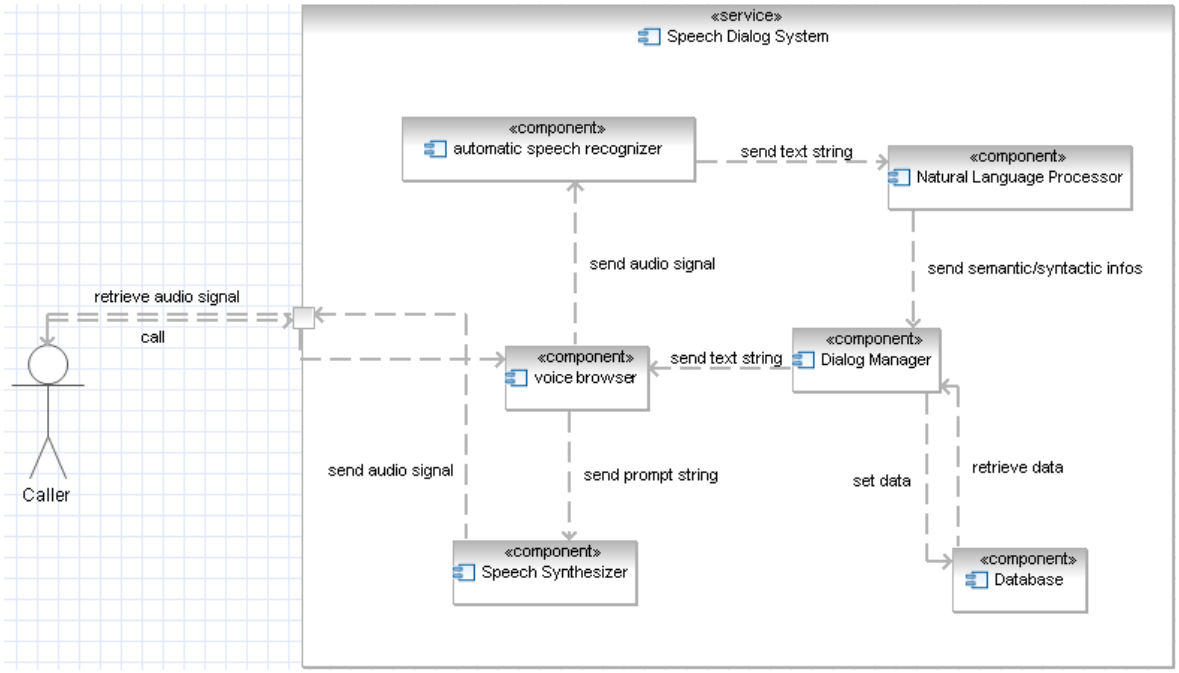


Abbildung 1: Komponenten eines Sprachdialogsystems

66 üblicherweise per serverseitig synthetisierter oder zuvor aufgenommener und konkatenierter Sprache
67 http://www.w3.org/ (überwacht sämtliche Web-Standards wie HTML, CSS etc.)

Auf funktionaler Ebene können die serverseitigen Vorgänge natürlich noch weiter aufgeteilt werden. Die Nutzereingaben werden also üblicherweise zunächst von einem automatischen Spracherkenner (ASR, *automatic speech recognizer*) in Textrepräsentationen (*Strings*) zerlegt, welcher diese Informationen an eine Komponente weitergeleitet, die syntaktische und semantische Informationen aus dem erkannten String extrahieren kann. Sobald das System erkannt hat, was der Nutzer bzw. der Anrufer fordert, kann über Folgeaktionen entschieden werden. Handelt es sich bespielsweise um eine Anfrage nach bestimmten Daten, so muss eine Datenanfrage von der entsprechenden Datenbank durchgeführt werden. Handelt es sich um eine Dateneingabe, so müssen die entsprechenden Eingaben entweder temporär oder permanent in der Datenbank festgeschrieben werden. Diese Aufgaben werden gewöhnlich über einen *Dialogmanager* durchgeführt, welcher als zentrales Modul der Sprachapplikation gesehen werden kann. Der Dialogmanager interpretiert also die erkannten Nutzereingaben und entscheidet über mögliche Folgezustände der Applikation.

Sobald die Folgezustände ausgewählt und entsprechende Backend-Operationen durchgeführt worden sind, kommt wiederum der Voice-Browser ins Spiel und entscheidet entsprechend der im Dialogdesign (z.B. VoiceXML) festgelegten Bedingungen, ob eine und gegebenenfalls welche Systemausgabe erfolgen muss. Die anhand erkannter und somit gefüllter Variablen (*Slots*) sowie in der VoiceXML-Datei festgelegten Spachelemente zusammengesetzte Textrepräsentation der Systemausgabe wird daraufhin an einen Sprachausgabe-Generator weitergeleitet, der entweder zuvor ausgenommene Sprachelemente zusammengefügt oder den String entsprechend mit einem *Text-To-Speech*-Modul synthetisiert und an den Nutzer sendet.

Da im Bereich der sprachbasierten Kommunikation auf eine möglichst flüssige Kommunikation zwischen den Kommunikationsteilnehmern Wert gelegt werden sollte und bereits eine verhältnismäßig kurze Verzögerung von wenigen Sekunden unter Umständen als sehr störend wahrgenommen werden kann, muss es bei einem SDS als essentiell angesehen werden, die Zeit zwischen Nutzerein- und Systemausgabe zu minimieren. Dies spielt vor allem dann eine ganz besonders große Rolle, wenn es zusätzlich zu Verzögerungen bei der Sprachübermittlung kommt. Bei Sprachdialogsystemen kommt natürlich hinzu, dass sich nicht nur die Bearbeitungszeit als essentiell für eine hohe Nutzerzufriedenheit erwiesen hat, sondern auch die Qualität der Informationsverarbeitung. So müssen bei der Erstellung von Sprachdialogsystemen für eine maximale Qualität alle beteiligten Komponenten sowohl für sich als auch im Zusammenspiel miteinander perfekt funktionieren. Dem entsprechend sind die Anforderungen an sämtliche Komponenten eines SDS ausgesprochen hoch und nicht zu vergleichen mit den Anforderungen, die beispielsweise an eine herkömmliche Internetseite gestellt werden.

Ein serverseitig noch so gut funktionierendes SDS kann sich allerdings beim alltäglichen Einsatz als völlig unbrauchbar erweisen, wenn nicht entsprechend auf etwaige Fehler reagiert wird oder beispielsweise die sprachliche Qualität der Systemausgaben zu wünschen übrig lässt. So sollte unabhängig vom serverseitig verwendeten System, also unter anderem der *Spracherkennung*, dem *Dialogmanager* und der *Sprachsynthese*, in einem nutzerzentrierten Designprozess[68] ein Interface erstellt werden, das den Wünschen eines überwiegenden Teils der zu erwartenden Nutzer entspricht und auf eine Vielzahl zu erwartender Probleme reagieren kann.

3.3 Voice-User-Interface-Design

Ein Voice User Interface arbeitet nach einem linguistischen Paradigma (Wörter als atomare Einheiten) und kann definiert werden als ein „unimodales Sprachein-/Sprachausgabesystem, das einen Dialog über ein einziges Thema mit einem einzigen Nutzer beschreibt, welcher in einer einzigen Sprache durchgeführt wird.“[69] Selbstverständlich lässt sich diese Definition noch weiter führen, ich möchte sie jedoch zunächst einmal so stehen lassen und habe mein System auf dieser Grundlage entwickelt.

Insbesondere im Hinblick auf die zumeist ausgesprochen zeit- (und damit vor allem auch) kostenintensive Entwicklung der serverseitigen Komponenten eines dialogorientierten SDS[70], bleiben oftmals nur sehr wenig Mittel für ein entsprechend nutzerfreundliches Design übrig und so wird dieses nicht selten vernachlässigt, oder es werden gar erst nach Inbetriebnahme des Systems wichtige Verbesserungen im Dialogdesign vorgenommen. Typische Fehler wie beispielsweise Endlosschleifen (*Deadlocks*) im Dialog oder sprachlich missverständliche Systemausgaben lassen sich durch entsprechend sorgfältiges Dialogdesign auf verhältnismäßig einfache Weise unterbinden.

Losgelöst von der hinter einem SDS liegenden Architektur soll im Folgenden dem Voice-User- Interface (VUI), also der Schnittstelle zwischen Nutzer und SDS, besondere Aufmerksamkeit zukommen. In dieser Arbeit werden dementsprechend die serverseitig für das Funktionieren einer Sprachapplikation benötigten Komponenten nicht weiter behandelt. Ich möchte dementsprechend den glücklichen Umstand, dass durch immer leistungsfähigere Hardware und immer ausgereiftere Softwarekomponenten eine dialogorientierte, natürlichsprachliche Kommunikation zwischen Mensch und Maschine nicht mehr undenkbar erscheint, an dieser Stelle ausnutzen und mich vollständig auf den Aspekt des Voice-User-Interface Designs be-

68 vgl. DIN EN ISO 13407 (1999)
69 http://www.acm.org/ubiquity/book/pf/v6i15_harris.pdf
70 Harris (2004), S. 26

schränken.

3.3.1 Designelemente im VUI-Design

Grundsätzlich kann beim Design von Voice User Interfaces, entsprechend der VoiceXML Spezifikationen, zwischen den drei Designelementen *Call-Flow*, *Grammatik* und *Prompts* unterschieden werden, wobei diese Elemente wiederum untereinander in Beziehung stehen. Besondere Aufmerksamkeit sollte im Designprozess zudem dem Aspekt der Fehlerbehandlung und -vermeidung zukommen.

3.3.2 Call-Flow Design

Als *Call-Flow* wird der per Entwurf festgelegte Ablauf aller möglichen Dialoge zwischen Benutzer und System bezeichnet. Bei einfachen Sprachdialogsystemen, also menübasierten oder ausschliesslich systeminitiierten Systemen, wird der Call-Flow in der Entwurfsphase explizit vorgegeben (als eine Art Flussdiagramm) und ist dementprechend fixiert. Bei aktuelleren Sprachdialogsystemen wird jedoch im Sinne einer dialogorientierten Kommunikation zwischen Nutzer und System die sogenannte gemischte Initiative (*Mixed-Initiative*) unterstützt und bevorzugt.

Bei Sprachdialogsystemen die nach dem Muster der gemischten Initiative arbeiten, wird der Nutzer zumeist nicht explizit zu einer bestimmten Eingabe aufgefordert, sondern es können auch auf Initiative des Nutzers mehrere Eingaben gleichzeitig abgefragt werden. In diesem Fall ist der Call-Flow weniger starr ausgelegt und es wird in der Regel auf einer höheren Ebene eine Struktur vorgegeben, wobei sich der Call-Flow viel mehr auf Basis eines Zustandsmodells ergibt. Je nach Dialogzustand werden bei einem Mixed-Initiative Dialog per Design definierte *Slots* gefüllt und gegebenenfalls fehlende Slots auf Initiative des Systems explizit nachgefragt. Die Auswahl des Dialogzustands ergibt sich in diesem Fall aus einer Relation aus dem aktuellen Systemzustand sowie den erkannten Nutzereingaben.

Da Mixed-Initiative-Systeme gewissermaßen die Grundlage für eine natürlichsprachliche Kommunikation zwischen Mensch und Maschine darstellen, habe ich mich dazu entschlossen, meine Applikation ebenfalls auf Grundlage der *Mixed-Initiative-Paradigmas* zu erstellen. Dementsprechend werden im vorhandenen Prototyp zur Zeit noch keine Menüstrukturen unterstützt.

3.3.3 Grammatik-Design

Die Grammatik enthält sämtliche unterstützten und vom System interpretierbaren Aussagen, die vom Nutzer verwendet werden können. Um die möglichen Eingaben zu einem bestimm-

ten Dialogzustand zu minimieren, bietet es sich allerdings an, bestimmte Schlüsselwörter (*Keywords*) nur während eines Dialogzustands zu aktivieren. Darüber hinaus existieren zumeist noch *globale Slots*, welche aus jedem beliebigen Dialogzustand heraus aktiviert werden können (z.B. „Hilfe“, „Menü“). Da sich aus den erkannten (oder nicht erkannten) Slots der jeweilige Folgezustand meistens direkt ergibt, ist das Grammatikdesign eng mit dem Call-Flow-Design verwoben.

Wird zu einem Zeitpunkt im Dialog nach einer zuvor spezifizierbaren Zeitspanne keine verwertbare Nutzereingabe erkannt, so wird ein sogenanntes *no-match*-Event ausgelöst. Erfolgt überhaupt keine Eingabe, wird analog dazu ein *no-input*-Event ausgelöst. Beide Fälle müssen entsprechend abgefangen und eine entsprechende Fehlerlösungsstrategie angewendet werden. Ein gutes Fehlermanagement ist für ein erfolgreiches Dialogdesign unabdingbar[71]. Grundsätzlich bietet sich für die Simulation solcher Fehler in einer Wizard-of-Oz-Applikation sehr gut an, im aktuellen Prototypen der von mir erstellen Rich Internet Application konnte dies jedoch noch nicht realisiert werden.

Insbesondere wenn solche Fehler wiederholt, oder sogar mehrfach hintereinander auftreten, könnte dies ein Hinweis auf eine mangelnde sprachliche Qualität der Systemausgaben (*Prompts*) sein[72].

3.3.4 Prompt-Design

Das Designelement, auf dem im Folgenden meine Hauptaufmerksamkeit liegen wird ist das Prompt-Design, also das Design der Systemausgaben. Als Systemausgaben dienen entweder zuvor aufgenommene und entsprechend konkatenierte Aufnahmen, welche auf dem Server hinterlegt sind, von einem *Text-To-Speech-Modul* synthetisierte Sprache oder eine Kombination aus beidem. Da sie, im hier betrachteten Szenario eines (unimodalen) SDS, das einzige Mittel der Kommunikation zwischen System und Nutzer sind, können sie als kritisch für die empfundene Qualität eines Sprachdialogsystems bezeichnet werden.[73]

Die vom Nutzer wahrgenommene Qualität der Systemausgaben ist dementsprechend neben dem Satzbau (Syntax) von Faktoren wie Sprecherauswahl (männlich/weiblich) und Prosodie (Satzrhythmus, Intonation, Tonhöhe, Lautstärke, Betonung) abhängig. So können sich aus unzulänglich intonierten Sätzen sehr einfach Mehrdeutigkeiten ergeben, welche unvermeidlich zu Fehlern im Dialog führen können. Beispielsweise führt eine fehlende Stimmhebung bei Fragestellungen in der Regel zu einer gesteigerten Anzahl an no-input Events, da der Prompt

71 vgl. Skantze (2005)
72 vgl. Jurafsky & Martin (2006)
73 vgl. Helander/Kamm (1997)

nicht als Frage verstanden wurde.

3.3.5 Fazit

Ähnlich wie bei grafischen Benutzerschnittstellen (GUIs) lässt sich konstatieren, dass sich, zumindest den Aspekt der Mensch-Maschine-Interaktion betreffend, durch gutes Design von Voice User Interfaces unter Umständen sogar ein minderwertiges und damit weniger kostenintensives Backend[74] ausgleichen lässt, und umgekehrt durch schlechtes VUI-Design selbst das beste und teuerste Backend letzten Endes wertlos sein kann. Wie im Folgenden gezeigt werden soll, lassen sich jedoch eine Vielzahl an mehr oder weniger gut vorhersehbaren Designfehlern in allen drei beschriebenen Designelementen des VUI Design durch sorgfältiges Design und frühes Testing verhindern. Da es sich jedoch als schwierig gestaltet, Designs bereits in der Konzeptionsphase eines Projekts zu testen, da etliche Funktionen des VUIs von bereits funktionierenden Schlüsselelementen (Spracherkennung, Dialogmanager, Text-To-Speech) abhängen, hat sich eine Methode durchgesetzt, welche den Kern meiner Arbeit darstellt und im Folgenden beschrieben werden soll. Der Wizard-of-Oz-Test.

3.4 Wizard-of-Oz-Testing von VUI-Prototypen

Beim sogenannten Wizard-of-Oz-Testing wird bewusst auf eine Implementation des Backends verzichtet, um sich Aspekten widmen zu können, die ausschließlich sprachlicher Natur sind und nur bedingt von der Funktionsweise der sonstigen Komponenten eines SDS abhängen. In einem Wizard-of-Oz Szenario wird ein Usability-Test auf der Grundlage zuvor mehr oder weniger umfassend definierter Elemente wie Call-Flow, Grammatik- und Prompt-Design durchgeführt. Den Part des sprachverarbeitenden Systems übernimmt jedoch ein Mensch, der *Wizard*, dessen Aufgabe es ist, auf Grundlage der zuvor spezifizierten Designelemente, Systemausgaben (*Prompts*) auszuwählen und zu tätigen. Damit die Ansprüche an das, durch den Wizard entsprechend simulierte System und den Wizard selbst nicht zu hoch sind, werden bei einem Test anhand von mehr oder weniger eingeschränkten Szenarien die möglichen Antworten und Folgezustände minimiert. Der Nutzer, der üblicherweise im Glauben gelassen wird, dass er mit einem echten System spricht, befindet sich in einem separaten Raum und ruft das „System“ auf Grundlage von, durch die Versuchsleitung erteilten Anweisungen an. Der Nutzer soll auf diese Art und Weise anhand fester Aufgabenstellungen, Schwachstellen des Systems ausloten, die oftmals erst durch ein Testing mit entsprechend ausgewählten Probanden sichtbar werden. Allerdings lassen sich durch weniger groß angelegte Tests mit 2-3 Versuchsteilnehmern (z.B. Mitarbeitern) bereits frühzeitig viele Schwachstellen im Design aufspüren,

74 serverseitiger und nicht direkt an der Kommunikation beteiligter Anteil des SDS-Gesamtsystems

welche somit schon zu einem sehr frühen Entwicklungsstadium korrigiert werden können.

3.4.1 Probleme bei Wizard-of-Oz-Tests

Das Test-Setup eines Wizard-of-Oz-Tests ist vergleichbar mit dem eines papierbasierten Usability Tests, wobei dem Anwender die Identität des Benutzers verborgen bleibt. Die Umsetzung gestaltet sich entsprechend schwierig, denn es erfordert einen sehr gut vorbereiteten Wizard, damit zum einen die Illusion eines *echten* Systems für den Nutzer aufrecht erhalten werden kann und zum anderen die Ergebnisse entsprechend vergleichbar bleiben. Weicht der Wizard zu stark vom eigentlichen Design ab, so lassen sich unter Umständen nur schwer wirkliche Rückschlüsse über die Qualität des aktuell spezifizierten Designs ziehen. Die Schulung des Wizards kann dementsprechend einer möglichen größeren Testfrequenz kleiner Dialogausschnitte im Wege stehen. Desweiteren ist der Personalaufwand bei einem herkömmlichen Wizard-of-Oz-Test verhältnismäßig hoch, da die Testperson von einer weiteren Person, der Versuchsleitung instruiert und gegebenenfalls noch zusätzlich einem Protokollführer begleitet werden muss.

3.4.2 Eine Rich Internet Application zum Testen von natürlichsprachlichen VUI-Prototypen.

Wie bereits in der Einführung erwähnt, existieren zwar bereits einige mehr oder weniger ausgereifte Tools für das Testen von VUI-Prototypen, es handelt sich jedoch ausschließlich um desktopbasierte Applikationen, die keine Unterstützung für ein ortsunabhängiges Testing bieten (Rick‘s Wizard-of-Oz Tool, Suede[75]). Die einzige bekannte internetbasierte Applikation (Voxeo Designer), ist zwar für ein Rapid Prototyping von Voice User Interfaces ausgelegt, bietet jedoch keine weitere Unterstützung für das Testing dieser Prototypen. Es kann zwar nicht ausgeschlossen werden, dass noch weitere Produkte auf dem Markt existieren, die ein Testing von Rapid Prototypes unterstützen, meine Recherchen lassen jedoch den Schluss zu, dass zur Zeit keine internetbasierte Applikation verfügbar ist, die für das Testing von VUI-Prototypen ausgelegt ist.

Aus diesem Mangel heraus, habe ich mich dazu entschlossen eine internetbasierte Applikation zum Testing von VUI-Prototypen zu entwerfen und prototypisch als „Proof of Concept“ zu implementieren. Hierbei habe ich auf die bereits beschriebene Flex Technologie in Verbindung mit einem Flash Media-Server (Wowza Media Server) zurückgegriffen, wobei ich beide Produkte kostenlos erhalten habe. Die hier beschriebenen Ergebnisse der Implementation stellen demnach einen Zwischenschritt in der Entwicklung eines möglichen Endproduktes dar und dienen in erster Linie der Veranschaulichung des von mir erdachten Konzeptes einer Rich Internet Application zum Testing von VUI-Prototypen. Es ist allerdings bereits möglich diesen Prototypen produktiv einzusetzen um Dialogausschnitte szenarienbasiert auf Fehler im Prompt- oder Grammatik-Design zu testen. Ein Hauptaugenmerk galt hierbei insbesondere der Demonstration des Client/Server Ansatzes und der Verknüpfung verschiedener Clients über einen Media-Server mit der Einbindung eines Voice-Over-IP ähnlichen Ansatzes des Live-Streamings von Audioinhalten.

75 http:// dub.washington.edu/2007/projects/suede/pubs/suede-chi2001.pdf

4. Konzeption und Umsetzung der RIA

Bei der Konzeption der Rich Internet Application habe ich mich für diese Arbeit auf die zentralen Aspekte des Dialogdesigns und insbesondere des Testings von Dialogprototypen beschränkt. Ein besonderes Augenmerk galt hierbei der Kommunikationsschnittstelle zwischen Testteilnehmer und Wizard, welche auch entsprechend implementiert wurde. Neben der Konzeption und Implementation einer Flex-basierten grafischen Benutzerschnittstelle für die verschiedenen Benutzertypen sowie einer XML-Datenbasis, wurde auch ein funktionsfähiges serverseitiges Modul erstellt, welches die Aufgabe übernimmt, die Aktionen des Wizards und des Testbenutzers zu synchronisieren sowie Daten zwischen diesen beiden Clients auszutauschen. Zusätzlich zu den für das Wizard-of-Oz-Testing entscheidenden Nutzertypen *Wizard* und *TestUser* habe ich ebenfalls, zumindest ansatzweise, die Nutzertypen *Admin* und *Designer* mit in den Prototypen eingebunden. Während der Admin die XML-Nutzerdaten der Applikation über ein entsprechendes Interface verwalten kann, wurde für den Designer ein Interface implementiert, das die Erstellung von Dialogdesigns erleichtern soll und ihn darüberhinaus befähigt, *Prompts* (Systemausgaben) aufzunehmen und auf dem Server zu speichern. Für die vereinfachte Verwaltung, das Debugging und zur Veranschaulichung wurde zusätzlich ein Interface für einen sogenannten *SuperUser* implementiert, welches die vier genannten Nutzertypen dieses Prototypen kombiniert.

4.1 Erweiterungsoptionen

Da grundsätzlich noch der Bedarf nach einer Möglichkeit zur vereinfachten Analyse von Testergebnissen besteht, sollte in den nächsten Konzeptions- und Entwicklungzyklen zusätzlich die Möglichkeit vorgesehen werden, die Testergebnisse über eine Flex-basierte Benutzerschnittstelle zu analysieren. Hierbei bietet sich insbesondere die Charting-Funktionalität[76] von Adobe Flex an, welche es ermöglichen würde, auf einfachste Art und Weise quantitative Testergebnisse wie Latenzzeiten oder die Nutzerzufriedenheit zu visualisieren. Die Latenzzeiten während eines Tests werden im aktuellen Prototypen bereits erfasst, eine ausgereifte und dynamische Nachbefragung ist zwar angedacht, muss jedoch noch implementiert werden. Natürlich sollten auch qualitative Daten aus einer solchen Nachbefragung analysiert und für eine Verbesserung der Dialogprototypen herangezogen werden können.

Des weiteren würde die ohnehin vorhandene serverbasierte Implementation über den Wowza Media Server die Möglichkeit bieten, einen zusätzlichen Nutzertypus zu integrieren, der es ermöglicht, die Tests *live* über das Internet zu verfolgen. Ein solcher *Spectator*-Modus

76 vgl. http://livedocs.adobe.com/flex/201/html/charting_012_1.html

könnte neben der Einbindung von Kunden in den Entwicklungsprozess für die Möglichkeit einer Erfassung zusätzlicher Informationen sorgen, um diese an die Testergebnisse und Audioaufnahmen anzuhängen. Einem unbeteiligten Betrachter fällt unter Umständen mehr auf als dem Wizard oder einer eventuell für die Unterstützung der Testperson anwesenden Versuchsleitung. Der Aufwand für solch einen Spectator-Modus wird zwar als nicht allzu hoch angesehen, es wurde jedoch ebenfalls zu diesem Zeitpunkt auf eine weitere Ausarbeitung verzichtet. Dennoch wird eine Einbindung in zukünftige Versionen der Applikation ausdrücklich empfohlen.

4.2 Mögliche Zustände der Applikation

Alle in diesem Prototyp ermöglichten Applikationszustände und -verzweigungen sind im Folgenden Zustandsdiagramm schematisch erfasst.

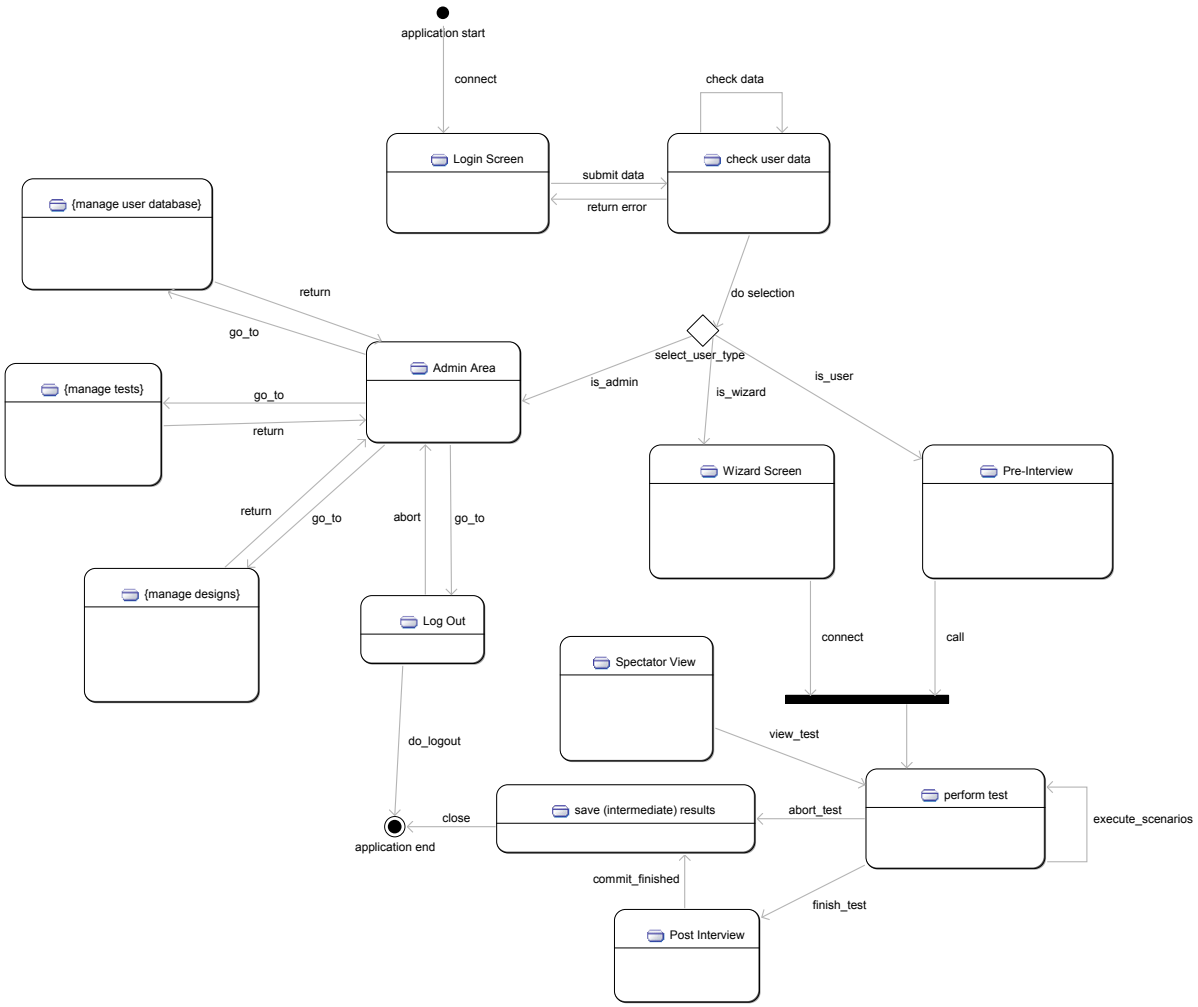


Abbildung 2: Zustandsdiagramm der Applikation

Die schematische Darstellung in diesem Use-Case Diagramm beschreibt den frontendseitigen Informationsfluss dieser Applikation inklusive aller möglichen Verzweigungen.

Ausgehend von diesem vereinfachten Use-Case Diagramm möchte ich zunächst die einzelnen Views (GUI Zustände) der Applikation schrittweise vorstellen. Danach erfolgt noch ein kurzer Überblick über die möglichen Use-Cases mit einem Use-Case Übersichtsdiagramm, bevor die Implementation des Prototypen auszugsweise beschrieben wird. Der bereits angesprochene Meta-Nutzertypus *SuperUser* wird gesondert vorgestellt.

4.2.1 Log-In

Zunächst gelangt der Nutzer über eine einfache URL per Internetbrowser mit installiertem Flash-Plugin auf die Startseite, bzw. den Log-In Screen.

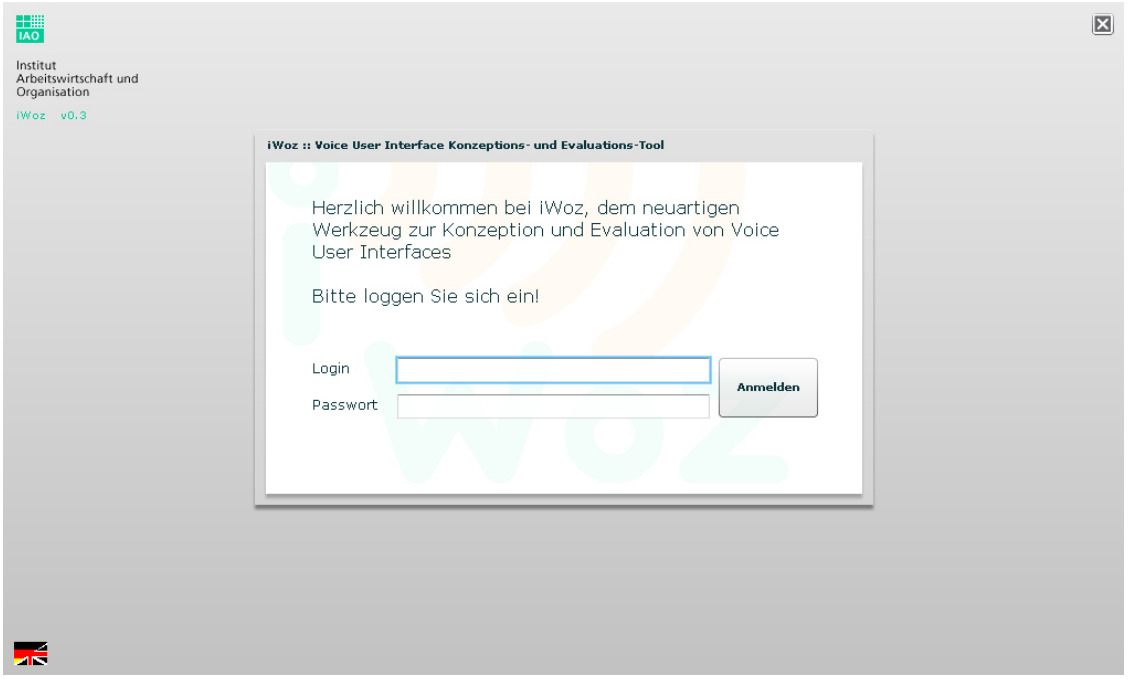


Abbildung 3: iWoz :: Login-Screen inkl. Anwendungslogo und Sprachauswahl

Dieser Screen enthält keine weiteren Auswahlmöglichkeiten, außer der Sprachauswahl (unten links) und einer Eingabemöglichkeit für einen Benutzernamen und ein Passwort. Werden vom Nutzer Logininformationen abgesendet, so wird diese Applikation entweder in den nächsten Zustand überführt, oder es erscheint eine entsprechende Fehlermeldung.

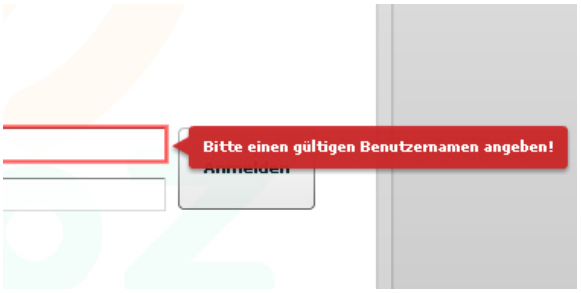


Abbildung 4: Fehlermeldung bei ungültigem Benutzernamen (iWoz :: Login-Screen)

Die Auswahl des Nutzertyps geschieht automatisch im Hintergrund und ist abhängig von den angegebenen Logininformationen. Jeder Nutzer kann nur einem Nutzertypus zugeordnet werden, wobei der Nutzertypus *SuperUser* für die Möglichkeit sorgt, sämtliche Nutzerrollen

auf einmal auszuüben. Sobald korrekte Logininformationen angegeben worden sind, wird eine Verbindung zum Media-Server hergestellt und es werden je nach erkanntem Nutzertypus zusätzliche für das Flex Interface benötigte Daten angefordert. Kann keine Verbindung zum Media-Server hergestellt werden, so erscheint eine entsprechende Fehlermeldung.

Abbildung 5: Fehlermeldung bei Verbindungsproblemen (iWoz :: Login-Screen)

4.2.2 Designer-Interface

Zunächst möchte ich das grafische Designer-Interface vorstellen. Einige Elemente, die hier Verwendung finden, wurden ebenfalls in leicht angepasster Form für die GUIs der Nutzertypen *Admin* und *Wizard* wiederverwendet. Das zentrale Element des *Designer*-Interfaces ist ein XML-Editor, der es dem VUI-Designer ermöglichen soll, die Dialogprototypen für das zu testende Voice-User-Interface bequem zu erstellen, zu editieren sowie Systemprompts aufzunehmen.

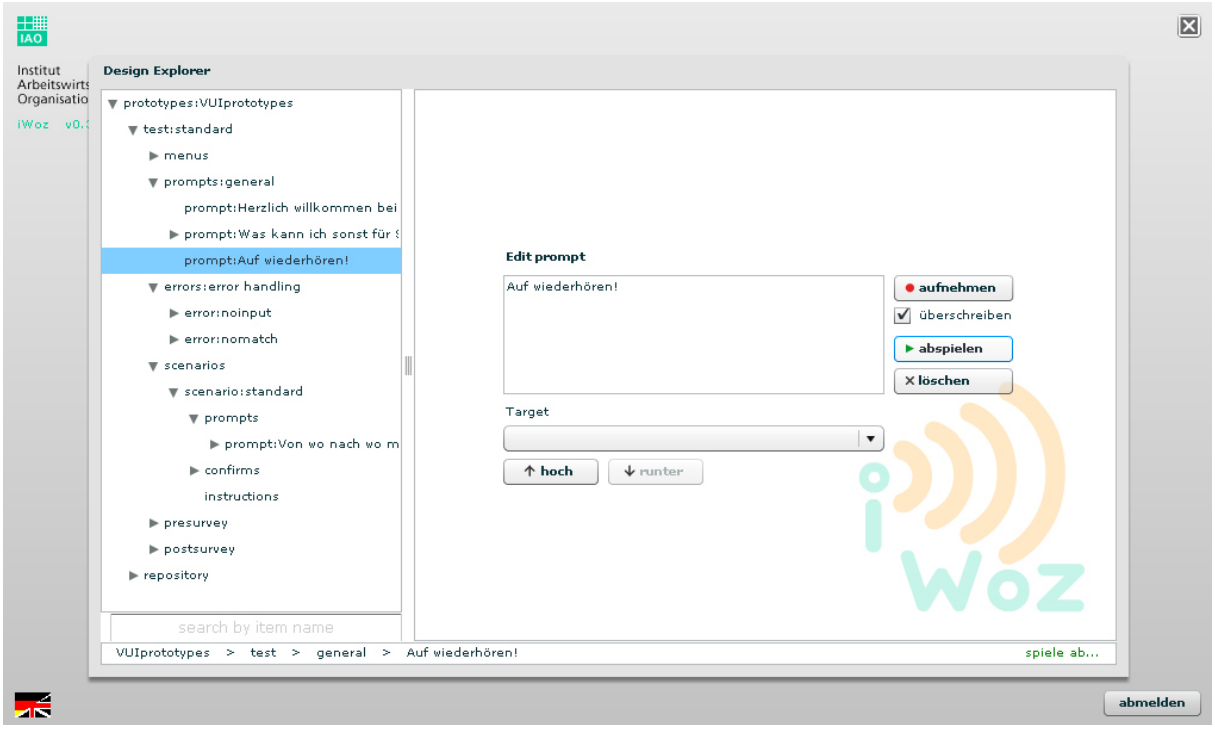


Abbildung 6: Design-Explorer (Designer-Interface)

Da das Hauptaugenmerk dieser Arbeit jedoch nicht auf dem VUI-Design liegen sollte, wurden diese Funktionen nur ansatzweise implementiert. Insbesondere die Aufnahme der Systemausgaben (*prompts*) über den angebundenen Media-Server wurde jedoch vollständig implementiert und kann entsprechend für die Erstellung von VUI-Prototypen verwendet werden. Der angesprochene XML-Editor verfügt über eine *Suchfunktion*, die es ermöglicht nach bestimmten XML-Elementen zu suchen (links unten; eine genauere Beschreibung folgt im nächsten Kapitel), eine *Baumansicht* (linke Seite) sowie eine *Detailansicht* (rechte Seite). Darüber hinaus wurde eine *Statusleiste* eingebunden, welche auf der linken Seite eine *Breadcrumb*-Anzeige[77] über den aktuellen Punkt in der im Baum angewählten XML-Struktur sowie auf der rechten Seite weitere Statusinformationen über die zuletzt ausgeführte Aktion bietet. Sämtliche verwendeten Interface-Elemente, wie das den Explorer beinhaltende *Panel* sowie die in der Abbildung zu sehende *Baumansicht*, die *Textfelder* und das *Drop-Down-Menü* wurden aus der Flex-SDK entnommen und vor allem im Falle der Baumansicht den Bedürfnissen angepasst. Auf ein weiteres Styling über *CSS-Styles*[78] wurde bei den meisten verwendeten GUI-Elementen bewusst zu diesem Zeitpunkt verzichtet, es kann jedoch ohne weiteres zu einem späteren Zeitpunkt in Erwägung gezogen werden.

Wird in der Baumansicht oder über die Suchfunktion ein Element ausgewählt, so erscheint die entsprechende Detailansicht auf der rechten Seite des Explorers. Der Call-Flow kann bislang nur über die entsprechende Auswahl eines *Targets* (Prompt mit eindeutiger ID) beeinflusst werden. Wünschenswert wäre es, für zukünftige Versionen eine Graphen-Ansicht einzubinden, welche die in Adobe Flex bereits vorhandenen *Drag&Drop*[79]-Funktionalitäten entsprechend ausnutzt. Um die Komplexität dieses Prototypens nicht zu sehr in die Höhe zu treiben und um nicht von der primären Aufgabe der Realisierbarkeit eines Wizard-of-Oz-Tests abzulenken, wurde darauf wiederum bewusst verzichtet.

Abbildung 7: Detailansicht Prompt (Designer-Interface)

77 Cooper, A. (1995), Seite 151
78 http://www.adobe.com/de/devnet/flex/quickstart/styling_components/
79 http://www.adobe.com/devnet/flex/quickstart/adding_drag_and_drop/

Mit der Detailansicht für die Systemausgaben können die existierenden Prompts sowohl textlich verändert als auch die aufgenommenen und auf dem Mediaserver hinterlegten Audiodaten abgespielt und überschrieben werden. Darüber hinaus ist eine Zielauswahl möglich (*Target*), der Prompt kann auf einer Ebene verschoben werden (hoch, runter) und er kann gänzlich entfernt werden (löschen). Analog zur Bearbeitung der Systemausgaben können bei entsprechender Auswahl in der Baumansicht neue *Prompts*, *Slots*, *Confirms* und *Keywords* hinzugefügt, bearbeitet und gelöscht werden.

Aufnahmen für Slots und Keywords fungieren während eines Wizard-of-Oz-Tests als Variablen, falls sie einer Bestätigung (*Confirm*) bedürfen. Bei der Aufnahme ist darauf zu achten, dass die entsprechenden Wörter gleichmäßig und mit korrekter Prosodie (Intonation, Lautstärke, Tonhöhe/-variation) eingesprochen werden. Aufgrund des serverseitig verwendeten Datenformats (*.flv*, Flash Video Format), ist zur Zeit leider keine nachträgliche Bearbeitung dieser Daten möglich[80]. Da es sich jedoch immer nur um kurze Aufnahmen handelt, halte ich es für vertretbar die Aufnahmen entsprechend zu wiederholen, bis ein zufriedenstellendes Ergebnis vorliegt.

Abbildung 8: Detailansicht HTML-Editor (Designer-Interface)

Wie in Abbildung 8 zu sehen, können zusätzlich Text-Elemente bearbeitet werden, wie in diesem Fall die Szenario-Instruktionen, wobei von Flex aktuell leider nur eine begrenzte Anzahl von HTML-Tags zur Verfügung gestellt werden. Im oberen Teil kann der HTML-Text eingegeben werden, woraufhin in einer Art *Live-Vorschau* (Preview) das Ergebnis angesehen

80 http://www.wowzamedia.com/forums/showthread.php?t=1954

werden kann. Damit die Änderungen übernommen werden, muss der Designer den *speichern*-Button betätigen. In der Abbildung wurde zudem die Suchfunktion verwendet (unten links). Existieren mehrere XML-Elemente mit demselben Namen, so erscheint automatisch ein Button, mit dem der Designer von einem Suchergebnis zum nächsten springen kann.

Abbildung 9: Suchfunktion (Designer-Interface)

Falls für ein Element der in der Bauansicht angezeigten XML-Struktur noch keine Detailansicht vorgesehen ist, so besteht die Möglichkeit die XML-Rohdaten zu editieren.

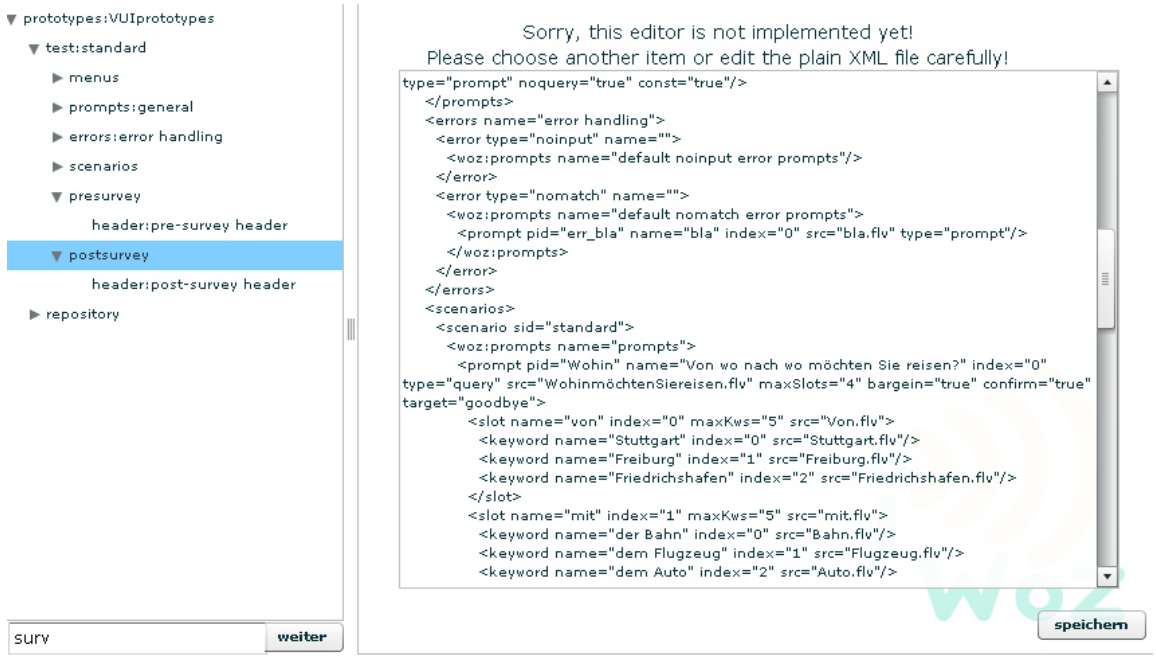


Abbildung 10: XML-Editor (Designer-Interface)

Ein Editieren der Rohdaten kann jedoch dazu führen, dass schwerwiegende Fehler in der XML-Struktur entstehen und die Daten schlimmstenfalls unbrauchbar werden. Aus diesem Grund ist diese Funktion standardmäßig deaktiviert und es wird entsprechend davor gewarnt. Leider ist die Validation der manipulierten XML-Daten frontendseitig über ActionScript 3 zur Zeit nicht möglich[81] und dementsprechend können invalide Eingaben, die nicht dem angegebenen XML-Schema entsprechen, in diesem Fall nicht erkannt werden. Wird versucht eine ungültige Datei zu speichern, so wird dies erkannt

81 http://bugs.adobe.com/jira/browse/ASC-3153

und es wird eine entsprechende Fehlermeldung im Infobereich angezeigt (vgl. Bild).

Abbildung 11: Fehler beim Schreiben ungültiger XML-Daten (Designer-Interface)

Allerdings ist das Editieren der XML-Rohdaten für essentielle Funktionen, wie der Vergabe von IDs für die eindeutige Kennzeichnung verschiedener Elemente (*Prompts*, *Slots*, *Keywords*) zur Zeit noch unumgänglich, da diese Funktionen noch nicht implementiert sind.

4.2.3 Admin-Interface (Admin-Explorer)

Für den Nutzertypus *Admin* wurde eine etwas vereinfachte Version des *Design-Explorers* wiederverwendet und entsprechend angepasst.

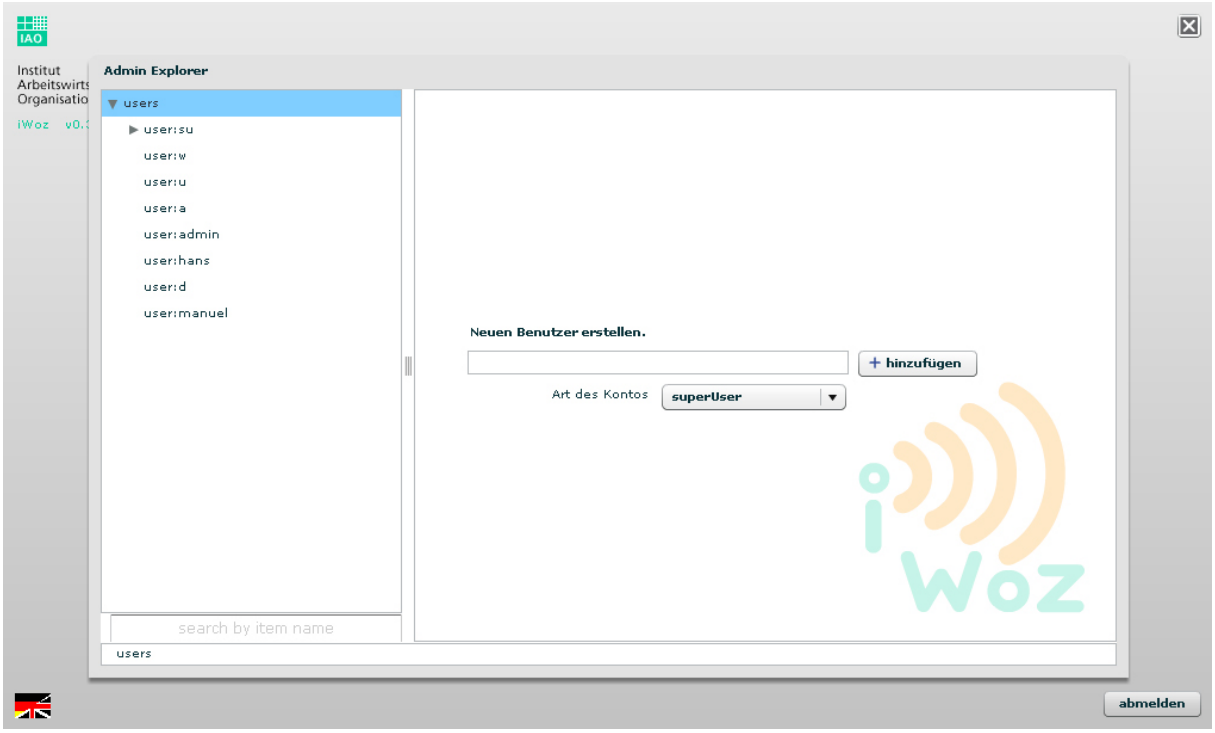


Abbildung 12: Admin-Explorer (Admin-Interface)

Mit Hilfe des Admin-Explorers ist es möglich, Nutzerdaten bequem anzupassen sowie neue Benutzer zu erstellen. Für alle Funktionen, die hier noch nicht implementiert sind, ist wiederum ein Editor für die XML-Rohdaten vorhanden.

4.2.4 Wizard-Interface

Ein zentrales Element für die Wizard-of-Oz-Testing-Funktionalität dieses Prototypen ist das *Wizard-Interface*. Das Wizard-Interface basiert ebenfalls auf der Explorer-Komponente, welche für die Admin- und Designerinterfaces verwendet worden ist, wobei sowohl auf der linken Seite in der Baumansicht, also auch in der Statuszeile im unteren Teil keine Eingaben möglich sind.

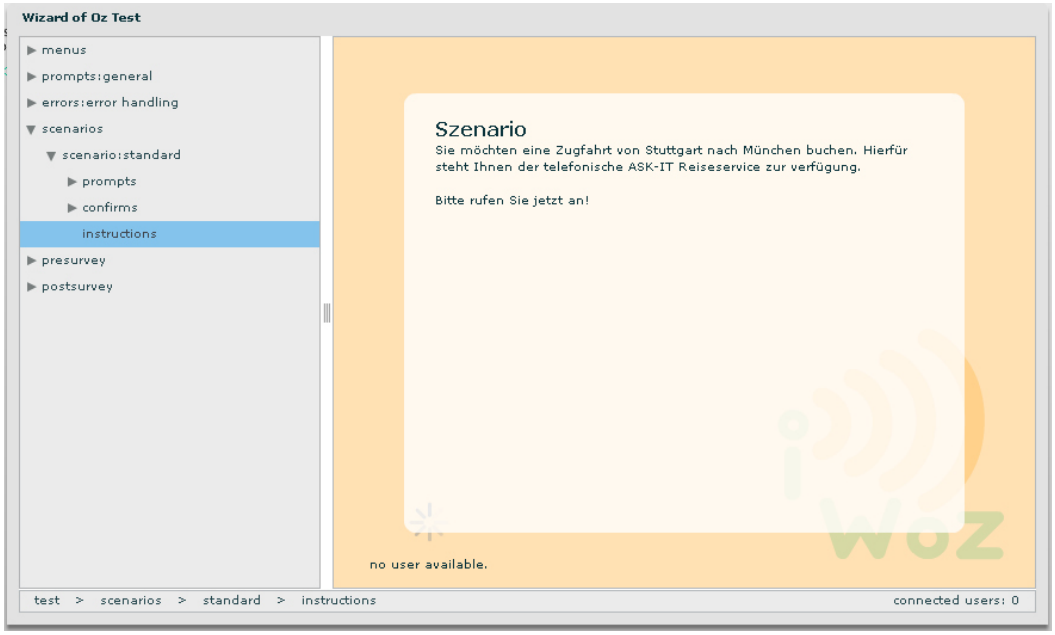


Abbildung 13: Wizard-Interface

Die Nutzerinteraktion erfolgt dementsprechend ausschließlich im rechten Teil des Fensters, wobei nach erfolgtem Login zunächst keine Aktionen seitens des Wizards notwendig sind. Damit sich der Wizard auf das anstehende Szenario entsprechend vorbereiten kann, bekommt er jeweils vor Beginn eines Szenarios die Instruktionen, die primär für die Testperson angelegt worden sind, angezeigt. Ist keine Testperson mit dem Server verbunden, so wird dies entsprechend unterhalb der Instruktionen links unten sowie im Infobereich auf der rechten Seite angezeigt.

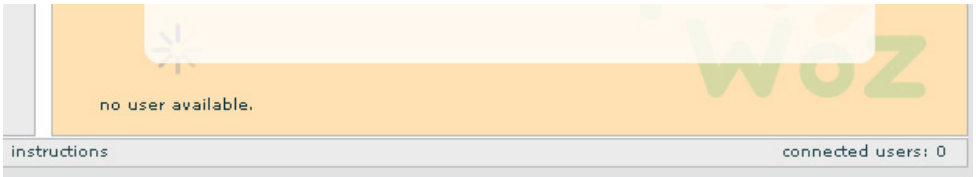


Abbildung 14: Information zu verbundenen Testpersonen (Wizard-Interface)

Um die Funktionsweise des Wizard Interfaces sowie das Wizard-of-Oz-Testing an sich besser zu verstehen, möchte ich zunächst eine kurze Einführung über das Testuser Interface geben und dann Schritt für Schritt und im Wechsel zwischen Wizard- und Testuser Interface einen Wizard-of-Oz-Test in der Form zu beschreiben, in der er mit dem implementierten Prototypen zur Zeit durchgeführt werden kann.

4.2.5 Testuser-Interface

Das Testuser-Interface wurde wie die anderen bereits vorgestellten GUIs auf Grundlage eines Panel-Containers (der grau hinterlegte Rahmen) erstellt, wobei es das einzige der vier Interfaces ist, das nicht auf dem XML-Explorer basiert. Während eines Testdurchlaufs kann das Testuser-Interface vier verschiedene Zustände einnehmen. Nach dem Loginvorgang erscheint zunächst die Vorbefragung, auf der, wie in diesem Beispiel, Kontaktdaten sowie weitere relevante Informationen zur jeweiligen Testperson erfasst werden können.

Abbildung 15: Testuser Interface - Vorbefragung

Sind sämtliche benötigten Informationen erfasst (obligatorische Felder sind hier mit einem Sternchen markiert), kann fortgefahren werden. Im nächsten Schritt erhält die Testperson allgemeine Informationen zum Testablauf, woraufhin im darauffolgenden Schritt mit dem eigentlichen Test begonnen werden kann.

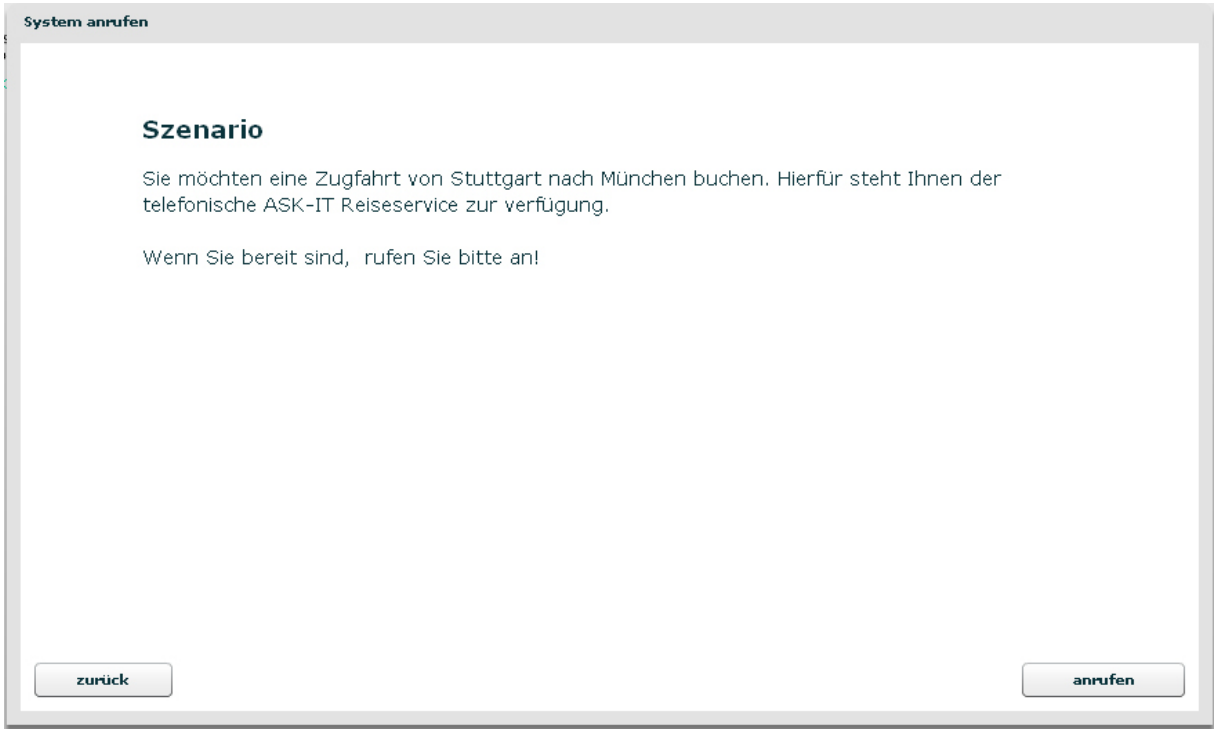


Abbildung 16: Testuser Interface - Szenario

Sobald sich die Testperon das Szenario eingeprägt hat, kann zum eigentlichen Test übergegangen werden. Ist ein Wizard eingeloggt, so erscheint wie in der vorangegangenen Abbildung unten rechts der *anrufen*-Button. Wird dieser Button betätigt, so wird serverseitig der nächste verfügbare, im System angemeldete Wizard selektiert und über den Anruf informiert. Der *anrufen*-Button wird durch einen entprechenden Hinweis ersetzt, wobei die Testperson den Anruf weiterhin abbrechen kann. Während des Anrufs wird ein Tonsignal abgespielt, welches zusätzlich über den laufenden Anruf informieren soll.

Abbildung 17: Anruf (Testuser-Interface)

Auch seitens des Wizards kann nun der Test beginnen. Unmittelbar nachdem ein Testteilnehmer mit dem Server verbunden ist, erscheint eine entsprechende Zahl in der Statusleiste des Wizard-GUIs. Auch die links unterhalb der Szenarioanweisungen befindliche Animation ändert sich, sobald ein Teilnehmer verfügbar ist.

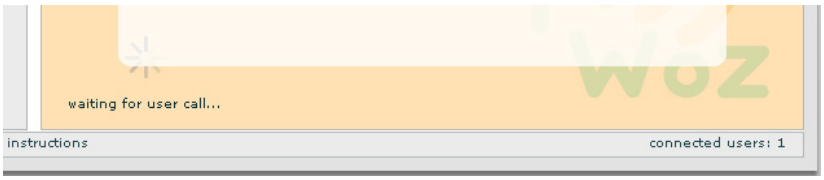


Abbildung 18: Testperson verfügbar (Wizard-Interface)

Wird seitens des Testteilnehmers der *anrufen*-Button betätigt, wird der Wizard sowohl grafisch, als auch akustisch durch den Server auf den Anruf aufmerksam gemacht.

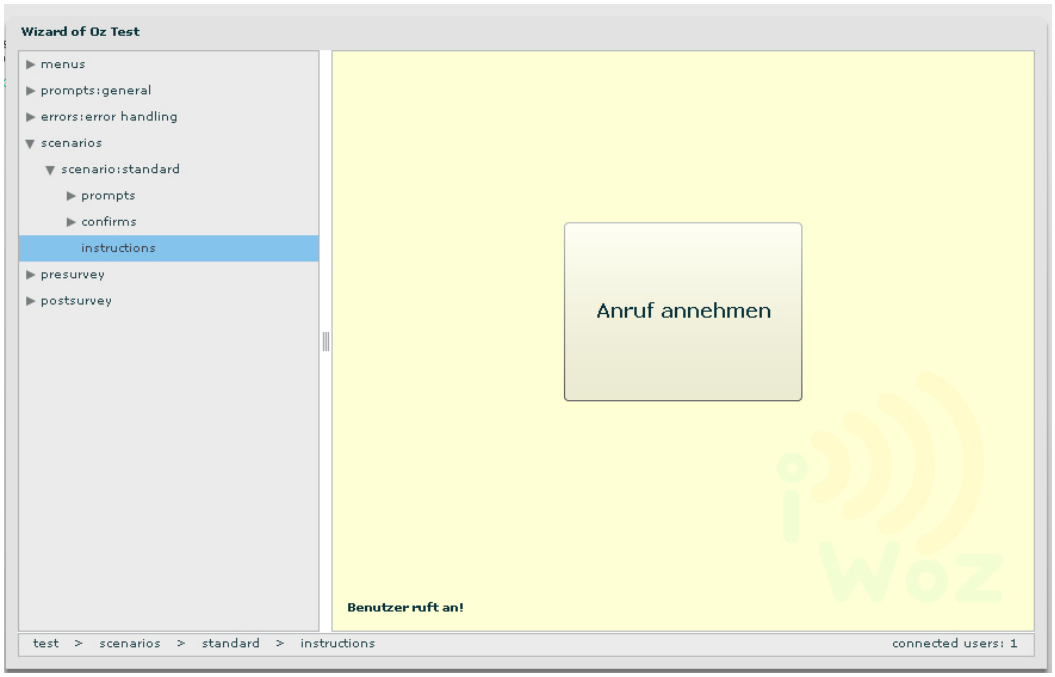


Abbildung 19: Anruf (Wizard-Interface)

Die Szenarienbeschreibung wird nun zunächst durch einen abwechselnd weiß und gelb blinkenden Hintergrund sowie einen großen Button ersetzt, mit dem der Anruf angenommen und somit der Test gestartet werden kann.

Ist der Test gestartet, springt die Anzeige im Wizard-GUI auf den im VUI-Prototypen festgelegten Willkommensprompt. Für das Springen zu den jeweils nächsten Dialogschritten wird eine leicht modifizierte Suchfunktion auf der im linken Teil des Wizard-GUIs dargestellten XML-Struktur angewendet. So kann auf einfache Weise zu den jeweils folgenden, als *Target* definierten Dialogschritten gesprungen werden. Sobald der jeweils nächste Schritt in der Baumanzeige auf der linken Seite vom System „ausgewählt“ worden ist, wird auf der rechten Seite entweder ein einfacher *Prompt*, eine Frage (*Query*) oder eine Bestätigung (*Confirm*) angezeigt. Gleichzeitig wird das zuvor im Designer-Interface aufgenommene Audiomaterial

auf beiden Seiten der „Leitung“ abgespielt.

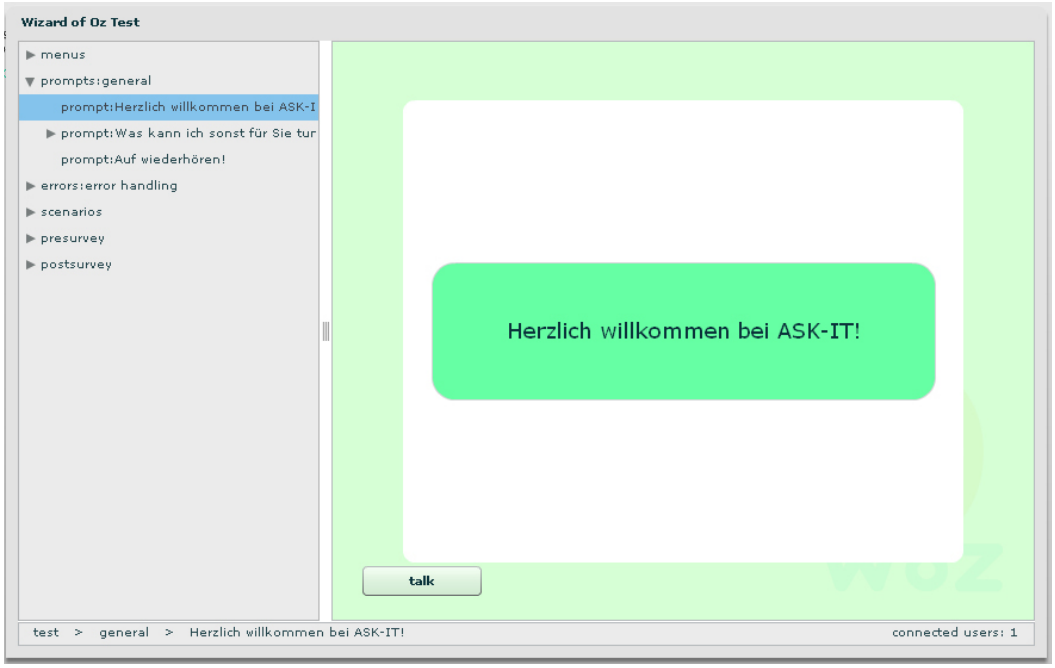


Abbildung 20: Prompt (Wizard-Interface)

Während des Tests hat der Wizard die Möglichkeit per Tastendruck mit Hilfe des *talk*-Buttons, direkt mit der verbundenen Testperson zu sprechen. Das Mikrophon der Testperson ist im Gegensatz zu dem des Wizards dauerhaft aktiv. Sämtliche Aussagen werden während eines Tests serverseitig gespeichert und können im Nachhinein über die serverseitige Client-ID, welche in den Testergebnissen festgehalten wird, abgerufen werden.

Abbildung 21: DTMF-Tastatur (Testuser-Interface)

Sobald der Test gestartet ist, erscheint auf Seiten der Testperson auf der unteren rechten Seite eine Tastatur, welche die Möglichkeit bietet *DTMF*-Signale an den Wizard zu übermitteln. Diese Funktionalität, mit der beispielsweise Menüpunkte, oder Shortcuts innerhalb des VUI-Prototypen angewählt werden können ist zwar grundsätzlich eingebunden, wird jedoch an dieser Stelle nicht weiter vorgestellt.

Unmittelbar nach dem der Willkommensprompt abgespielt worden ist, wird mit dem eigentlichen Szenario begonnen. Der Anwendungsfall in diesem Szenario ist in Anlehnung an das ASK-IT Projekt, für das ich mit dem Tool Voxeo Designer bereits einen Prototypen erstellt habe, eine einfache Reiseauskunft. Da das Szenario lediglich der Veranschaulichung dienen

soll, werden hier nur zwei weitere Dialogschritte beschrieben.

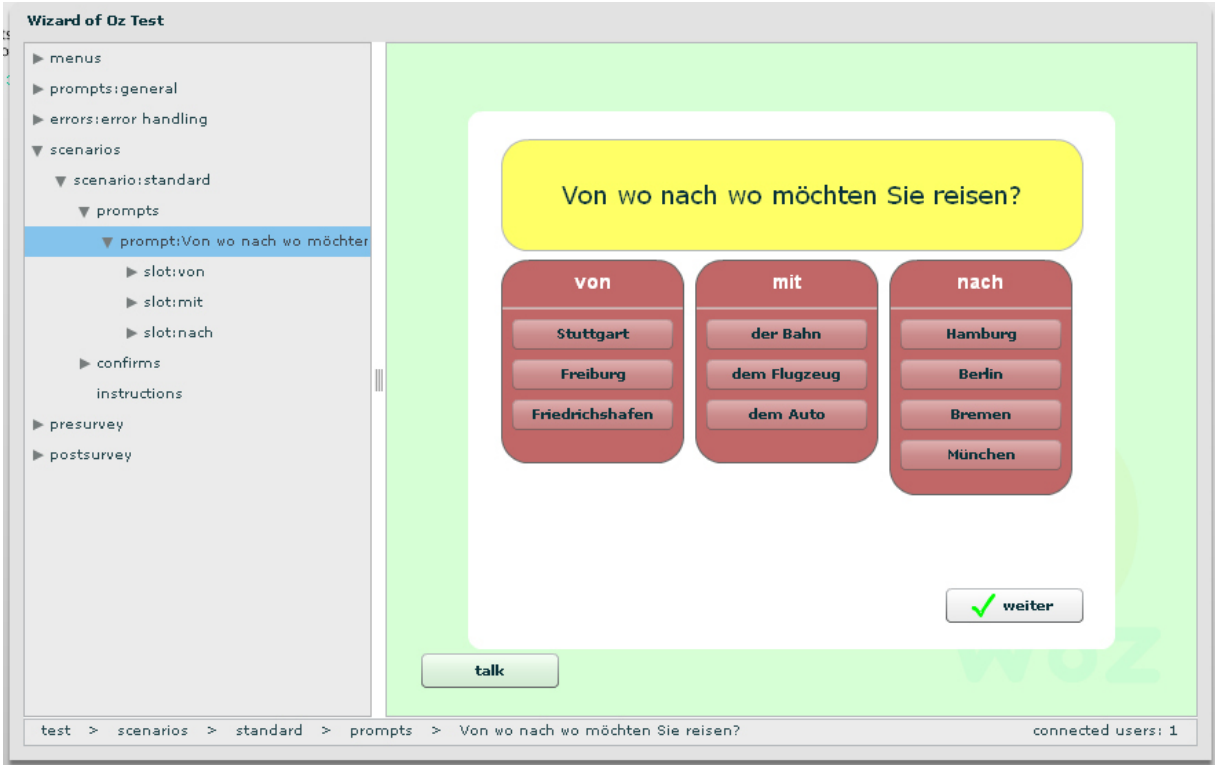


Abbildung 22: Query (Wizard-Interface)

Die auf den Willkommensgruß folgende Frage beinhaltet sowohl die Fragestellung, einen Prompt-Spezialfall (Typ `query`), welcher auf gelbem Grund dargestellt wird sowie die vorhandenen Slots inklusive der Schlüsselwörter (*Keywords*), welche auf rotem Grund dargestellt sind.

An dieser Stelle ist es die Aufgabe des *Wizards*, die Texterkennung eines Sprachdialogsystems zu simulieren und die erkannten *Keywords* per Mausklick auszuwählen. Da es zum aktuellen Entwicklungsstatus leider noch nicht möglich ist, globale Elemente, wie Hilfe oder ein Hauptmenü, einzubinden und zudem eine Fehlerbehandlung (no-match, no-input) noch nicht vorhanden ist, muss der Wizard in solchen Fällen zur Zeit noch *manuell* eingreifen, indem er den *talk*-Button betätigt. Eine Einbindung dieser sicherlich wichtigen Funktionalitäten wird weiterhin angestrebt, ist aber in diesem Prototypen noch nicht verfügbar.

Sobald die erkannten Keywords vom Wizard ausgewählt worden sind, kann der *weiter*-Button betätigt werden. Die zu den ausgewählten Slots und Keywords hinterlegten Audioaufnahmen werden daraufhin zusammengefügt und als eine Art Abspielliste (*Playlist*) an den Server gesendet. Dieser initiiert den nächsten Dialogschritt und sendet die zusammengefügten Audiodateien an die verbundenen Clients (*Wizard* und *TestUser*).

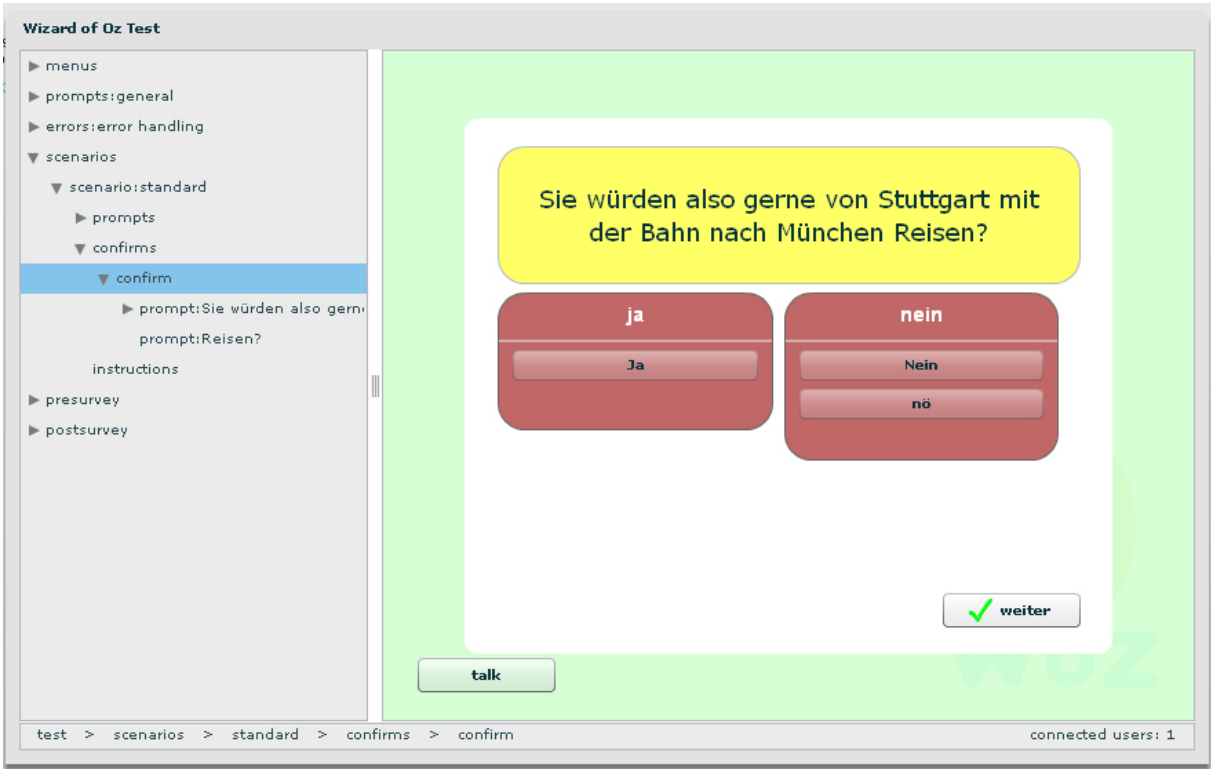


Abbildung 23: Confirm (Wizard-Interface)

Nach Auswahl aller relevanten Keywords erscheint als nächster Dialogschritt die Bestätigung der Auswahl. Erfolgt in diesem Fall keine Bestätigung, so wird die vorherige Frage erneut gestellt. Wird die Auswahl bestätigt, so kann der Dialog mit dem nächsten Schritt (z.B. Buchung) fortgeführt werden. Im aktuell in der VUI-Prototypen-Datei implementierten Fall, ist der Dialog beendet und es erfolgt eine Verabschiedung.

Ist die Verabschiedung abgespielt, so wird der Test beendet und das Ergebnis abgespeichert. Für den Wizard geht es in diesem Fall wieder von vorne los (es erscheinen die Instruktionen für das nächste Szenario), während die Testperson mit der unmittelbar darauf folgenden Nachbefragung beginnen kann. In einem echten Wizard-of-Oz-Tests sollte es möglich sein, mehrere Szenarien hintereinander durchzuführen. Im aktuellen Prototypen wurde dies jedoch bewusst zunächst auf ein einzelnes Szenario begrenzt.

In einer Nachbefragung werden bei Usability Tests gewöhnlich qualitative Daten, wie die allgemeine Benutzerzufriedenheit sowie konkrete Kritik und Verbesserungsvorschläge erfasst. Für zukünftige Versionen der Applikation wird empfohlen bei mehr als einem Szenario zwischen den Szenarien sowie nach dem Test eine solche kurze Befragung, analog zur beschriebenen und implementierten Vorbefragung durchzuführen. Eine Nachbefragung ist zum aktuellen Stand der Implementation zwar vorgesehen, jedoch nicht weiter implementiert.

4.2.6 SuperUser-Interface

Das SuperUser-Interface vereint sämtliche zuvor beschriebenen Nutzerrollen in einem einzigen Interface. Unmittelbar nach erfolgtem Login als SuperUser erscheint zunächst eine gekachelte Übersicht, welche eine Auswahl des gewünschten Nutzer-Interfaces ermöglicht.

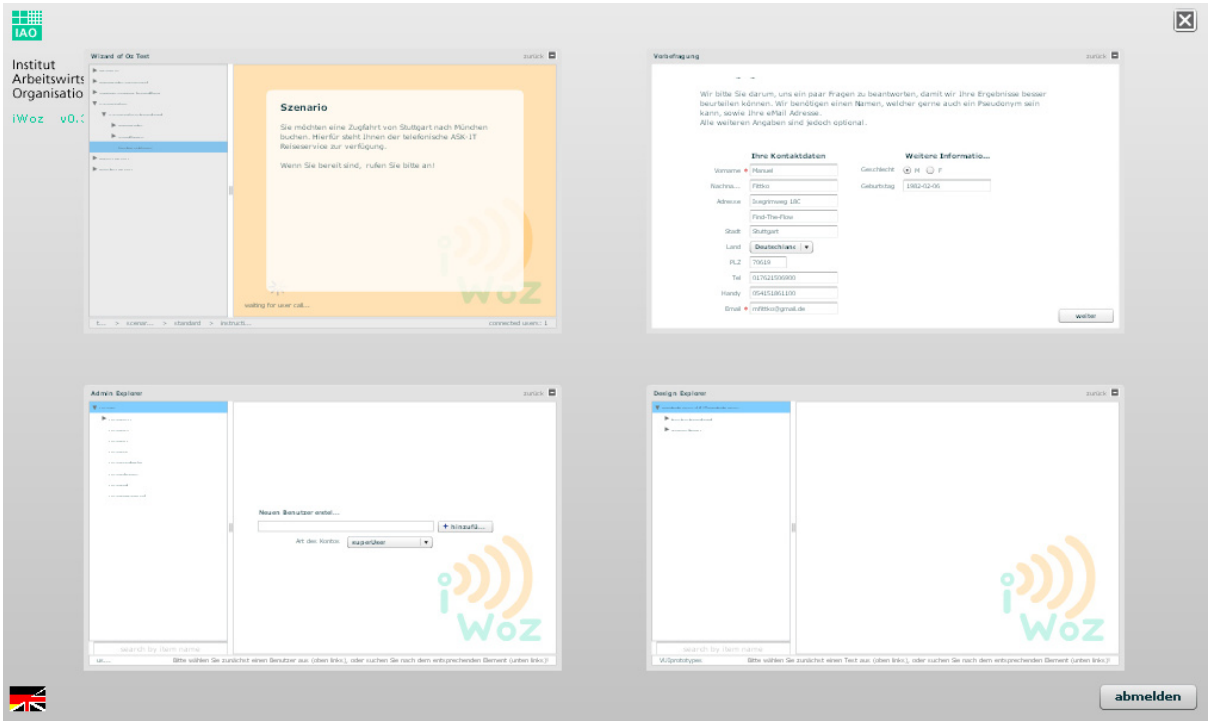


Abbildung 24: Übersicht SuperUser-Interface

Fährt der Nutzer mit dem Mauscursor über ein Icon, so wird dieses blau hinterlegt und es erscheint eine Anzeige, welche darüber informiert, über welchem Nutzerinterface sich der Cursor zur Zeit befindet. Damit klar wird, dass es sich um einen Button handelt, erscheint der Mauscursor als Hand[82].

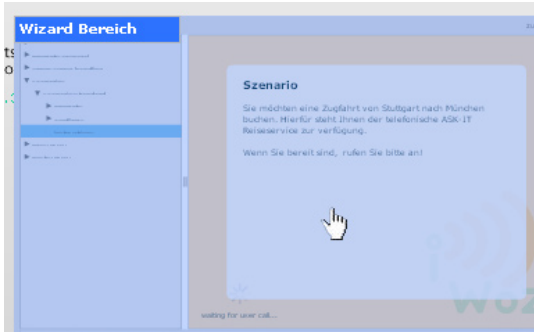


Abbildung 25: Rollover-Effekt Auswahl (SuperUser-Interface)

82 http://www.adobe.com/cfusion/communityengine/index.cfm?event=showdetails&productId=2&postId=1687

Wird das entsprechende Interface ausgewählt, so vergrößert sich das ausgewählte Interface, während sich die anderen Interfaces zum Rand hin verkleinern, bis sie nicht mehr sichtbar sind.

Abbildung 26: Auswahl Wizard-Interface, Zoom-Effekt (SuperUser-Interface)

Die Nutzerinterfaces unterscheiden sich grundsätzlich nicht von den bereits Beschriebenen. Damit jedoch eine Rückkehr zur Übersicht ohne weiteres ermöglicht wird, haben die Panels der jeweiligen Interfaces einen Button erhalten, mit dem wieder zur Übersicht zurückgekehrt werden kann (oben rechts).

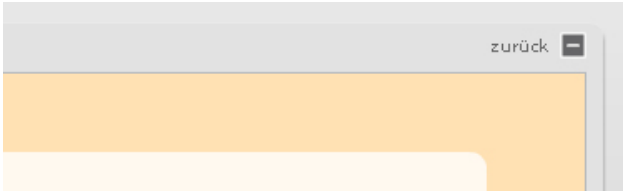


Abbildung 27: Zurück-Button (SuperUser-Interface)

Das SuperUser-Interface wurde vor allem zu Testzwecken erstellt, damit ein Testdurchlauf entsprechend simuliert werden kann, ohne dass man sich zusätzlich einloggen muss. Bei einer solchen Test-Simulation kommt es allerdings bei Nichtverwendung von Kopfhörern unweigerlich zu einer Rückkopplung, da Mikrophon und Lautsprecher in diesem Fall gleichzeitig aktiv sind.

4.3 Use Case Übersicht

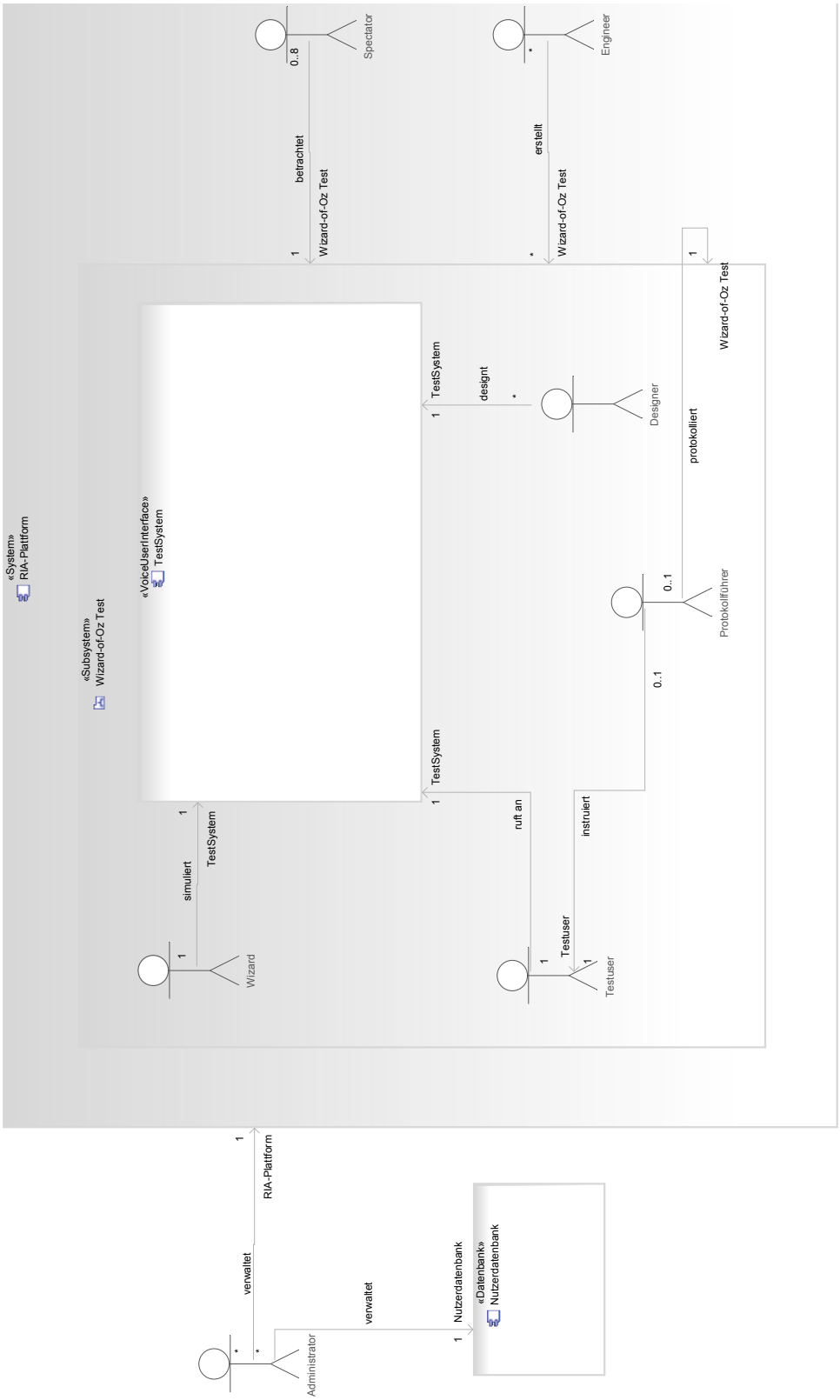


Abbildung 28: Use-Case-Übersichtsdiagramm

Dieses schematische Use-Case-Übersichtsdiagramm soll einen Überblick über die in der RIA konzeptionell eingebundenen Nutzerrollen und Funktionen bieten. Es wurde wie das bereits vorgestellte

Zustandsdiagramm in der Konzeptionsphase und somit vor Beginn der Implementation des Prototypens erstellt. Optionale Nutzerrollen, wie der *Spectator* sowie der *Protokollführer* wurden nicht in den Prototypen der RIA eingebunden, während die Nutzerrolle des *Engineers* mit der des SuperUsers gleichgesetzt werden kann.

5. Datenbasis und Datenbankdesign

Als Datenbasis für den Prototypen wurden ausschließlich, sowohl backend- als auch frontendseitig, XML-Dateien mit selbst erstellten *Strukturen* (*XML-Schemata*)[83] verwendet. Dies ist meines Erachtens alleine deshalb naheliegend, weil mir mit der Unterstützung des *E4X-Standards* (*ECMA-357*)[84] durch *ActionScript 3* (*AS3*) ein gleichermaßen intuitives wie auch mächtiges Werkzeug mitgeliefert wird, mit dem sich die Daten auf einfache Weise clientseitig manipulieren lassen. Da es sich, bei dem für diese Arbeit implementierten, System um einen ersten Prototypen handelt, habe ich die sicherlich vorhandenen Nachteile, die eine ausschließlich auf XML basierende Datenbasis mit sich bringt, erst einmal außer Acht gelassen.

Ein Nachteil des verwendeten Ansatzes ist es, dass ich aufgrund der fehlenden Unterstützung von E4X oder eines vergleichbaren Ansatzes in Java, darauf verzichtet habe, die XML-Daten serverseitig zu parsen und zu manipulieren. Die verwendeten und clientseitig manipulierten/generierten XML-Dateien werden also folglich in Form eines *Strings*, also in Textform als ganzes zum Server geschickt und dort auf dem Dateisystem abgelegt, bzw. vom dortigen Dateisystem abgerufen. Diese vereifachte Form des Datenaustauschs bringt zwar den Nachteil mit sich, dass die Validität der übermittelten XML-Daten nicht noch einmal serverseitig überprüft wird. Es ist also denkbar, dass die durch den Client manipulierten Daten nicht mehr der in der jeweiligen XML-Schema Datei (*.xsd*) entsprechen und somit nicht mehr valide (s.o.) sind. Die Wohlgeformtheit (s.o.) der Daten wird jedoch durch das in AS3 verwendete *Strict-Typing* sichergestellt. Näheres dazu im Abschnitt über die Adobe Flex Entwicklungsumgebung.

In folgenden Implementationen wäre es eventuell vorzuziehen, zumindest einen Teil der Datenmanipulationen, gegebenenfalls in Kombination mit einer *relationalen Datenbank* wie *mySQL*[85] auf dem Server zu implementieren, damit das Datenvolumen der zu transferrierenden Daten minimiert und somit der Client entlastet werden kann. Bei zunehmender Komplexität der Anwendung ist eine relationale Datenbank sicherlich vorzuziehen[86], wobei ich in diesem konkreten Anwendungsfall nicht davon ausgehe, dass die Komplexität der Daten zu irgendeinem Zeitpunkt diesen Status tatsächlich erreichen wird.

Ein weiterer Grund dafür, dass ich mich bei der Datenbasis des Prototypen für die ausschließliche Verwendung XML basierter Strukturen entschieden habe, ist die Tatsache, dass die am weitesten verbreiteten Datenstrukturen im Bereich Sprachdialogsysteme ebenfalls XML ba-

83 vgl. https://www.bg.bib.de/portale/xml/pdf/XML-Schema.pdf
84 vgl. http://www.ecma-international.org/publications/standards/Ecma-357.htm
85 http://www.mysql.de/
86 vgl. http://eprints.otago.ac.nz/323/2/AnneWilliamsOCR.pdf

siert[87] sind. Der Einfachheit halber habe ich allerdings, entgegen meiner ursprünglichen Absicht, in dieser Arbeit darauf verzichtet, mich zu stark an diesen Standards zu orientieren. Ein Datenexport in Form von VoiceXML, SCXML oder SGML könnte allerdings ohne größere Umstände in zukünftige Versionen der von mir implementierten RIA integriert werden, um die mittels der RIA erstellten Prototypen unter Realbedingungen zu testen oder gar in ein *echtes* Sprachdialogsystem zu überführen. Da jedoch in dieser Arbeit der Fokus auf die Konzeption und vor allem auf das ortsunabhängige Testing von Dialogkonzepten sowie insbesondere die Interaktion zwischen *System* und Anrufer gelenkt werden sollte, wurde auf eine Standardkonformität der verwendeten Daten bewusst verzichtet.

5.1 Entwicklungsumgebung für die Erstellung der XML Schemata

Als Entwicklungsumgebung für die XML-Schemata wurde die Eclipse IDE mit der Erweiterung WTP (*Web Tools Platform*), bzw. dem untergeordneten Projekt *Web Standard Tools* (WST)[88] verwendet. Diese Erweiterung der Eclipse IDE bietet insbesondere eine gute Unterstützung, wie eine automatische Code-Completion von XML- und XML-Schema-Dateien sowie die automatischen Validierung und das Parsing dieser Dateitypen im Hintergrund. Auf diese Art und Weise wird unmittelbar auf Fehler oder invalide Elemente im Quelltext aufmerksam gemacht.

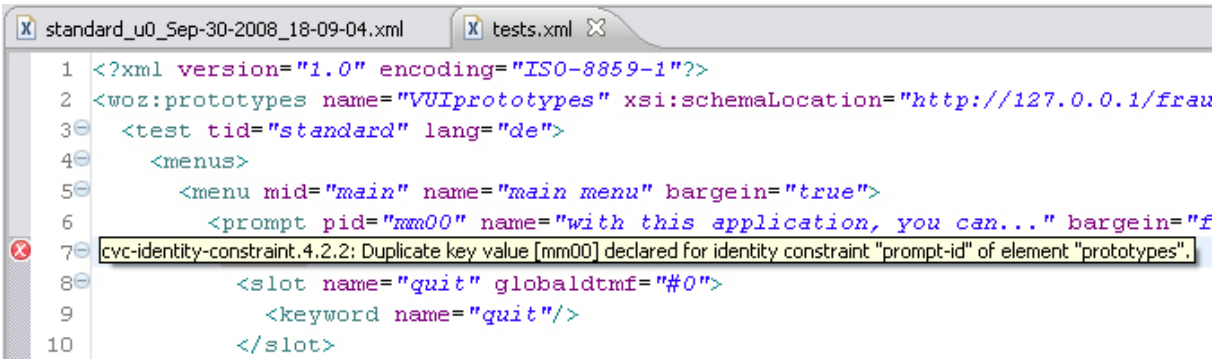


Abbildung 29: Hinweis bei invalidem XML Element (Eclipse WST)

Darüber hinaus wird ein grafischer Editor für XML- und XML-Schema-Dateien angeboten, wobei vor allem der XML-Schema-Editor einen hilfreichen Wizard für die Erstellung von *regulären Ausdrücken* beinhaltet. Mittels regulären Ausdrücken können eigene Datentypen erstellt werden und so kann beispielsweise eine Email-Adresse oder eine Telefonnummer auf Gültigkeit überprüft werden.

87 VoiceXML, SCXML (State Chart XML), SGML (Speech Grammar Markup Language)
88 http://www.eclipse.org/webtools/wst/main.php

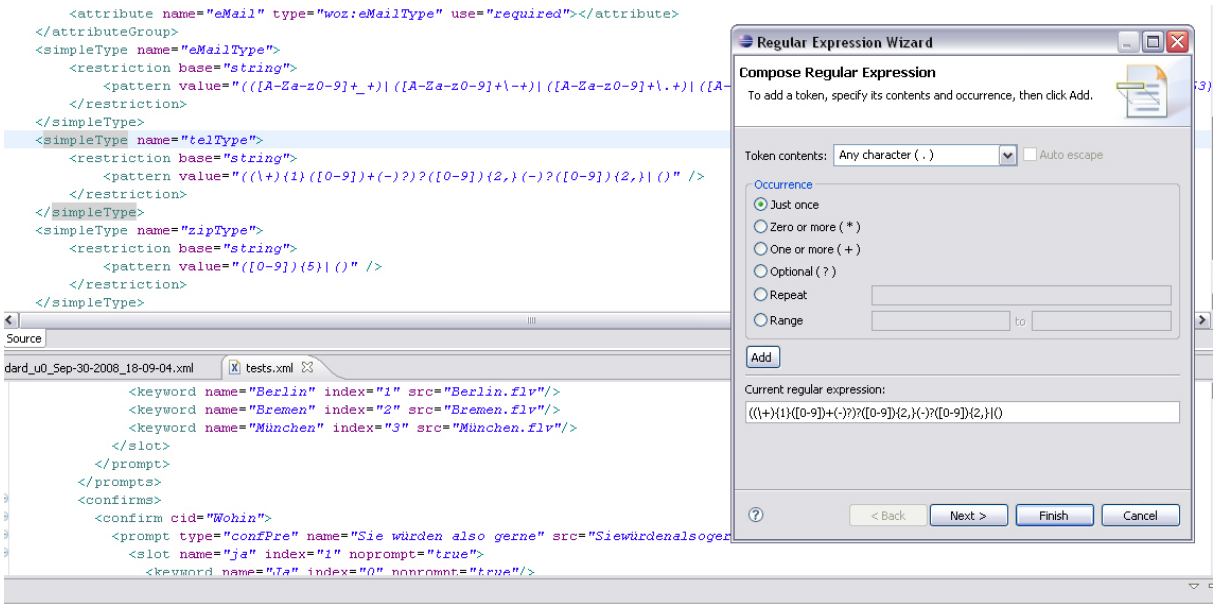


Abbildung 30: Wizard für die Erstellung von regulären Ausdrücken (Eclipse WST)

5.2 Datenaustausch Client/Server

Der Datenaustausch zwischen einem Flex Client und dem Server erfolgt in dem von mir implementierten System im Allgemeinen asynchron[89], d.h. im Hintergrund und ohne, dass der Nutzer davon erfährt. Dieser Ansatz ist grundsätzlich vergleichbar mit der als AJAX[90] bekannten Technologie. Außer der eigentlichen Applikation, welche als *.swf*-Datei auf einem HTTP Server liegt und beim Aufruf der URL zunächst gänzlich an den Client übertragen wird, werden die Daten je nach Bedarf vom Server geladen oder auf diesem abgelegt. Für das einmalige Laden der Nutzerdaten sowie der lokalisationsspezifischen Daten (mehr dazu später) wird die URLLoader-Komponente[91] aus dem Flex SDK verwendet. Für den weiteren Datenaustausch wird ausschließlich direkt mit dem Media-Server kommuniziert. Die Verbindung mit dem Server geschieht erst im Rahmen des Logins und wird so lange aufrecht erhalten, wie der Benutzer auf dem Server eingeloggt ist.

5.3 XML-Datei für dynamische Texte und Lokalisation (Internationalisierung)

Um die spätere Anpassung der in der Applikation verwendeten Texte sowie eine mögliche Erweiterung auf verschiedene Sprachen zu erleichtern, habe ich mich dafür entschieden, eine externe XML-Datenstruktur einzubinden, mit deren Hilfe sich zu jedem Element in Flex nachträglich Eigenschaften definieren lassen, welche in Abhängigkeit der verwendeten Brow-

89 http://www.adaptivepath.com/ideas/essays/archives/000385.php
90 vgl. oben
91 http://livedocs.adobe.com/flex/2/langref/flash/net/URLLoader.html

ser- bzw. Systemsprache angewendet werden.

Zur Veranschaulichung wird im Folgenden ein Ausschnitt aus der jeweiligen XML-Datei im *Design*-Modus des Eclipse-WST-Editors dargestellt. Die jeweiligen XML-Elemente sind in der Baumstruktur auf der linken Seite dargestellt und durch das e-Icon gekennzeichnet. Attributen ist das a-Icon vorangestellt. In der rechten Spalte finden sich sowohl die inhalte der ausgewählten Elemente und Attribute sowie ausgegraut die aus der entsprechenden Schema-Instanz (hier: locales.xsd) übernommenen Einschränkungen.

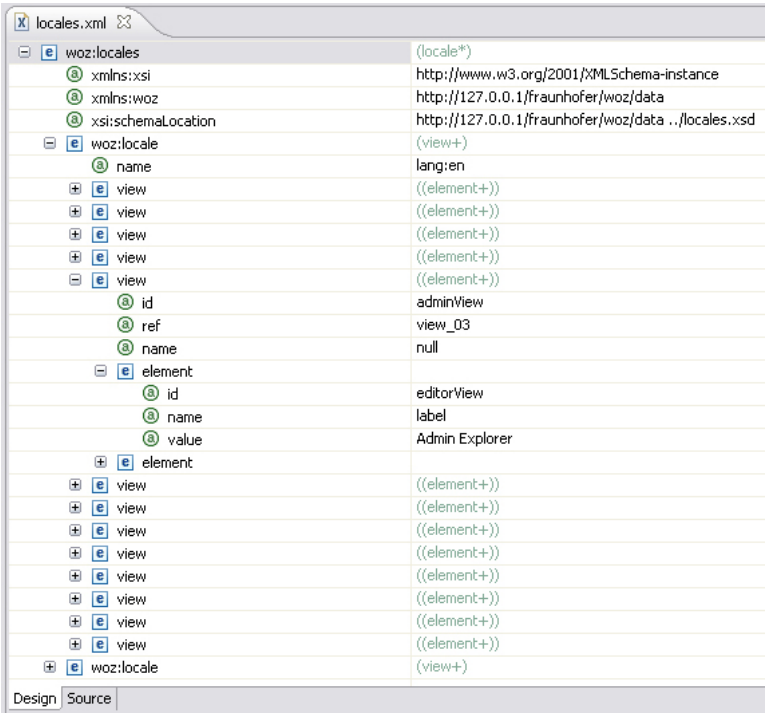


Abbildung 31: Design-Ansicht der Datei locales.xml (Eclipse WST)

Beliebige, mit einer ID versehene Objektinstanzen in Flex, hier das Panel, welches den Admin-Explorer beinhaltet, lassen sich über das Attribut `id` ansprechen, wobei mittels des Namens der Objekteigenschaft über das Attribut `name` (hier die *label*-Eigenschaft) ein beliebiger Wert (Attribut `value`) gesetzt werden kann.

5.3.1 Einschränkung

Die Möglichkeit der Einbindung verschiedenster lokalisationsspezifischer Eigenschaften und Texte ist also grundsätzlich vorgesehen, es wurde jedoch in diesem Prototypen lediglich eine deutsche sowie ansatzweise eine englische Lokalisation erstellt. Weitere Lokalisationen können nach Bedarf, vorzugsweise nach Fertigstellung der Applikation, hinzugefügt sowie die bestehenden vervollständigt werden. Wie genau die Einbindung der in dieser XML-Datei abgelegten Lokalisationsdefinitionen realisiert worden ist, möchte ich im folgenden Kapitel kurz beschreiben. Die externe Definition der Beschriftungen in dieser Flex-Applikation ist bereits weitestgehend realisiert. Bei Elementen welche hier noch nicht unterstützt werden, sind vereinzelt noch englischsprachige Beschriftungen zu finden.

5.4 XML-Datei mit Logindaten der Benutzer

Genau wie die Lokalisationsinformationen werden die Logininformationen der Benutzer ebenfalls gleich zu Beginn per *URLLoader* (s.o.) abgerufen. Aus Sicherheitsgründen sollte dies gegebenenfalls in zukünftigen Versionen der Applikation erst nach Eingabe der Benutzerdaten serverseitig geschehen. Darauf wurde jedoch in Hinblick auf die untergeordnete Relevanz sicherheitstechnischer Überlegungen für das aktuell beschriebene und implementierte Konzept an dieser Stelle verzichtet.

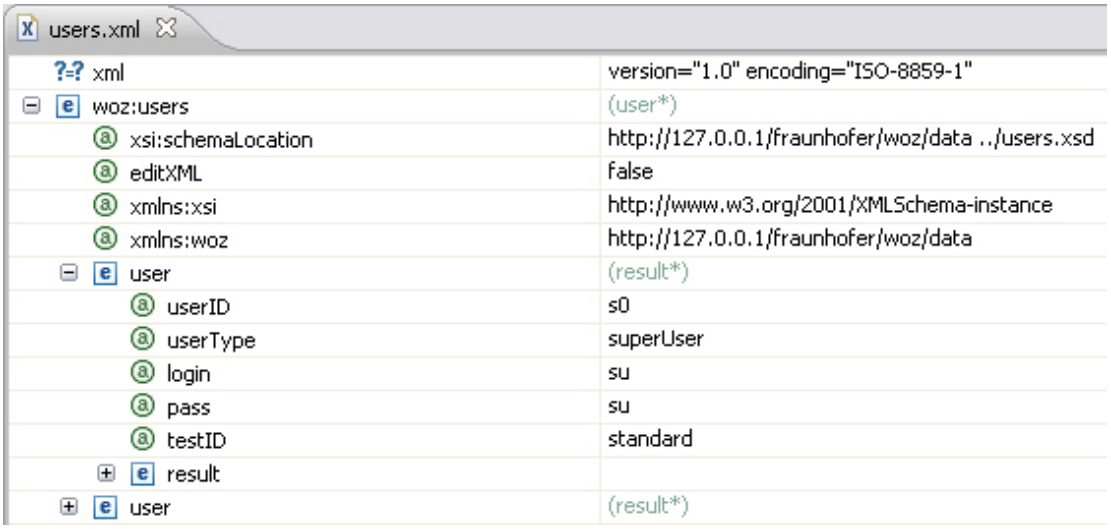


Abbildung 32: Ausschnitt der Datei users.xml (Eclipse WST)

In dieser flachen XML-Struktur sind alle für den Loginprozess relevanten Benutzerdaten abgelegt. Jeder Benutzer hat eine eindeutige ID (Attribut `userID`). Die aktuell vorhandenen Nutzertypen (Attribut `userType`), hier ein Benutzer vom Typ *superUser*, die Logindaten (Attribute `login` und `pass`) sowie die ID des Tests, zu dem dieser Nutzer aktuell zugeordnet ist (Attribut `testID`). Zudem ist es vorgesehen, jedoch noch nicht im aktuellen Proto-

typen realisiert, die Testergebnisse der Nutzertypen *superUser* und *testUser* über das Element `result` zu verlinken.

Neben dem Benutzertypus *superUser* existieren noch die Typen *testUser*, *wizard*, *admin* und *designer*, auf deren genauere Bedeutung ich wiederum erst zu einem späteren Zeitpunkt eingehen möchte. Außer der `testID` sind sämtliche Attribute für alle Nutzertypen obligatorisch und müssen angegeben werden. Die `testID` ist obligatorisch für die Nutzertypen *superUser*, *testUser* und *wizard*.

5.5 XML-Dateien mit nutzerspezifischen Daten

Die nutzerspezifischen Daten, also Kontakt- und sonstige persönliche Informationen sowie eine beliebige Anzahl von Tags werden in separaten XML Dateien abgelegt.

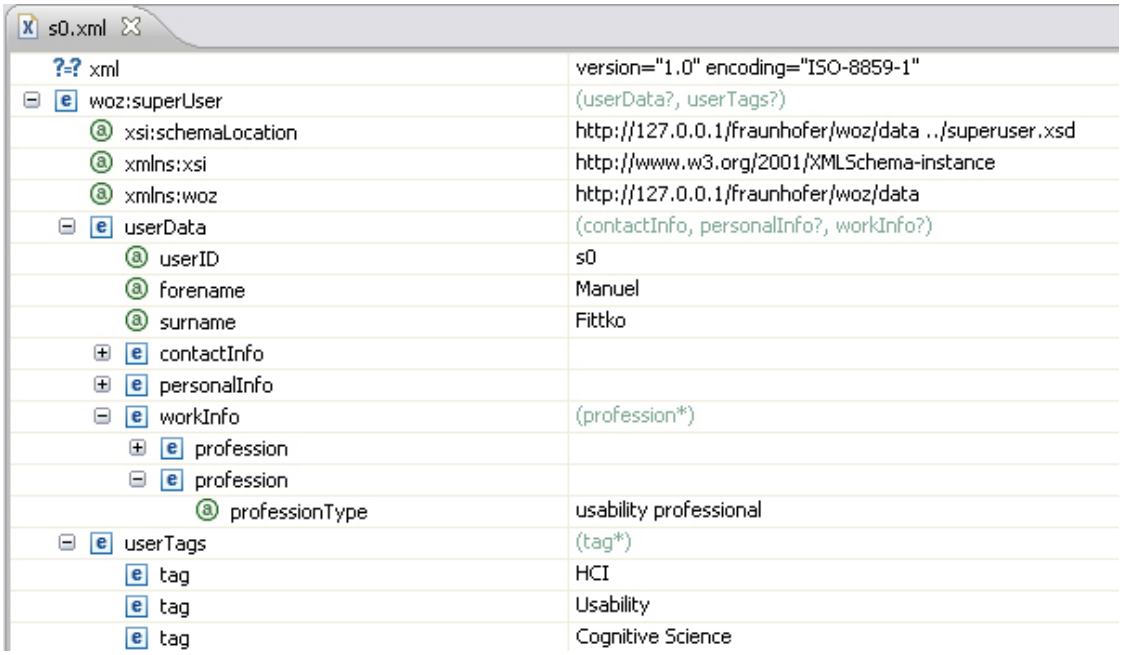


Abbildung 33: Ausschnitt der Datei s0.xml (Eclipse WST)

Die nutzerspezifischen Daten sind der Einfachheit halber nach der `userID` benannt. Auch hier wäre es aus sicherheitstechnischen Überlegungen in Zukunft sinnvoll die entsprechenden XML-Daten serverseitig abzufragen. Für eine genauere Beschreibung der einzelnen Elemente und Attribute, bitte gegebenenfalls zusätzlich im Quelltext bei der entsprechenden Datei nachsehen.

Die Datenstruktur der nutzerspezifischen XML-Daten ist zwar grundsätzlich für alle Nutzertypen vergleichbar, die Elemente `personalInfo`, `workInfo` und `userTags` werden

jedoch üblicherweise nur während der Vorbefragung bei einem Wizard-of-Oz-Test[92] gefüllt. Diese Vorbefragung kann nur bei den Nutzertypen *testUser* und *superUser* durchgeführt werden. Für alle anderen Nutzertypen können allerdings im *Admin-Interface*[93], welches sowohl über den Nutzertypus *superUser*, als auch über den *admin* selbst erreicht werden kann, Kontaktinformationen spezifiziert und angepasst werden.

92 vgl. Kapitel 4.2.5
93 vgl. Kapitel 4.2.3

5.6 XML Datei für die Dialogdesigns

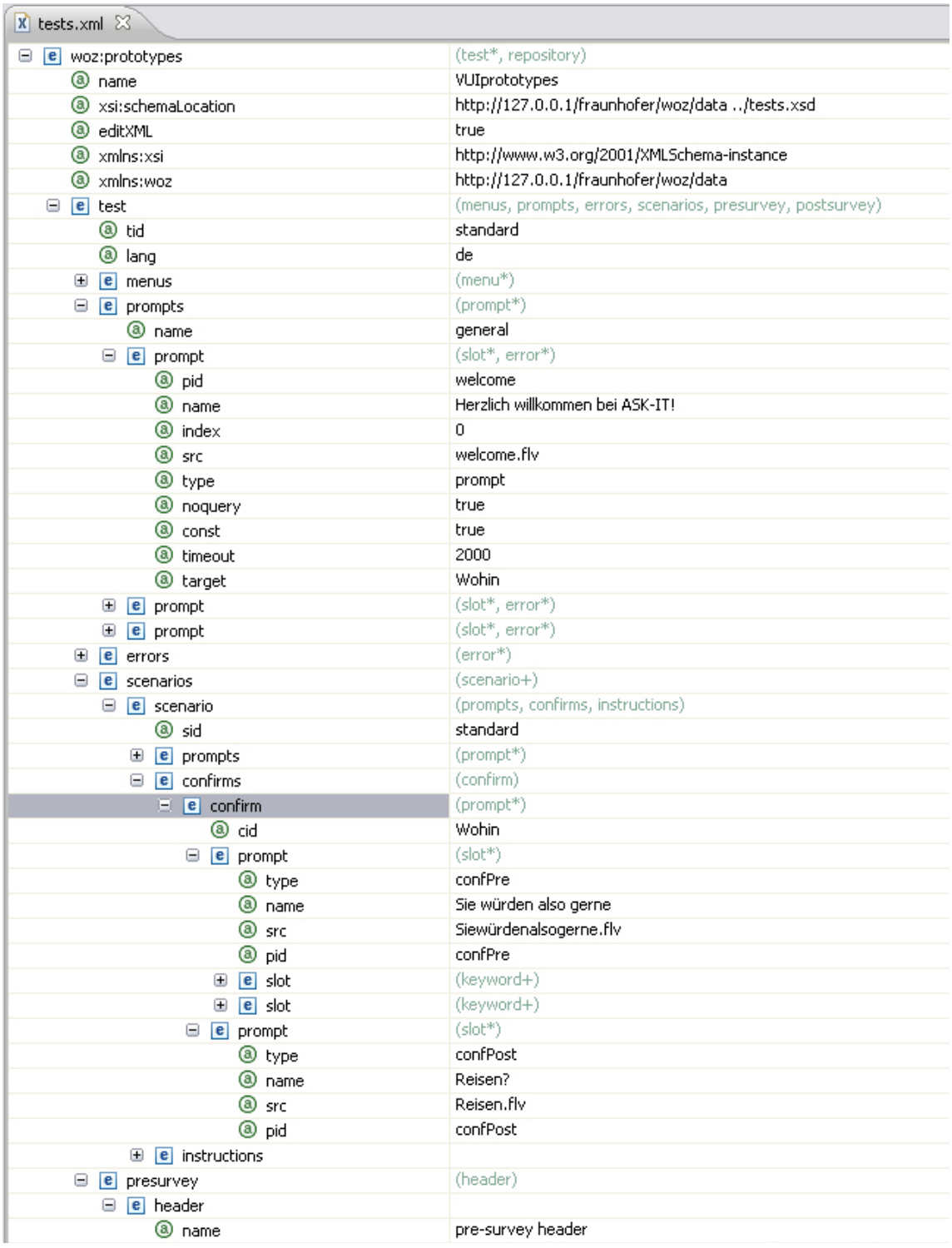
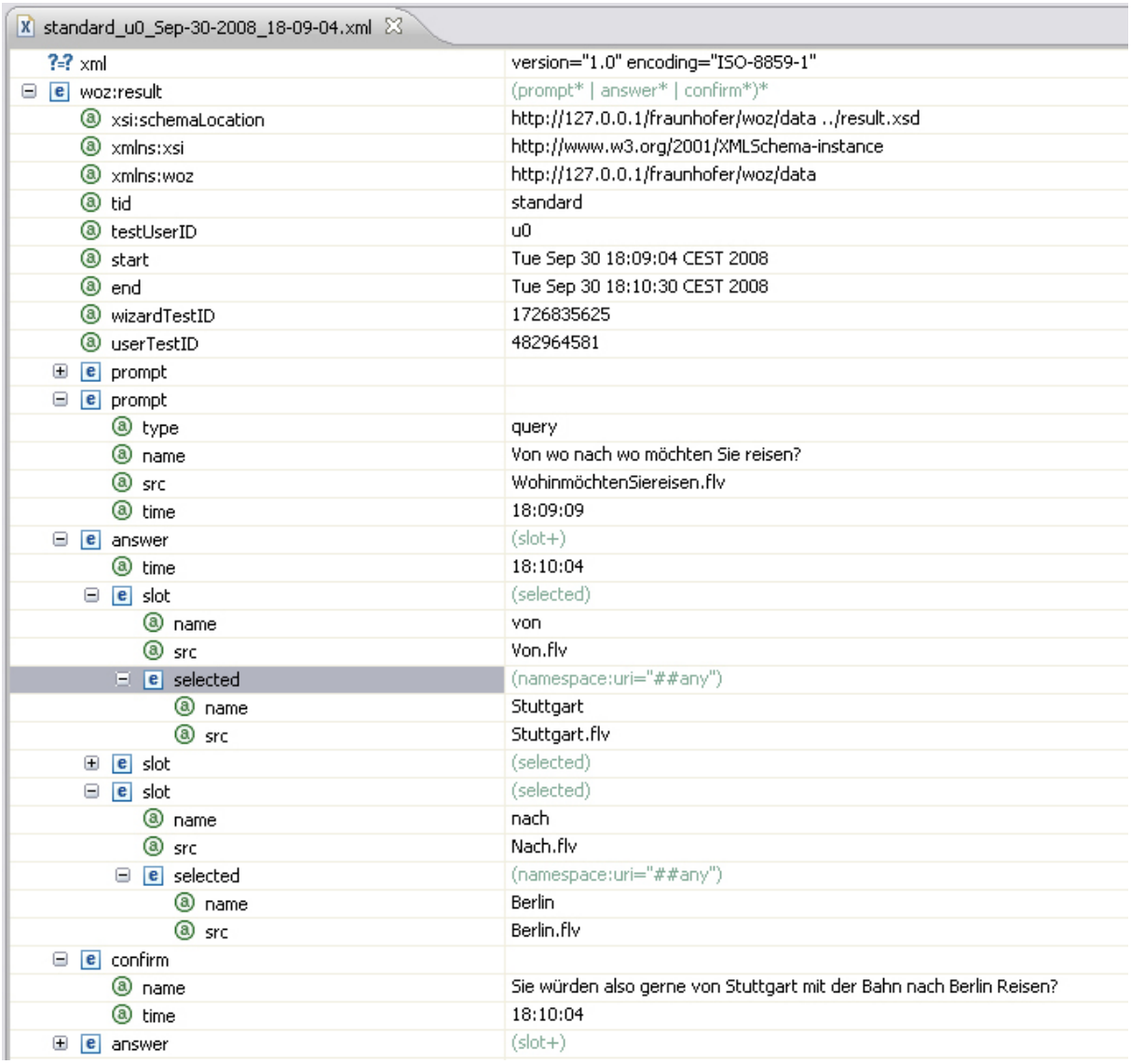


Abbildung 34: Ausschnitt der Datei s0.xml (Eclipse WST)

Für eine genauere Beschreibung der einzelnen Elemente und Attribute, bitte im Quelltext bei der entsprechenden Datei bzw. in Kapitel 4.2.2 nachsehen.

5.7 XML-Dateien für die Testergebnisse



standard_u0_Sep-30-2008_18-09-04.xml

xml	version="1.0" encoding="ISO-8859-1"
woz:result	(prompt* | answer* | confirm*)*
xsi:schemaLocation	http://127.0.0.1/fraunhofer/woz/data ../result.xsd
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xmlns:woz	http://127.0.0.1/fraunhofer/woz/data
tid	standard
testUserID	u0
start	Tue Sep 30 18:09:04 CEST 2008
end	Tue Sep 30 18:10:30 CEST 2008
wizardTestID	1726835625
userTestID	482964581
prompt	
prompt	
type	query
name	Von wo nach wo möchten Sie reisen?
src	WohinmöchtenSiereisen.flv
time	18:09:09
answer	(slot+)
time	18:10:04
slot	(selected)
name	von
src	Von.flv
selected	(namespace:uri="##any")
name	Stuttgart
src	Stuttgart.flv
slot	(selected)
slot	(selected)
name	nach
src	Nach.flv
selected	(namespace:uri="##any")
name	Berlin
src	Berlin.flv
confirm	
name	Sie würden also gerne von Stuttgart mit der Bahn nach Berlin Reisen?
time	18:10:04
answer	(slot+)

Abbildung 35: Ausschnitt der Datei standard_u0_Sep-30-2008_18-09-04.xml (Eclipse WST)

Die Dateinamen werden nach dem Schema `testID` + `userID` + `Datum` + `Startzeit` generiert. Es ist vorgesehen und bereits im entsprechenden XML-Schema spezifiziert, die *Dateinamen* sowie die `testID` in der Datei `users.xml` zu verlinken. Dies ist allerdings momentan noch nicht frontendseitig implementiert. Für eine genauere Beschreibung der einzelnen Elemente und Attribute, bitte ebenfalls im Quelltext bei der entsprechenden Datei nachsehen.

6. Implementation in Adobe Flex

6.1 Entwicklungsumgebung Adobe Flex Builder 3

Für die frontendseitige Implementation des RIA-Prototypen wurde ausschließlich der auf der bereits für das Datendesign verwendeten *Eclipse*-Entwicklungsumgebung (IDE) basierende *Adobe Flex Builder 3* verwendet. Für die Nutzung dieser IDE und der im Flex SDK[94] enthaltenen Komponenten habe ich von Adobe eine unbegrenzte *Educational*-Lizenz erhalten. Eine Testversion der verwendeten Flex IDE ist ebenso wie die verwendete kostenfreie Eclipse WST IDE auf der beigelegten DVD enthalten, deshalb wird an dieser Stelle nicht auf sämtliche Funktionen der Entwicklungsumgebungen eingegangen.

Neben einem Source Code-Editor ist zwar ein Design-Editor vorhanden, in dem visuelle Komponenten über eine grafische Benutzerschnittstelle per Drag&Drop verschoben und ineinander geschachtelt werden können, dieser wurde jedoch nur äußerst selten im Rahmen der frontendseitigen Implementation des RIA-Prototypen verwendet, da die meisten visuellen Elemente per Actionscript- oder MXML-Code platziert worden sind. Sobald eine vorgegebene oder selbst erstellte Flex-Komponente auf die Bühne des Design-Editor gezogen wird, wird ein der Komponente entsprechendes XML-Element in den Quelltext eingefügt.

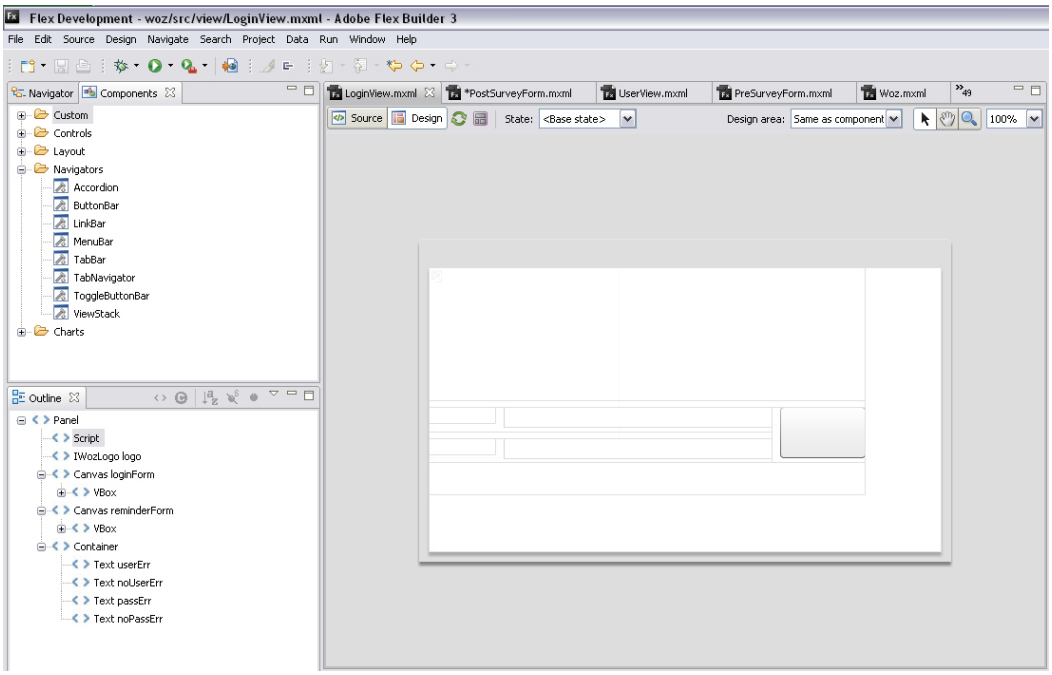


Abbildung 36: Ausschnitt des Design-Editors mit Komponentenauswahl und Outline (Flex Builder)

94 http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK

In der Abbildung des Design-Editors ist insbesondere die Komponentenauswahl (links oben), mit der sämtliche verfügbaren Komponenten auf die Bühne (rechte Seite) gezogen werden können, hervorzuheben. Im eigentlichen Design-Editor auf der rechten Seite ist das Panel für den *Login* sowie u.a. die Rahmen der Textfelder für Benutzername und Passwort zu erkennen.

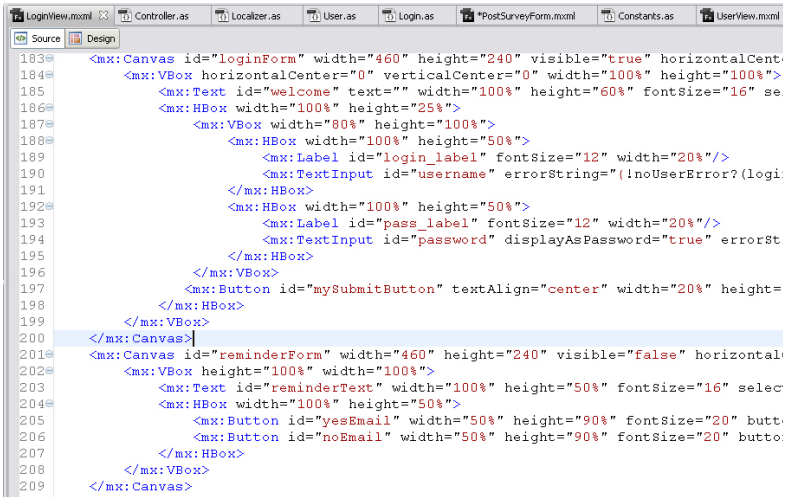


Abbildung 37: Ausschnitt des Source-Editors mit der Datei LoginView.xml (Flex Builder)

Alle Elemente, die im Design-Editor zu sehen sind, können auch im Source-Editor bearbeitet werden, wobei als unsichtbar definierte Elemente (visible="false") nicht im Designmodus editiert werden können.

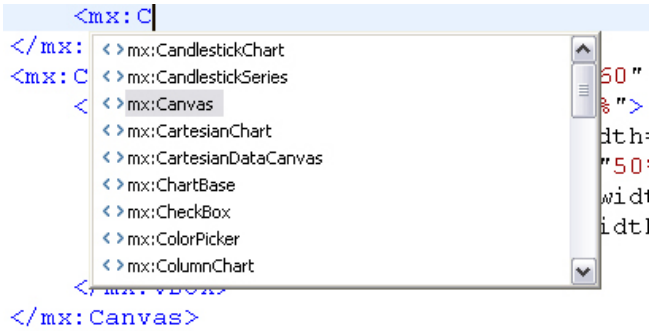


Abbildung 38: Ausschnitt automatische Code-Completion im MXML Editor (Flex Builder)

Der MXML-Editor bietet ebenfalls eine automatische Code-Completion an, wobei immer nur die ersten Buchstaben eingegeben werden müssen und mit der Tastenkombination <Strg>+<Leertaste> alle Elemente angezeigt werden, die mit den eingegebenen Buchstaben beginnen. Neben den visuellen Elementen kann in einer MXML-Datei ebenfalls AS3-Code eingebunden werden. Hierfür muss, ähnlich wie in HTML, ein `script`-Block eingefügt werden, wobei der AS3-Code zusätzlich in einen `CDATA`-Block eingebettet wird, damit sämtliche

Zeichen (inklusive <,> oder &) bedenkenlos innerhalb dieses Blocks verwendet werden können.

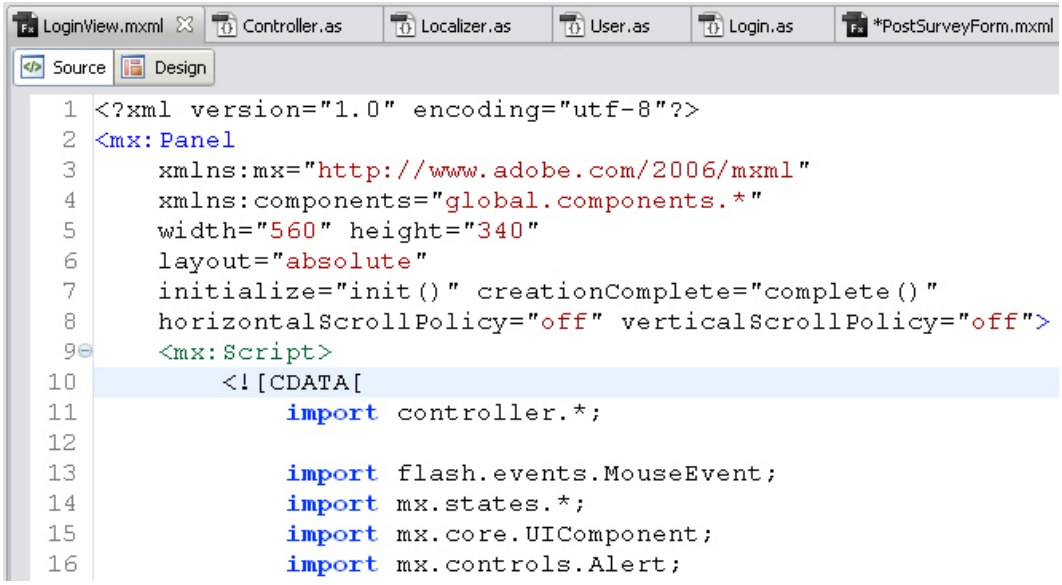


Abbildung 39: Ausschnitt Script- und CDATA-Wrapper-Elemente für AS3-Code (Flex Builder)

Es kann zwar auch außerhalb solcher Script-Blöcke AS3-Code innerhalb von MXML-Attributen eingebunden werden, indem der Attributinhalt in geschweifte Klammern gefasst wird. Dies bringt jedoch den Nachteil, dass für XML reservierte Zeichen (<,> oder &) nicht in dieser Form verwendet werden können.[95]

Neben den visuellen Komponenten können per MXML auch Style-Blöcke, ebenfalls mit CSS-Styles bei HTML Seiten sowie weitere nicht-visuelle Komponenten, wie beispielsweise XML-Daten eingebunden werden. Beispiele hierfür finden sich im Quelltext.

Der innerhalb eines Script-Blocks eingebundene AS3-Code unterscheidet sich kaum von dem in reinen AS3-Klassen verwendeten Code. Es können sowohl externe Pakete oder Komponenten über `import`-Statements eingebunde, als auch alle Arten von Methoden und Variablen definiert werden. Über das Schlüsselwort `this`, kann die Instanz der in der jeweiligen MXML-Datei beschriebenen Klasse angesprochen werden. Die Struktur einer MXML Datei kann zudem auch durch reinen AS3-Klassen-Code beschrieben werden, wobei beispielsweise im Falle eines *Panel*-Containers, die Klasse `mx.containers.Panel`, oder bei einem Button-Element, wie im Folgenden Beispiel, die Klasse `mx.controls.Button` erweitert werden müsste.

95 https://bugs.adobe.com/jira/browse/SDK-12930

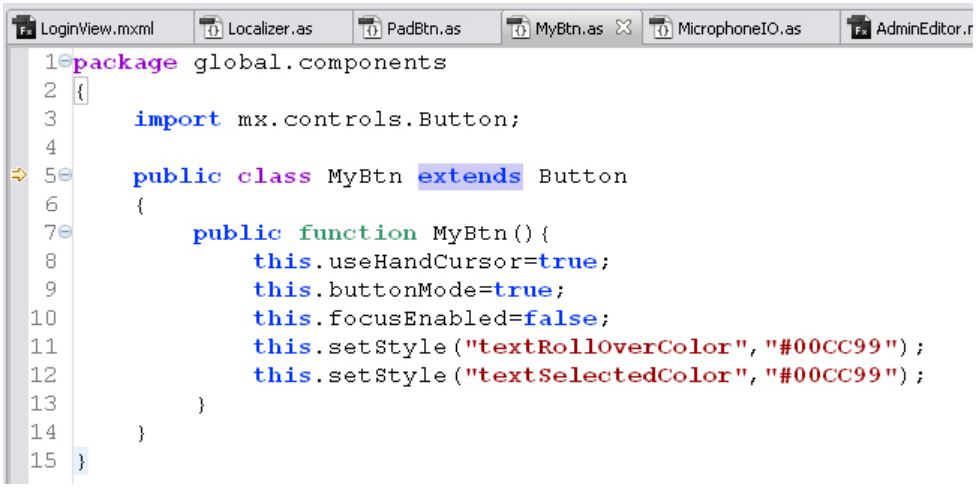


Abbildung 40: Erweiterung der im Flex-SDK enthaltenen Klasse Button.as (Flex Builder)

Wie aus anderen objektorientierten Programmiersprachen bekannt, erbt die Klasse MyBtn sämtliche Eigenschaften von der mit dem Schlüsselwort `extends` erweiterten Klasse `mx.controls.Button`. Im Konstruktor der MyBtn-Klasse (`public function MyBtn()`), werden öffentliche Eigenschaften der Button-Klasse überschrieben und *CSS-Styles* gesetzt (`setStyle()`).

6.1.1 Referenzierung der über MXML eingebundenen Komponenten

Wird zu einem per MXML eingebundenen Element das `id`-Attribut angegeben, so ist dies gleichbedeutend mit dem Instanznamen des Objekts. Dieses Attribut kann dann direkt verwendet werden um das entsprechende Objekt in Objektnotation zu referenzieren. Im Vergleich dazu muss in JavaScript in diesem Fall immer zunächst die JS-Funktion `document.getElementById()`[96] oder eine vergleichbare Funktion verwendet werden, um eine Objekt direkt referenzieren zu können.

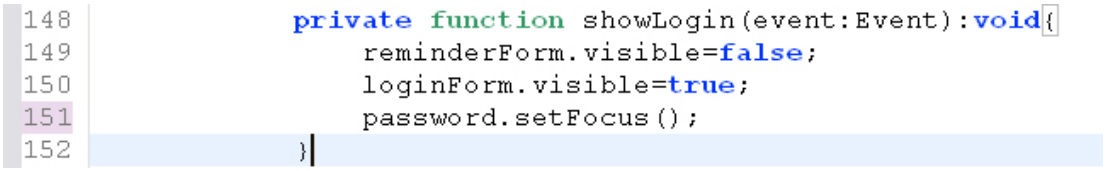
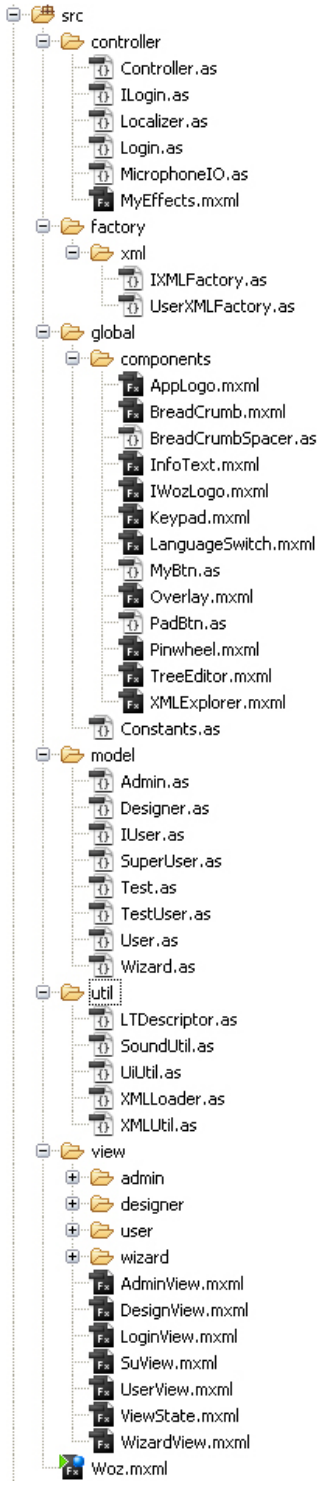


Abbildung 41: Direkte Referenzierung von MXML Elementen nach ID (`LoginView.mxml`)

96 http://de.selfhtml.org/javascript/objekte/document.htm#get_element_by_id

6.2 Aufbau der Flex-Applikation

Beim Aufbau der Flex-Applikation wurde versucht, streng nach dem Model-View-Controller Prinzip (Gamma et al. 1995) vorzugehen. Sämtliche ausschließlich die Dartellung, bzw. das grafische Interface betreffenden Klassen, sind als MXML-Dateien im Paket „`view`", bzw. als globale Komponenten im Paket „`global`" angelegt.



Die Hauptklasse `Woz.mxml` findet sich im Quellcode Stammverzeichnis „`src`" und ist die Basis dieser Applikation. Aus dieser Applikations-MXML Klasse wird schließlich auch die entsprechende *swf*-Datei dieser Applikation erstellt.

Alle anderen Klassen sind als *.as*-Klassen (außer der Effekt-Sammlung `MyEffects.mxml`) angelegt und in die Pakete `controller`, `model` sowie `util` und `factory` verteilt.

Im Paket `controller` sind die Klassen abgelegt, die für die Zustände der Applikation sowie die Zustandsübergänge zuständig sind abgelegt.

Im Paket `model` finden sich entsprechend ausschließlich die für die Nutzerrollen (`IUser.as`, `User.as`, `Admin.as`, `Designer.as`, `TestUser.as`, `Wizard.as` und `SuperUser.as`) sowie den Test (`Test.as`) verwendeten Klassen und Interfaces. Diese Klassen beinhalten auch die Schnittstelle zum Wowza Media Server.

Während im Paket `factory` die Klassen abgelegt sind, die zum Beispiel für die valide Erstellung von XML-Daten zuständig sind (`UserXMLFactory.as`), sind im Paket `util` alle Klassen zu finden, die u.a. für allgemeine Aktionen, wie das Laden und Speichern von XML-Daten (`XMLLoader.as/IOUtil.as`), die Manipulation von XML-Daten (`XMLUtil.as`) oder die Acquisition[97] von verwandten UI-Elementen (`UiUtil.as`) verwendet werden.

97 http://www.onflex.org/ted/2007/01/acquisition-within-flex.php

Abbildung 42: Aufbau der Applikation (Flex Builder)

Im Folgenden sollen exemplarisch einzelne Klassen aus den genannten und auf der vorherigen Seite dargestellten Paketen herausgegriffen und Ausschnitte davon kurz beschreiben werden. Für weitere Informationen zu den hier nicht mehr näher beschriebenen Komponenten und Klassen bitte im Quelltext nachsehen.

6.3 Datei Woz.mxml

Die Datei `Woz.mxml` ist die Datei, auf der die gesamte Flex-Applikation aufgebaut ist. Sämtliche visuellen Komponenten und Actionscript-Klassen, die in der Implementation verwendet werden sind dieser Datei untergeordnet und sie ist somit die oberste Klasse in der Klassenhierarchie. Über das Schlüsselwort `parentApplication` kann die Instanz dieser Klasse von jeder ihr untergeordneten Komponente referenziert werden.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <mx:Application
3      xmlns:mx="http://www.adobe.com/2006/mxml"
4      xmlns:view="view.*"
5      xmlns:comp="global.components.*"
6      layout="absolute"
7      name="Fraunhofer Wizard of Oz Tool"
8      backgroundColor="0xdddddd"
9      currentState="initState"
10     usePreloader="true"
11     initialize="init();"
12     preinitialize="pre();"
13     creationComplete="complete();"
14     frameRate="45"
15     verticalScrollPolicy="off" horizontalScrollPolicy="off" viewSourceURL="srcview/index.html">
```

Abbildung 43: *Kopf* der Datei `Woz.xml` (Flex Builder)

Über die im `mx:Application`-Element festgelegten Attribute können globale Eigenschaften, wie die Bildwiederholungsrate (`frameRate`), die Hintergrundfarbe (`backgroundColor`) oder die Anzeige eines Preloaders (`usePreloader`) festgelegt werden. Die Funktion `init()` wird aufgerufen, sobald die Applikation initialisiert ist, die Funktion `complete()`, sobald allein dieser Applikation direkt eingebundenen visuellen Elemente erstellt worden sind.

```
66         /* initialize application */
67         private function init():void{
68             this.langStates = new Array();
69             var lang:String=Application.application.parameters.pageLang!=undefined?
70                             Application.application.parameters.pageLang:Capabilities.language;
71             this.currLang="lang:"+lang;
72             closeBtn.addEventListener(MouseEvent.CLICK,this.contr.close);
73         }
```

Abbildung 44: lokale Funktion `init()` der Datei `Woz.xml` (Flex Builder)

Sobald die Applikation initialisiert ist, wird die Lokalisation, also die Sprache „`lang`" mit der die Applikation gestartet wird, festgelegt. Das Kürzel für die jeweilige Sprache (z.B. `de`

oder `en`) kann der Applikation entweder per Parameter bei der Einbettung in eine Website mitgeliefert werden[98] oder über die Flex-eigene Variable `Capabilities.language`[99] abgerufen werden.

```
167     <!-- app states-->
168     <mx:states>
169         <mx:State name="initState">
170             <mx:AddChild relativeTo="{this}">
171                 <view:LoginView id="loginView" name="view_00"
172             </mx:AddChild>
173         </mx:State>
174         <mx:State name="sState" id="sState"></mx:State>
175         <mx:State name="wState" id="wState"></mx:State>
176         <mx:State name="tState" id="tState"></mx:State>
177         <mx:State name="aState" id="aState"></mx:State>
178         <mx:State name="dState" id="dState"></mx:State>
179     </mx:states>
```

Abbildung 45: Haupt-Applikationzustände in der Datei `Woz.xml` (Flex Builder)

Sämtliche Haupt-Zustände, die diese Applikation annehmen kann sind in der Datei `Woz.xml` festgelegt. Der aktuelle Zustand einer Flex-Applikation oder einer MXML-Komponente kann über die Eigenschaft `currentState`[100] festgelegt werden. Dieses Instrument wird in dieser Form nur in der Applikations-MXML verwendet, da die Flex-Zustände (`<mx:states/>`) in den untergeordneten Komponenten ausschließlich für die Internationalisierung verwendet werden.

6.4 Paket „view“

Im „`view`“-Paket befinden sich sämtliche MXML-Komponenten, die direkt mit den grafischen Benutzerschnittstellen der fünf Benutzertypen in Verbindung stehen. Ich möchte zunächst exemplarisch die Vererbung und Erweiterung bestehender Komponenten anhand der Datei `ViewState.mxml` vorstellen, auf deren Basis sämtliche nutzerspezifischen Ansichten erweitert werden.

6.4.1 Abstrakte Klassen und Methoden in AS3

Da es in AS 3 nicht vorgesehen ist, wie beispielsweise in Java *abstrakte Klassen* und *Methoden* zu definieren, welche in untergeordneten Klassen überschrieben werden müssen, habe ich abstrakte (hier leere) Funktionen lediglich entsprechend im Quelltext markiert. Es existieren zwar Möglichkeiten, eine Überschreibung der entsprechenden Klassen und Methoden zu erzwin-

98 vgl. http://code.google.com/p/swfobject/wiki/documentation (flashVars Object)
99 vgl. http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/system/Capabilities.html
100 vgl. http://livedocs.adobe.com/flex/201/html/states_039_07.html

gen[101], wobei mir der Aufwand hierfür jedoch nicht gerechtfertigt erschien.

```
<mx:Canvas
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="86%" height="86%"
    preinitialize="preInit()" initialize="init()" creationComplete="complete()"
    verticalScrollPolicy="off" horizontalScrollPolicy="off" xmlns:comp="global.components.*"
    <mx:Script>
        <![CDATA[
            import controller.*;
            import model.TestUser;
            import mx.events.CloseEvent;
            import mx.controls.Alert;
            import mx.controls.Text;

            public var isSU:Boolean=false;

            // set true if component contains other components to be localized
            public var bubbles:Boolean=true;

            [Bindable]
            public var statusTxt:Text;
            [Bindable]
            protected var currState:Canvas;
            protected var pseudoStates:Array;
            [Bindable]
            protected var numOfStates:int;
            [Bindable]
            protected var currStateNo:int;

            protected var contr:Controller;
            protected var FX:MyEffects=new MyEffects();

            protected function preInit():void{
                this.visible=false;
                this.contr=this.parentApplication.contr;
            }

            // ABSTRACT functions!!!
            protected function init():void{}        // to be overridden in subcomponent!
            protected function complete():void{}    // to be overridden in subcomponent!
```

Abbildung 46: Komponente `ViewState.mxml` mit abstrakten Funktionen (Flex Builder)

101 vgl. http://joshblog.net/2007/08/19/enforcing-abstract-classes-at-runtime-in-actionscript-3/

6.4.2 Das [Bindable]-Metatag

Alle Variablen, für welche zur Laufzeit Änderungen zu erwarten sind, sollten über das Meta-Tag `[Bindable]`[102] (hier: Zeilen 20, 22, 25 und 27) entsprechend markiert werden. Damit kann sicher gestellt werden, dass alle Elemente, welche die entsprechende Variable verwenden benachrichtigt werden, sobald sich der Inhalt der Variable ändert.

Die Variable `bubbles` vom Typ `Boolean` muss von sämtlichen Komponenten gesetzt werden, die wiederum Komponenten enthalten, welche bei der Lokalisation einbezogen werden sollen.

Exemplarisch für die visuellen Komponenten, welche die soeben beschriebene `ViewState`-Komponente erweitern, werden im Folgenden Auszüge der `DesignView`-Komponente beschrieben.

```
2  <ViewState xmlns="view.*" xmlns:mx="http://www.adobe.com/2006/mxml"
3      <mx:Script>
4          <![CDATA[
5              import controller.*;
6              import model.*;
7              import flash.events.*;
8              import mx.events.*;
9              import mx.controls.*;
10
11             [Bindable]
12             public var closeBtn:Boolean;
13
14             [Bindable]
15             public var designer:Designer;
16
17             override protected function init():void{
18                 this.visible=false;
19                 this.statusTxt=new Text();
20                 this.pseudoStates=pseudoStateCnt.getChildren();
21                 this.numOfStates=pseudoStates.length;
22                 this.currState=pseudoStates[0];
23                 this.contr=this.parentApplication.contr;
24             }
```

Abbildung 47: Komponente `DesignView.mxml` mit überschriebener `init()`-Funktion

6.4.3 Erweiterung von Komponenten und Überschreiben von Funktionen

Der Name der Komponente `ViewState` ist bei einer Erweiterung entsprechend als Root-Element (hier: Zeile 2 in `DesignView.xml`) zu verwenden. Die in der erweiterten Komponente definierte abstrakte Funktion `init()` wird über das Schlüsselwort `override` überschrieben (hier: Zeile 17).

102 vgl. http://livedocs.adobe.com/flex/2/docs/00001653.html

```
70    <mx:TitleWindow id="statesCnt" maxWidth="1500" maxHeight="1000" minWidth="750"
71        <mx:Canvas name="states" id="pseudoStateCnt" width="100%" height="100%">
72            <mx:Canvas name="state_00" id="editorView" visible="true" width="100%"
73                <designer:TestExplorer id="testExplorer" name="view_testExplorer"
74            </mx:Canvas>                    view.designer.TestExplorer
75        </mx:Canvas>
76    </mx:TitleWindow>
77    <mx:Container visible="false">
78    </mx:Container>
79 </ViewState>
```

Abbildung 48: Komponente `DesignView.mxml` mit Kind-Elementen

Werden, wie im Fall der `DesignView`-Komponente, in einer Komponente weitere visuelle Komponenten per MXML eingebunden, so können in einer möglichen Unterklasse dieser Komponente keine zusätzlichen visuellen Komponenten per MXML-Definition eingebunden werden[103]. Allerdings kann diese Limitation per Skript-Definition innerhalb eines Skript-Tags umgangen werden.

Für weitere detaillierte Informationen zu den im Paket `view` enthaltenen Komponenten, bitte im Quelltext an den entsprechenden Stellen nachsehen.

6.5 Paket „controller“

Im `controller`-Paket sind primär AS-Klassen enthalten, die für Aktionen zuständig sind, die unmittelbar mit den visuellen Komponenten zusammenhängen. Exemplarisch möchte ich hier in Ausschnitten die Klasse `Login.as` beschreiben, welche für die Anwendung von Lokalisationszuständen der View-Komponenten verwendet wird.

103 http://www.tink.ws/blog/extending-your-own-mxml-components/

```
1  package controller
2  {
3      import mx.core.UIComponent;
4      import mx.states.*;
5      /**
6       *  Die Klasse Localizer.as dient ausschließlich der Anwendung von Lokalisationszuständen.
7       *  Mittels in der externen XML Datei locales.xml beschriebender Definitionen, welche bei
8       *  der Erzeugung einer Instanz dieser Klasse als XMLList-parameter übergeben werden.
9       */
10     public class Localizer
11     {
12         public var statesSet:Boolean;   // set States only once
13
14         private var states:Array;
15         private var localesArr:Array;
16         private var locales:XMLList;
17         private var refObj:UIComponent;
18
19
20         /**
21          *  Konstruktor der Klasse Localizer.as
22          *  @param locales:XMLList      XML-Liste mit Lokalisierungen
23          *  @param refObj:UIComponent   UI-Komponente auf die die Lokalisierungen angewendet werden
24          */
25
26         public function Localizer(locales:XMLList,refObj:UIComponent)
27         {
28             this.localesArr=new Array();
29             this.locales=locales;
30             this.refObj=refObj;
31             if(locales&&refObj){
32                 this.statesSet=this.setStates();
33             }
34         }
35         /* Getter- and Setter-Methods for private Variables */
36         public function getStates():Array{
37             return this.states;
38         }
39         public function getLocales():Array{
40             return this.localesArr;
41         }
42
```

Abbildung 49: oberer Teil der Klasse `Localizer.as` (Flex Builder)

Anhand dieser Klasse möchte ich lediglich kurz die von mir verwendete Kommentierweise sowie das in AS3 verwendete *Strict-Typing* erläutern.

6.5.1 Kommentierweise

In Anlehnung an die aus Java (JavaDoc[104]) bekannte sowie der in den AS-Klassen des Flex SDK verwendeten ähnlichen Kommentierweise (ASDoc[105]) wurden die wichtigsten Klassen und MXML-Komponenten umfassend kommentiert und per *ANT-Skript*[106] in eine HTML-Basierte *API-Dokumentation* überführt. Wie im vorangegangenen Codeausschnitt aus der Klasse `Localizer.as` (u.a. Zeile 5-9) zu sehen, sind die Kommentare, welche später auch in der mittels ASDoc erstellten API-Dokumentation wiedergefunden werden können, im Quelltext Editor des Flex-Builders hellblau eingefärbt und in deutscher Sprache. Einfache Kommentare innerhalb des Quelltexts, wie hinter zwei `//`-Zeichen (Zeile 12) oder zwischen den Sternchen in `/* */` (Zeile 35), sind grün eingefärbt und in englischer Sprache verfasst.

104 http://java.sun.com/j2se/javadoc/
105 http://livedocs.adobe.com/flex/3/html/asdoc_3.html#189145
106 http://labs.zeroseven.de/2008/07/asdoc-mit-ant-in-flexbuilder-eclipse/

6.5.2 Strict-Typing in ActionScript 3

Der Datentyp von in ActionScript 3 definierten Variablen sollte bei der Variablendefinition mit der Schreibweise `:Typ` („Typ“ durch entsprechenden Datentyp ersetzen), wie im Folgenden Beispiel aus der Datei Localizer.xml angegeben werden. Dies ist eines der zentralen Merkmale in dem sich die ECMA-Scriptsprache ActionScript 3 von anderen Skriptsprachen wie *JavaScript* unterscheidet und aus diesem Grund möchte ich an dieser Stelle darauf näher eingehen.

```
14          private var states:Array;
15          private var localesArr:Array;
16          private var locales:XMLList;
17          private var refObj:UIComponent;
```

Abbildung 50: Strict-Typing in AS3 (`Localizer.as`, Flex Builder)

Geschieht dies nicht („dynamic-“ oder „loose-typing“)[107], so kann die Applikation zwar kompiliert werden, es erscheint jedoch ein entsprechender Hinweis auf die fehlende Typendeklaration.

```
14          private var states:Array;
15  1008: variable 'localesArr' hat keine Typdeklaration.sArr;
16          private var locales:XMLList;
17          private var refObj:UIComponent;
```

Abbildung 51: Loose-Typing in AS3 (`Localizer.as`, Flex Builder)

Eine fehlende Typenangabe bei der Variablendeklaration oder bei den Parametern einer Funktion muss zwar nicht weiter problematisch sein, der Compiler kann jedoch in diesem Fall nicht überprüfen, ob die der entsprechenden Variable zugewiesenen Daten gültig sind.

```
14          private var states:Array;
15          private var localesArr="noarray"
16          private var locales:XMLList;
17          private var refObj:UIComponent;
```

Abbildung 52: Fehlerhafte Wertzuweisung (`Localizer.as`, Flex Builder)

Wird einer dynamisch deklarierten Variable mit fehlender Typendeklaration ein Wert zugewiesen (hier: Zeile 15 der Datentyp `String`), so kann dies beispielsweise dann zu Folgefehlern führen, falls diese Variabe zur Laufzeit als Parameter an eine Funktion übergeben wird, die eigentlich einen Wert vom Datentyp *Array* erwartet.

107 http://www.learning-how.to/OOP_in_AS/Voraussetzungen/content-docs/voraus_04.php

In dem für diese Arbeit implementierten Prototypen wurden in Flex sämtliche Variablen per Strict-Typing deklariert.

6.6 Pakete „`factory`" und „`util`"

Im Paket `factory` sind Klassen enthalten, die ausschließlich der Erzeugung von Datenstrukturen dienen. Dies bietet sich insbesondere für die Erzeugung spezieller XML-Strukturen an und wurde für diesen Prototypen exemplarisch und frei nach dem sog. „Factory-Method-Pattern"[108] für die Erzeugung von Nutzerdaten realisiert. In diesem Fall wird dadurch sicher gestellt, dass die erzeugte XML-Struktur valide ist und somit dem jeweiligen XML-Schema entspricht. Dieses Vorgehen wurde für diesen Prototypen nur im Falle der im Folgenden vorgestellten Klasse `UserXMLFactory.as` verwendet, es ist jedoch geplant dies in weiteren Programmversionen auch auf andere Datenstrukturen anzuwenden. Darüber hinaus möchte ich anhand des folgenden Beispiels kurz die Verwendung der E4X[109]-Implementation in AS3 demonstrieren.

108 vgl. Sanders & Cumaranatunge (2007), S. 65 ff
109 http://livedocs.adobe.com/flex/2/docs/00001912.html

```
package factory.xml
{
    public class UserXMLFactory implements IXMLFactory
    {

        public function createXML(xmlNode:XML):XML
        {
            var wozNS:Namespace=xmlNode.namespace("woz");
            var xsiNS:Namespace=xmlNode.namespace("xsi");
            var userXML:XML=new XML("<"+xmlNode.@userType+"/>");
            userXML.addNamespace(xsiNS);
            userXML.setNamespace(wozNS);
            userXML=addSchema(userXML);
            userXML=addChildren(userXML,xmlNode.@userID);
            return userXML;
        }

        private function addSchema(xml:XML):XML{
            var userXML:XML=xml;
            var schemaLocation:String=userXML.namespace("woz").toString()+" ../"+userXML.localName()
            userXML["@schemaLocation"]=schemaLocation;
            userXML.@schemaLocation.setNamespace(userXML.namespace("xsi"));
            return userXML;
        }
        private function addChildren(xml:XML,id:String):XML{
            xml=createUserData(xml,id);
            xml.appendChild(<userTags/>);
            return xml;
        }
        private function createUserData(xml:XML,id:String):XML{
            var userData:XML=<userData userID="" forename="" surname=""/>;
            userData.@userID=id;
            userData.appendChild(<contactInfo eMail=""/>);
            if(xml.localName()=="testUser"||xml.localName()=="superUser"){
                userData.appendChild(<personalInfo birthDay="" gender=""/>);
                userData.appendChild(<workInfo/>);
            }
            xml.appendChild(userData);
            return xml;
        }
        public function addUserProfession(userXML:XML,type:String=null, years:int=0):XML{
            userXML.workInfo.appendChild(<profession professionType="" sinceYears=""/>);
            userXML.workInfo.@professionType=type;
            userXML.workInfo.@sinceYears=years;
            return userXML;
        }
        public function addUserTag(userXML:XML,name:String=null):XML{
            var newTag:XML=userXML.userTags.appendChild(<tag/>);
            newTag.appendChild(name);
            return userXML;
        }
    }
}
```

Abbildung 53: Factory-Klasse `UserXMLFactory.as` (Flex Builder)

Bei der Erstellung der XML-Struktur in der öffentlichen Methode `createXML()` werden hier zunächst Rohdaten übergeben (`xmlNode:XML`), aus denen schließlich in lokalen Methoden (`addSchema()`, `addChildren()` und `createUserData()` schrittweise die korrekte XML-Struktur erzeugt und schließlich zurückgegeben wird. Desweiteren werden zwei öffentliche Methoden angeboten, die das Hinzufügen weiterer Details zum entsprechenden Nutzer ermöglichen.

6.6.1 E4X in ActionScript 3

Bei der Arbeit mit XML-Daten in Flex hat sich die Unterstützung des neuen ECMA-Script for XML (E4X)[110]-Standards in AS3[111] als äußerst hilfreich erwiesen. Zwar konnte bereits in früheren ActionScript Versionen mit XML-Daten gearbeitet werden, der neue Standard bringt jedoch einige wesentliche Verbesserungen mit sich, welche die Arbeit mit XML sehr vereinfachen können.

XML ist seit AS3 ein eigenständiger einfacher Datentyp und muss nicht wie bisher als `String` angegeben werden, der dann in ein XML-Objekt umgewandelt (geparsed) wurde.

```
30        private function createUserData(xml:XML,id:String):XML{
31            var userData:XML=<userData userID="" forename="" surname=""/>;
32            userData.@userID=id;
33            userData.appendChild(<contactInfo eMail=""/>);
34            if(xml.localName()=="testUser"||xml.localName()=="superUser"){
35                userData.appendChild(<personalInfo birthDay="" gender=""/>);
36                userData.appendChild(<workInfo/>);
37            }
38            xml.appendChild(userData);
39            return xml;
40        }
```

Abbildung 54: Lokale Methode `createUserData()` (Klasse `UserXMLFactory.as`)

In der Methode `createUserData()` wird auf die beschriebene Weise ein XML-Element ohne Anführungszeichen (z.B. `<workInfo/>`, Zeile 36) an ein anderes XML-Element (`userData`) angehängt. Attribute können über die `.@`-Syntax (vgl. Zeile 32) adressiert werden. Ebenso können Attribute bereits bei der Variablendeklaration angegeben werden (`<userData userid=""...>`, Zeile 31).

Der lokale Name eines Elements kann ohne eventuelle Namespace-Deklarationen über die Methode `localName()` erhalten werden (Zeile 34).

110 http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-357.pdf
111 http://livedocs.adobe.com/flex/2/docs/00001912.html

Eine erweiterte E4X-Syntax wurde im Paket `util` in der Klasse `XMLUtil` und der statischen Methode `findNodes()` verwendet, welche ich hier exemplarisch vorstellen möchte.

```
115        /**
116        * Die globale statische Funktion findNodes kann zum Auffinden bestimmter Knoten oder
117        * Attributen in einer XML-Struktur verwendet werden.
118        *
119        * benötigte Parameter:
120        * @param xmlData:XML        zu durchsuchende XML Struktur
121        * @param attName:String     Attribut nach dem gesucht wird (falls kein Element gesucht wird)
122        * @param searchString:String    String nach dem gesucht wird
123        *
124        * optionale Parameter:
125        * @param tagName:Boolean    falls true, wird nicht in attributnamen, sondern in Elementnamen ges
126        * @param ignorecase:Boolean falls true, wird in der Suche nicht nach Groß-/Kleinschreibung unter
127        */
128        public static function findNodes(xmlData:XML,attName:String,searchString:String,
129                                          tagName:Boolean=false,ignoreCase:Boolean=true):XMLList{
130            var re:RegExp;
131            var flags:String=ignoreCase?"ig":"g";    // "i"=ignore case, "g"=global
132            re=new RegExp(searchString,flags);
133            var foundNodes:XMLList=null;
134            if(tagName){     // find searchString in Elements!
135                try{
136                    foundNodes=xmlData?xmlData.(name().toString().search(re)
137                                ||descendants().name().toString().search(re)):null;
138                }catch(e:Error){
139                    trace(e.getStackTrace());    // @debug: print stack trace!
140                }
141            }else{       // find searchString in attributes specified in attName-param!
142                foundNodes=
143                    xmlData?xmlData.descendants().(attribute(attName).toString().search(re)!=-1):null;
144            }
145            return foundNodes;
146        }
```

Abbildung 55: Globale statische Methode `findNodes()` (Klasse `XMLUtil.as`)

ActionScript 3 bietet zwei zentrale XML-Datentypen (`XML` und `XMLList`) an, wobei in einer Variable vom Typ `XML` XML-Strukturen mit genau einem *Wurzel*-Knoten enthalten sind, während der auf dem Typ `XML` basierende Datentyp `XMLList` eine Array-artige Kollektion solcher XML Strukturen darstellt.

Über die sog. *Dot-Syntax* können XML-Objekte direkt mit ihrem jeweiligen lokalen Namen angesprochen werden. So liefert beispielsweise die Syntax

```
xmlData.kindknoten.localName()
```

den String `"kindknoten"` bei einem direkt untergeordneten Element `kindknoten` des XML-Elements `xmlData`. Die Syntax

```
xmlData..kindknoten.(@name=="kind")
```

liefert hingegen alle untergeordneten Elemente mit dem lokalen Namen `kindknoten`, die ein Attribut mit dem Namen `name` enthalten, welches `"kind"` heißt.

6.7 Paket „global“

Im Paket „global.components“ sind sämtliche globalen Komponenten abgelegt, die nicht direkt und ausschließlich einer bestimmten Nutzerrolle zugeordnet werden können. Eine besonders erwähnenswerte Komponente in diesem Paket ist die Komponente XMLExplorer.mxml. Diese Klasse bildet die Grundlage für die XML-Editoren der nutzerspezifischen GUIs Admin-Explorer, Design-Explorer und Wizard-Explorer, von welchen sie erweitert wird.

```
168    /**
169    * Die geschützte Funktion tree_labelFunc bietet die Möglichkeit, den Elementen
170    * in einer Tree-Komponente (mx.controls.Tree) individuelle Labels zu geben.
171    *
172    * @param item:XML
173    */
174
175    protected function tree_labelFunc(item:XML):String {
176        var label:String;
177        var itemName:String=item.localName();
178        var itemNameAtt:String=item.attribute(labelAtt);
179        switch(itemNameAtt){
180            case(""):
181                if(item.@type.toString()!=""){
182                    label=item.localName()+":"+item.@type;
183                    // use attribute "type" as Element-Label
184                }else if(item.@tid.toString()!=""){
185                    label=item.localName()+":"+item.@tid;
186                    // use attribute "tid" as Element-Label
187                }else if(item.@pid.toString()!=""){
188                    label=item.localName()+":"+item.@pid;
189                    // use attribute "pid" as Element-Label
190                }else if(item.@mid.toString()!=""){
191                    label=item.localName()+":"+item.@mid;
192                    // use attribute "mid" as Element-Label
193                }else if(item.@sid.toString()!=""){
194                    label=item.localName()+":"+item.@sid;
195                    // use attribute "sid" as Element-Label
196                }else if(itemName==""){
197                    label=cdataTxt.text;
198                    // CDATA Element
199                }else{
200                    label=itemName;
201                }
202            break;
203            default:
204                if(itemName==item.attribute(labelAtt).toString()){
205                    label=itemName;
206                    // use attribute specified in labelAtt as Element-Label
207                }else{
208                    label=item.localName()+":"+item.attribute(labelAtt);
209                    // use localName and labelAtt as Element-Label
210                }
211            break;
212        }
213        return label;
214    }
```

Abbildung 56: Geschützte *Labeling*-Funktion tree_labelFunc() (XMLExplorer.mxml)

Diese *Labeling*-Funktion für die Kompontente mx.controls.Tree wird verwendet, um in den Baumansichten der jeweiligen grafischen Schnittstellen unterschiedliche Elementnamen

anzeigen zu können[112].

```
322     <mx:Tree id="myTree" width="100%" percentHeight="{noinput?100:95}" dataProvider="{this.treeData}"
323         iconField="@icon" showRoot="true" labelField="@name" labelFunction="tree_labelFunc"
324         editable="true" verticalScrollPolicy="auto" itemEditBeginning="startEditing(event);"
325         itemEditEnd="processData(event);" itemOpen="onItemOpen(event)" itemClose="onItemClose(event)"
326         allowMultipleSelection="true" itemRenderer="mx.controls.treeClasses.MyTreeItemRenderer"/>
```

Abbildung 57: MXML-Definition der Tree-Komponente (`XMLExplorer.mxml`)

Bei der Einbindung der Tree-Komponente können neben der bereits angesprochenen labeling-Funktion `tree_labelFunc()` weitere zentrale Eigenschaften, wie die XML-Datenbasis (`dataProvider`) oder Events, wie `itemEditBeginning` definiert werden. Die Funktion `startEditing(event)` wird bei Auswahl eines Elements in der Baumansicht ausgeführt, wobei es auf diese Weise möglich ist, einen eigenen Editor für jedes einzelne Element innerhalb der Baumansicht zu erstellen. Diese Art und Weise habe ich auch die Editor-Komponenten `view.admin.AdminEditor`, `view.designer.TestEditor` und `view.wizard.TestSystemUI` eingebunden, welche allesamt die globale Komponente TreeEditor.mxml erweitern.

Der in Abbildung 57, Zeile 326 eingebundene `itemRenderer` `mx.controls.treeClasses.MyTreeItemrenderer` ist der von mir erweiterte Standard-ItemRenderer aus dem Flex-SDK, der es ermöglicht, dass ein *Hand-Cursor* über einem Baum-Element angezeigt werden kann.

```
85      /**
86       *  Die Funktion StartEditing wird aufgerufen, sobald der Benutzer ein Tree-Element
87       *  auswählt. Dies wird benötigt, um einen eigenen ItemEditor verwenden zu können.
88       *  @param event:Event
89       */
90      public function startEditing(event:Event):void {
91          event.preventDefault(); // prevent use of default item editor
92          currNode=XML(myTree.selectedItem);
93          /*initiate item editing in right box treeEditor)*/
94          treeEditor.editItem(currNode,true);
95          setInfoText(null);
96          if(currNode){
97              setBreadCrumbs(currNode);
98          }
99      }
```

Abbildung 58: Event-Handler Funktion `startEditing()` (`XMLExplorer.mxml`)

6.8 Paket „`model`"

Im „`model`"-Paket sind unter anderem die nutzerspezifischen Klassen `Wizard.as`, `TestUser.as`, `Admin.as`, `Designer.as` und `SuperUser.as` abgelegt. Alle nutzerspezifischen Klassen erweitern die Klasse `User.as`, welche wiederum die Klasse `mx.events.`

112 http://blog.flexexamples.com/2007/10/29/defining-a-custom-label-function-on-a-flex-tree-control/

`EventDispatcher` erweitert und das Interface `IUser.as` implementiert.

6.8.1 Definition und Verwendung eigener Events

Die Klasse `EventDispatcher` muss in diesem Fall erweitert werden, da die jeweiligen nutzerspezifischen Views über alle Änderungen an Model-Instanzen mit selbstdefinierten Events informiert werden müssen.

```
18    public class User extends EventDispatcher implements IUser
19    {
20        NetConnection.defaultObjectEncoding = ObjectEncoding.AMF0;
21
22        public var nc:NetConnection = null;
23        public var client:Object = new Object();
24        public var shared_object:SharedObject;
25        public var connectedUsers:Array;
26        public var connectedWizards:Array;
27
28        public var myView:Object;
29
30        [Bindable]
31        public var test:Test;
32        public var contr:Controller;
33
34        [Bindable]
35        public var peerAvailable:Boolean;
36
37        [Event(name="onSync",type="flash.events.Event")]
38        [Event(name="nextState",type="flash.events.Event")]
39        [Event(name="onDisconnect",type="flash.events.Event")]
40        [Event(name="testStarted",type="flash.events.Event")]
41        [Event(name="testAborted",type="flash.events.Event")]
42        [Event(name="testFinished",type="flash.events.Event")]
43        [Event(name="peerAvailable",type="flash.events.Event")]
44        [Event(name="peersDisconnected",type="flash.events.Event")]
```

Abbildung 59: `Event`-Definitionen in der Klasse `User.as`

Ist ein entsprechender Event in einer anderen Klasse oder Komponente registriert, so kann bei Auslösung des Events eine Event-Handling-Funktion in dieser Komponente ausgeführt werden.

```
36  /* set current localizations (state) AFTER creation!*/
37  override protected function complete():void{
38      this.isSU=this.contr.app.currentView.name=="view_05";
39      this.testUser=isSU?TestUser(SuperUser(this.contr.currentUser).testuser):TestUser(this.contr.
40      this.testUser.addEventListener("peerAvailable",setWizardAvailable);
41      this.testUser.addEventListener("peersDisconnected",setWizardsDisconnected);
42      this.testUser.addEventListener("callEstablished",onCallEstablished);    // call established
43      this.testUser.addEventListener("callAborted",onCallAborted);    // call aborted
44      this.testUser.addEventListener("testStarted",onStartTest);  // test started
45      this.testUser.addEventListener("testFinished",onFinishTest);    // test started
46      this.testUser.addEventListener("testAborted",onAbortTest);  // wizard disconnected
47      this.keypad.addEventListener("DTMFCommand",onDTMF);
48      this.waitForWiz.statusText="bitte warten";
49      this.waitForWiz.enable();
```

Abbildung 60: Registrieren von Events in der Komponente `UserView.mxml`

Sobald ein Event-Typ registriert ist, wie hier der Event `callEstablished` in Zeile 42,

kann darauf reagiert werden, wenn der entsprechende Event in der dazugehörigen Klasse (`model.TestUser`) ausgelöst wird.

```
89        /**
90         * Diese auch serverseitig aufrufbare responder-Funktion initialisiert mit
91         * den Serverseitigen IDs von Testperson und Wizard einen neuen Wizar-of-Oz Test.
92         * Daraufhin wird ein "callEstablished"-Event ausgelöst.
93         *
94         * @param resultObj:Object serverseitiges Objekt mit
95         */
96        override protected function onCallEstablished(resultObj:Object):void{
97            try{
98                var clientID:String=resultObj["userID"];
99                var peerID:String=resultObj["wizardID"];
100           }catch(e:Error){
101               trace("Call established, could not get peer-IDs!");
102               return;
103           }
104           this.test.initTest(clientID,peerID);
105           dispatchEvent(new Event("callEstablished"));
106       }
```

Abbildung 61: Auslösen des Events `callEstablished` in der Klasse `TestUser.as` (Zeile 42)

Die Funktion `onCallEstablished()` ist an das `client`-Objekt der `NetConnection` (`nc`, `User.as` Zeile 22) gebunden, welches die Funktion von der verbundenen Server-Instanz[113] (Wowza Media Server) aus aufrufbar macht.

```
88        /**
89         *  In der geschützten Funktion initNC() werden diverse Funktionen dieser Klasse
90         *  an ein client-Objekt gebunden. Dadurch ist es möglich, diese Funktionen vom
91         *  über das NetConnection-Objekt angebundenen Server aus aufzrufen.
92         */
93        protected function initNC():void{
94            /* add event-handling functions to client Object */
95            client.onDTMF = onDTMF;
96            client.onTestFinished = onTestFinished;
97            client.onTestStarted = onTestStarted;
98            client.onTestAborted = onTestAborted;
99            client.onCallIncoming = onCallIncoming;
100           client.onCallEstablished = onCallEstablished;
101           client.onCallAborted = onCallAborted;
102           client.onMetaData = onMetaData;
103           client.onBWDone = onBWDone;
104           nc.client=this.client;  // add client-Object to NetConnection (nc)
105       }
```

Abbildung 62: Bindung der Event-Handling Funktionen an das `client`-Objekt (`User.as`)

6.8.2 Herstellung einer RTMP-Verbindung mit dem Media-Server

Die RTMP[114]-Verbindung zum Media-Server wird aufgebaut, sobald sich ein Benutzer einloggt und somit eine Instanz der entsprechenden `model`-Klasse erstellt wird. Sie wird so lange aufrecht erhalten, bis sich der Nutzer wieder ausloggt.

113 vgl. http://blogs.warwick.ac.uk/stevencarpenter/entry/flex_and_wowza/

114 „Realtime Messaging Protocol“: proprietäres Netzwerkprotokoll (vergleichbar mit HTTP), welches von Adobe speziell für die Verbindung von Flash-clients und (Flash-) Media-Servern entwickelt wurde.

```as
57        /**
58         *  Konstruktor der Klasse User. Sobald auf Grund eines erfolgreichen Login-Vorgangs
59         *  ein Benutzer erstellt wird, wird versucht eine  über eine RTMP-Verbindung mit
60         *  dem Media-Server aufzubauen. Die Adresse des Servers ist in der Konstante HOST
61         *  in der Datei global.Constants abgelegt.
62         *
63         *  @param userNode:XML     das zum Benutzer gehörige Element in der Datei users.xml
64         *  @param contr:Controlle  die Controller-Instanz
65         *  @param fullXMLSpec:XML  bereits existierende Nutzerdaten (nur für SuperUser relevant)
66         */
67        public function User(userNode:XML,contr:Controller,fullXMLSpec:XML)
68        {
69            this._userNode=userNode;
70            this.contr=contr;
71            this.setView();
72            this.initUser(userNode,fullXMLSpec);
73            // create a connection to the wowza media server
74            if (nc == null) {
75                this.nc = new NetConnection();
76                nc.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
77                nc.objectEncoding = ObjectEncoding.AMF0;
78                nc.connect("rtmp://"+this.contr.app.HOST+"/woz");
79            }
80        }
```

Abbildung 63: Herstellung einer Verbindung zum Wowza-Media-Server (`User.as`)

6.9 Schlussbemerkung zur Implementation der Flex-Applikation

Es wurde versucht, einen Überblick über die clientseitigen Pakete und Klassen des entwickelten Prototypen zu geben und auf, meiner Meinung nach, besonders wichtige und interessante Stellen in den einzelnen Klassen und Komponenten einzugehen. Eine noch detailliertere Betrachtung der Implementation hätte wohl den Rahmen dieser Arbeit gesprengt, deshalb bitte ich darum, für die Klärung möglicher Fragen in der Dokumentation nachzusehen.

Um einen ebenfalls kurzen und exemplarischen Überblick in die serverseitige Implementation zu gewähren, folgt im nächsten Kapitel eine Einführung über die für das Live-Streaming der Audioinhalte notwendigen serverseitigen Funktionen des Wowza Media-Servers.

7. Implementation für den Wowza Media Server

Für sämtliche backendseitigen Funktionalitäten wurde, wie bereits erwähnt, der Java-basierte Wowza Media Server eingesetzt. Hierbei habe ich mich auf die für die konzipierte RIA zwingend benötigten Funktionen, wie das Live-Streaming und gleichzeitige Aufnehmen von Audioinhalten, das Lesen und Schreiben von XML-Daten sowie die Synchronisation der verbundenen Flex-Clients konzentriert. Da das Hauptaugenmerk dieser Arbeit der Implementation der frontendseitigen Funktionalitäten in Adobe Flex gelten sollte, möchte ich im Folgenden nur in aller Kürze auf zentrale Funktionen in den serverseitigen Modulen eingehen.

7.1 Entwicklungsumgebung Wowza IDE

Für die Entwicklung der serverseitigen Module wurde die von Wowza Media Systems für Entwickler kostenlos bereitgestellte und ebenfalls auf der Eclipse-Plattform basierende *Wowza IDE* der Version 1.0 eingesetzt. Sie bietet sowohl umfassende Debugging-Funktionalitäten für die Java Klassen, als auch die aus den bereits vorgestellten IDEs bekannten Standard-Funktionen, wie beispielsweise automatische Builds im Hintergrund, die Anzeige von Compiler -Fehlern, eine Konsolenansicht und automatische Code-Completion.

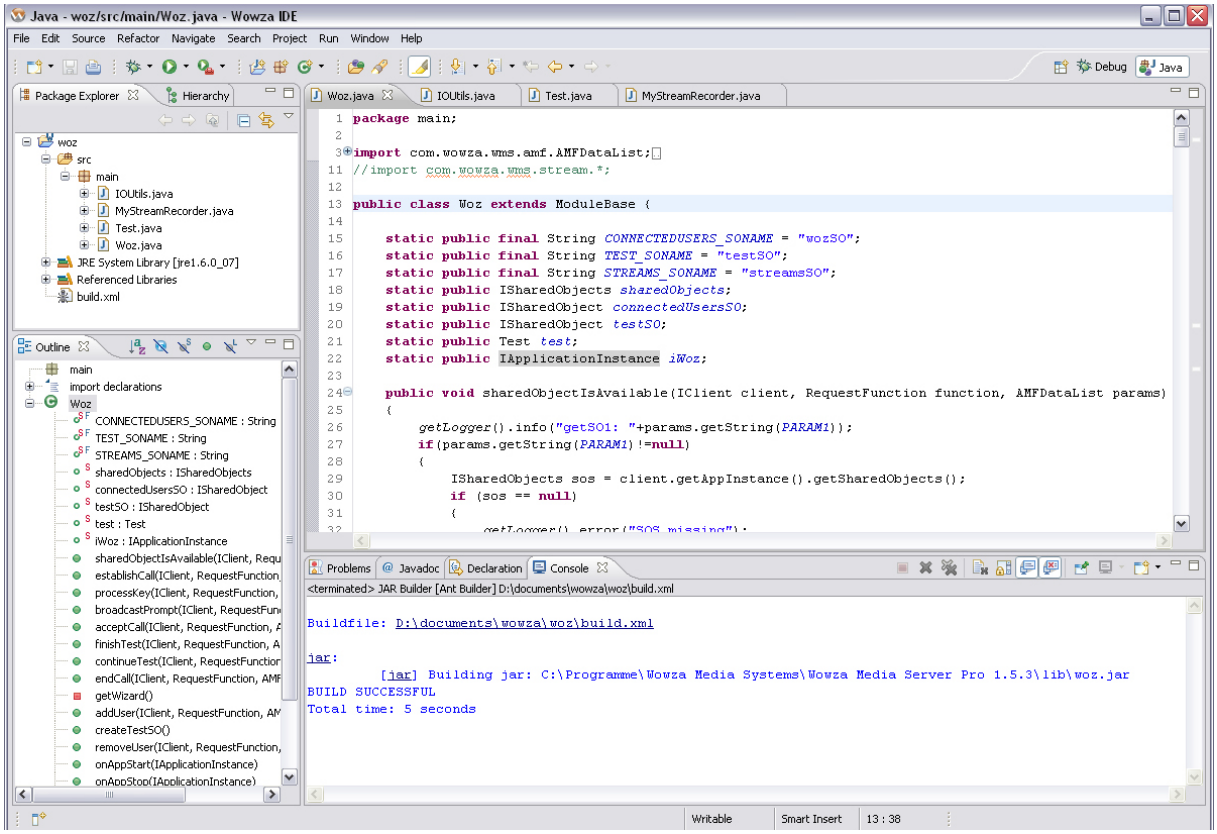


Abbildung 64: Wowza IDE

Der entscheidende Vorteil der Wowza IDE im Vergleich zur aktuellen Eclipse IDE, welche

für die Erstellung der XML-Schemata eingesetzt wurde, ist die Out-Of-The-Box Einbindung des Wowza Media Servers über einen entsprechenden Wizard für neue „Wowza Media Server Pro Java"-Projekte, sowie das automatische *Deployment* der implementierten Module (*.jar-Erstellung*).

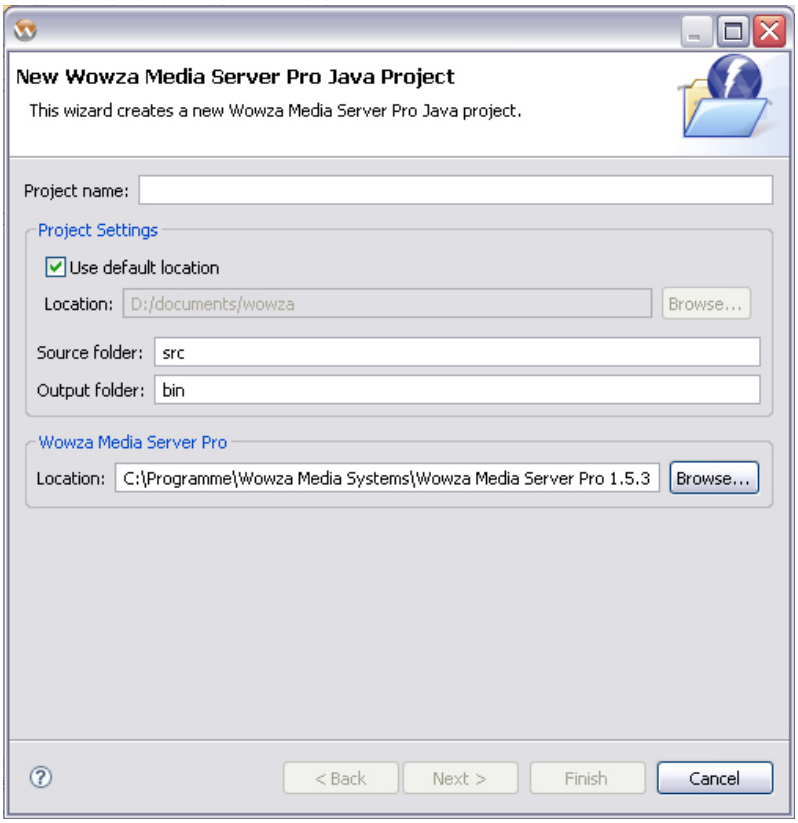


Abbildung 65: Wizard für die Erstellung neuer WMS-Java-Projekte (Wowza IDE)

Sobald ein Projekt angelegt worden ist, muss es jedoch noch händisch im Dateisystem erstellt werden[115]. In der Datei `Application.xml` im für das Projekt vorgesehenen Verzeichnis (hier: `...Wowza Media Server Pro 1.5.3/conf/woz`) können dann sämtliche selbst erstellten Module zusätzlich zu den Standard-Modulen verlinkt werden. Neben den Standard-Modulen wird von mir im Projekt „`woz`" zusätzlich das serverseitige Modul `com.wowza.wms.module.ModuleFLVPlayback` sowie die von mir implementierten bzw. erweiterten Module `main.Woz`, `main.IOUtils` und `main.MyStreamRecorder` eingebunden. Darüber hinaus wird in dieser Konfigurationsdatei das Verzeichnis für die Streams (`${com.wowza.wms.AppHome}/user/woz/streams`), welches in diesem Fall auch die Aufnahmen der Tests beinhaltet, festgelgt.

115 http://www.wowzamedia.com./quickstart.html#coding

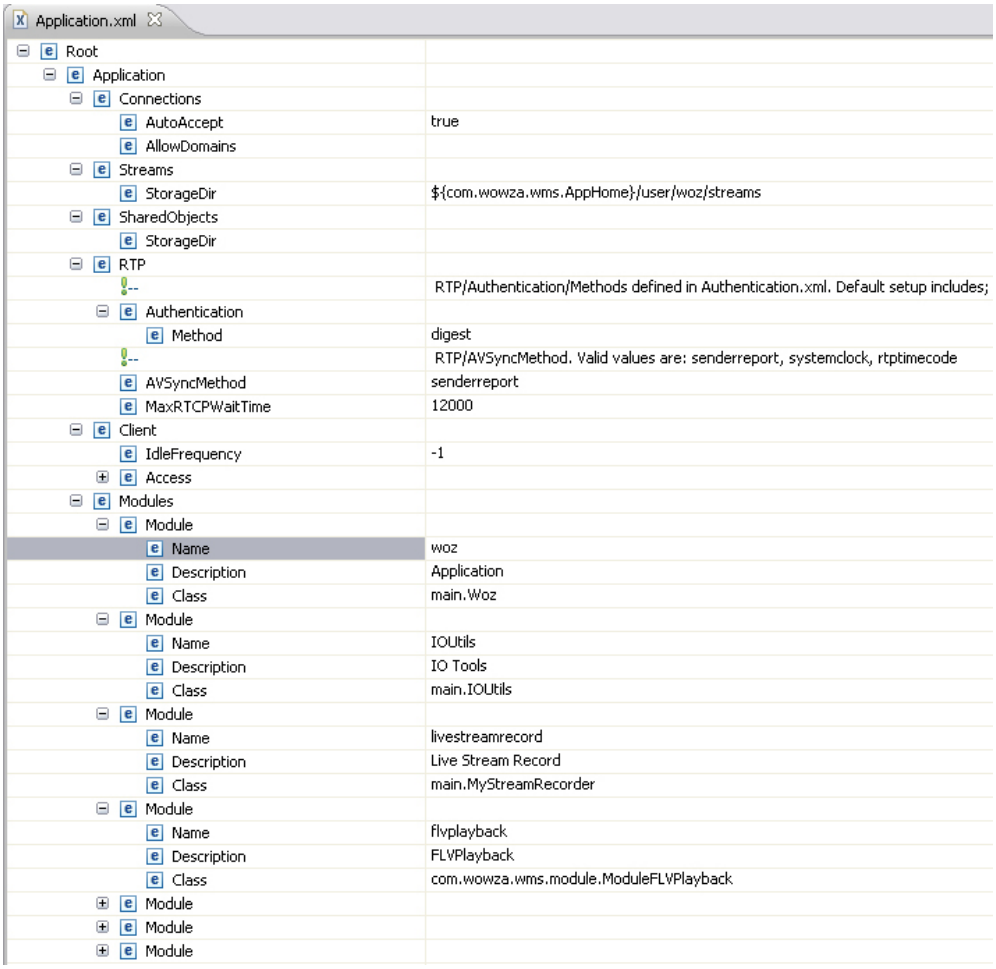


Abbildung 66: Die Datei conf/woz/Application.xml (Eclipse WST)

Üblicherweise kann im Element „`Streams`" noch ein Element „`StreamType`" festgelegt werden. Dies ist jedoch im Falle der von mir erstellten Applikation nicht notwendig, da der Typ der jeweils verwendeten Streams dynamisch angepasst werden muss[116]. Dies geschieht über die serverseitige Funktion `setStreamType`, welche jeweils unmittelbar vor der clientseitigen Verbindung mit einem Stream aufgerufen werden muss. Für das Streamen einer Datei (aufgenommen *.flv*-Datei) muss dementsprechend der Stream-Typ `file` verwendet werden, während für das ebenfalls verwendete Live-Streaming der Stream-Typ `live-lowlatency` verwendet werden sollte.[117]

116 vgl. http://www.wowzamedia.com/forums/showthread.php?t=990&highlight=setStreamType
117 vgl. http://www.wowzamedia.com/quickstart.html

```
124         /* initialize streams and call server-side function setStreamType() */
125         private function initNetStreams(streamType:String):void{
126             nc.call("setStreamType",null,streamType);
127             this.incomingNetStream = new NetStream(nc);
128             incomingNetStream.bufferTime=BUFFERTIME;     // set Buffer Time
129             prepNetStream(incomingNetStream);    // prepare incoming NetStream
130         }
131
132         /**
133          * Die öffentliche Funktion startTalk() startet einen Live-Streaming
134          * Vorgang und erstellt eine Audioverbindung mit dem verbundenen Client.
135          * Hierfür muss der Streamtyp "live-lowlatency" verwendet werden.
136          *
137          * @param rec:Boolean   falls ja, starte Aufnahme
138          * @param play:Boolean=true     falls ja, spiele Stream des verbundenen
139          *                              Clients ab. (Standard:"true")
140          */
141         public function startTalk(rec:Boolean,play:Boolean=true):void {
142             initNetStreams("live-lowlatency");
143             if(!rec){
144                 defaultMute?muteMic():activateMic();
145             }else{
146                 autoRecord=true;
147                 activateMic();
148             }
149             play?incomingNetStream.play(inStreamName):false;
150             trace("start talk");
151         }
```

Abbildung 67: Herstellung einer `NetStream`-Verbindung in der AS3-Klasse `controller.MicrophoneIO` (Flex Builder)

Serverseitige Funktionen lassen sich, wie hier in der AS3-Klasse `MicrophoneIO.as` (Zeile 126) die Funktion `setStreamType()`, über die `NetConnection`-Methode `call()` aufrufen. Dabei kann neben einer „`Responder`“-Funktion (hier nicht spezifiziert, `null`) ein Parameter eines einfachen (z.B. `Number`, `String`) oder komplexen Datentyps (z.B. `Array`, `Object`) an die angesprochene Funktion eines serverseitigen Moduls übergeben werden.

Sobald nach erfolgtem Login eine Verbindung mit dem Media-Server hergestellt ist, wird der verbundene Client entsprechend seines Nutzertyps vom Server in einem *Shared Object* registriert[118]. Ein Shared Object bietet den Vorteil, dass Änderungen an einem solchen serverseitigen Objekt unmittelbar von den verbundenen Clients erfasst werden können (`onStatus`-Event), ohne dass mögliche Änderungen explizit in einem vorgegeben Zeitintervall abgefragt werden müssen. Dies sorgt unter anderem dafür, dass ein Wizard unmittelbar über verbundene oder abgemeldete Testteilnehmer informiert wird sowie dass ein Test abgebrochen werden kann, sobald sich die Gegenseite vom System abmeldet.

118 http://livedocs.adobe.com/flex/2/docs/00001020.html

7.2 Die Klasse Woz.java

Das serverseitige Modul `main.Woz` ist die Schaltstelle für die Kommunikation zwischen den Flex-Clients und dem Media-Server. Die meisten Anfragen über die `call()`-Methode des clientseitigen `NetConnection`-Objekts werden von dieser Java-Klasse verarbeitet, wobei die Hauptaufgabe des Moduls `main.Woz` darin besteht, die verbundenen Clients miteinander zu synchronisieren und die Tests zu verwalten. Auch während eines Tests werden sämtliche Anfragen über dieses Modul an die Java-Klasse `main.Test` weitergeleitet.

```
46    /**
47     * Diese vom verbundenen Flex-Client (einem Testteilnehmer) über das TestUser-
48     * Interface aus aufgerufene Methode verbindet den Aufrufenden Testteilnehmer
49     * mit dem nächsten verfügbaren Wizard-Client und startet einen Testdurchlauf.
50     *
51     * @param client    Referenz zum aufrufenden Flex-Client
52     * @param function  Responderfunktion des Flex-Clients, die automatisch nach Beendigung
53     *                  der Funktion aufgerufen wird
54     * @param params    vom Client übergebenes Parameter-Objekt
55     */
56    public void establishCall(IClient client, RequestFunction function,
57            AMFDataList params) {
58        String res="";
59        WMSLogger log = getLogger();    // get Logger Reference
60        AMFDataObj request = (AMFDataObj)getParam(params, PARAM1);  // get request parameter
61        IClient wizard = getWizard();   // get next available Wizard-Reference
62        if(wizard!=null){   // start Test iff Wizard is available
63            /* start Test iff no Test is currently running */
64            if(!testSO.containsSlot("test")){
65                res=String.valueOf(wizard.getClientId());
66                AMFDataArray wizArray=(AMFDataArray)connectedUsersSO.getProperty("wizards");
67                AMFDataObj metaData=new AMFDataObj();
68                String testID = request.getString("testID");    // get client-side TestID
69                String userID = request.getString("userID");    // get client-side userID (TestUser's ID)
70                String wizardID = wizArray.get(0).toString();   // get client-side Wizard ID
71                /* add IDs to metaData-Object*/
72                metaData.put("userID", userID);
73                metaData.put("wizardID", userID);
74                log.info("establishCall: " + userID +" with "+wizardID);
75                test=new Test(client,wizard,metaData,testID,userID);    // start new Test!
76                testSO.lock();
77                try
78                {
79                    testSO.acquire();
80                    testSO.setProperty("test",request);     // register the test in a shared Object (testSO)
81                    getLogger().info("establishCall - add property: "+testSO.getProperty("test").toString());
82                }
83                catch (Exception e)
84                {}
85                finally
86                {
87                    testSO.unlock();
88                }
89            }else{
90                res="Error:Cannot run more than one test at a time!";
91                log.info(res);
92            }
93        }else{
94            res="Error:Could not establish call!";
95            log.info(res);
96        }
97        sendResult(client, params, res);
98    }
```

Abbildung 68: Die öffentliche Funktion establishCall() im Modul main.Woz (Wowza IDE)

Ein Test-Objekt wird über die Funktion `establishCall()` genau dann erstellt (hier: Zeile 75), wenn zur Zeit kein anderer Test im Shared Object `testSO` registriert ist.

Auf diese Weise wird sichergestellt, dass immer nur ein Test parallel auf einem Server ab-

laufen kann (frei nach dem *Singleton-Design-Pattern*[119]). Zukünftig könnte dies auch auf eine begrenzte Anzahl von Tests erweitert werden, wobei zum einen im aktuellen Anwendungsszenario ein simultan durchgeführter Test völlig ausreicht und zum anderen darauf geachtet werden sollte, dass durch eine zu starke Auslastung des Servers, die für eine reibungslose Kommunikation essentielle kurze Latenz nicht mehr gegeben sein könnte.

7.3 Die Klasse Test.java

Die für die Testdurchführung verwendete Klasse `main.Test` erweitert zwar ebenso wie die soeben vorgestellte Haupt-Klasse `main.Woz`, die Klasse `com.wowza.wms.module.ModuleBase`, da sie jedoch nicht in der Konfigurationdatei `Application.xml` als Modul eingebunden ist, müssen sämtliche Anfragen über die Klasse `main.Woz` erfolgen. Dies ist deswegen sinnvoll, da die Kontrolle über diese Klasse ausschließlich dem Hauptmodul dieser Applikation `main.Woz` obliegt und verhindert werden soll, dass das Modul `main.Test` direkt von einem beliebigen verbundenen Client addressiert werden kann.

```
34    /**
35     * Die öffentliche Methode broadcastPrompt() wird verwendet um die Systemprompts
36     * an beide in diesem Test miteinander verbundenen Clients zu senden. Falls der
37     * übergebene request-Parameter mehrere Pfade zu auf dem Server hinterlegten
38     * Audio (bzw. .flv-Dateien) beinhaltet, so werden diese Pfade hier konkateniert,
39     * und ein Array mit den vollen Pfaden der entsprechenden Dateien an beide Clients
40     * übergeben.
41     *
42     * @param client        die client-Referenz zum verbundenen Wizard
43     * @param function      muss nicht spezifiziert werden
44     * @param params        die vom Wizard (aktiv) ausgewählten Prompts
45     */
46    public void broadcastPrompt(IClient client, RequestFunction function,
47            AMFDataList params){
48        sendResult(wizard,params,getTime());
49        WMSLogger log = getLogger();
50        AMFDataMixedArray request = getParamMixedArray(params, PARAM1);
51        AMFDataArray playlist = new AMFDataArray();
52        for(int i=0;i<request.size();i++){
53            String promptFile=request.getString(i);
54            String fullPath = PROMPTPATH+testID+"/"+promptFile;
55            log.info("broadcasting "+promptFile);
56            playlist.add(STREAMTYPE+":"+fullPath);  // add full file-path to Playlist
57        }
58        /* send broadcast-playlist to connected clients */
59        user.call("onBroadcastPrompt",null,playlist);
60        wizard.call("onBroadcastPrompt",null,playlist);
61    }
```

Abbildung 69: Senden einer Playliste an die Testteilnehmer im Modul main.Test (Wowza IDE)

119 Gamma et al. (1995), S. 128

7.4 Das Modul main.IOUtils

Für sämtliche Schreib-Lese Vorgänge auf dem Dateisystem des Servers, genauer für die im Prototypen verwendeten XML-Daten, wird die Methode `getOrSetXML()` verwendet. Je nach den im übermitteltem Parameter-Objekt angegebenen Flags wird entweder ein Lese- (`sGetOrSet="get"`) oder ein Schreibvorgang (`sGetOrSet="set"`) ausgeführt.

Zur Überprüfung der Korrektheit der als *String* übermittelten XML-Daten werden diese Daten bei einem Schreibvorgang zunächst von einem *SAX-Parser*[120] geparsed.

Tritt beim Schreiben oder Lesen ein Fehler auf, wird ein entsprechender Fehler-XML-String zurückgegeben. Andernfalls erhält der verbundene Client den Inhalt der im Parameter `fileName` spezifizierten XML-Datei als String zurück.

120 http://www.saxproject.org/

8. Schlusswort und Ausblick

Ziel der Arbeit war es, einen funktionstüchtigen Flex-basierten, vertikalen Prototypen einer Rich Internet Application für die Durchführung eines Wizard-of-Oz-Tests, auf Grundlage eines Dialogprototypen für ein Voice-User-Interface, zu konzipieren und zu implementieren. Dies ist mir in vollem Umfang gelungen, wobei sogar darüber hinaus eine Schnittstelle zum Design XML-Basierter Dialogprototypen sowie zur Verwaltung der Nutzerdaten erstellt wurde.

Das System lässt sich über die auf der mit herausgegebenen DVD enthaltenen Programmpakete lokal testen, wobei es darüber hinaus angestrebt wird, die Applikation zu Testzwecken gänzlich über das Internet verfügbar zu machen. Die vorgestellten Code-Ausschnitte konnten nur exemplarisch behandelt werden und es wird daher empfohlen für genauere Informationen im ebenfalls auf der DVD enthaltenen Quelltext oder in der Dokumentation nachzusehen.

Es ließ sich leider auf Grund des begrenzten dreimonatigen Zeitfensters für diese Arbeit nicht verhindern, dass gewisse für einen Produktiv-Einsatz des Systems sicher wichtige Detailfragen, wie die Behandlung von Fehlern während eines Dialogs oder eine Nachbefragung, noch nicht abschließend geklärt und implementiert werden konnten. Da es sich jedoch beim für diese Arbeit implementierten System ausdrücklich um ein *Proof-Of-Concept*, also eine Durchführbarkeitsstudie handeln sollte, ist dies zu diesem Zeitpunkt zu vernachlässigen.

Die bekannten Schwachstellen der implementierten RIA wurden ausdrücklich erwähnt und es wird empfohlen diesen Prototypen als Ausgangsbasis für die zukünftige Implementation eines internetbasierten Produktivsystems zum Wizard-of-Oz-Testing von (frühen) Dialogprototypen zu sehen und zu verwenden.

Ich denke, dass die Vorzüge für ein internetbasiertes System im Bereich des Usability Testings auch allgemein deutlich geworden sind und es wird angeregt, dass die beschriebene Herangehensweise über Flex-basierte Clients und einen Media-Server zukünftig auch für andere, nicht zwingend auf Voice User Interfaces begrenzte Konzepte zum webbasierten Usability-Testing Anwendung findet. Es besteht meinerseits großes Interesse daran, in diese Richtung weiter zu forschen und auf diesem Gebiet neuartige Konzepte mit einem interdisziplinär ausgelegten Team zu kreieren und für den Produktiveinsatz zu realisieren.

9. Anhang

A. Literaturverzeichnis

Ansorge, P. & Haupt, U. (1997). Ergonomie-Reviews und Usability-Testing als Beratungs- und Qualifizierungsinstrumente, in: R. Liskowsky, B. M. Velichkowsky & W. Wünschmann (Hrsg.) Software-Ergonomie ,97, Berichte des German Chapter of the ACM 49. S. 55 - 69, Stuttgart : B. G. Teubner

Boyce, S. J. (2000). Natural Spoken Dialogue Systems for Telephony Applications. COMMUNICATIONS-ACM. 43, 29-35.

Cohen, M. H., Giangola, J. P., & Balogh, J. (2004). Voice user interface design. Boston: Addison-Wesley.

Cooper, A., & Reimann, R. (2003). About face 2.0: the essentials of interaction design. New York: Wiley.

Gamma, E. (1995). Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series. Reading, Mass: Addison-Wesley.

Harris, R. A. (2004). Voice interface design: crafting the new conversational speech system. San Francisco, Calif: Morgan Kaufmann.

International Organization for Standardization. (1999). Human-centred design processes for interactive systems. International standard, ISO 13407. [Geneva]: International Organization for Standardization.

Jurafsky, D., & Martin, J. H. (2000). Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall series in artificial intelligence. Upper Saddle River, N.J.: Prentice Hall.

Kamm, C. (2000), Design Issues for Interfaces using Voice Input. In Tzafestas, S. G. Handbook of Human-Computer Interaction, Martin G. Helander, Thomas K. Landauer and Prasad V. Prabhu (eds.). Journal of Intelligent and Robotic Systems. 28 (3), 291-292.

Nielsen, J., Usability Engineering (1993), Academic Press/AP Professional, Cambridge, MA

Virzi, Robert A, Sokolov, Jeff, and Karis, Demetrios (1995), Usability Problem Identification Using Both Low- and High-Fidelity Prototypes

Nielsen, J. (1990), Paper versus Computer Implementations as Mockup Scenarios for Heuristic Evaluation‘, Human-Computer Interaction-Interact `90, D. Diaper et al. (ed.) Elsevier

Science Publishers B.V. North Holland

Noble, J., Taivalsaari, A., & Moore, I. (1999). Prototype-based programming: concepts, languages, and applications. Singapore: Springer.

Pawar, S. A. (2004). A common software development framework for coordinating usability engineering and software engineering activities. Blacksburg, Va: University Libraries, Virginia Polytechnic Institute and State University. http://scholar.lib.vt.edu/theses/available/etd-05202004-124527/.

Peissner, M., Biesterfeldt, J., Heidmann, F. (2004), Akzeptanz und Usability von Sprachapplikationen in Deutschland, Hrsg.: Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO, Stuttgart

Rifkin, J. (2007). Access: das Verschwinden des Eigentums ; warum wir weniger besitzen und mehr ausgeben werden. Frankfurt am Main: Campus-Verl.

Sanders, W. B., & Cumaranatunge, C. (2007). ActionScript 3.0 design patterns. Adobe developer library. Beijing: O‘Reilly.

Skantze, G. (2005). Exploring human error recovery strategies: Implications for spoken dialogue systems. SPEECH COMMUNICATION. 45 (3), 325-341.

Smith, R. W., & Hipp, D. R. (1994), Spoken natural language dialog systems: a practical approach. New York: Oxford University Press.

B. Abbildungsverzeichnis

C. Verzeichnis verlinkter Webseiten und Dokumente

Aufgrund der Aktualität der in dieser Bachelorthesis eingesetzten Techniken sind zum Beleg viele Internetseiten und verlinkte pdf-Dokumente aufgeführt. Da sich die Erreichbarkeit und der Inhalt dieser verlinkten Seiten schnell ändern können, kann nicht sichergestellt werden, dass die unter den Links zu findenden Informationen noch aktuell sind. Zum Zeitpunkt der Fertigstellung dieser Bachelorthesis am 14.10.2008 waren alle Quellen unter den in den Fußnoten aufgeführten Links erreichbar.

Die verlinkten Webseiten und Dokumente sowie die Literaturangaben sind im Format *Zotero RDF* inclusive Meta-Informationen als Snapshot auf der beigelegten DVD enthalten.

D. Inhaltsverzeichnis der Daten-DVD

Verzeichnis *thesis*	**Bachelorthesis**
thesis/pdf/	pdf-Version incl. Deckblatt
thesis/indesign/	Adobe-Indesign-Version incl. Deckblatt
Verzeichnis *docs*	**Dokumentation**
docs/asdoc/	API-Dokumentation des Flex-Prototypen
docs/zotero/	Quellen, Links und Webseiten zu dieser Arbeit
Verzeichnis *projekte*	**Eclipse-/Flex-Projekte (Quellcode)**
projekte/flex/woz/	Projekt zum Import im Adobe Flex Builder
projekte/wowza/woz/	Projekt zum Import in der Wowza IDE
projekte/xml/	Projekt zum Ablegen auf einem (lokalen) Webserver
projekte/uml/	Projekt für die Ansicht der erstellten UML-Diagramme
Verzeichnis *software*	**Benötigte Software**
software/iwoz/bin-release/	letzte Version der kompilierten Flex-Applikation
software/wowza/	gepackte Version 1.5.3 des Wowza Media Servers
software/adobe/flex/	Testversion des Adobe Flex Builders 3
software/adobe/air/	AIR-Runtime-Umgebung
software/adobe/flash-player/	Adobe Flash-Player
software/mozilla/firefox/	Version 3 des Firefox-Browsers
software/java/	Java-SDK und Rundime-Umgebung
software/eclipse/	Eclipse Ganymede incl. Web-Standard-Tools
software/eclipseUML/	EclipseUML zum Bearbeiten der UML-Diagramme
software/xampp/	Lokaler Apache-Webserver

Für die mitgelieferten Software-Pakete wird Microsoft Windows XP oder Vista benötigt.

Erklärung

Hiermit erkläre ich, dass ich diese Bachelorthesis selbstständig angefertigt und keine anderen Quellen und Hilfsmittel außer den in der Arbeit angegebenen benutzt habe.

Stuttgart, den 14.10.2008 _____

Manuel Fittko