

2 Informations-Technologie

Eine Online-Seminar-Verwaltung als Sprachapplikation

Sprachapplikationen

Eine Sprachapplikation ist eine Anwendung, welche von einem menschlichen Benutzer mit Hilfe **natürlicher Sprache** gesteuert und bedient wird. Sie unterscheidet sich von gängigen Anwendungen vor allem dadurch, dass dem Benutzer als Schnittstelle zum Computer-System ein Telefon oder Headset an Stelle von Maus, Tastatur und Bildschirm zur Verfügung steht.

Durch die **weite Verbreitung von Telefonen** und insbesondere Mobiltelefonen eröffnen Sprachapplikationen die Möglichkeit von überall und zu jeder Zeit auf Geschäftsanwendungen zuzugreifen. Auch technisch unbedarfte Personen können mit diesem einfach zu benutzenden Medium jederzeit Informationen abrufen, Buchungen tätigen oder andere Dienste wahrnehmen, die bisher einen Internetzugang und entsprechende Skills vorausgesetzt haben.

Eine Sprachapplikation kann, da sie über eine Sprachschnittstelle (**Voice-User-Interface, VUI**) bedient wird, im Gegensatz zu einer herkömmlichen Applikation mit einer grafischen Bedienoberfläche (Graphical-User-Interface, GUI), auch in Situationen genutzt werden, in welchen die Aufmerksamkeit des Benutzers nicht ausschließlich auf die Bedienung der Applikation gerichtet sein darf, wie etwa beim Steuern eines Fahrzeugs.

Ein ideales, neues **Anwendungsszenario** ist der Ausbau einer bestehenden Web-Applikation zu einer Sprachapplikation. Dies ist besonders einfach zu bewerkstelligen, wenn es sich dabei um ein mehrschichtiges System, wie zum Beispiel eine J2EE-typische 3-Tier-Architektur, handelt, kann aber andererseits den Kundenkreis für die bestehende Applikation, wie oben erläutert, erheblich erweitern. In Abb.1 sieht man die Application-Tier (blau) und die Client-Tier (grün, gelb), letztere wurde um sprachspezifische Anteile (rot) erweitert (Die Data-Tier wird hier nicht gezeigt).

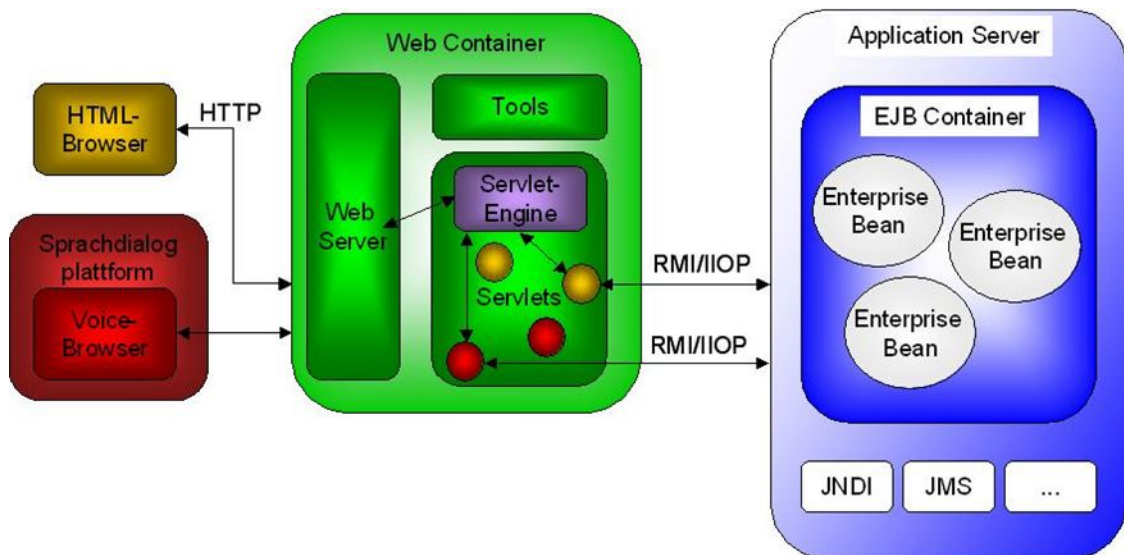


Abb.1: J2EE-Architektur zur Sprachapplikation ausgebaut

VoiceXML

Bei einer Sprachapplikation erfolgt die Informationsübertragung zwischen dem menschlichen Benutzer und dem Computersystem durch einen natürlichsprachlichen **Dialog**. Dieser Dialog besteht aus einer Folge von Dialogschritten, welche durch die Spracheingaben des Benutzers und die Sprachausgaben des Systems gebildet wird.

Seit einigen Jahren existiert mit VoiceXML 2.0 eine vom W3C standardisierte **Programmiersprache**, in der sich solche Dialoge einfach und schnell plattformunabhängig realisieren lassen. VoiceXML ist damit für Sprachapplikationen so essentiell wie HTML für Web-Applikationen. Weitere mit VoiceXML assoziierte Sprachen werden für die Steuerung von Telefonie (CCXML), Sprachausgabe (SSML) und Spracherkennung (grXML) eingesetzt.

OSV: Das Voice-Projekt von K&A

Zielsetzung

Um zu demonstrieren, wie einfach eine J2EE basierte Web-Anwendung mit einem zusätzlichen VUI ausgerüstet werden kann, hat die Firma Kölsch & Altmann ihre interne [Online-Seminar-Verwaltung \(OSV\)](#) um eine solche Schnittstelle erweitert. Das neue Interface sollte zwar nicht die volle Funktionalität des bestehenden Web-Interface bieten, aber es doch einem Seminarkunden ermöglichen, sich umfassend über unser Seminarangebot zu informieren und Reservierungen zu tätigen.

Projektergebnis

Nicht nur das Primärziel, die Funktionalität einer Sprachapplikation herzustellen wurde erreicht. Auch das Sekundärziel, **Best Practices** des Dialogdesigns herauszuarbeiten, Einschränkungen und Fallgruben bei der Anbindung an J2EE-Applikationen kennenzulernen, und sich mit der verwendeten Sprachsoftware und zugehörigen Werkzeugen vertraut zu machen, wurde vollends erfüllt.

System- und Entwicklungsumgebung

Ein Voice-Client erfordert naturgemäß zusätzliche Hard- und Software, deren Auswahl projektabhängig überlegt sein will. Nach einem kurzen Überblick über die **sprachspezifischen Komponenten** wird unsere persönliche Wahl vorgestellt:

Bei einer Web-Applikation fordert ein Webbrowser HTML-Dokumente von einem Application-Server an und interpretiert sie zu einer grafischen Ausgabe. Eingaben des Benutzers erfolgen über Formulare in den Webseiten. Bedient wird der Webbrowser mit Maus und Tastatur.

Analog fordert in einer Sprachapplikation ein sogenannter Voicebrowser VXML-Dokumente von einem (meist dem gleichen) Application-Server an und interpretiert sie zu einer akustischen Ausgabe. Eingaben des Benutzers erfolgen über einen Spracherkenner. Bedient wird der Voicebrowser mit einem Telefon oder einem Headset am PC.

Eine Voice-Plattform hat die Aufgabe, alle Komponenten zur Verfügung zu stellen und zu verbinden, welche für einen Voice-Client benötigt werden. Sie enthält den beschriebenen Voicebrowser als Kernstück, leistet seine Anbindung an das Telefonnetz, und integriert Module für die Spracherkennung (Automated Speech-Recognition, ASR) und Sprachausgabe (Text-To-Speech, TTS). Die beteiligten Module können von verschiedenen Herstellern kommen.

- Die **Voice-Plattform** ist von unserer Partnerfirma, der Clarity AG, gestellt. Diese "Clarity V/3 Dialog Platform" bietet volle [VoiceXML 2.0](#) Unterstützung und erlaubt den flexiblen Einsatz von Spracherkennern (ASR) und Sprachsynthetisierern (TTS) verschiedener Hersteller.
- Als **Spracherkenner (ASR)** kam das Produkt "OpenSpeech Recognizer 2.06" der Firma [Scansoft](#) zum Einsatz.
- Für die **Sprachausgabe (TTS)** wurde anfangs die Text-To-Speech-Engine "RealSpeak 3.5.1" der Firma Scansoft verwendet und später durch "rVoice 4.3" der Firma Rhetorical ersetzt.
- Als **Grammatiksprache** wurde der SRGS Standard in der XML-Notation gewählt. Diese ist zwar weniger eingänglich und schwerer zu lesen als die ABNF-Notation, dafür aber portabel, da sie dem Standard folgend von jeder Voice-Plattform unterstützt wird.
- Als **Application-Server** ist der bereits bei Vorgängerprojekten verwendete "JBoss", nun in der Version 4.0.0, mit integriertem Webserver "Tomcat" im Einsatz.
- Als **Datenbanksysteme** werden wahlweise Oracle oder mySQL verwendet.
- Ein mit dem **Betriebssystem** "Linux Redhat 7.3" ausgerüsteter Server dient als HW-Plattform für die genannten Komponenten.

Als Werkzeuge kamen "Together Central Center" (Modellierung), Eclipse (Implementierung) und das [HP OCMP vXML Developer Toolkit](#) (Dialogmetriken) zum Einsatz. Als Entwicklungsprozess diente der Rational Unified Process (RUP).

Analyse, Design

In der Inception-Phase des RUP wurden die grundlegenden Anforderungen an den Voice-Client definiert. Es sollten zwei Use-Cases betrachtet werden.

- Eine **Seminarberatung**, in der ein Anrufer einen Überblick zu den angebotenen Seminaren sowie Detailinformationen zu einem von ihm ausgewählten Seminar (Beschreibung, Inhalt, Termine) abfragen kann.
- Eine **Reservierung** von Plätzen für ein von einem registrierten Seminarkunden ausgewähltes Seminar sollte entweder ausgehend von einer Seminarberatung, oder auch direkt möglich sein.

Diese beiden Use-Cases, dargestellt in Abb.2, wurden nicht nur im Hinblick auf die reine Benutzbarkeit des Systems gewählt, sondern auch unter Berücksichtigung technischer Aspekte. So war es Vorgabe, neben dem rein lesenden Zugriff auf die Datenbasis, auch den schreibenden Zugriff mit seinen Anforderungen an das Session- und Transaktionsmanagement zu demonstrieren.

- In der **ersten Iteration** des Projekts lag der Schwerpunkt auf der Anbindung der Sprachapplikation an den Application-Server, der die notwendigen Seminaraten für den Use-Case *Seminarberatung* bereitstellt. Diese Daten müssen zur Laufzeit der Sprachapplikation ermittelt und in den Dialog eingebunden werden.
- In der **zweiten Iteration** lag der Fokus auf dem Ausbau der VXML-Dialoge. Die bisher zustandslose Sprachapplikation lernte im Rahmen des Use-Case *Reservierung* Buchungsvorgänge mit schreibenden Zugriff zu bearbeiten, welche ein Session- und Transaktionsmanagement erforderten.
- Die **dritte Iteration** wurde dem "Hear&Feel", also der Benutzerfreundlichkeit gewidmet: Die bislang relativ starren, geleiteten Dialoge des Use-Case *Seminarberatung* wurden durch gemischt-initiative Dialoge ersetzt, welche dem Benutzer mehr Flexibilität und eine natürlichere Steuerung der Folge der Dialogschritte ermöglichen. Die Qualität der Spracherkennung und -ausgabe wurde erheblich erhöht.

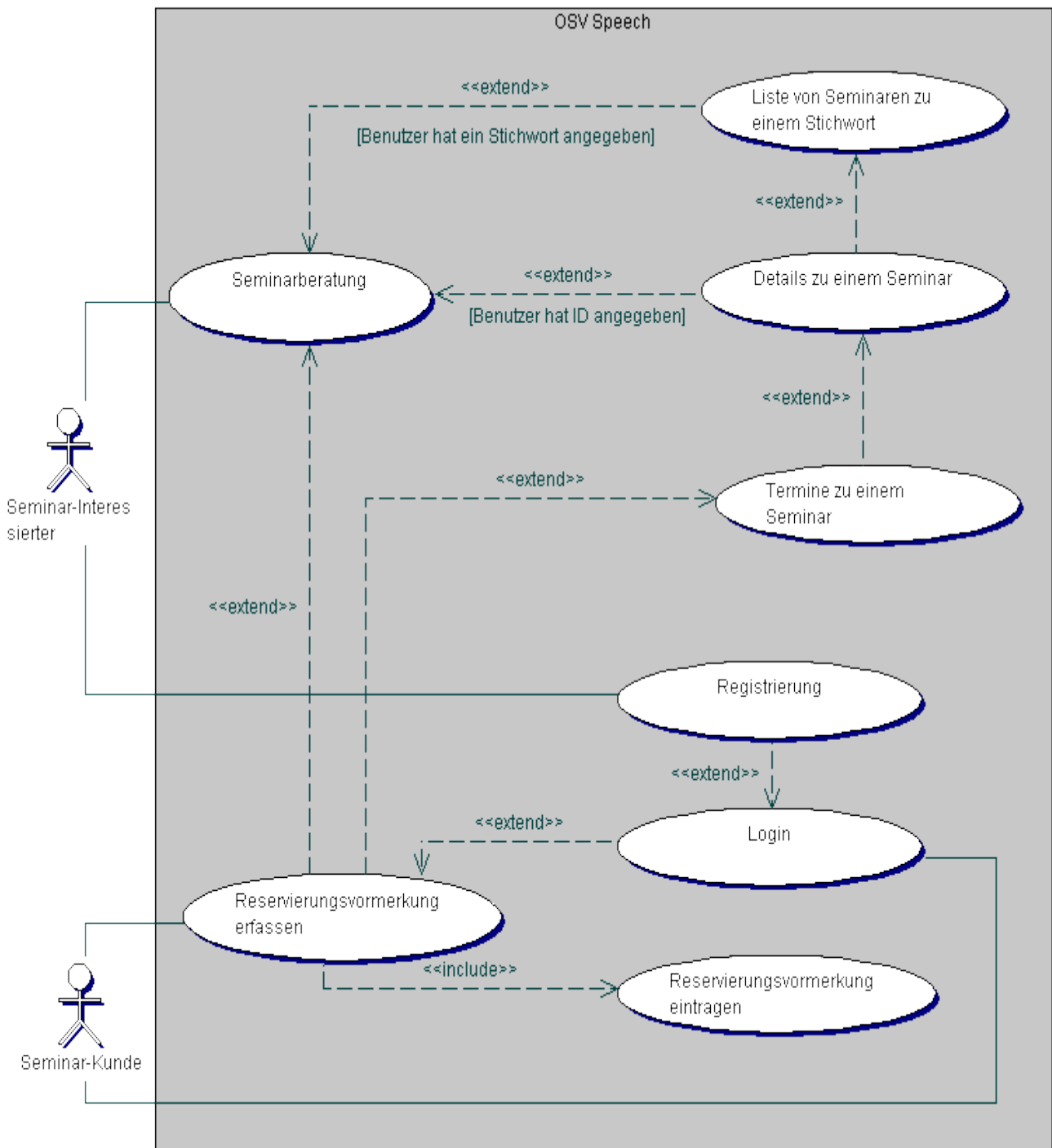


Abb.2: Use-Cases für die OSV-Sprachapplikation

Eine wesentliche **Anforderung** an unsere Sprachapplikation ist ihre Fähigkeit zur dynamischen Generierung von Grammatiken (Spracherkennungsregeln) und Prompts (Sprachausgaben) mit Semindaten, welche zur Laufzeit des Systems aus Datenbanktabellen gelesen werden. Das System ist somit in der Lage, die statischen Anteile der Sprachausgaben um generierte Anteile zu ergänzen und so dem Benutzer stets die aktuellen Informationen zu den angebotenen Seminaren zur Verfügung zu stellen.

Dies wurde durch den Einsatz von Java-Server-Pages (JSP) erreicht, welche statt HTML-Seiten, wie aus Web-Applikationen bekannt, VXML- bzw. grXML-Seiten lieferten. Als direkte Folge der sich ständig ändernden Datenbasis schied die Verwendung von echten Sprach-Samples (Audiodateien) an Stelle einer TTS-Engine für die Sprachausgabe des Systems aus. Jede Ergänzung des Datenbestands um neue Seminare würde sonst neue Aufnahmen und entsprechende Kosten mit sich bringen.

Unsere Sprachapplikation wurde weiterhin unter der **Anforderung** entwickelt, die bestehenden Enterprise Java Beans der existenten Web-Applikation unverändert weiter zu nutzen, d.h. keine sprach- bzw. VoiceXML-spezifischen Anteile in den Kern der Anwendungslogik einfließen zu lassen.

Da unsere Web-Applikation "Online-Seminar-Verwaltung" auf einer sauberen 3-Tier-Architektur basiert, war dies ohne weiteres möglich. Alle Erweiterungen blieben auf die Client-Tier beschränkt, Application-Tier und Data-Tier blieben von den Änderungen unberührt.

Für die **grafische Modellierung** von Sprachapplikationen gibt es bisher keine Standards. Neben den unabhängig von der Realisierung des User-Interfaces (GUI oder VUI) für die Modellierung des Systems verwendeten UML-Diagrammarten wie Use-Case-, Komponenten-, Deployment-, Sequenz- und Klassendiagrammen wurde nach einer Möglichkeit gesucht, den Ablauf der Sprachdialoge, also die Interaktion des Benutzers mit dem System, ebenfalls mit den Mitteln der UML darzustellen.

Als besonders geeignet hierfür erwiesen sich **Aktivitätsdiagramme mit Swimlanes**, in denen jeweils die Aktivitäten des Benutzers und der Sprachanwendung übersichtlich dargestellt werden können. In Abb.3 wird als Beispiel ein Aktivitätsdiagramm mit Swimlanes gezeigt, welches den Dialogverlauf während einer Seminarberatung skizziert.

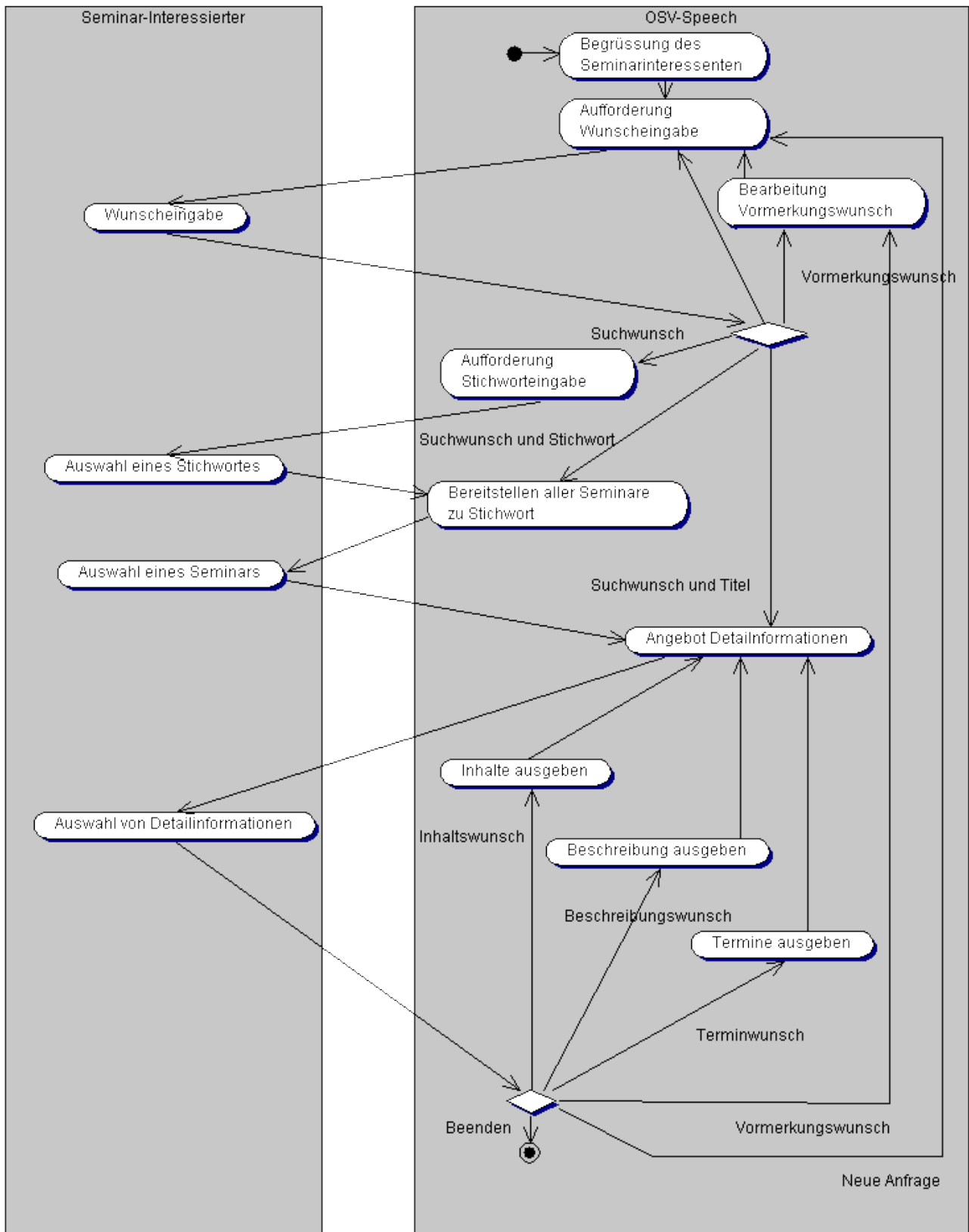


Abb.3: Aktivitätsdiagramm Seminarberatung

Besonderheiten beim Design von Sprachdialogen

Wichtig für die **Akzeptanz** einer Sprachapplikation durch den Benutzer ist in erster Linie ihre Benutzerfreundlichkeit. Wird der Benutzer nicht unterstützt, wird er überfordert, oder gelangt er nicht in angemessener Zeit zum gewünschten Ergebnis so wird er die Anwendung nicht annehmen. Dies gilt unabhängig davon wie gut ihre sonstige Funktionalität realisiert ist.

Im Gegensatz zum Design einer grafischen Benutzeroberfläche ergeben sich durch die Verwendung von Sprache als einzigem Medium völlig neue und einzigartige Probleme, die gelöst werden müssen.

- Selbst ein aufmerksamer Benutzer kann sich nur eine gewisse Anzahl an Optionen merken, zumal er unter Zeitdruck steht, da die Anwendung nur begrenzt lange auf eine Antwort wartet.
- Antwortet die Sprachapplikation nicht sofort, werden z.B. mögliche Ladezeiten von Webseiten nicht durch Sprachausgaben überbrückt, so entsteht für den Benutzer der Eindruck, die Anwendung würde nicht mehr arbeiten.
- Ein Benutzer antwortet nicht immer mit der von der Sprachapplikation erwarteten Wortwahl.
- Es kann passieren, dass ein Benutzer, obwohl er die erwartete Spracheingabe geliefert hat, dennoch von der Sprachapplikation nicht verstanden wird. Dies kann auch durch äußere Umstände wie Nebengeräusche und schlechte Verbindungen verursacht bzw. verstärkt werden.
- Ein Benutzer kann den Überblick über den aktuellen Zustand des Dialogs verlieren. Dies kann geschehen, falls die Sprachapplikation zu viele Informationen auf einmal liefert bzw. fordert, oder falls dem Benutzer nicht mehr klar ist, welche Eingabe- bzw. Steuerungsmöglichkeiten er hat.
- Der Benutzer kann gelangweilt oder genervt sein, falls zu lange oder wiederkehrende Sprachausgaben erfolgen. Das gilt besonders, falls er diese nicht unterbrechen kann.

Aus diesen Gründen ergeben sich für ein akzeptables VUI erfahrungsgemäß folgende Anforderungen:

- Die Sprachausgaben der Sprachapplikation sollen einfach und klar sein.
- Es sollen dem Benutzer zur Steuerung des Dialogs nicht mehr als 4 Auswahlmöglichkeiten auf einmal geboten werden.
- Es sollen möglichst wenig Dialogschritte erforderlich sein, bis der Benutzer sein Ziel erreicht hat, oder bis die Sprachapplikation wieder in den Startzustand zurückgekehrt ist.
- Die Auswahlmöglichkeiten in einem einzelnen Dialogschritt müssen klar und eindeutig sein und sollten sich nicht mit den Auswahlmöglichkeiten anderer Dialogschritte überschneiden, um die Orientierung des Benutzers innerhalb des gesamten Dialogs nicht zu gefährden.
- Dem Benutzer ist in jedem Dialogschritt Hilfe anzubieten. In dem Eingangsdialogschritt ist dem Benutzer eine Übersicht über die Funktionalität zu geben, welche von der Sprachapplikation angeboten wird.
- Der Benutzer bekommt eine Rückmeldung darüber, welche Informationen seiner Spracheingabe von der Sprachapplikation verstanden und wie sie interpretiert wurden. Er wird über seine Möglichkeiten zur Steuerung des Dialogs fortwährend informiert.

Auf die Erfüllung dieser Anforderungen wurde bei der Entwicklung des VUI für die K&A Online Seminarverwaltung großer Wert gelegt.

Ein weiterer Designaspekt bei Sprachdialogen betrifft die **Art der Dialogführung** durch die Sprachapplikation:

In den klassischen **geleiteten Dialogen** hat der Benutzer zur Steuerung des Dialogablaufs nur die Möglichkeit, aus mehreren angebotenen Optionen auszuwählen ("Sie wählen eine Waffel mit 5 Kugeln Eis. Bitte nennen sie mir die Geschmacksrichtung der ersten Kugel: Schoko, Vanille oder Erdbeer..."). Das System behält die Initiative bei sich. Die Applikation fragt, der Benutzer antwortet - und liefert dabei mit jeder Antwort genau eine Information. An dem Beispiel mit dem Eis ist erkennbar, dass dies nicht in allen Situationen die Art und Weise widerspiegelt, wie ein normaler Mensch gerne vorgehen würde.

Mit sogenannten **gemischt-initiativen** Sprachdialogen kann dem Benutzer eine weit natürlichere Art des Dialogs geboten werden, die einer Unterhaltung mit einem menschlichen Wesen schon recht nahe kommt ("Ich hätte gerne ein Eis in einer Waffel mit 3xSchoko und 2xVanille!"). Der Benutzer hat hier die Möglichkeit, die Folge und Anzahl der Dialogschritte aktiv zu beeinflussen. Er kann mehrere Eingabeinformationen in einer einzigen Äußerung an die Sprachapplikation übergeben. Diese bestimmt den Folgedialogschritt auf Grund der empfangenen Informationen. Da die Sprachapplikation keinen Wert auf Höflichkeit seitens des Benutzers legt, kann ein Power-User den Dialog dadurch verkürzen, dass er in jede Spracheingabe möglichst viele Informationen hineinpackt und Höflichkeitsfloskeln und Füllwörter weglässt.

Am Beispiel der Reservierung eines Seminars bedeutet dies, dass statt den Seminartitel, den Termin und die gewünschte Anzahl Plätze abzufragen, der (im vorliegenden Fall höfliche) Benutzer die Möglichkeit hat, alle oder auch nur einige dieser Informationen in einer Äußerung unterzubringen: "*Buchen Sie mir bitte das Seminar 'Programmieren in Java' für den 15. November*". Das System kann dann abhängig von den bereits erkannten, für die Reservierung nötigen Informationen, die restlichen Informationen, hier die gewünschte Anzahl Plätze, abfragen.

Darüber hinaus spielt die **Persona** einer Anwendung, also die Wortwahl, Betonung und Stimmlage der Sprachausgabe, eine große Rolle für die Akzeptanz der Sprachanwendung. Je nach Unternehmen und Zielgruppe sollte die Persona seriös oder jugendlich salopp wirken. Die Hersteller von TTS-Engines bieten meistens mehrere Stimmen an, die nach den jeweiligen Bedürfnissen zu wählen sind, bzw. auch Werkzeuge, welche die Bearbeitung und Anpassung der Stimmen ermöglichen.

Eine Alternative zum Einsatz einer TTS-Engine bietet das Aufzeichnen der von einem Sprecher vorgetragenen eigenen Texte in Audiodateien, die statt der Synthetisierung von Text in den Dialogen verwendet werden. Damit kann dem VUI noch besser der gewünschte Charakter verliehen werden. Allerdings ist diese Variante mit deutlich höherem Zeit- und Kostenaufwand versehen und scheidet bei Anwendungen mit ständig neuen Inhalten aus, falls man den Sprecher nicht dauerhaft verpflichten will.

Technische Realisierung: Überblick

Ausgangspunkt für die Realisierung der Sprachdialoge war die Wiederverwendung der EJBs (Enterprise Java Beans) unserer bestehenden J2EE-Web-Applikation "Online-Seminar-Verwaltung", welche in einer 3-Tier-Architektur angelegt ist. Die relevante Komponente ist in diesem Fall eine Session-Bean namens *UserController*, welche alle Aktionen, die ein Kunde auslösen kann, in der Application-Tier kapselt und somit eine Schnittstelle zur Client-Tier bildet.

In der Client-Tier existierten drei wichtige Komponenten:

Kernstück des Voice-Clients bilden **VXML-Dialoge**, welche direkt auf gesprochene Eingaben des Benutzers reagieren und entsprechende Antworten ausgeben können. Um diese VXML-Dialoge mit der Java-Welt zu verknüpfen, wurden sie als Java-Server-Pages konstruiert. Dies machte es theoretisch möglich, direkt Daten aus dem *UserController* in die VXML-Dialoge zu schleusen und umgekehrt Benutzereingaben an ihn weiterzuleiten.

Die **grXML-Grammatiken** waren eine weitere wichtige Komponente des Voice-Clients und dienten dazu, die Menge der Äußerungen des Benutzers zu definieren, welche die Anwendung zu einem bestimmten Zeitpunkt verstehen sollte. Auch sie wurden als JSP realisiert, so dass es möglich wurde, dynamisch Seminarnamen oder Suchbegriffe aus der Datenbank zum Sprachschatz der Applikation hinzuzufügen.

Um eine saubere Trennung von (akustischer) Darstellung und Logik zu erreichen und den Rest der Client-Tier gegen das EJB-Lifecycle-Management abzuschotten, wurden für den Web-Container Java-Klassen nach dem **Business-Delegate**-Pattern entworfen. Diese Business-Delegates bündeln die relevanten Methoden des *UserControllers* und stellen sie dem Rest der Client-Tier als Dienste zur Verfügung. Die JSPs greifen also, wie in Abb.4 zu sehen, nicht direkt auf den *UserController* zu, sondern gehen den Umweg über einen geeigneten Business-Delegate.

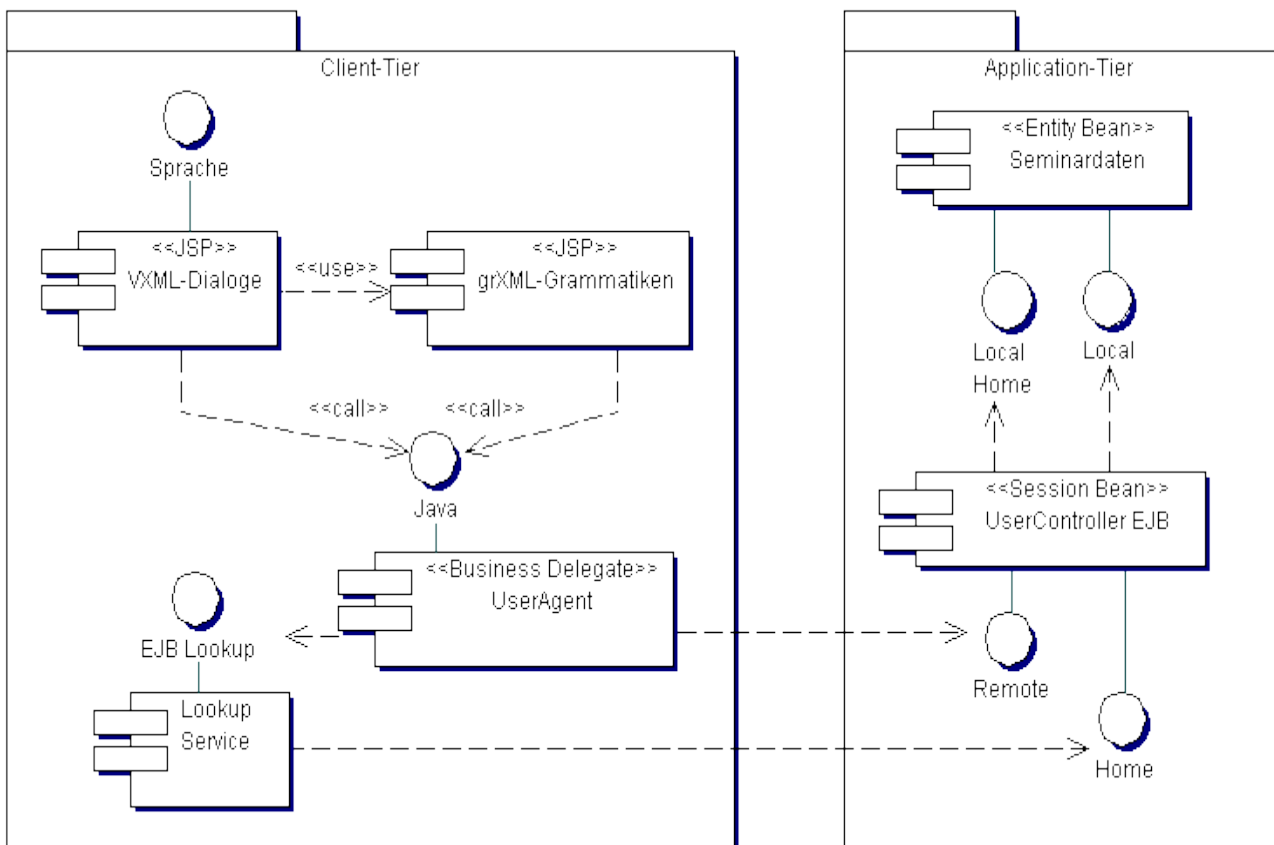


Abb.4: Intra-Tier-Architektur der Seminarverwaltung

Für den Use-Case *Seminarberatung* hat dieses Business-Delegate den Namen *ConsultingAgent*, für den Use-Case *Reservierung* den Namen *ReservationAgent*. Da die Anbindung an den *UserController*, sowie das Zwischenspeichern von Kursdaten gemeinsame Anliegen darstellen, wurden sie in einer Vererbungshierarchie verflochten, wie man in Abb.5 erkennt.

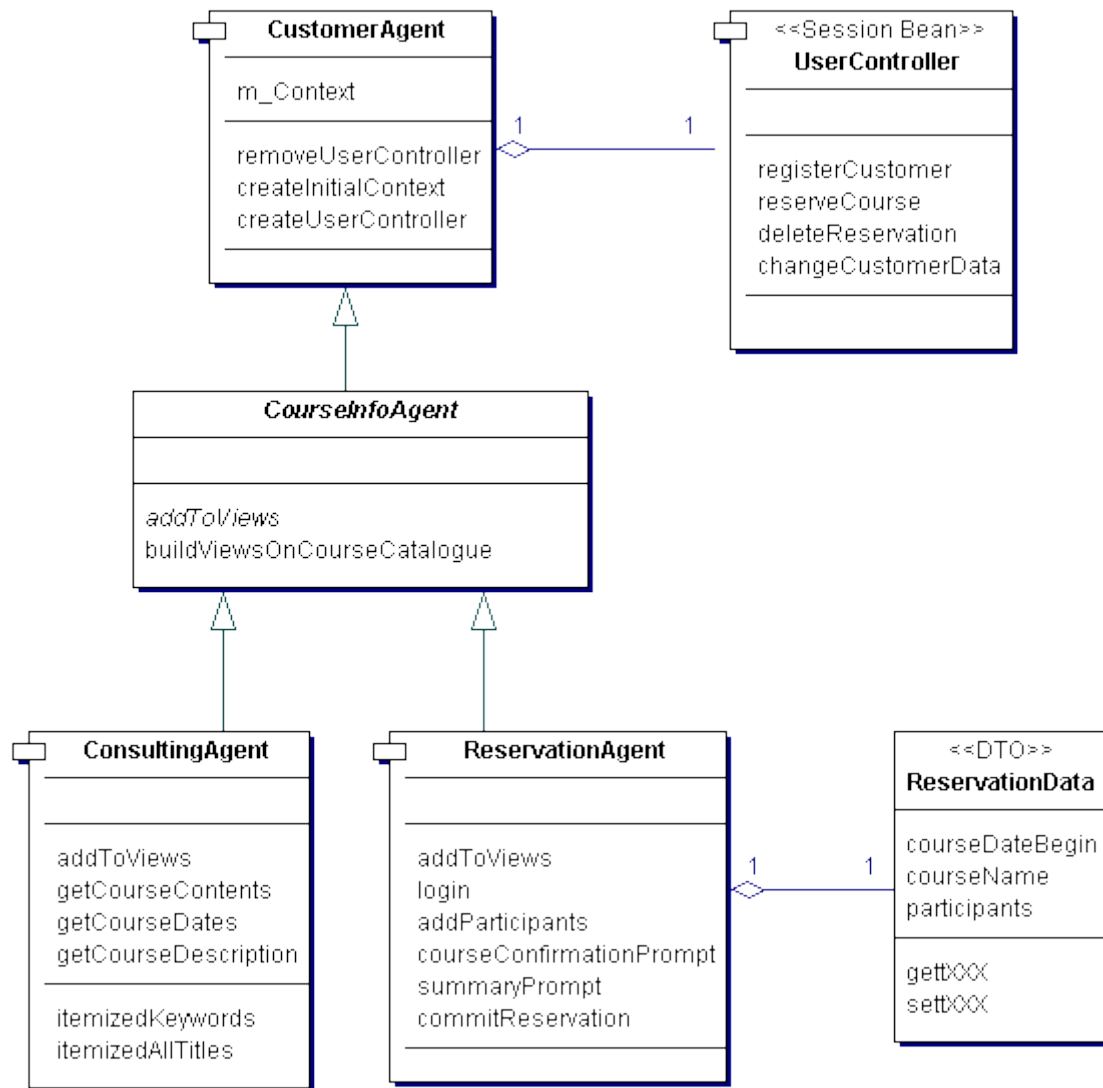


Abb.5: Vererbungshierarchie der CustomerAgents

Die VXML-Dialoge enthalten ähnlich wie HTML-Seiten Ausgaben (Prompts) und Eingaben (Verweise auf Grammatiken, welche vorher deklarierte Felder füllen). Zusätzlich gibt es noch Sprachelemente für die Ablaufsteuerung. Die **dynamischen Anteile der VXML-Dialoge** waren in unserer Anwendung vor allem Prompts, welche Seminarinformationen mit einbeziehen ("Wir können 3 Seminare zum Thema *Java* anbieten: *X, Y und Z*" oder "Die Seminarveranstaltung zum *15. Oktober* ist leider ausgebucht").

Ähnliches gilt für die **dynamischen Anteile der grXML-Grammatiken**: Sie müssen die erlaubten Suchbegriffe und Seminarnamen aus der Datenbank kennen, um diese Wörter in den Äußerungen des Benutzers wiedererkennen zu können.

Die folgende JSP zeigt einen VXML-Dialog, in dem der Anrufer nach einem Stichwort gefragt wird, zu dem dann später eine Liste von interessanten Seminaren ermittelt werden kann. Dies ist praktisch das einfachste **Beispiel** für einen in VoiceXML geschriebenen Dialog mit Ein- und Ausgaben. Im letzten Prompt wird eine Instanz des Business-Delegates *ConsultingAgent* verwendet, um eine Liste von erlaubten Stichworten aus dem Datenbestand zu erzeugen.

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Import Anweisungen und Deklarationen (ausgelassen) -->

<vxml version="2.0" ... >
  <form id="keywordSearch">
    <field name="stichwort">
      <grammar src="grammarMenu2.jsp#ROOT2"
type="application/srgs+xml" />
      <prompt bargein="true">
        Zu welchem Thema suchen Sie ein Seminar?
      </prompt>
      <filled>
        <prompt>
          Ich stelle Ihnen nun Seminare
          zum Thema <value expr="stichwort"/> zusammen.
        </prompt>
        <submit expr="application.consulting_keywordResultURL"
          method="get"
          namelist="stichwort"
          fetchtimeout="10s"/>
      </filled>
    </field>
    <nomatch>
      <prompt>
        Ich habe leider kein Thema verstanden, das ich kenne.
        Ich gebe ihnen mal eine Liste der verfügbaren Themen:
        <%= consultingAgent.getKeywords() %>
      </prompt>
      <reprompt/>
    </nomatch>
  </form>
</vxml>

```

Für eine konsequente Umsetzung des Model-View-Controller-Patterns wurde der Einsatz des **Struts-Frameworks** von Jakarta evaluiert, welches sich in dem Web-Client unserer Anwendung bewährt hat. Da dieses Framework jedoch nie für den Einsatz in Voice-Clients konzipiert wurde, waren wir auch wenig überrascht, dass es sich als ungeeignet erwies.

Unabhängig davon wurde der Nutzen eines Einsatz von **Tag-Libraries** überprüft, um Java-Scriptlets aus den JSPs fernzuhalten. Kandidaten dafür waren sowohl eigens von uns für den Web-Client entwickelte Tags, als auch solche, die das Struts-Framework bereitstellt. Während die selbstgeschriebenen Tags (z.B. um den Kontrollfluss je nach Login-Zustand des Anrufers zu verändern) durchaus Sinn ergaben, waren die meisten der Struts-Tags nicht zu gebrauchen: Die Tiles- und HTML-Tags beziehen sich auf grafische Elemente, die es in einem Voice-Client nicht gibt, die Logic-Tags hingegen werden durch die Programmiersprache VoiceXML obsolet. Einzig die Bean-Tags konnten nutzbringend verwendet werden.

Session-Management

Ein Webserver hat kein Gedächtnis. Sendet ein Browser erneut Daten an den Webserver, so sind diese für den Server in keine Beziehung zu früher gesendeten Daten zu bringen. Dies macht

komplexere Buchungsvorgänge in mehreren Schritten eigentlich unmöglich. Um dem abzuweichen, ordnet der Webserver jeder Anfrage und den mitgesendeten Daten eine eindeutige Nummer zu, die sogenannte Session-ID. Zwei verschiedenen Techniken, Cookies und URL-Rewriting, sorgen dafür, dass der Browser dem Server bei der nächsten Anfrage wieder die richtige Session-ID mitschicken kann.

In unserem Voice-Client war Session-Management für den zweiten Use-Case *Reservierung* ein wichtiges Thema, da hier festgehalten werden musste, ob der Kunde sich schon eingeloggt und welche Wünsche er bzgl. Seminar, Datum und Teilnehmerzahl schon geäußert hatte.

Auf der Seite des Servers war die Verwendung von Cookies in der `server.xml` von Jboss leicht zu konfigurieren. Während aber auf der Seite der Clients alle gängigen Webbrowser diese Technik schon lange beherrschen, ist dies bei Voicebrowsern noch nicht selbstverständlich. Hier mussten wir zur etwas umständlicheren Technik des URL-Rewritings greifen.

Hat man den Web-Container erst einmal konfiguriert, stellt er Sessions als Java-Objekte im Rahmen seiner Infrastrukturdienste zur Verfügung. Gegen die technischen Details abgekapselt, ist dann in den JSPs die Verwendung von Objekten, welche die ganze Session über bestehen, äußerst simpel.

Subdialoge

VoiceXML bietet mit Subdialogen die Möglichkeit, ähnlich wie mit einer Funktion einer herkömmlichen Programmiersprache, zu verzweigen und nach Abarbeitung des Subdialogs an die Stelle des Aufrufs zurückzukehren. Von Subdialogen aus lassen sich wiederum weitere Subdialoge aufrufen, ganz wie in den erwähnten Funktionen. Subdialoge können mit Parametern versorgt werden und liefern Werte in einem [ECMA-Script](#) Objekt zurück.

So lassen sich Funktionalitäten gemeinsam nutzen, wie z.B. die Registrierung oder die Anmeldung eines registrierten Benutzers. Damit können dann sogar Bibliotheken von Subdialogen aufgebaut werden, die in verschiedenen Sprachapplikationen Anwendung finden.

Sprachausgabe

Eine weitere Besonderheit in dem vorgestellten Projekt stellte die Techniklastigkeit der auszugebenden Texte dar, die stark mit (zumeist englischsprachigen) Fachausdrücken wie "Threads", "J2EE" oder "Java Beans" durchsetzt war.

Hier galt es, eine geeignete Text-To-Speech-Engine auszuwählen, welche englische Vokabeln erkennt und richtig ausspricht. Falsche Aussprachen ließen sich durch ein benutzerdefiniertes **Aussprache-Dictionary** korrigieren.

Während mit einem solchen Dictionary die Aussprache einzelner Wörter gesteuert werden kann, wird mit der Verwendung der **Speech Synthesis Markup Language (SSML)** Einfluss auf weitere Sprachmerkmale genommen, welche man unter dem Stichwort **Prosodie** zusammenfasst. Mit den in SSML definierten Tags können Betonung, Geschwindigkeit, das Stimmvolumen und die Tonhöhe festgelegt werden.

Spracherkennung

Auch für das Spracherkennungsmodul stellten die oben vorgestellten Fachausdrücke und Abkürzungen ein Problem dar. Die Abhilfe brachte hier ebenfalls ein benutzerdefiniertes **Aussprache-Dictionary**. Leider verwenden verschiedene Hersteller auch verschiedene Phonem-Alphabete und Datei-Formate, so dass wir das Wörterbuch für die Sprachausgabe nicht wiederverwenden konnten.

Bei kompletten Sätzen ("Geben Sie mir die *Beschreibung*, bitte"), wie sie vornehmlich in gemischt-initiativen Dialogen vorkommen, muss der Spracherkennung die interessante **Kerninformation** vom sogenannten "Garbage" trennen. Dazu steht innerhalb der Grammatiken eine ECMA-Script-ähnliche Programmiersprache (Semantic Interpretation-Script) zur Verfügung, welche vom Spracherkennung interpretiert wird. Obwohl es Bemühungen des W3C gibt, zwei Standards für diese Programmiersprachen zu etablieren, sind diese Spezifikationen leider noch zu neu, um sich gegen die vorherrschenden proprietären Lösungen durchzusetzen.

Funktionaler Test

Ein weiterer interessanter Aspekt war der Test der Sprachapplikation und der verwendeten Komponenten. Hier stellte sich die Frage, ob solche Anwendungen ganz neue Ansätze für den Test erfordern oder ob zumindest Teile davon wie gewohnt **Regressionstests** unterzogen werden können. Da der Voice-Client und der Web-Client gemeinsame Technologien wie Java und Java-Server-Pages verwenden, waren zumindest diese mit herkömmlichen Mitteln zu testen.

Die **Business-Delegates** als reine Java Klassen wurden mit "JUnit" getestet. Um die Regressionstestfähigkeit zu gewährleisten, wird dabei in der setup-Methode die verwendete Datenbank mit einem definierten Inhalt gefüllt. Die Business-Delegates konnten sogar außerhalb des Web-Containers getestet werden. Zu beachten war dabei nur eine geeignete Initialisierung des Environments für das Lookup auf die verwendete Session-Bean *UserController*.

Für den **Test der Java-Server-Pages** wurde das Jakarta Test-Framework "Cactus" eingesetzt. Auch hier kann ein Regressionstest durchgeführt werden, der prüft, ob die von den Java-Server-Pages generierten VoiceXML-Dokumente, samt den dynamischen, anhand der Datenbankinhalte erstellten Anteilen, genau den vorgegebenen Erwartungen entsprechen. Mit Hilfe dieses Cactus-Tests konnte das Ergebnis der bereits "deployten", d.h. im Web-Container ablaufenden Java-Server-Pages, abgefragt und, wie bei einem normalen JUnit-Test, mit `assertEquals()` mit einem erwarteten Ergebnis verglichen werden.

Der **Test der Grammatiken** wurde mit Hilfe der mit der OpenSpeech ASR mitgelieferten Werkzeuge *parsetool* und *test_parser* durchgeführt. Diese ermöglichen es, die Grammatiken isoliert, also nicht im Kontext der Sprachapplikation, zu testen. Insbesondere können damit die von den Grammatiken erkannten Äußerungen einem Regressionstest unterzogen werden.

Der **Systemtest**, bei dem die gesamte Anwendung getestet wird, wurde anhand eines Testdrehbuchs durchgeführt. Pro Testfall wurden die vom Benutzer zu tätigen Eingaben und die daraufhin vom System erwarteten Reaktionen definiert und die tatsächlichen Ergebnisse während der Testdurchführung festgehalten. Auch für solche manuellen Tests stehen bereits, wie beispielsweise

mit dem "Rational Manual Tester", geeignete Werkzeuge für die Definition der Testfälle und die Dokumentation der Testergebnisse zur Verfügung.

Akzeptanztest

Mit den bisher aufgeführten Tests wäre dann zwar die Funktionalität der Anwendung getestet, jedoch noch nicht die Güte der Spracherkennung, die Vollständigkeit der Grammatiken und weitere Aspekte, welche die Benutzbarkeit bzw. Benutzerfreundlichkeit unerfahrene und erfahrene Benutzer beeinflussen.

Dieser Benutzertest, oder auch Akzeptanztest nach dem Rational Unified Process, ist bei Sprachapplikationen jedoch sehr viel wichtiger als bei einer Anwendung mit grafischer Benutzeroberfläche, da ein frustrierter Benutzer nie wieder anrufen wird. Der Test beginnt sehr früh in Form von Wizard-Of-Oz-Simulationen (Mensch simuliert Maschine) und dauert auch noch weit bis in den laufenden Betrieb hinein an (Alpha-Release). Denn erst in dieser Phase kann automatisiert (z.B. durch Mitschnitte oder Auswertung von Logfiles) eine wirklich große Masse an Testdaten gesammelt werden, die dann wiederum zum Tuning des Voice-Clients führt.

Die gesammelten Daten sollten nach den folgenden Fragestellungen ausgewertet werden, die es erlauben, Schwachstellen in Dialogdesign, Spracherkennung oder Prompts aufzudecken:

- in wievielen Schritten kommt der Benutzer zum Ergebnis?
- wie oft schweigt ein Benutzer?
- wie oft erkennt der Spracherkennung Äußerungen nicht?
- wie oft und an welchen Stellen wird die Hilfe aufgerufen?
- werden bestimmte Dialogteile nicht benutzt?

Der Zugriff auf die Clarity-Plattform erfolgte dabei über H.323-Clients (OpenPhone 1.8.1). Durch diese Möglichkeit ist es nicht nötig, für den Test die Verbindung über ISDN herstellen zu müssen.

Die Erkenntnisse aus dem Test und der Befragung der Benutzer wurden zum Tuning der Anwendung genutzt, um so den künftigen Benutzern den größtmöglichen Nutzen zu bieten.

Performance-Test

Der Performance-Test kann in zwei Stufen vorgenommen werden.

In der **ersten Stufe** wird durch gleichzeitigen Zugriff auf die Java-Server-Pages die Performance des Webservers unabhängig von der VoiceXML-Plattform gemessen. Der gleichzeitige Zugriff kann entweder über einen selbst geschriebenen Testtreiber erreicht werden, der z.B. mit Hilfe einer konfigurierbaren Anzahl Threads parallel den Cactus-Test für die Java-Server-Pages in einer beliebigen Frequenz startet. Es gibt aber auch eine Anzahl Werkzeuge, die für solche Tests verwendet werden können, wie z.B. eines, das auf der Seite www.softwareqatest.com beschrieben ist.

In der **zweiten Stufe** wird die VoiceXML-Plattform mit einbezogen und ihre Performance während der Ausführung der VoiceXML-Dokumente, der Spracherkennung und -ausgabe unter Lastbedingungen getestet. Wenn im produktiven Betrieb eine geringe Last erwartet wird, so kann

der Test durch gleichzeitig anrufende Testpersonen erreicht werden. Für größere Last ist ein sogenannter Call-Generator geeignet, der eine voreingestellte Anzahl an parallelen Anrufen erzeugt.

Deployment

Das Deployment für den Voice-Client unterscheidet sich nur geringfügig vom Deployment eines Web-Clients. Es wird genau wie dort ein **Web-Archiv** (war-File) erzeugt und auf dem Webserver installiert. Das Web-Archiv enthält die Java-Server-Pages, übersetzte Java-Klassen, eventuell abzuspielende Audiodateien, benötigte Java-Archive, die Deployment-Deskriptoren, Tag-Library-Deskriptoren und ggf. die für den Cactus-Test benötigten Dateien.

In einer **Konfigurationsdatei** der Voice-Plattform wurden unter anderem die Anzahl der Voice-Prozesse, die zu verwendenden Spracherkenner und Sprachsynthetisierer und deren Parameter sowie die Anbindung an ein VoIP-Netzwerk oder ISDN-Telefonnetz eingestellt.

Fazit und Ausblick

Die gesammelten Erfahrungen und die erfolgreiche Durchführung des Projekts zeigen, dass sich eine Sprachapplikation hinsichtlich der Projektdurchführung nicht wesentlich von den bisher bekannten Projekten unterscheidet, außer in folgendem Punkt: Es werden typischerweise noch viele Anpassungen des Sprachdialogs unter "Einsatzbedingungen" durchgeführt.

Die Anbindung an einen Webserver war problemlos möglich. Das in dieser Hinsicht bereits vorhanden Know-how konnte auch für die Entwicklung der Sprachapplikation genutzt werden.

Ebenso wie bei herkömmlichen Projekten eine Rollentrennung von Webdesigner und Java-Entwickler möglich ist, ließen sich in unserem Voice-Projekt die Rollen von Sprachdialog-Designer und Java-Entwickler trennen.

Zusätzliches Wissen ist über die Sprache VXML und die Grammatiken (SRGS) sowie die diversen Standards aus dem Umfeld wie z.B. SSML und die sich damit eröffnenden Möglichkeiten nötig. Bei der Auswahl der Hard- und Software-Komponenten ist darauf zu achten, welche Hersteller die noch sehr jungen Standards überhaupt schon vollständig unterstützen.

Den Hauptunterschied bei der Entwicklung einer Sprachapplikation stellt das Design des Voice-User-Interface dar, bei dem auf die speziellen Anforderungen eingegangen werden muss. Dazu kommt das Tuning der Sprachausgabe, die Prosodie, um dem System einen möglichst natürlichen, angenehm klingenden, dem Thema und der Zielgruppe angemessenen Auftritt zu verschaffen.

Für die Entwicklung, den Test und das laufende Tuning des VUI sind mehrere Iterationen nötig, wofür sich die Projektdurchführung im Rahmen des Rational Unified Process naturgemäß besonders gut eignet.

Die System-Modellierung kann wie gewohnt mit dem Mitteln der UML erfolgen.

Mit dem "W3C Speech Interface Framework" bietet das W3C-Konsortium weitere Standards für die Entwicklung, Verfeinerung von Sprachapplikation und die Erweiterung der Funktionalität.

Das sind im Einzelnen:

- [VoiceXML 2.0](#) für die Definition der Dialoge und den Austausch von Informationen zwischen dem Benutzer und der Sprachapplikation.
- [VoiceXML 2.1](#) mit zusätzlichen Funktionen.
- [Speech Recognition Grammar Specification](#) (SRGS) für Grammatiksprachen
- [Speech Synthesis Markup Language](#) (SSML) für die Aufbereitung der Sprachausgabe.
- [Semantic Interpretation Script](#) für die Extraktion von Kerninformationen während der Spracherkennung
- [CCXML](#) für Call-Control-Funktionen wie Telefonkonferenz, Rufweiterleitung, etc.

Abb.6 zeigt, wo die vom W3C standardisierten Sprachen ihren Platz in der gesamten Sprachtechnologie einnehmen.

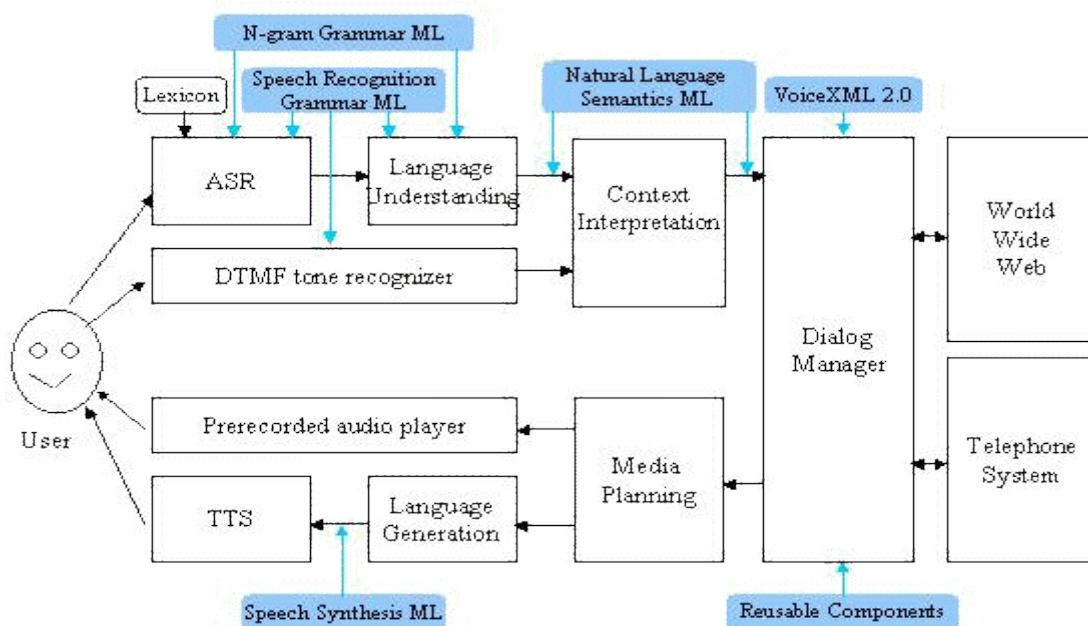


Abb.6: W3C Speech Interface Framework (Quelle: W3C)

Darüber hinaus gibt es weitere Bestrebungen, Voice-User-Interface und Graphical-User-Interface nicht isoliert zu betrachten, sondern in sogenannten multimodalen Anwendungen zu vermischen. Hier bieten SALT, die [Speech Application Language Tags](#), eine geeignete Alternative zu VoiceXML, da die Tags direkt in HTML-Seiten eingeflochten werden können. Einen guten Überblick der Visionen zu diesem Thema und dem aktuellen Stand bietet das [W3C](#) auf der Seite [Multimodal Interaction Activity](#).

Die Firma Kölsch & Altmann wird diese Aktivitäten auch weiterhin verfolgen und diese Erfahrungen zum Nutzen ihrer Kunden einsetzen.