Barrierefreier Webkalender

Projektpraktikum aus Software-Engineering
Evaluierungsergebnisse

Martin Scharrer, 0655546

Johannes Kepler Universität Linz
Juli bis September 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Barrierefreiheit von HTML Seiten an sich ist mittlerweile ein Forschungsgebiet, an dem viele Wissenschaftlerinnen und Wissenschaftler arbeiten. Trotzdem gibt es noch einige Teilbereiche zu welchen noch Lösungen fehlen.
Genau das ist ein Ziel dieses Praktikums – Technologien einzusetzen, welche mittlerweile weit verbreitet und zu einem Web 2.0 Standard geworden sind aber nicht unbedingt Barrierefreiheit unterstützen bzw. gewährleisten.
Deshalb sollen in diesem Praktikum Elemente evaluiert werden, welche als Grundlage für die Entwicklung eines Webkalenders dienen können. Im Fokus stehen Elemente, welche überwiegend zum Aufbau der Seite, bzw. für die Benutzerinteraktion eingesetzt werden.

2 Evaluierung bestehender Technologien

2.1 JQuery UI

Ein wesentliches Merkmal von Rich Internet Applications (RIA) ist neben dem umfangreichen funktionalen Aspekt vor allem eine interaktive, einfach bedienbare, einer Desktop – Anwendung nachempfundene Benutzerschnittstelle.
Um ein User Interface (UI) diesen Anforderungen gerecht werden zu lassen, bietet die Website jqueryui.com [3], basierend auf der bekannten Javascript Bibliothek jquery [4], ein Framework zum Download an, welches unterschiedlichste Bedienelemente zur Verfügung stellt, um den Anforderungen einer RIA gerecht zu werden.
Im ersten Schritt wurden von diesem Framework (jQueryUI) Elemente ausgewählt, welche für die Implementierung eine Kalenders als hilfreich erschienen, und deren Zugänglichkeit anhand der Web Content Accessibility Guidelines 2.0 (WCAG 2.0) [5] evaluiert. Die Ergebnisse sowie die notwendigen Verbesserungen werden in den folgenden Kapiteln ausführlich diskutiert.

3 Evaluierungsergebnisse zu jQuery UI

3.1 Datepicker

Eine absolut notwendige Funktionalität für einen Kalender ist die unterstützte Eingabe eines Datums in ein Textfeld. Ein Datepicker ist dabei praktisch zum Standard geworden und wird in diesem Abschnitt auf dessen Zugänglichkeit geprüft.
Ausgangspunkt für diese Analyse ist das folgende Beispiel:

/demos/datepicker/icon-trigger.html

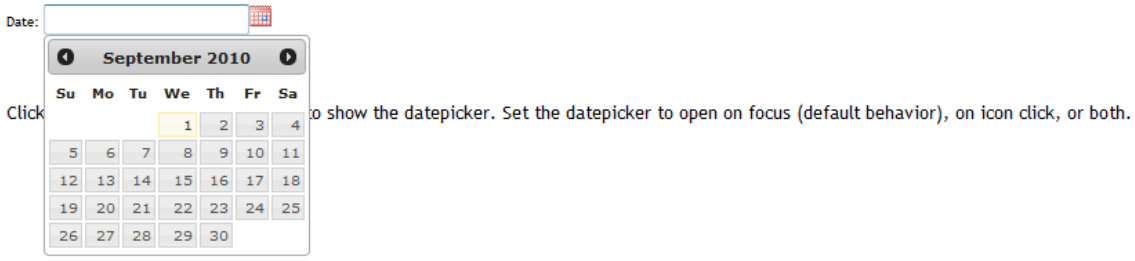

Abbildung 1: Datepicker mit Icon Trigger

3.1.1 Prinzip 1: Wahrnehmbar

Richtlinie 1.1 besagt, dass für alle Nicht-Text-Inhalte Textalternativen zur Verfügung gestellt werden müssen, insbesondere bei Steuerelemente und Eingabe muss das Textfeld einen Namen haben, der seinen Zweck beschreibt.
Ein Punkt der bei diesem Beispiel nicht erfüllt ist – der einfache Label-Tag fehlt.
Des Weiteren sind bei der Deaktivierung der Grafiken zwar Textalternativen vorhanden, diese werden aber nicht angezeigt oder sind nicht bedienbar. Konkret betrifft das die beiden Pfeile zur Navigation zwischen den Monaten, und das Icon über welches man grundsätzlich den Datepicker einblendet.

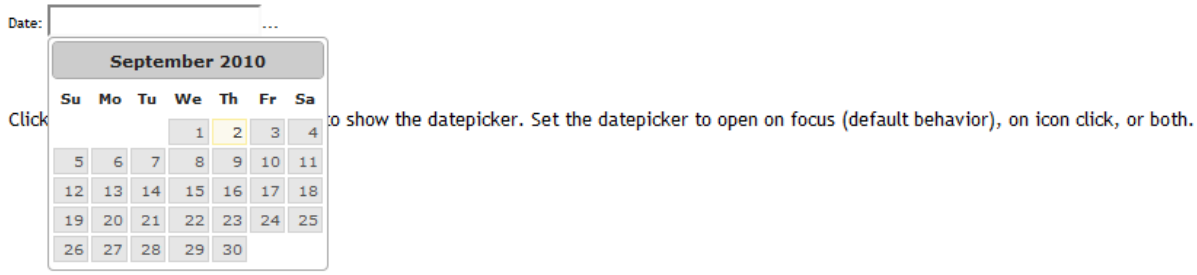

Abbildung 2: Datepicker mit Alternativtexten für die Grafiken

Bei den Pfeilen zur Navigation wurde zwar das das title – Attribut beim Link Tag verwendet, jedoch ist die Texteinrückung so definiert, dass der Text an sich aus dem Bild verschwindet. Können nun Grafiken nicht geladen - oder von Benutzerinnen und Benutzer nicht interpretiert werden, so kann dieses Element nicht mehr verwendet werden.
Das Icon ist als <img /> mit einem onclick Event ausgeführt. Das Problem hierbei ist, dass bei der Verwendung der Textalternativen das onclick Event nicht ausgelöst wird, bzw. für Text einfach kein onclick Event möglich ist.
Richtlinie 1.2 behandelt zeitbasierte Medien und ist für Datepicker nicht ausschlaggebend.
Richtlinie 1.3 definiert, dass Inhalte auf unterschiedliche Arten dargestellt werden können – ohne Informationsverlust. Insbesondere muss gewährleistet sein, dass Abhängigkeiten zwischen einzelnen Teilen – also z.b. zwischen Eingabefeld und Datepicker von Software bestimmt werden können – in großes Problem bei dieser Anwendung.
Bei der Verwendung eines Screenreaders wird weder darauf hingewiesen, dass die Möglichkeit besteht das Datum über einen Datepicker auszuwählen, noch kann die Reihenfolge bestimmt werden. Werden beispielsweise sämtliche Stylesheets deaktiviert, so erscheint der Datepicker am Ende der Seite – siehe folgende Abbildung.

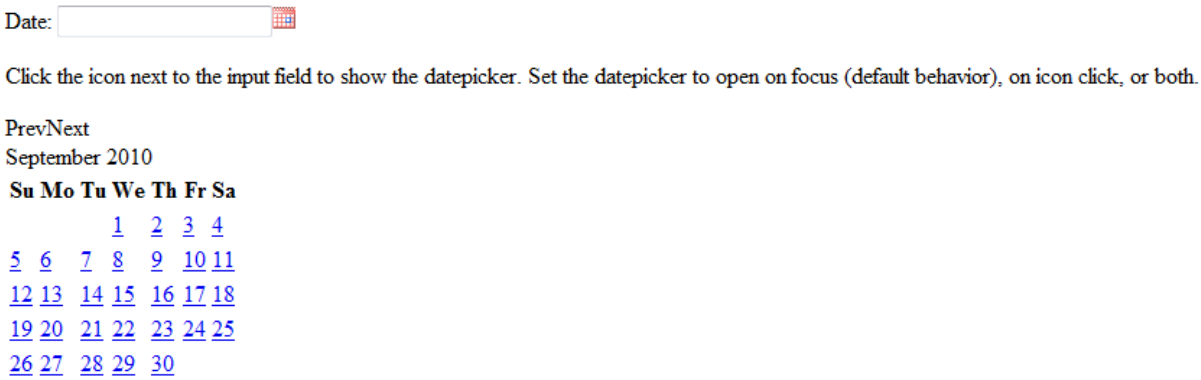

Abbildung 3: Datepicker ohne Stylesheets

Die Benutzung von Farbe als einziges visuelles Mittel und ein entsprechendes Kontrastverhältnis sind beim Datepicker gegeben - getestet mit [6] (dunkelgrau auf hellgrauem Grund - #4a4a4a, #e6e6e6 – 7,1:1).
Auch die Änderung der Textgröße ist einwandfrei möglich.

3.1.2 Prinzip 2: Bedienbar

Richtlinie 2.1 besagt, dass alle Funktionalitäten per Tastatur zugänglich sind. Beim Datepicker ist das bei mehreren Elementen nicht der Fall. Wie schon im Abschnitt 3.1 beschrieben, ist das Icon als Grafik ausgeführt – und Grafiken können nie den Focus erhalten. Wenn man die Seite mit Tab durchgeht, so kann man den Datepicker nie einblenden. Des Weiteren ist die Navigation der Monate nicht per Tastatur bedienbar, weil auch diese Elemente nicht fokusierbar sind.
Geht man davon aus, dass der Datepicker zwar angezeigt werden kann (dadurch, dass man den Datepicker schon einblendet sobald das entsprechende Textfeld den Fokus erhält) kann man noch immer nicht durch navigieren, weil der Datepicker sofort wieder ausgeblendet wird, wenn der Fokus vom Textfeld weg, zum nächsten Element wechselt.
Diese Tatsache widerspricht auch der Richtlinie 2.2 – Ausreichend Zeit.
Der Benutzer kann die „automatische Aktualisierung“ (das Einblenden / Ausblenden des Datepickers) nur bedingt kontrollieren.
Die letzte Richtlinien 2.4 (2.3 ist nicht relevant) besagt, dass die Seite navigierbar sein muss. Einige dieser Problem wurden bereits beschrieben – wie Fokus-Reihenfolge und Linkzweck – und werden somit bereits berücksichtigt.
Ein Problem welches hier noch festgestellt wurde ist, dass – sobald der Datepicker einmal geöffnet wurde – und der Fokus auf einem Element im Datepicker steht, dieser nicht mehr geschlossen werden kann. Erhält ein Element außerhalb des Datepickers den Fokus, so kann der Datepicker mit Esc geschlossen werden – jedoch wird diese Möglichkeit nicht dargestellt bzw. dem Benutzer oder der Benutzerin vermittelt.

3.1.3 Prinzip 3: Verständlich

Lesbar – Richtlinie 3.1 ist der Datepicker nur bedingt. Das Datumsformat beispielsweise ist nicht abhängig der bevorzugten Spracheinstellung im Browser (Deutsche Sprache, englisches Datum). Gleiches gilt für die Abkürzungen bei den Wochentagen – die sind auch in englischer Sprache und nicht genauer erläutert. Des weiteren werden als Alternativtexte englische Abkürzungen verwenden („Prev“ - für das navigieren zum vorherigen Monat).
Richtlinie 3.2 widerspricht der Möglichkeit, den Datepicker sofort einzublenden, wenn das

entsprechende Textfeld den Focus erhält – „3.2.1 Bei Fokus: Wenn irgendein Bestandteil den Fokus erhält , dann löst dies nicht eine Änderung des Kontextes aus. (Stufe A)“.
Die Richtlinie zur Fehlerkennung und Fehlervermeidung (3.3) wird praktisch erfüllt. Der Datepicker an sich hilft bei der Vermeidung eines Eingabefehlers (Datum im falschen Format). Natürlich wäre ein Validierung der Eingabe und entsprechende Rückmeldung an den Benutzer sinnvoll (3.3.3 Fehlerempfehlung).

3.1.4 Prinzip 4: Robust

Die einzige Richtlinie dieses Prinzips besagt zusammengefasst erstens, dass eine Syntaxanalyse erfolgreich abgeschlossen werden muss und zweitens, dass Name, Rolle und Wert von Bestandteilen der Benutzerschnittstelle durch Software bestimmt werden können. Getestet wurde das mit dem Screenreader von JAWS. Soweit die Elemente mit Tastatur erreichbar waren, konnten diese auch vom Screenreader interpretiert werden. Natürlich steht diese Tatsache im Zusammenhang mit der Erreichbarkeit einzelner Elemente per Tastatur. Das momentan angezeigte Monat kann in dieser Ausführung beispielsweise nicht von Software bestimmt werden.

3.2 Dialog

Eine, für den User komfortable Interaktion mit einer Anwendung ist eine Grundvoraussetzung, damit ein bestimmtes Produkt auch von den Anwendern akzeptiert und genutzt wird. Die von Betriebssystemen bekannten Dialogfelder zur Bestätigung von Eingaben, Systemzuständen oder Änderungen erwartet ein User auch von Webanwendungen. Diese Anforderung kann mit dem Dialog Feature von jQuery UI erfüllt werden. Dabei können nicht nur Mitteilungen an den Benutzer und die Benutzerin weitergegeben werden, vielmehr dienen diese Dialoge auch zur asynchronen Eingabe von Daten – beispielsweise zwischen der Bearbeitung einer bestimmten Aufgabe. Die Zugänglichkeit dieser Dialog wird in diesem Abschnitt diskutiert und evaluiert. Als Beispiel wurde folgendes verwendet:

/demos/dialog/modal-form.html

3.2.1 Prinzip 1: Wahrnehmbar

Die in Richtlinie 1.1 festgelegten Kriterien sind nicht zur Gänze erfüllt. Dabei wird festgelegt, dass man für alle Nicht-Text-Inhalte Textalternativen zur Verfügung stellen muss. Ein diesbezügliches Problem tritt beim Icon, welches zum Schließen des Dialogs im rechten, oberen Bereich angezeigt werden soll, auf. Die folgende Abbildung zeigt den Dialog im ursprünglichen Layout, bei welchem die Grafik zum Schließen des Dialogs eindeutig zu erkennen ist.

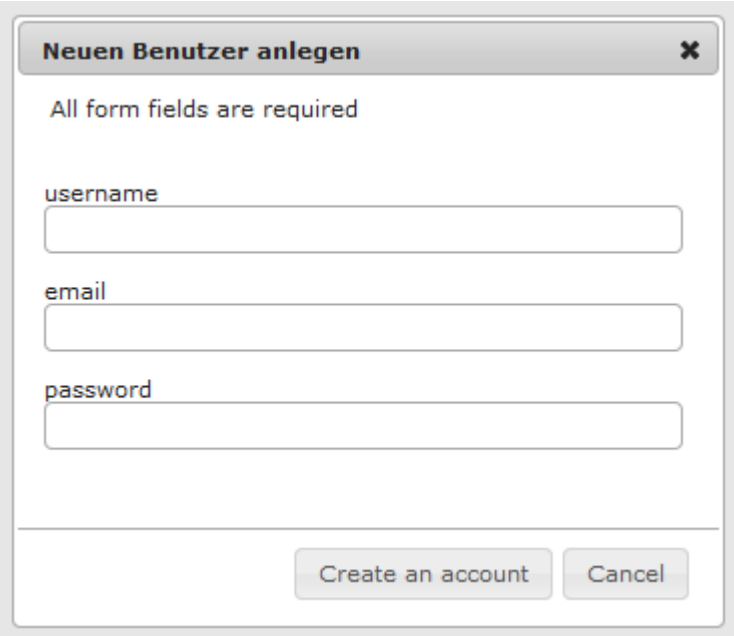

Abbildung 4: Dialog bei dem die verwendeten Grafiken sichtbar sind

Benutzerinnen und Benutzer die einen Browser, oder assistierende Technologien verwenden, welche nicht mit Grafiken arbeiten können sehen die Möglichkeit den Dialog dort zu schließen nicht. Die folgende Abbildung veranschaulicht diese Tatsache. Dabei wurden sämtliche Grafiken im Browser unterdrückt bzw. mit dem Text des alt-Attributes ersetzt.

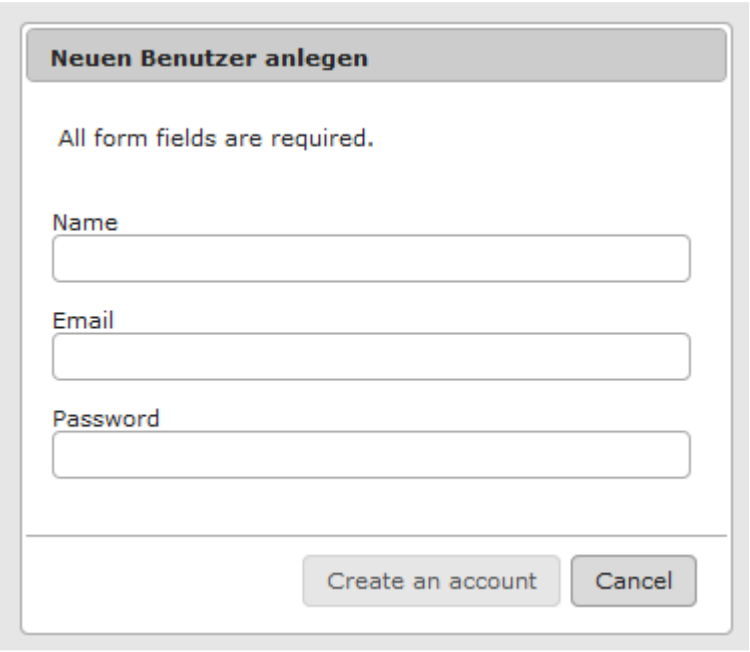

Abbildung 5: Dialog bei dem Grafiken durch alt - Attribute ersetzt wurden

Richtlinie 1.2 behandelt zeitbasierte Medien und ist, wie auch schon für den Datepicker, nicht ausschlaggebend.
Wählt man ein „einfacheres Layout“, wie es bei Richtlinie 1.3 gefordert ist, so hält die Seite trotzdem die Struktur bei. Der Dialog wird innerhalb der Seite ein- bzw. ausgeblendet, der Fokus steht nach dem Öffnen des Dialogs direkt im ersten Eingabefeld.

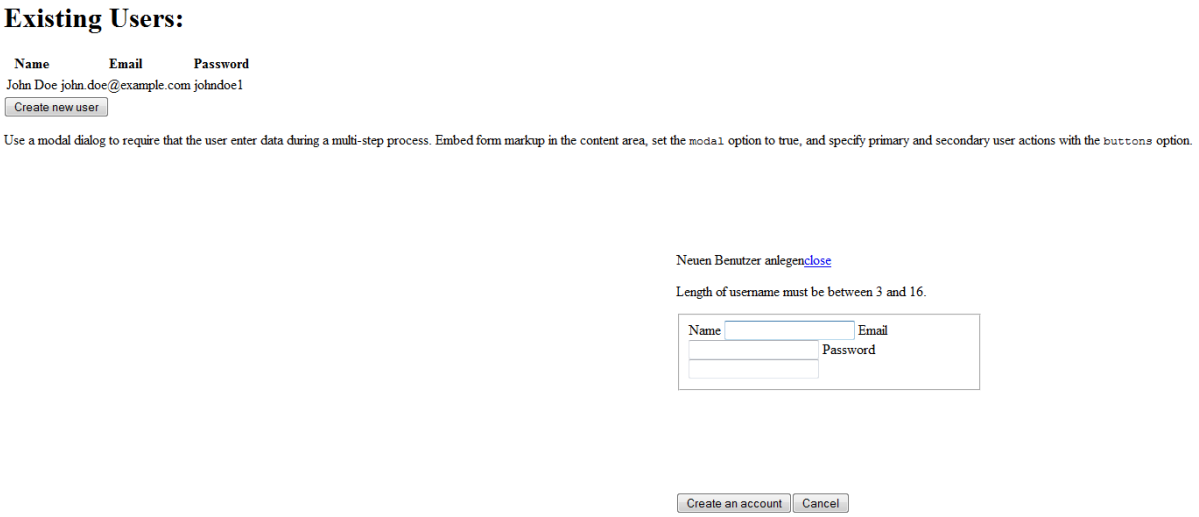

Abbildung 6: Dialog bei deaktivierten Stylesheets

Um das Erreichen der Stufe AA der Zugänglichkeit zu gewährleisten ist gefordert, dass die visuelle Darstellung von Text und Bildern die Text enthalten ein Kontrastverhältnis von mindestens 4,5:1 hat. Getestet wurde die Textfarbe (#555555) von „Create an account“ auf dem hellgrauen Grund (#e6e6e6). Diese hat ein Kontrastverhältnis von 6:1 – auch die Farbänderung im fokussierten Zustand entspricht den Vorgaben (#dadada : #212121 = 11,5 : 1). Die Textänderung um 200 Prozent – ohne Verlust von Inhalt oder Funktionalität - ist gewährleistet. Bei kleineren Displays kann das Problem auftreten, dass durch Verwenden der Zoom Funktion im Browser das Dialog Feld aus dem Fenster praktisch rausrutscht. Durch entsprechendes Scrollen, kann aber trotzdem damit gearbeitet werden.

3.2.2 Prinzip 2: Bedienbar

Die wichtigste Richtlinie dieses Prinzips besagt, dass alle Funktionalitäten des Inhalts durch eine Tastaturschnittstelle bedienbar sein müssen.
Grundsätzlich ist der Dialog und dessen Inhalte gut mit der Tastatur bedienbar mit der Ausnahme, dass die Möglichkeit zum Ändern der Größe und zum Verschieben des Dialogs nur mit der Maus zu bedienen ist.
Richtlinie 2.2 ist nur teilweise erfüllt. Das Ein- und Ausblenden des Dialogs kann zwar ausschließlich vom Benutzer gesteuert werden, somit hat er bzw. sie ausreichend Zeit um die Inhalte zu lesen und zu benutzen. Eine automatische Aktualisierung ist aber insofern vorhanden, dass die Validierung der Eingabe eine automatische Aktualisierung des Textes über den Eingabefeldern hervorruft. Näheres dazu folgt im nächsten Abschnitt.
Richtlinie 2.3 ist auch wie beim Datepicker irrelevant, Richtlinie 2.4 muss genauer betrachtet werden. Das Umgehen von Blöcken ist nicht ausschlaggebend, weil im Dialog nichts wiederholt wird. Der Seitentitel ist ausreichend gut deklariert, die Fokus-Reihenfolge entspricht größtenteils der natürlichen Vorgehensweise. Ein Problem besteht beim Schließen des Dialogs, da danach, nicht erkennbar ist welches Element fokussiert ist. Fakt ist, dass nach dem Schließen des Dialogs das gleiche Element den Fokus erhalten soll, welches auch vor dem Öffnen des Dialogs den Fokus hatte – nämlich der Button zum Öffnen.
Links und Überschriften (Titel des Dialogs) können vom Screenreader gelesen werden – das wurde mit JAWs getestet.

Problematisch ist 2.4.7 – die Sichtbarkeit des Focus ist nicht überall gegeben. Der Button zum Anzeigen des Dialogs hat genauso wenig einen gut sichtbaren Fokus, wie die beiden Buttons im Dialog selbst.

3.2.3 Prinzip 3: Verständlich

Die in der ersten und zweiten Richtlinien beschrieben Notwendigkeiten sind erfüllt. Die Sprache der Seite ist im einleitenden html-Tag ausgezeichnet, der Kontext wird nicht abhängig des Fokus eines bestimmten Elements geändert. Konsistenz von Navigation (praktisch nicht vorhanden) und Bestandteile mit der gleichen Funktionalität ist gegeben. Die Fehlererkennung welche in Richtlinie 3.3 beschrieben ist, wird nicht erfüllt. Es wird zwar der Fehler erkannt und dem Benutzer, der Benutzerin mitgeteilt jedoch nicht so, dass jeder und jede diese Mitteilung erfassen kann.
Bei der Verwendung einer assistierenden Technologie z.B. des JAWs Screenreaders merkt der User nicht, dass er eine falsche Eingabe vorgenommen hat. Der Fokus wird zwar auf das entsprechende Textfeld gesetzt, jedoch wird die Fehlermeldung nicht vorgelesen, weil diese im Absatz oberhalb angezeigt wird, und der Screenreader keinen Zusammenhang zwischen dem Textfeld und dem Absatz herstellen kann. Siehe folgende Abbildung.

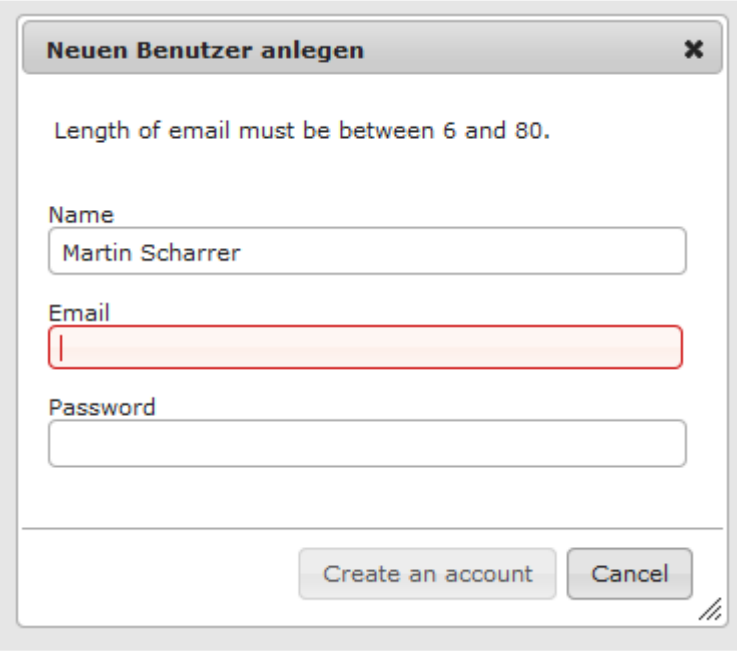

Abbildung 7: Eine falsche Eingabe bewirkt eine automatische Content-Aktualisierung, Farbe wird als einziges visuelles Mittel benutzt

Die Art der Fehlerdarstellung widerspricht auch teilweise einem Unterpunkt von Richtlinie 1.4.1. Diese besagt, dass Farbe nicht als einziges visuelles Mittel benutzt werden darf, um Informationen zu vermitteln. Dem könnte man entgegenhalten, dass dafür eine Textalternative lt. Richtlinie 1.1 zur Verfügung steht – die aber wenig hilft, wenn der Screenreader diese nicht erkennen bzw. lesen kann.

3.2.4 Prinzip 4: Robust

Die Richtlinie 4.1 besagt erstens, dass eine Syntaxanalyse erfolgreich abgeschlossen werden muss und zweitens, dass Name, Rolle und Wert von allen Bestandteilen der Benutzerschnittstelle durch Software bestimmt werden kann. Bei beiden Punkte besteht

Verbesserungsbedarf. Obwohl die Syntaxanalyse lt. [8] erfolgreich abgeschlossen wird, ist nicht sichergestellt, dass von allen Elementen der Wert bestimmt werden kann. Auch besagt die Richtlinie, dass Benachrichtigungen über Änderungen an Elementen, den Benutzeragenten, einschließlich assistierender Techniken, zur Verfügung steht. Genau das ist hier ein Problem. Erstens ist es mit einem Screenreader nicht möglich, den ersten Absatz im Dialog zu lesen. Das ist jener Absatz, in dem auch mögliche Mitteilungen über Eingabefehler dem Benutzer und der Benutzerin mitgeteilt werden. D.h. auch über diese Wert – Änderung des ersten Absatzes wird der Benutzer nicht informiert.

4 Verbesserungsvorschläge und Workarounds zu jQuery UI

4.1 Datepicker

4.1.1 Lösungen zum Prinzip 1: Wahrnehmbar

Bei dem Beispiel fehlt, wie oben beschrieben, der Label-Tag des Eingabefeldes, in welches ein entsprechendes Datum eingegeben werden muss. Wird hier ein Screenreader verwendet so kann dieser keinen Bezug zwischen Textfeld und Beschriftung herstellen, weil in dem Beispiel das folgendermaßen ausgeführt ist.

```
<p>Date: <input type="text" id="datepicker"></p>
```

Um zwischen „Date:“ und dem Textfeld einen Bezug herstellen zu können muss der Label-Tag verwendet werden.

```
<label for="datepicker">Date:</label><input type="text" name="appointmentdate"
id="datepicker" />
```

Die Pfeile um in das nächste bzw. vorherige Monat zu navigieren, können nicht den Focus erhalten. Die Pfeile sind zwar als Link ausgeführt, jedoch fehlt bei den Links das href – Attribut.
Um das Problem zu lösen, fügt man bei den Link – Tags das href – Attribut hinzu – mit einem Anker auf sich selbst. jquery.ui.datepicker.js Zeile 1394 und Zeile 1403

```
'<a href="#" class="ui-datepicker-prev ui-corner-all" onclick="DP_jQuery_' +
dpuuid +
```

Der Titel für die beiden Links kann dem Datepicker beim Aufruf als Argument übergeben werden. Dabei empfiehlt es sich, als Titel das Monat und das Jahr zu wählen, welches durch auswählen des Links angezeigt wird – also das vorherige bzw. das nachfolgende Monat des im Moment angezeigten. Das erreicht man durch folgende Parameter

```
navigationAsDateFormat: true,
prevText: 'MM yy',
nextText: 'MM yy'
```

Dadurch wird das Titel Attribute mit z.b. „September 2010“ gesetzt.

4.1.2 Lösungen zum Prinzip 2: Bedienbar

Das Icon um den Datepicker zu öffnen, darf nicht rein als Grafik im <img /> Tag ausgeführt sein. Grafiken können grundsätzlich nicht, wie in 3.1.2 beschrieben, fokussiert werden. Das Problem löst man, indem man das Icon als Button ausführt. Das lässt sich durch den Parameter buttonImageOnly erreichen.

```
buttonImage: 'images/calendar.gif',
buttonImageOnly: false
```

Ein weiteres in 3.1.2 beschriebenes Problem ist die Tatsache, dass der Datepicker sofort wieder geschlossen wird, sobald nach dem Öffnen eine Taste gedrückt wird. Das ist insbesondere problematisch, wenn der User als Eingabeinterface nur die Tastatur verwenden kann – durch Drücken der TAB – Taste wird der Datepicker geschlossen – und nicht wie angenommen das nächste Element fokussiert.
Folgender Lösungsweg wurde gewählt:

In jquery.ui.datepicker.js werden die Zeile 499 – 501 auskommentiert.

```
case 9:      $.datepicker._hideDatepicker();
             handled = false;
             break; // hide on tab out
```

Dadurch wird der Datepicker nur mehr durch das Auswählen eines Datums oder durch Drücken des close-Buttons geschlossen.

Eine weitere Verbesserung wird erreicht, indem man Monat und Jahr zusätzlich, neben der Navigation zum nächsten und vorherigen Monat, in Select-Boxen auswählen kann. Das erreich man indem die folgenden Parameter bei der Instanziierung verwendet werden.

```
changeMonth: true,
changeYear: true
```

Diese Tatsache bringt wieder das Problem mit sich, dass für diese beiden Elemente (Select-Box für Monat und Select-Box für die Jahrauswahl) kein Label-Tag verwendet wird.

In jquery.ui.datepicker.js, Zeile 1535 wurde deshalb das Label für den Monat hinzugefügt.

```
monthHtml += '<label for="monthSelect'+inst.id+'" class="ui-helper-
hidden">Month</label>'
```

Die dazu notwendige ID setzt sich aus einem Schlüsselwort (monthSelect, yearSelect) und aus jener ID die der Anwender übergibt (die ID des Textfeldes selbst). Dadurch ist sichergestellt – bzw. ist der Programmierer dafür verantwortlich, dass eine ID nicht doppelt in einer Seite vorkommt. Als Klasse wird ui-helper-hidden verwendet (definiert in ./themes/base/jquery.ui.core.css Zeile 13) – dadurch werden die Labels nicht angezeigt, können aber trotzdem von einem Screenreader gelesen werden.

In Zeile jquery.ui.datepicker.js, 1567 wurde das Label für das Jahr hinzugefügt

```
yearHtml += '<label for="yearSelect'+inst.id+'" class="ui-helper-
hidden">Year</label>'
```

Um den Datepicker auch ohne Datumsauswahl, z.b. bei einem optionalem Textfeld schließen zu können, wurde das Button-Panel eingeblendet. Dabei werden 2 Buttons angezeigt. Die folgenden Attribute wurden hinzugefügt:

```
showButtonPanel: true,
closeText: 'close',
currentText: 'dd.mm.yy'
```

Um den Datepicker auch einfach (nach dem öffnen) schließen zu können wurde ein Tastenkürzel für den close Button definiert.

In Zeile 1411 beim <button> Tag wurde das Attribute accesskey="c" hinzugefügt. Im Firefox kann das Tastenkürzel mit Alt-Shift-c angesprochen werden.

```
var controls = (!inst.inline ? '<button type="button" accesskey="c" …
```

4.1.3 Lösungen zum Prinzip 3: Verständlich

Der in 3.1.3 beschriebene Sachverhalt lässt sich insofern lösen, dass abhängig von der im Browser als bevorzugt festgelegte Sprache, die entsprechende Bibliothek geladen wird. Bibliotheken in unterschiedlichen Sprachen findet man auf der jQuery Website [3].
Die Tatsache, dass für die Navigation keine eindeutigen Titel verwendet wurden, wurde bereits in Abschnitt 5.1.1 gelöst.

4.1.4 Lösungen zum Prinzip 4: Robust

Das Prinzip besagt, dass eine Syntaxanalyse erfolgreich abgeschlossen werden muss. Dass ist, abhängig vom DOCTYPE, möglich oder auch nicht. Jedenfalls fällt auf, dass bei der Tabelle, welche die einzelnen Tage des ausgewählten Monats darstellt, einige wichtige Strukturinformationen fehlen. Wichtig im Bezug auf die Verwendung von assistierende Technologien. Bei der Verwendung eines Screenreaders, z.B. JAWS sind folgende Verbesserungen unumgänglich.

In der Zeile 1454 wurde das summary Attribute für die Tabelle hinzugefügt.

```
summary="In the table you can select a day of '+monthNames[drawMonth]+'
'+drawYear+'"
```

Diese Summary enthält immer das aktuell ausgewählte Monat / Jahr. Aktualisierung erfolgt automatisch. In der Zeile 1455 wurde die Caption (Tabellenüberschrift) für die Tabelle hinzugefügt.

```
'<caption class="ui-helper-hidden">Select the date from this table. You can choose
a day of '+monthNames[drawMonth]+' '+drawYear+'</caption>
```

Die Summary kann genauso wie die Caption durch eine Tastenkombination vom Screenreader gelesen werden (Shift + T).

Der zweite wichtige Punkt in der einzigen Richtlinie dieses Prinzips ist jener, dass sowohl Name, Rolle als auch der Wert von Bestandteilen der Benutzerschnittstelle durch Software bestimmt werden können muss. Das ist nicht der Fall, da ein Screenreader nicht den Zusammenhang zwischen Datenzelle und Headerzelle herstellen kann. Anders formuliert, kann nicht festgestellt werden, zu welchem Wochentag die Zahl eines bestimmten Tages gehört.
Das löst man, indem man einen Gültigkeitsbereich der Tabellen Header festlegt.
In der Zeile 1460 wurde beim <th> Tag das Attribute scope="col" hinzugefügt – der Tabellenheader mit den Wochentagen gilt somit für jeweils die Spalte. Eine genauere Erläuterung findet man in [7].

```
thead += '<th' + ((dow + firstDay + 6) % 7 >= 5 ? ' scope="col" class="ui-
datepicker-week-end"' : '') + '>' +
```

Wird bei der Instanzierung des Datepickers auch die Kalenderwoche mit Attribute showWeek: true eingeblendet, so muss auch diese Zelle als Kopfzelle referenziert werden. D.h. eine Datenzelle (ein Tag) hat nun zwei Kopfzellen – den Wochentag und die Kalenderwoche in welcher dieser Tag liegt.

Dazu muss in Zeile 1457 beim <th> Tag das Attribute scope="row" hinzugefügt werden – der Tabellenheader gilt für die Zeile.

```
var thead = (showWeek ? '<th class="ui-datepicker-week-col" scope="row">' +
this._get(inst, 'weekHeader') + '</th>' : '');
```

4.2 Dialog

Zur Lösung und zum Verständnis einiger Probleme im Zusammenhang mit dem Dialog Feature wurden insbesondere die jQuery UI Dokumentation zum Dialog und zu grundsätzlichen Teile von jQuery sowie die Quellen [9] bis [12] verwendet bzw. genauer betrachtet.

4.2.1 Lösungen zum Prinzip 1: Wahrnehmbar

Das Problem bei der Wahrnehmbarkeit des Icons zum Schließen des Dialogs kann gelöst werden, indem das Stylesheet angepasst wird. Durch

```
text-ident: -9999px
```

in jquery.ui.core.css Zeile 34, wird der Text unsichtbar. Weiters muss die Verwendung der Grafik zum Erzeugen des Icons unterdrückt werden. Dazu kann das Stylsheet /demos/accessibility.css verwendet werden.
Das ist nur notwendig, wenn man nicht mit Grafiken arbeiten möchte. Assistierende Technologien wie der hier verwendete Screenreader von JAWs können das Icon zum Schließen sehr wohl interpretieren.
Was hier noch fehlt, ist das title – Attribute beim Link zum Schließen des Dialogs. Das wurde in jquery.ui.dialog.js Zeile 124 hinzugefügt.

```
.attr('title', options.closeText)
```

4.2.2 Lösungen zum Prinzip 2: Bedienbar

Das Problem, dass sowohl die Größe des Dialogs, als auch die Position im Fenster nur mit der Maus geändert werden können, wird bei dieser Anwendung außen vor gelassen, da diese Möglichkeiten nicht für die hier betrachtete, tatsächliche Funktion notwendig sind.
Um sicherzustellen, dass es zu keinen möglichen Fehlinterpretationen oder hier nicht beschriebenen Zuständen – insbesondere bei der Verwendung von assistierenden Technologien kommt, können die beiden Funktionen deaktiviert werden.
Dazu müssen die beiden Parameter resizeable und draggable mit false initialisiert werden. Siehe modal-form.html Zeile 90 und 91.
Um den Fokus nach dem Schließen des Dialogs wieder auf den Button zu setzten, welcher den Dialog geöffnet hatte, wurde eine Callback Funktion implementiert, welche beim Auftreten eines Close-Events aufgerufen wird. Dadurch wird die in Punkt 2.4.3 geforderte natürliche Fokus-Reihenfolge gewährleistet. Siehe modal-form.html Zeile 140:

```
$( "#dialog-form" ).dialog({
   close: function(event, ui) { $("#create-user").focus(); }
});
```

Die Lösung zum Unterpunkt 2.4.7 wurde auch in accessibility.css implementiert. Wie beim Icon zum Schließen des Dialogs, wird einfach für die Buttons eine Outline angezeigt sobald diese den Fokus erhalten.

```
button.ui-state-focus:focus{
      outline:dotted 1px #000000;
}
```

4.2.3 Lösungen zum Prinzip 3: Verständlich

Das ausschlaggebendste Problem dieses Prinzips besteht darin, dass der Benutzer und die Benutzerin nicht über Eingabefehler informiert werden.
Die Lösung liegt darin, dass Fehlermeldungen genau dort angezeigt werden, wo auch der Fehler aufgetreten ist – und nicht in einem umliegenden Absatz, welcher von der assistierenden Technologie nicht einem entsprechendem Eingabefeld zugeordnet werden kann. Dazu muss die Validierung der Eingaben in modal-form.html angepasst werden.
Die bereits bestehende Funktion updateTips(t) zeigt, den Text im Parameter t im Absatz oberhalb der Eingabefelder an.
Angelehnt an diese Funktion wurde eine weitere implementiert updateLabel(o,t) welche zum Objekt o, das dazugehörige Label ermittelt, und den Label-Text mit dem Text im Parameter t ersetzt.
Die neue Funktion wird jeweils nach updateTips(t) aufgerufen, sofern ein Fehler erkannt wurde. Wird kein Fehler erkannt, so wird die ursprüngliche Beschriftung der Eingabefelder wieder hergestellt (Zeile 62, bzw. Zeile 82).
Dazu muss bei der Funktion checkRegexp(o,n,regexp,t) der Parameter n hinzugefügt werden, welcher auch bei der Funktion checkLength(o,n,min,max) zum Einsatz kommt und die ursprüngliche Beschriftung für das Eingabefeld beinhaltet. Natürliche müssen auch die Aufrufe der Funktion entsprechend angepasst werden, Zeilen 109, 111 und 112.

Um sicherzustellen, dass nach einer falschen Eingabe der Text der Fehlermeldung auch vom Screenreader vorgelesen wird, muss noch der Focus auf jenes Element gesetzt werden, in welchem der Fehler aufgetreten ist. Das erledigt o.focus() in Zeile 59, bzw. Zeile 79.
Die folgende Abbildung zeigt die Reaktion auf einen Eingabefehler.

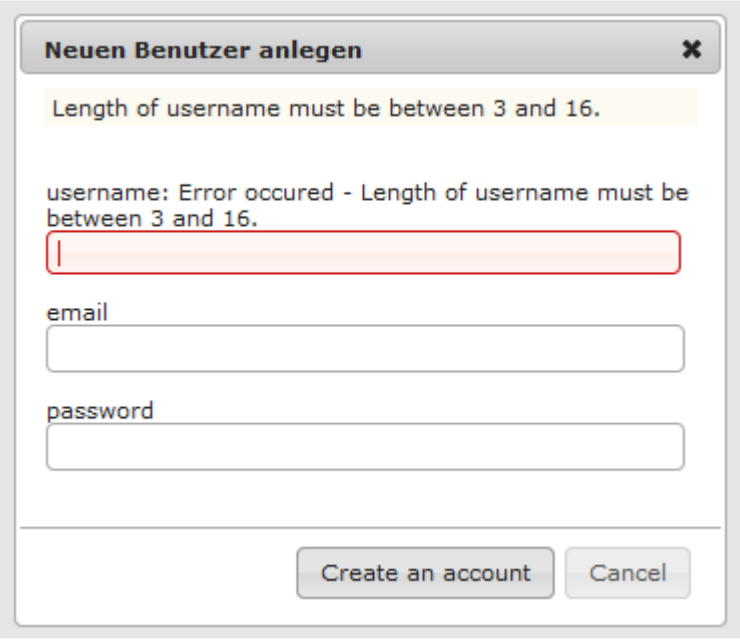

Abbildung 8: Anzeige eines Eingabefehlers im dazugehörigen Label

4.2.4 Lösungen zum Prinzip 4: Robust

Um das Problem zu lösen, dass der Screenreader den ersten Absatz nicht vorließt wurden mehrere Möglichkeiten versucht zu implementieren und auch getestet. Die folgenden Absätze beschreiben die Idee hinter den einzelnen Implementierungen sowie die notwendigen Änderungen und die Testergebnisse.

Ausgehend von der Tatsache, dass ein Screenreader immer genau jene Texte, Beschriftungen, etc. vorliest die momentan fokussiert sind, wurde der erste Lösungsansatz entwickelt. Eine Möglichkeit ist, den Beschreibungstext in einer nur lesbaren Textarea anzuzeigen, anstatt in einem Absatz. Das Textarea bekommt automatisch beim Durchgehen des Formulars mit der Tab – Taste den Fokus und der Screenreader kann den Inhalt vorlesen. Um auch für den sehenden Benutzer den Text ansprechend darzustellen, bzw. so wie in einem Absatz darzustellen wurde die Textarea entsprechend formatiert, siehe accessibility.css. Damit auch der Screenreader den Text automatisch vorließt, muss entweder ein Label mit dem gleichen Text wie in der Textarea hinzugefügt werden, oder der Titel der Textarea mit dem gleichen Text entsprechend mit dem title Attribut gesetzt werden. Das Label wird mit Hilfe der ui-helper-hidden Klasse ausgeblendet.

```
<label for="additionalInfo" class="ui-helper-hidden">All form fields are
required</label>
<textarea id="additionalInfo" readonly="readonly" class="paragraphTextarea"
rows="1">All form fields are required</textarea>
```

Die folgende Abbildung zeigt den Dialog mit einer Textarea als Ersatz für einen Absatz.

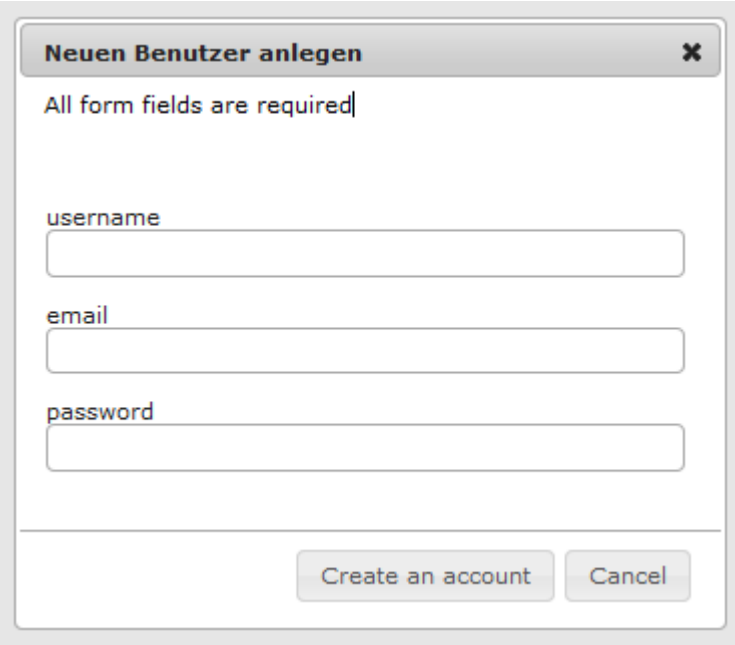

Abbildung 9: Dialog mit einer Textarea anstatt einem Absatz als Tag für den Beschreibungstext

Wie aus der Abbildung zu erkennen ist, sieht man den Cursor hinter dem letzten Wort stehen. Ein weiterer Nachteil dieser Variante ist, dass der Screenreader die Textarea natürlich als Eingabefeld erkennt und das auch verdeutlicht („Geben Sie Text ein“), obwohl die Möglichkeit gar nicht besteht, Text einzugeben (das readonly Attribut wurde verwendet) – und das auch nicht so gewollt ist.

Deshalb wurde versucht, mit dem Paragraph Tag weiter zu arbeiten. Die erste Idee war, auf den Absatz auch mit dem tabindex Attribut den Fokus zu setzen.
Ähnliches wird ja auch mit dem Dialogfeld an sich gemacht, bei welchem der Screenreader ja auch den Titel vorlesen kann - Siehe Zeile 92 bis 94 in jquery.ui.dialog.js. Dabei wird ermöglicht, dass der gesamte Block den Fokus erhalten kann. Mit dem Hinzufügen des tabindex Attributes, und des title Attributes -

```
<p class="validateTips" tabindex="0" title="All form fields are required.">All
form fields are required.</p>
```

wird nach dem Öffnen des Dialogs sowohl der Dialogtitel, als auch der Beschreibungstext vorgelesen. Der Absatz erhält in der natürlichen Reihenfolge den Fokus, und wird praktisch wie ein Formularelement behandelt.

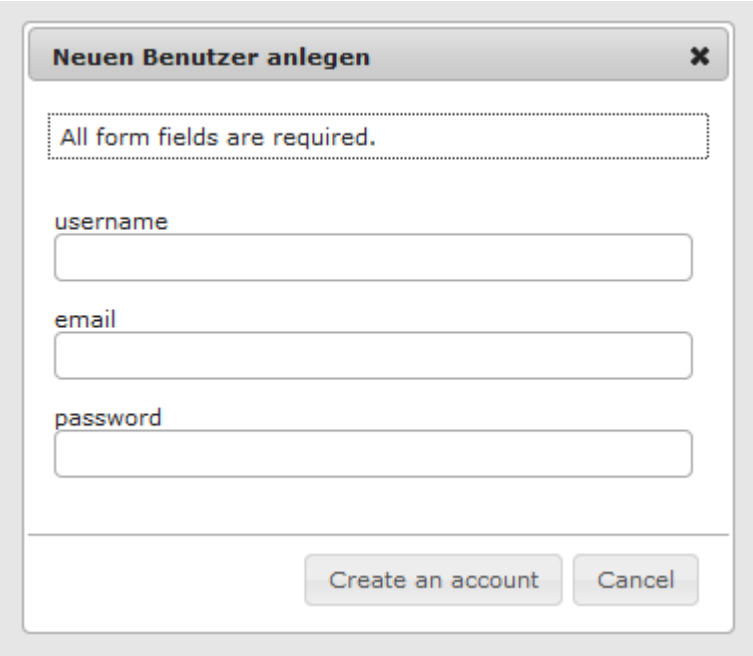

Abbildung 10: Dialog mit Beschreibungstext der fokussiert werden kann

Der Nachteil an dieser Lösung ist, dass das Absatz Element redundante Daten enthält, da sowohl das title Attribute als auch der Tag an sich mit dem gleichen Inhalt ausgefüllt werden müssen. Dadurch können z.B. bei einer Aktualisierung durch die Eingabe-Validierung leicht Inkonsistenzen entstehen.
Des weiteren interpretiert der Screenreader das Absatz Element als Eingabefeld und weist den Benutzer darauf hin, dass er respektive sie doch bitte Text eingeben möchte – obwohl die Möglichkeit dazu gar nicht besteht.

Um dieses Problem zu lösen, wurde nochmal die Variante genauer betrachtet, welche bei dem Titel des Dialogs an sich zu Einsatz kommt. Dort wird durch einmaliges setzten des title Attribute beim alles umgebenden Block Element beides erzielt. Erstens wird der Titel im Element angezeigt und zweitens kann der Screenreader diesen Titel auch Vorlesen, wobei – wenn auch nur augenscheinlich – der sichtbare Titel nie den Focus erhält.
Wie das funktioniert kann folgendermaßen erklärt werden. Erstens bekommt der Block wie vorher schon beschrieben doch einmal den Fokus, der Titel aber nicht. Den Focus an sich erhält der Dialog beim öffnen – in moveToTop, Zeile 294.
Damit nun die Verbindung zwischen Block und Titel des Blocks hergestellt werden kann wird die Kombination der Attribute role und aria-labelledby verwendet – jquery.ui.dialog.js Zeile 102 bis 105.

```
.attr({
            role: 'dialog',
            'aria-labelledby': titleId
})
```

Das role Attribute wurde in XHTML 2 geschaffen um unter anderen diesen assistieren Technologien Informationen über die Semantik vermitteln zu können. Näheres dazu liest man am besten in [5], [13] und [15] nach. Das aria-labelledby Attribute gibt die ID des Labels zu dem jeweiligen Element an. Näheres dazu findet man in [14] und [15].

Im konkreten Fall wurde nun genau diese Technik auch für die Anzeige der Beschreibung bzw. Statusinformation verwendet. Als ersten Schritt ersetzt man den Absatz – Tag mit einem Block Tag. Den gewünschten Text schreibt man in das title Attribute – das sieht dann folgendermaßen aus:

```
<div id="dialog-form" title="Neuen Benutzer anlegen">

	<div id="tips" class="validateTips" title="All form fields are required">
	</div>

	<form action="#">
	<fieldset>
		<label for="name">username</label>
		<input type="text" name="name" id="name" class="text ui-widget-
		content ui-corner-all" />
		<label for="email">email</label>
		<input type="text" name="email" id="email" value="" class="text ui-
		widget-content ui-corner-all" />
		<label for="password">password</label>
		<input type="password" name="password" id="password" value=""
		class="text ui-widget-content ui-corner-all" />
	</fieldset>
	</form>

</div>
```

In jquery.ui.dialog.js betrachten wir die neu eingefügten Zeilen 160 bis 173.

```
var	tips = $("#tips");
	tipsTitle = tips.attr('title');

	tips.attr({
			role: 'note',
			'aria-labelledby': 'tips-title'
		})
		.attr('tabIndex', 0)
		.removeAttr('title');

	uiTipsTitle = $('<span></span>')
		.attr('id', 'tips-title')
		.html(tipsTitle)
		.prependTo(tips);
```

Als erstes wird der Titel vom entsprechenden Block-Element gelesen. Anschließend wird festgelegt, welches Element den Block beschreibt, und welche Rolle dieses Element hat – siehe dazu [16].
Damit das Block –Element welches die Beschreibung enthält fokussiert werden kann, wird der tabindex auf 0 gesetzt. Somit wird auch erreicht, dass nach dem Öffnen des Dialogs zuerst der Titel (tabindex=-1) und anschließend die Beschreibung vorgelesen wird. Das Entfernen des title Attributes kann und muss deshalb erfolgen, weil es anschließend ohnehin in einem span Tag dem Beschreibungsblock hinzugefügt wird, und vor allem auch deshalb, dass keine Redundanzen auftreten und damit den im vorherigen Absatz beschriebenen möglichen Inkonsistenzen vorgebeugt wird.
Der letzte Codeabschnitt fügt den schon angesprochenen span Tag hinzu, setzt den entsprechenden Titel, den anzuzeigenden Text und fügt den Tag dem Block Element hinzu.

Durch die zuvor festgelegte Abhängigkeit zwischen Block Element und span Element – mit Hilfe von role und aria-labelledby kann nun trotzdem der Screenreader den Inhalt des Blockelements vorlesen – ohne dass der Benutzer auf ein nicht vorhandenes Textfeld hingewiesen wird.
Mit dieser Lösung sind somit alle vorher aufgetretenen Probleme ausgeräumt. Der Beschreibungstext muss im Code nur einmal angegeben werden, wird auch nur einmal gespeichert. Er kann sowohl von Screenreader als auch von sehenden Benutzern gelesen werden. Weiters gliedert sich der Text in die natürliche Fokusreihenfolge des Formulars ein und ist somit fixer, bedienbarer Bestandteil des Dialogs.

5 Quellenverzeichnis

[1] Vitaly Friedman: Praxisbuch Web 2.0: Moderne Webseiten programmieren und gestalten; Galileo Press Bonn 2008 http://yatil.de/Artikel/ajax-und-zugaenglichkeit
[2] jQuery UI Project and jQuery UI Team – official Website http://jqueryui.com/
[3] jQuery Project – official Website http://jquery.org/
[4] Web Content Accessibility Guidlines (WCAG 2.0) – World Wide Web Consortium (W3C), autorisierte deutsche Übersetzung http://www.w3.org/Translations/WCAG20-de/
[5] Accessible Rich Internet Applications (WAI-ARIA) 1.0 – World Wide Web Consortium (W3C), http://www.w3.org/TR/wai-aria/
[6] Accessibility Color Wheel http://gmazzocato.altervista.org/colorwheel/wheel.php–
[7] Kopfzellen in Zellen von HTML Tabellen referenzieren – Selfhtml.org http://de.selfhtml.org/html/tabellen/nicht_visuell.htm#referenzen
[8] http://validator.w3.org/check
[9] http://www.geedew.com/2010/02/25/jquery-ui-dialog-accessibility/
[10] Posting in google Groups: http://groups.google.com/group/free-aria/browse_thread/thread/cedc98b7f1351772?pli=1
[11] http://old.nabble.com/jQuery-Modal-Dialog-boxes-td28090174.html
[12] http://www.mail-archive.com/fluid-talk@fluidproject.org/msg00450.html
[13] http://www.w3.org/TR/xhtml-role/
[14] http://www.filamentgroup.com/lab/update_jquery_ui_slider_from_a_select_element_now_with_aria_support/
[15] http://www.w3.org/TR/wai-aria/roles
[16] http://www.w3.org/TR/xhtml-role/#s_role_module_attributes