



smart help

Jan Oevermann (46594)

Projektarbeit an der
Hochschule Karlsruhe
WS 2013/14

Inhaltsverzeichnis

Einführung	4
Einordnung	5
Geführte Fehlersuche	5
Situation	6
Zielgruppe	7
Technisches Konzept	8
XML-basiertes Format	8
Entwicklungswerzeuge	8
Workflow	9
Erstellung	9
Qualitätssicherung	10
Laufzeitumgebung	11
Anforderungen	11
Umsetzung	12
API	13
Smartphone-App	14
Aufbau	14
Gestaltung	14
Gesten	15
Spezialfunktionen	16
Website Widget	20
Einführung	20
Fazit	21
Ausblick	21
Anhang	22
Kurzeinführung tr3	22
Vergleich Bordliteratur und Baum	23
Quellen	24

Einführung

Abstract

Gerade bei der Bordliteratur von Fahrzeugen sind die Benutzer oft mit der Fülle an angebotenen Informationen überfordert. Oft wird die Dokumentation nur dann konsultiert, wenn bereits eine Fehlfunktion vorliegt, die im schlimmsten Fall die Fahrsicherheit des Nutzers beeinträchtigt. Um in solchen Situationen schnell Abhilfe zu schaffen und den Nutzer zur richtigen Problembehandlung zu lenken eignet sich das Verfahren der Geführten Fehlersuch bzw. der geführten Hilfe bei der dem Nutzer auf Basis eines Entscheidungsbaums nacheinander ein Reihe von einfachen Fragen gestellt werden, die das Problem immer weiter eingrenzen bis schließlich eine passende Problemlösung angeboten werden kann.

Moderne Smartphones stellen hierfür die ideale Plattform dar, da sie „immer dabei“ sind und auch im mobilen Umfeld die Möglichkeit bieten, multimediale und interaktive Inhalte darzustellen. Des weiteren kann auf die zahlreichen Zusatzfunktionen zurückgegriffen werden, wie etwa den Zugriff auf Position, Kamera oder Telefon um den Nutzer auf dem schnellsten Weg zu lenken ohne die eigentlich Hilfe-Anwendung verlassen zu müssen.

Ziel

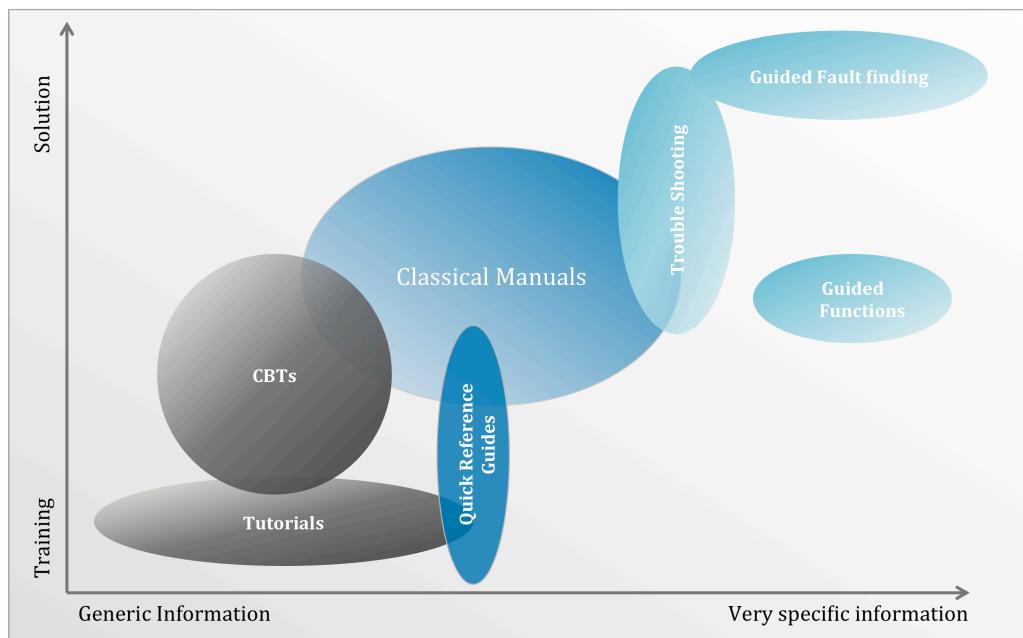
Das Ziel der vorliegenden Arbeit ist die Konzeption und Entwicklung einer webbasierten App, die den Nutzer mit dem Mittel der geführten Fehlersuche auf Basis eines Entscheidungsbaums durch eine Pannensituation führt und ihm auf dem schnellsten Weg Abhilfe bzw. eine einschätzende Diagnose des Problems schaffen kann. Das Ergebnis soll sich in Optik und User Experience am Markenbild smart orientieren und sowohl Mittel der Technischen Dokumentation als auch Marketinginstrument sein.

Einordnung

Geführte Fehlersuche

Die *Geführte Fehlersuche* ist eine relativ junge Disziplin in der Technischen Dokumentation, obwohl die Methodik von Entscheidungsbäumen im Bereich der Informatik schon seit Jahrzehnten erforscht ist. Im Umfeld anderer Übermittlungsarten lässt sich die Geführte Fehlersuche als sehr lösungsorientiertes und spezifisches Instrument einordnen (siehe Abbildung)

Classification of Guided Fault Finding



1 Classification of Guided Fault Finding (Quelle: arvato TI GmbH / H. Nitsche)

Die Geführte Fehlersuche kann ihre Stärken in den Bereichen einsetzen, in denen auf Basis eines komplexen Fehlerbildes schnell und zuverlässig eine Diagnose des Problems herbeizuführen ist und das behandelte Thema überwiegend technischer Natur ist.

Durch den Einsatz einer Geführten Fehlersuche können Hersteller Kosten z.B. bei Service und Logistik einsparen und die Kundenzufriedenheit erhöhen. So liegt ein großer Interessenschwerpunkt in der Vermeidung von falsch positiven Reparatureinsendungen, die durch den Kunden hätten selbst behoben werden können (z.B. durch Neustart des Geräts o.Ä.). Klassische Anleitungen werden hier oft nicht konsultiert oder auf Grund einer falschen Diagnose nicht richtig interpretiert. Ein weiterer Anwendungsfall ist der Einsatz in Call-Center oder POS (Point-of-Sale) Situationen in denen Mitarbeiter im Umgang mit Kunden geleitet werden, was einen standardisierten Ablauf des Gesprächs gewährleistet und die Einarbeitungszeit verkürzt.

Situation

Der spezielle Einsatzzweck der Geführten Fehlersuche bei einer Pannenhilfe greift an den oben aufgeführten Vorteilen an. Allerdings sind in dieser ungewöhnlichen Situation weitere Besonderheiten zu beachten.

Anforderungen an die Methodik:

- Das Bordhandbuch ist oft zu umfangreich. Die gesuchten Informationen sind darin zwar enthalten, werden aber nicht gefunden. Allerdings kann darauf weiter verwiesen werden wenn die Problemdiagnose gestellt wurde (z.B. Problem erkannt: *Reifen defekt* -> Bordhandbuch zum Wechsel konsultieren). Dadurch wird bestehender Content / bestehende Medien weiterverwendet und die „Content-Fülle“ im Entscheidungsbaum selbst reduziert
- Das zugrundeliegende Problem ist für Laien oft schwer einzugrenzen. Defekte können an sehr vielen Komponenten in unterschiedlichen Schweregraden auftreten. Wichtig an dieser Stelle ist, dass der Nutzer die Problemdiagnose nicht exakt stellen müssen muss, sondern nur die entsprechenden Anweisungen zum weiteren Vorgehen benötigt (z.B. *weiterfahren* vs. *Pannendienst rufen*)
- Mehrdimensionale Entscheidungen sind mit reinem Text schwer abzubilden. So können z.B. Displayleuchten mehrfach belegt sein, je nach Kontext oder in Kombination mit einem anderen Symbol verschiedene Bedeutungen zu haben. Diese mehrfachen Abhängigkeiten sind in rein linearem Textfluss oder in Tabellenform schwer abzubilden.

Anforderungen für die Aufbereitung:

- Die Benutzer befinden sich bei der Benutzung in einer enormen Stresssituation. Die Hauptbedürfnisse sind ein schnelles Eingrenzen des Problems und klare Anweisungen zum weiteren Vorgehen. Der Fokus einer Anwendung muss deshalb schnell und zuverlässig funktionieren.
- Texte müssen eindeutig, kurz und prägnant sein (vgl. Hamburger Verständlichkeitsmodell) um von den Nutzern sofort verstanden zu werden. Bilder und Grafiken können eine schnelle Informationsaufnahme hier unterstützen. Das Smartphone selbst kann ebenfalls zur Nutzerunterstützung eingesetzt werden (vgl. Spezialfunktionen wie Abstandsmesser oder Anrufinitialisierung).
- Bei der Gestaltung der Bedienoberfläche muss der Fokus auf Inhalt und guter Usability liegen. Das bedeutet konkret: große Schriftgröße, große, voneinander abgegrenzte Buttons, hoher Kontrast, klare Bereichsabgrenzung.

Anforderungen an Technische Umsetzung

- In ländlichen Gegenden ist die Netzabdeckung oft unzuverlässig. Deswegen muss die Größe der zu übermittelnden Daten so klein wie möglich gehalten werden (kompaktes Speicherformat, *Lazy Loading*, Vektorgrafiken, etc.)
- Zusatzcontent, der zur eigentlichen Knotenfrage bzw. -anweisung eingeblendet wird sollte unabhängig von der eigentlichen Baumlogik gelagert, geladen und angezeigt werden. Dadurch wird gewährleistet, dass der Entscheidungsbaum und damit die Laufzeitumgebung bzw. die Smartphone-App weiterhin bedienbar bleiben, auch wenn ein dynamisches Nachladen dieser Inhalte fehlschlägt (z.B. auf Grund eines Funklochs)

Zielgruppe

Als Auto, das sich in Gestaltung und Werbung von anderen Kleinfahrzeugen abhebt, spricht die Marke smart eine bestimmte Zielgruppe an. Angepasst auf diese Zielgruppe soll auch die Smartphone-App das Markengefühl weitertragen und ihre Charakteristik berücksichtigen. Im Folgenden sind einige Aussagen zur Zielgruppe zusammengefasst:

- 68 Prozent Frauen vs. 32 Prozent Männer favorisieren das Zweisitzer-Modell. (a)
- Smartfahrer wollen ihr Leben aktiv und individuell gestalten. (a)
- Mehrheit der Zielgruppe begreift das Auto als Gebrauchsgegenstand. (a)
- Fast 60 Prozent besitzen noch ein weiteres Fahrzeug. (a)
- Die Zielgruppe legt Wert darauf, dass es [das Auto] zuverlässig funktioniert. (a)
- Vor Allem Junggebliebene sollen angesprochen werden. (a)
- Anteil der Jungen ist überdurchschnittlich hoch. (b)
- Erlebnisorientiert und dynamisch. (b)
- Als Berufstätige fast täglich mit ihrem Auto unterwegs. (a)

Die oben stehenden Aussagen wurden aus den folgenden Markenanalysen entnommen:

- a) RIEKE, Jennifer: „Markenanalyse »smart«“, Seminararbeit an der FH Düsseldorf, SS 2009
- b) ROLAND BERGER STRATEGY CONSULTANTS / BURDA COMMUNITY NETWORK:
„Smart: So stilorientiert ist die Zielgruppe“ in Absatzwirtschaft 04/2009, S. 33

Aus den untersuchten Zielgruppenanalysen lässt sich ableiten, dass die überwiegend junge und weibliche Zielgruppe vor allem Wert auf Zuverlässigkeit legt und täglich mit dem Auto unterwegs ist. Das mehrheitliche Zweitfahrzeug wird dabei vor allem als Gebrauchsgegenstand betrachtet.

Durch diese Zielgruppen-Charakteristik lässt sich auf einen Bedarf im Bereich der geführten Pannenhilfe schließen, da diese sowohl das Bedürfnis nach Zuverlässigkeit (schneller Wieder-verfügbarkeit) und Dynamik befriedigt.

Technisches Konzept

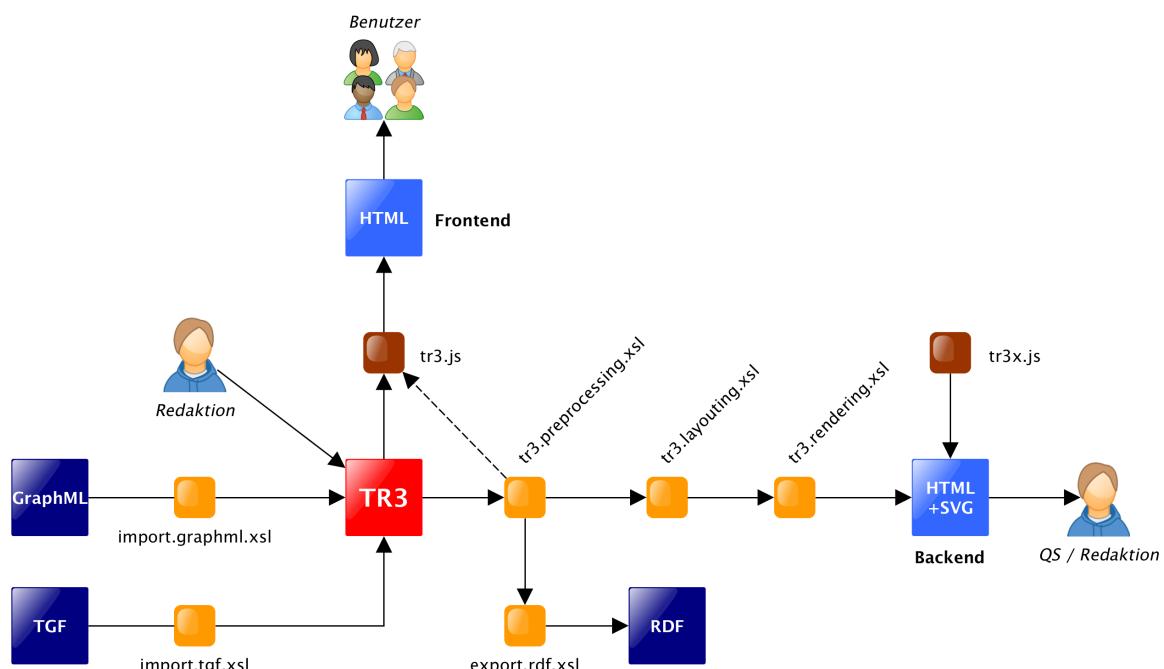
XML-basiertes Format

In einem verknüpften Projekt der Vorlesung *XML-basiertes Informations- und Content-Management* bei Prof. Dr. Ziegler wurden die Grundlagen für den Einsatz XML-basierter Entscheidungsbäume erarbeitet. Als eines der Ergebnisse wurde das Format *tr3* spezifiziert, das es erlaubt Entscheidungsbäume für den Einsatz in Applikationen in einem kompakten und leicht zu verarbeitenden XML-Format zu speichern. Die Spezifikation wird als XSD- und DTD-Schema zur Verfügung gestellt.



Entwicklungswerkzeuge

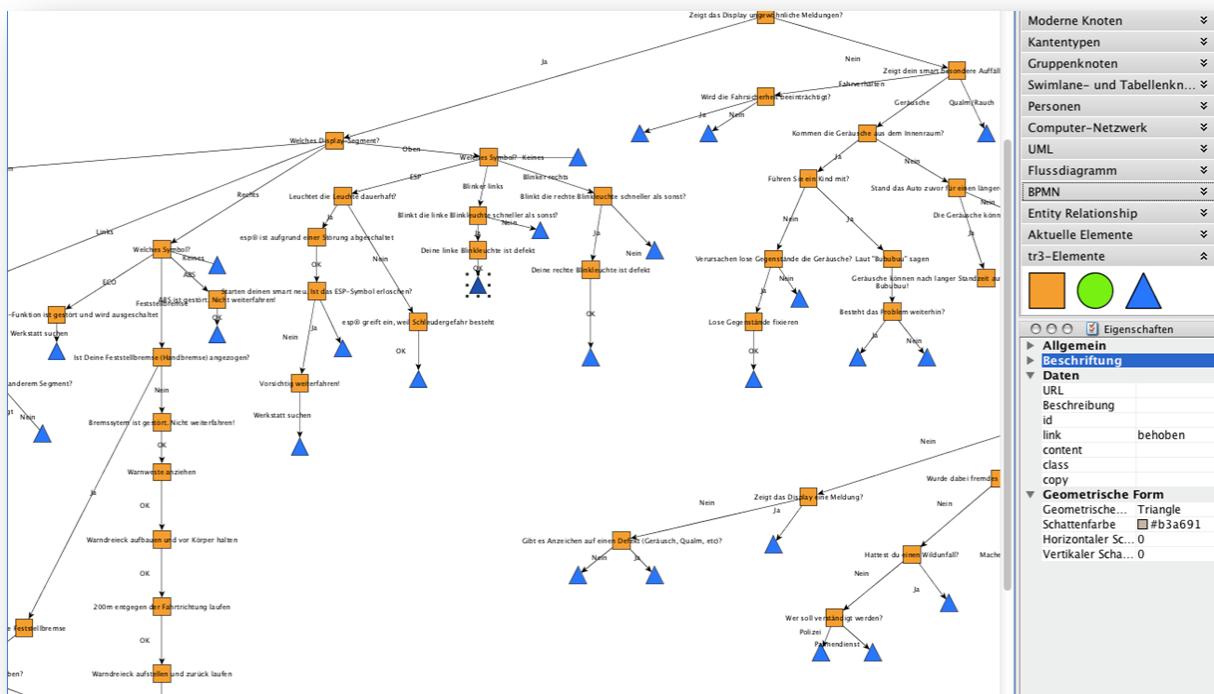
Für den Erstellungsprozess des Formates werden einige Entwicklungswerkzeuge zur Verfügung gestellt. Darunter Anpassungen für einen grafischen Editor (*yED*), Import- und Export-Möglichkeiten für andere Formate und eine Testumgebung zur Qualitätssicherung (*tr3x*). Dazu Tools zur Fehleranalyse, Validierung und Vorverarbeitung des XML-Formats (siehe Abbildung)



Workflow

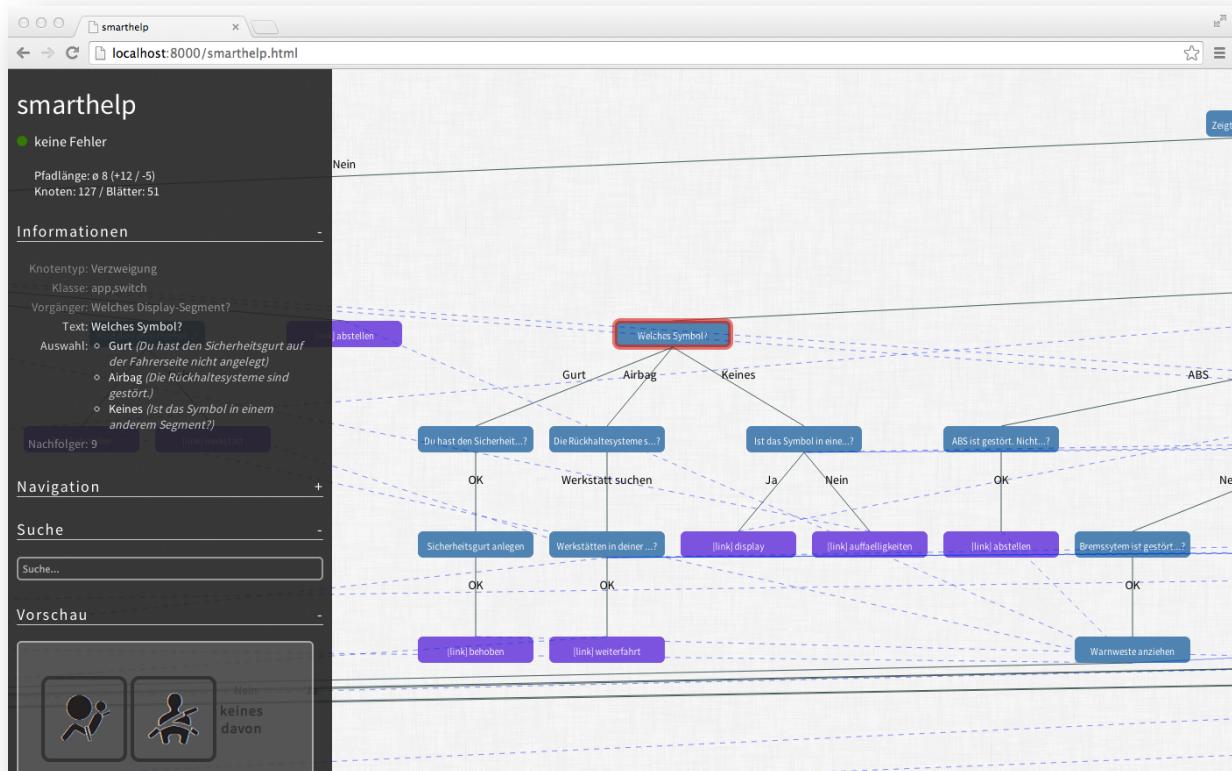
Erstellung

Üblicherweise erfolgt die Erstellung eines Entscheidungsbaumes in einem grafischen Editor. Dafür bietet sich die kostenlose Software *yED* an, deren natives Speicherformat *GraphML* in *tr3* umgewandelt werden kann. Beim Erstellungsprozess gelten lediglich für den Einsatz verschiedener Knotenformen Konventionen (Rechteck für normale Knoten, Dreieck für Links, etc). Im Bezug auf andere gestalterische Mittel (z.B. Farbe) hat der Autor eine freie Auswahl.



Qualitätssicherung

Um dem Autor bei der Erstellung komplexer Bäume ein Werkzeug zur Qualitätssicherung zur Hand zu geben, wurde eine Testumgebung zum Prüfen und Konstruieren von tr3-Bäumen entwickelt. Darin kann der Autor durch den Baum navigieren, Teilbäume aus- und zuklappen, eine Volltextsuche über den Baum durchführen und einzelne Knoten genauer untersuchen. Verknüpfter Inhalt kann in einer Vorschau betrachtet werden. Zusätzlich werden Baumfehler angezeigt und Statistiken über den Baum berechnet.



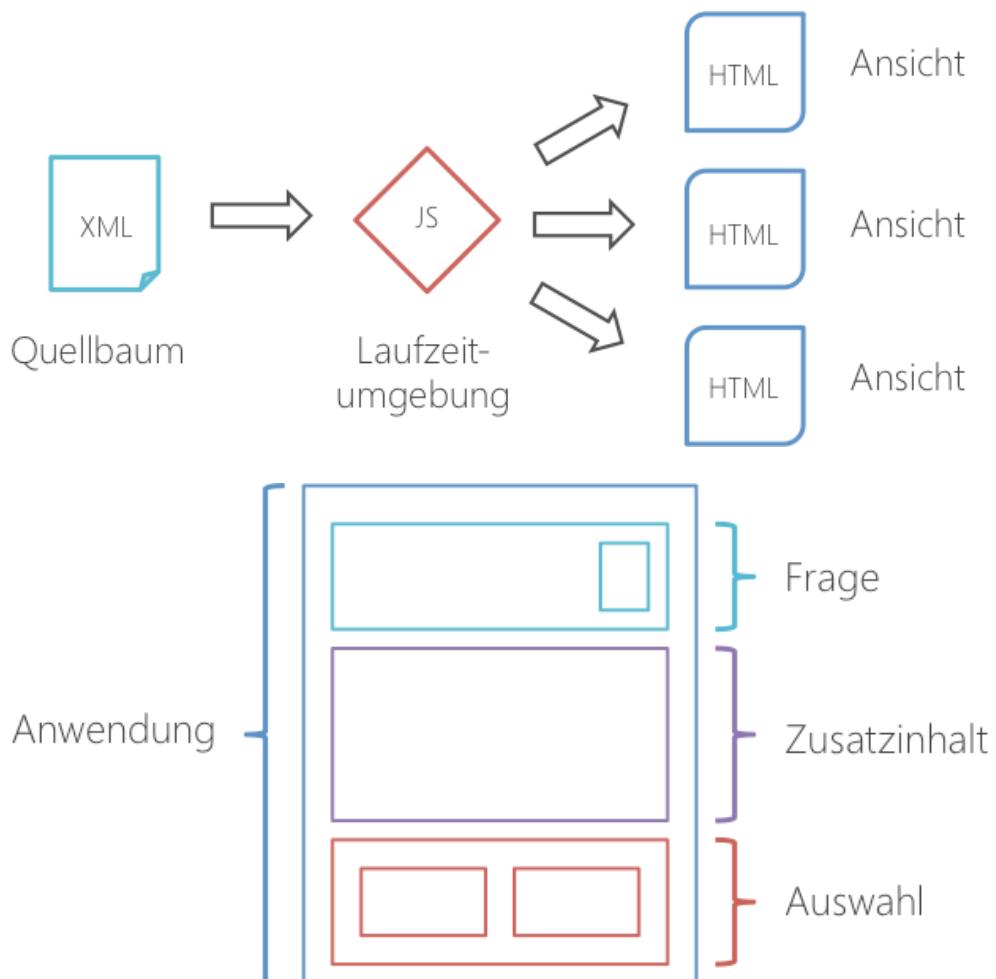
Laufzeitumgebung

Anforderungen

Um unabhängig von Gestaltung und Art der späteren Implementierung (Website, App, Anwendung) zu sein, muss die Laufzeitumgebung beliebige Bäume im tr3-Format verarbeiten können. Das heißt, zum einen den internen Verlauf und Weg durch den Baum zu verwalten als auch den externen Verlauf durch das Rendern von Einzelansichten (Fragen/Antworten) übernehmen.

Stichpunktartig zusammengefasst lassen sich die Anforderungen an die Umgebung wie dargestellt formulieren:

- Rendern der Einzelansichten
- Dynamisches Nachladen von externen Inhalten
- Bedienung (Zurück, Start, anderer Baum)
- Kommunikation mit Rahmenanwendung
- Auflösen von Referenzierungen (Links)
- Prüfen von Gültigkeiten/Parametern
- Tracking



Umsetzung

Die Laufzeitumgebung wurde als JavaScript-Bibliothek umgesetzt, die als einzelne Datei (tr3.js) in eine beliebige HTML-Seite eingebettet werden kann. Die Funktionen der Bibliothek bauen dabei auf der weit verbreiteten JavaScript-Bibliothek jQuery auf. Die Einbindung kann entsprechend so erfolgen:

```
<script src="pfad/zu/jquery.js"></script>
<script src="pfad/zu/tr3.js"></script>
```

Angesteuert werden kann die Laufzeitumgebung über eine Objektbasierte API (siehe Abschnitt API), die öffentliche Eigenschaften (Konfiguration und Informationen) und Methoden (Funktionen) zur Verfügung stellt. Für den Standardbetrieb reicht eine einmalige Initialisierung allerdings aus. Diese erfolgt mit dem Befehl:

```
tr3.init('pfad/zu/baum.xml');
```

Innerhalb der HTML-Seite der Anwendung sucht die Laufzeitumgebung nach einem Element mit der Klasse **tr3**. Innerhalb dieses Elements werden die einzelnen Ansichten gerendert. Alle von der Laufzeitumgebung erzeugten Elemente besitzen semantische Klassen mit dem Präfix **tr3-** (z.B. **tr3-question**, **tr3-answer**, usw.). Damit lassen sie sich gekapselt von der Rahmenanwendung gestalten und Namenskonflikte werden vermieden.

Die Kommunikation mit der Rahmenanwendung (also z.B. der umgebenden App) erfolgt über native JavaScript *events*. Diese werden immer dann am tr3-Element ausgelöst, wenn ein bestimmter Anwendungszustand erreicht ist (z.B. Content fertig geladen). Die Rahmenanwendung kann diese abfangen und entsprechend reagieren. Diese event-Handler können mit jQuery definiert werden. So kann beim Aufruf der Seite zunächst abgewartet werden bis der Baum vollständig geladen ist um danach die App zu starten oder einen Ladebildschirm auszublenden.

```
tr3.init('.tr3').^on('tr3-ready', function() {
    smarthelp.init();
});
```

Es ist möglich die Laufzeitumgebung mit selbstdefinierten Optionen aufzurufen und damit z.B. die Art der gerenderten Elemente oder das Verhalten in mobilen Anwendungen zu steuern. Diese Optionen werden in einem zusätzlichen Funktionsparameter beim Aufruf als Objekt mitgegeben. Ein Aufruf der Anwendung mit Optionen kann z.B. so aussehen:

```
tr3.init ('pfad/zu/baum.xml', {
    trigger: 'ontouchend',
    data: ['smartphone']
});
```

API

Übersicht

Die Laufzeitumgebung stellt eine objektbasierte JavaScript-Schnittstelle zur Verfügung, die von anderen Skripten oder über die Browser-Konsole durch das tr3-Objekt angesteuert werden kann.

Eigenschaften

Eigenschaft	liefert	Beschreibung
tr3.data	Array	Enthält alle Baum-Parameter, die bei der Verarbeitung berücksichtigt werden (in der Regel die beim Start übergebenen Parameter)
tr3.doc	String	Enthält den relativen Pfad zum Quelldokument (XML)
tr3.gui	Objekt	Enthält für eine Zuordnung der zu erzeugenden HTML-Elemente (z.B. <i>div</i>) für die semantischen GUI-Elemente (z.B. <i>question</i>). Kann beim Aufruf mitgegeben werden,
tr3.history	Array	Enthält eine Auflistung aller bisherigen Knoten ohne Sprünge zurück (für Auswertung durch <i>back</i> -Funktion)
tr3.log	Array	Enthält eine Auflistung aller bisherigen Knoten mit Sprüngen zurück (für Auswertung durch ein Analyse-Backend)
tr3.options	Objekt	Enthält die beim Aufruf übergebenen Optionen als tr3-Optionen-Objekt
tr3.root	Node	Enthält den Wurzelknoten als DOM-Node
tr3.trigger	String	Enthält das <i>trigger</i> -Event für die Bedienung (z.B. <i>click</i> für Desktop und <i>ontouchstart</i> für Mobile)
tr3.version	String	Enthält die Versionsbezeichnung der aktuellen tr3-Instanz

Methoden

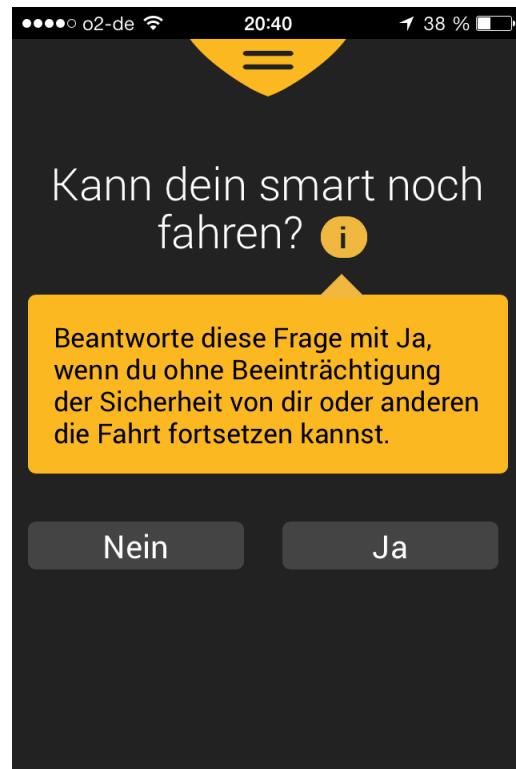
Methode	Parameter	Beschreibung
tr3.back	-	Springt in der bisherigen Abfolge einen Schritt (bzw. Knoten) zurück
tr3.build	Node	Erzeugt die Elemente der Oberfläche, bindet Events, erzeugt Hilfsobjekte und initialisiert das Rendering des ersten Knotens
tr3.init	Quelle, Optionen	Initialisiert die Laufzeitumgebung des übergebenen Baums (<i>Quelle</i>) mit den übergebenen Optionen
tr3.render	Node	Rendert die Ansicht für den übergebenen Knoten entsprechend der eingestellten Optionen und Knoteneigenschaften. Initialisiert ggf. das Nachladen externen Contents.
tr3.send	Node	Plazhalter-Funktion, zum Verschicken der Trackingdaten (tr3.log), die aufgerufen wird, nachdem ein Blattknoten erreicht wurde. Kann durch eine externe Funktion ersetzt werden (für die Anbindung an ein Backend)

Smartphone-App

Aufbau

Die Smartphone-App ist als webbasierte Single-Page-Application aufgebaut. Das bedeutet, dass sie auf einer Haupt-HTML-Seite basiert, in die zusätzlicher Inhalt bei Bedarf nachgeladen und integriert werden kann. Diese Web-Anwendung kann entweder direkt per Browser aufgerufen werden oder z.B. auf Geräten mit iOS-Betriebssystem als *Safari Web App* mit einem Icon auf dem Home-Screen des Geräts geöffnet werden (siehe Abschnitt *Safari Web App*). Dadurch kann die Website durch einen Klick im Vollbildmodus geöffnet werden und der umliegende Browser ist nicht mehr sichtbar.

Durch den direkten Zugriff auf Internetinhalte können laufend Updates eingespielt werden, die allen Benutzern einer solchen Web App ohne weiteres Zutun (wie z.B. einem manuellen Update) sofort zur Verfügung stehen.



Das Design der App ist gekapselt in einem separaten Stylesheet (*smartheight.css*) definiert, dass über Klassen mit dem Präfix *sh* die Elemente gestaltet. Funktionalität, die auf Scripting zurückgreift, wird in der JavaScript-Datei *smartheight.js* definiert. Dies betrifft hauptsächlich das Event-Binding (Touch-Gesten), das Starten der App und der *tr3*-Laufzeitumgebung sowie die Kommunikation mit der Laufzeitumgebung über Events.

Gestaltung



Das Design der App orientiert sich am Corporate Design der Marke *smart* und übernimmt auch deren Leitlinien hinsichtlich Einfachheit und Klarheit. Fragen und die dazugehörigen Antwortmöglichkeiten sind zentrale Gestaltungselemente, die groß und zentral auf dem Bildschirm platziert sind. Als farblichen Akzent ist ein Teil der *smart*-Bildmarke als Menü-Drawer (mit dem man das Menü „herunterziehen“ kann) am oberen Bildrand platziert.

Das Applikationsmenü selbst legt sich bei Aktivierung über den aktuell angezeigten Inhalt und bietet nur die nötigsten Auswahlmöglichkeiten. Dazu gehören die sogenannten Quicklinks, die Spezialfunktionen der App ohne den Weg über den Gesamtbaum schnell zugänglich machen und die Grundfunktionen der App bzw. der Laufzeitumgebung: Einen „Zurück“ sowie einen „Home“-Button. Auch eine Verlinkung der offiziellen *smart*-Website wurde in diesem Bereich vorgenommen.

Gesten

Bei der Steuerung der App wurde darauf geachtet, die Bedienung so intuitiv wie möglich zu gestalten. So „blättert“ ein Wischen mit einem Finge nach rechts eine Seite zurück, ein Ziehen des Menü-Drawers oder des Applikationskörpers nach unten lässt das Menü von oben hereinfallen und ein Fingertipp auf den dahinter liegenden Hintergrund lässt es wieder verschwinden. In eigenen Tests mit verschiedenen Personen konnte festgestellt werden, dass diese Bedienung der erwarteten entspricht und deshalb auch nicht vorher erklärt werden muss.

Zur technischen Implementierung der Gestenerkennung wurde die JavaScript-Bibliothek Hammer.js verwendet. Diese implementiert Endgeräteunabhängig die gewünschten Gesten und bietet entsprechende Eventhandler und –listener an, um diese anzufangen. Ein typisches Event-Binding mit Hammer.js ist im folgenden Beispiel dargestellt:

```
//swiperight on body
Hammer($('.tr3')[0], {swipe_velocity: 0.2}).on("swiperight", function() {
    tr3.back();
});
```

Zu beachten war bei der Entwicklung der Gestensteuerung auch das Browserverhalten bei mobilen Endgeräten. Prinzipiell sind die meisten für Desktop-Webbrowser entwickelten Anwendungen auch auf mobilen Endgeräten ohne Einschränkungen lauffähig, da der click-Event, das viele Websites als Auslöser für Buttons verwenden auf mobilen Endgeräten umreferenziert wird auf einen „Single Tap“, also das einzelne Antippen eines Elements auf dem Touchscreen. Da bei mobilen Browsern der „Double Tap“ das Vergrößern der aktuellen Seite bewirkt, muss das System nach einem ersten Antippen abwarten, ob ein zweiter folgt um zwischen Single und Double Tap unterscheiden zu können. Dieser Zeitabstand beträgt 300ms, die in der normalen Anwendung deutlich spürbar sind und eine nicht angepasste Anwendung träge erscheinen lassen. Referenziert man die Event-Handler der Applikation um auf das Event „ontouchend“, also das Loslassen des ersten Taps, kann ein deutlich beschleunigtes Anwendungsgefühl erreicht werden.

Spezialfunktionen

Um alle Vorteile und Funktionen von modernen Smartphones auszunutzen, wurden moderne HTML5- bzw. JavaScript-APIs verwendet, die den Zugriff auf spezielle Features (Lokalisierung, Kamerazugriff, etc.) der Geräte zulassen. Die damit implementierten Spezialinhalte der App werden im Folgenden aufgeführt:

Abstandsmesser



Um dem Nutzer beim Aufstellen eines Warndreiecks zu unterstützen wurde ein GPS-basierter Abstandsmesser integriert, der über Positionsdaten den Abstand zwischen Start- und Zielpunkt anzeigt und bei erreichen der Zielvorgabe von 200m den Nutzer optisch und akustisch informiert.

Bewegt sich der Nutzer, ändert sich die Anzeige auf seinem Display in 0,5m-Schritten. Die Genauigkeit hängt hierbei stark von den äußeren Bedingungen ab. In Räumen oder bei stark bedeckten Verhältnissen kann es hier – wie bei allen Positionierungsanwendungen – zu Ungenauigkeiten kommen.

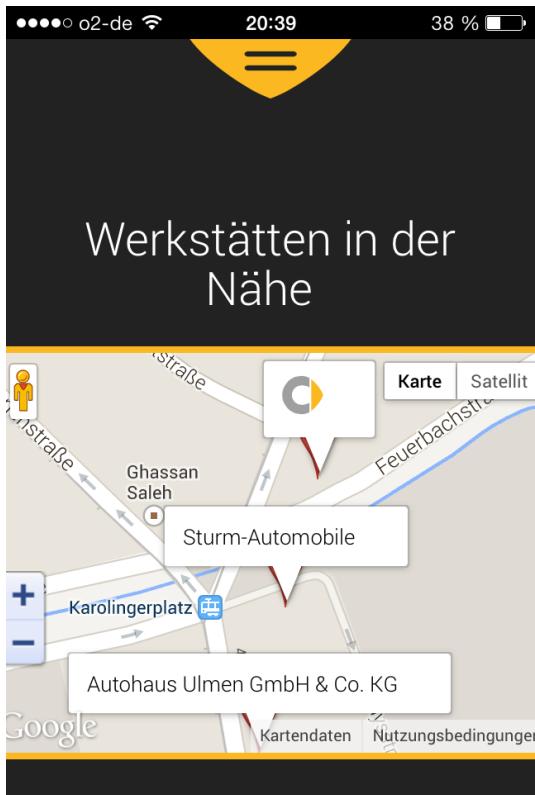
Dabei wird auf die **geolocation** API moderner Webbrowser zurückgegriffen, die auf die Positionierungsfunktionen der Host-Hardware zugreifen kann. Die Position kann so entweder über das verwendete WLAN-Netzwerk, die IP-Adresse (GeolP-Verfahren) oder die GPS-Position ermittelt werden.

```
navigator.geolocation.getCurrentPosition(function(position) {
    startPos = position;
}, function(error) {
    alert("Position kann nicht ermittelt werden");
});
```

Nach Ermittlung der Startposition wird bei jeder Veränderung der aktuellen Position geprüft ob bereits der berechnete Abstand zur Startposition (200m) erreicht ist. Ist dies der Fall wird der Nutzer durch Blinken und das Abspielen eines Tons darüber informiert.

```
navigator.geolocation.watchPosition(function(position) {
    // ...
    distance = calculateDistance(startPos.coords.latitude,
        startPos.coords.longitude, position.coords.latitude,
        position.coords.longitude);
    $("#userDistanceValue").html(distance); // show distance
    checkDistance(distance); // check if already reached target distance
});
```

Geo-Dienste



Neben der Abstandsmessung können dem Nutzer auf Basis der aktuellen Position auch weitere nützliche Informationen angeboten werden. So zum einen die Lage der nächsten Werkstätten, Tankstellen, Krankenhäuser oder Parkflächen. Als kategorisierte Datenbank und zum Rendering der aktuellen Karte wird Google Maps API verwendet.

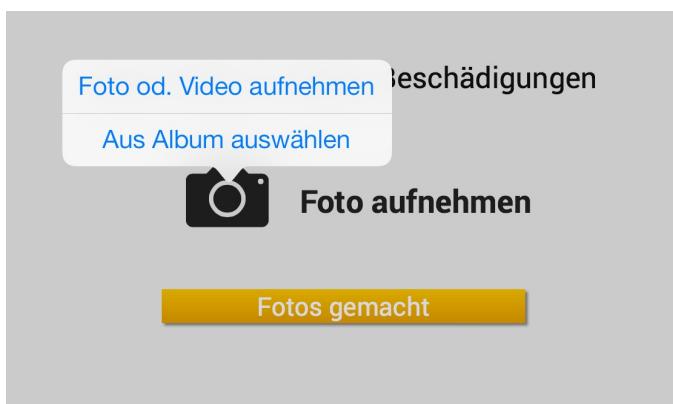
Diese API wird innerhalb des Zusatzcontents als JavaScript-Library über ein CDN (Content Delivery Network) eingebunden und verwendet. Nach dem Initialisieren der Kartenanwendung wird die aktuelle Position über die **geolocation** API des Browser ermittelt und anschließend eine Umgebungssuche in einem bestimmten Suchradius nach Kategorien (z.B. Werkstätten) durchgeführt. Die Ergebnisse werden mit Beschriftungen als „Pins“ auf die Karte gesetzt und die Karte an der ermittelten Position mit einer festgelegte Zoom-Stufe zentriert.

```
// defining the search request by location, radius and type of category
var request = {
  location: geolocate,
  radius: '500',
  types: ['car_repair']
};

// using google maps/places API for performing search
service = new google.maps.places.PlacesService(map);
service.nearbySearch(request, callback);

// center map at current position
map.setCenter(geolocate);
```

Kamera-Integration



Alle aktuellen Smartphones besitzen mindestens eine integrierte Kamera, die z.B. dazu verwendet werden kann, bei einer Kollision mit einem anderen Auto Beweisbilder zu machen. Durch eine neue HTML5-API ist dies mit Hilfe des input-Formularfelds möglich:

```
<input type="file"
       accept="image/*;capture=camera">
```



Dabei wird dem Benutzer nach einem Klick auf das Feld bzw. seine jeweilige Repräsentation angeboten ob er ein Bild aufnehmen oder ein bestehendes hochladen möchte. In beiden Fällen lassen sich durch den Einsatz der neuen JavaScript FileReader API die Inhalte des Bildes auslesen und in der App integriert als Vorschau anzeigen. Damit kann für jedes angefertigte oder hochgeladene Foto dynamische ein Thumbnail erzeugt werden. Dem Nutzer wird somit ein Überblick über die bisher vorhandenen Fotos gegeben.

Zum jetzigen Stand der App werden diese Fotos allerdings nicht weiterverwendet oder gespeichert, sie werden nach Beenden der App wieder verworfen.

Display-Symbole



Gerade beim Aufleuchten von unbekannten Symbolen im Fahrzeug-Display ist bei vielen Nutzern eine große Unsicherheit zu beobachten. Fahrzeughersteller versuchen dem entgegenzuwirken indem sie in die Bordliteratur Tabellen zur schnellen Deutung von Symbolen integrieren. Allerdings sind diese oft zu umfangreich und unübersichtlich und in Pannen- oder Gefahrensituationen ungeeignet als Problemlösungsinstrument. Oft sind Symbole in Ihrer Bedeutung auch doppelt belegt oder nur in Kombination mit anderen Symbolen gültig. Eine gezielte Fragestellung kann hier helfen diese mehrdimensionale Problematik schnell einzuschränken.

Gibt der Nutzer an, dass in seinem Fahrzeug-Display ein Symbol leuchtet, muss er zunächst auswählen in welchem Bereich des Displays das Symbol ist. Dadurch wird die Liste der in Frage kommenden Symbole stark eingeschränkt und übersichtlicher.

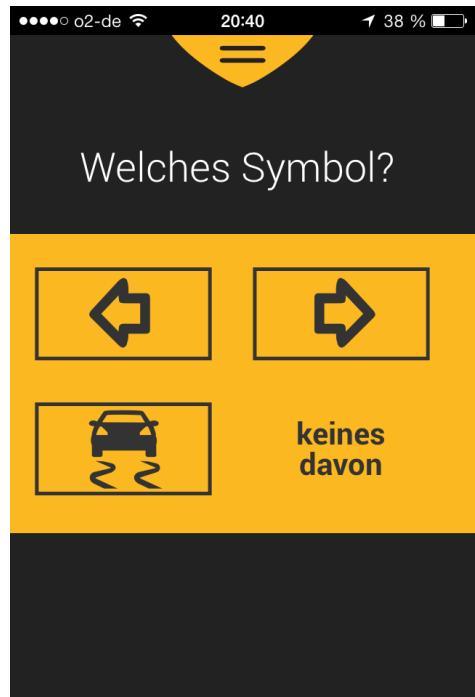
Realisiert wurde diese Darstellung durch die Erstellung eines fünfteiligen SVGs, dessen Bestandteile mit Funktionen belegt wurden, die wiederum die nächste Auswahl treffen. Dabei werden die gerenderten Elemente der Antworten ausgeblendet und die Auswahl der richtigen Antwort durch den Zusatzinhalt übernommen.

Nachdem das entsprechende „Panel“ gewählt wurde bekommt der Nutzer die Symbole angeboten, die in diesem Bereich aufleuchten können. Ist ein Symbol doppelt belegt oder wird er nach einer erstmaligen Auswahl weiter befragt (z.B. „leuchtet das Symbol dauerhaft?“).

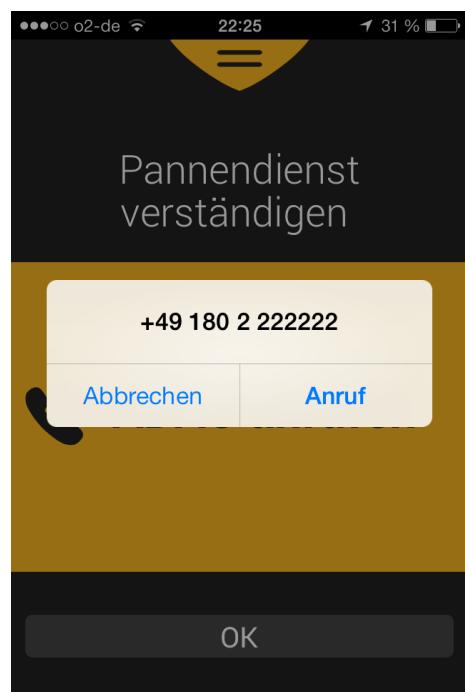
Um die Symbole verwenden zu können, die sowohl im Cockpit als auch in der Bordliteratur auftauchen wurde eine kleine CSS-Library (`mbsymbols.css`) entwickelt, die den Daimler-Symbolfont semantisch auf CSS-Klassen referenziert und den Einsatz als Icons erlaubt:

```
<i class="mb-icon mb-airbag"></i>
```

ergibt:



Anruf-Integration

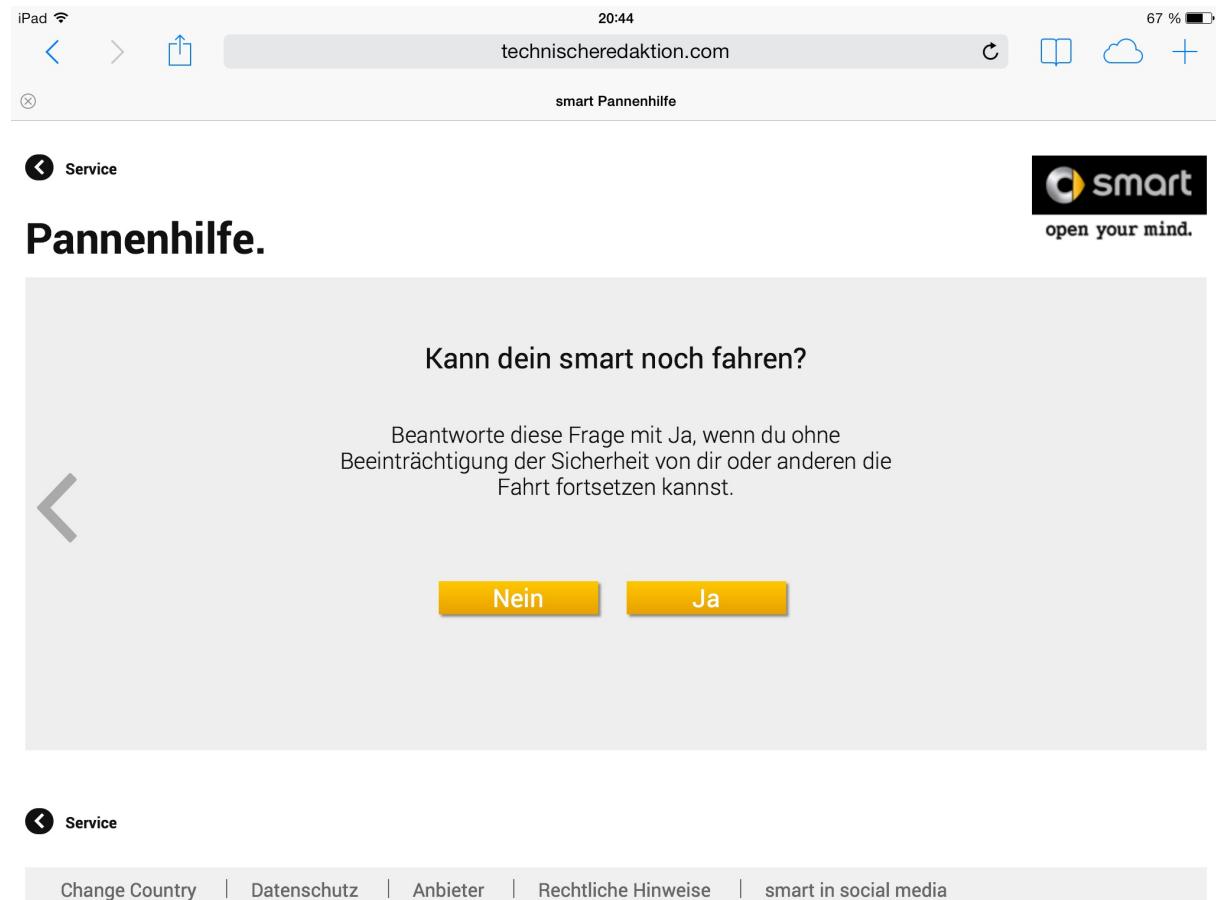


Tritt ein Problemfall ein, bei dem der Benutzer nicht mehr weiterfahren kann oder soll, muss je nach Lage ein Pannendienst oder die Polizei telefonisch verständigt werden. Um diesen Schritt zu erleichtern und direkt aus der App heraus ausführbar zu machen, kann auf eine Neuerung in HTML5 zurückgegriffen werden, die Telefonnummern als mögliches Protokoll für Hyperlinks zulässt. Bei Klick auf einen solchen Link wird der Nutzer gefragt ob diese Nummer tatsächlich angerufen werden soll. Bestätigt er das, beginnt der Wählvorgang.

Website Widget

Einführung

Um darzustellen, dass das Rendering durch die Laufzeitumgebung von der endgültigen Darstellung unabhängig ist, wurde der gleiche Entscheidungsbaum mit der Laufzeitumgebung als „Widget“ in ein Mock-up der aktuellen smart-Website integriert. Hierbei wurden die in der Webpräsenz verwendeten Gestaltungselemente wiederverwendet und so ein einheitliches Erscheinungsbild gewährleistet.



Zusätzlich wurden einige wenige applikationsspezifische Funktionen in einem separaten Skript integriert (z.B. der animierte Übergang beim Aufruf der geführten Hilfe): smartweb.js.

Durch diese zweite Anwendung, die mit sehr wenig Aufwand realisiert werden konnte, soll gezeigt werden, dass einmal entwickelte Entscheidungsbäume problemlos mit der Laufzeitumgebung wiederverwendet werden können. Lediglich Anpassungen an den Stylesheets der Rahmenanwendungen sind zu einer nahtlosen Integration nötig.

Fazit

Bei der Entwicklung der App und insbesondere der Laufzeitumgebung konnten viele Erfahrungen im Bezug auf Web-Technologien und Gestaltung von (mobilen) Anwendungen gesammelt werden. Dabei konnte bestehendes Wissen im Bereich JavaScript und XML ausgebaut werden und zusätzliches Wissen im Bereich neuartiger HTML5-Funktionen und der webbasierten App-Entwicklung gewonnen werden.

Abschließend betrachtet ist das Projekt sehr gut verlaufen, so dass bereits weitere Implementierungen der Laufzeitumgebung für andere Hersteller im Rahmen von Anwendungsdemonstrationen vorgenommen wurden. Hier stieß das Gesamtkonzept auf großen Zuspruch.

Ausblick

Neben dem Testen und Ausbauen der Laufzeitumgebung wird vor allem die Entwicklung eines Backends für die Auswertung des Nutzerverhaltens einer der nächsten Schritte sein, um auch für Hersteller, einen weiteren Anreiz zu geben, diese Technik einzusetzen. Auch der Einsatz außerhalb des Endkundenbereichs, z.B. in Call-Centern wird betrachtet, da dort unter Umständen spezielle Anforderungen an eine Anwendung gestellt werden.

Im Bereich der App-Entwicklung ist die Umwandlung in native Apps der jeweiligen Plattformen (iOS, Android, etc.) mit Apache Cordova geplant. Damit lassen sich die Vertriebsmechanismen der jeweiligen App-Stores nutzen und eine einfache Verteilung gewährleisten

Anhang

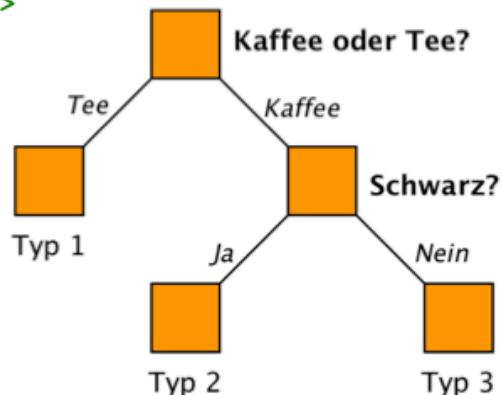
Kurzeinführung tr3

tr3: Grundlagen

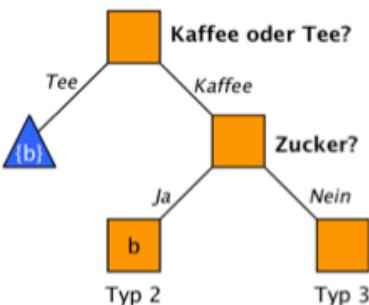
```
<tr3 out="Kaffee oder Tee?" ...>
  <branch in="Kaffee" out="Schwarz?">
    <leaf in="Ja" out="Typ 2" />
    <leaf in="Nein" out="Typ 3" />
  </branch>
  <leaf in="Tee" out="Typ 1" />
</tr3>
```

zusätzliche Attribute:

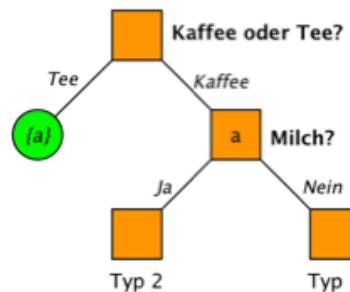
```
id="f67"
content="lorem.html"
class="danger"
```



tr3: Wiederverwendung



```
<tr3 out="Kaffee oder Tee?" ...>
  <branch in="Kaffee" out="Zucker?">
    <leaf in="Ja" out="Typ 2" id="b"/>
    <leaf in="Nein" out="Typ 3"/>
  </branch>
  <ref in="Tee" link="b"/>
</tr3>
```



```
<tr3 out="Kaffee oder Tee?" ...>
  <branch in="Kaffee" out="Zucker?" id="a">
    <leaf in="Ja" out="Typ 2"/>
    <leaf in="Nein" out="Typ 3"/>
  </branch>
  <ref in="Tee" copy="a"/>
</tr3>
```

Vergleich Bordliteratur und Baum

		Warn-/Kontrollleuchte	Mögliche Ursachen/Folgen und ► Lösungen
(1)	Die rote Bremsystem-Warnleuchte leuchtet bei Einschalten der Zündung.	Die Warnleuchte geht aus: ▪ nach dem Starten des Motors ▪ nach spätestens zehn Sekunden	
(1)	Die rote Bremsystem-Warnleuchte leuchtet bei angezogener Feststellbremse.	Die Feststellbremse ist angezogen. ► Die Feststellbremse lösen.	⚠️ Unfallgefahr Es ist zu wenig Bremsflüssigkeit im Flüssigkeitsbehälter. ► Nicht weiterfahren. ► Das Fahrzeug umgehend verkehrssicher abstellen. ► Das Fahrzeug mit der Feststellbremse gegen Wegrollen sichern, wenn Sie es verlassen. ► Einen Pannendienst anrufen, z. B. smartmove Assistance oder eine qualifizierte Fachwerkstatt, z. B. ein smart center.
(1) (2)	Die rote Bremsystem-Warnleuchte und die gelbe abs-Kontrollleuchte leuchten, während der Motor läuft.	⚠️ Unfallgefahr abs ist auf Grund einer Störung abgeschaltet. Damit ist auch esp® abgeschaltet. ► Unverzüglich eine qualifizierte Fachwerkstatt aufsuchen, z. B. ein smart center.	⚠️ Unfallgefahr Der Bremskreis ist ausgefallen. ► Nicht weiterfahren. ► Das Fahrzeug umgehend verkehrssicher abstellen. ► Das Fahrzeug mit der Feststellbremse gegen Wegrollen sichern, wenn Sie es verlassen. ► Einen Pannendienst anrufen, z. B. smartmove Assistance oder eine qualifizierte Fachwerkstatt, z. B. ein smart center.



Quellen

GOOGLE DEVELOPERS: Google Maps Javascript API v3.

< <https://developers.google.com/maps/documentation/javascript/> >

HAMMER.JS: A Javascript library for multi-touch gestures.

< <http://eightmedia.github.io/hammer.js/> >

IMURA, Tomomi (2013): HTML5 Mobile: Hardware Access and Device APIs. In: JS Conf 2013

< <http://girliemac.com/presentation-slides/html5-mobile-approach/deviceAPIs.html> >

MAHEMOFF, Michael (2010): A Simple Trip Meter using the Geolocation API.

< http://www.html5rocks.com/en/tutorials/geolocation/trip_meter/ >

OEVERMANN, Jan (2014): XML-basierte Entscheidungsbäume für Applikationen.

Projektarbeit an der Hochschule Karlsruhe. WS 2013/14

RIEKE, Jennifer (2009): Markenanalyse »smart«. Seminararbeit an der FH Düsseldorf, SS 2009

< http://weblogs.mk1.fh-duesseldorf.de/greenci/archive/jenniferrieke_525787_smart_final.pdf >

ROLAND BERGER STRATEGY CONSULTANTS / BURDA COMMUNITY NETWORK (2009):

Smart: So stilorientiert ist die Zielgruppe“ in Absatzwirtschaft 04/2009, S. 33

< http://www.absatzwirtschaft.de/content/_p=1004040,an=040901061 >

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig angefertigt, keine anderen als die angegebenen Mittel benutzt und alle wörtlichen oder sinngemäßen Entlehnungen als solche gekennzeichnet habe.

Karlsruhe, 13. März 2014