Interner Projektbericht zum Themenfeld mimesis.ui

Ansatz zur technologieneutralen Beschreibung von Benutzerschnittstellen für dialogbasierte Anwendungen in einem Multikanalumfeld

Stand 06.2014 (draft)

Dipl.-Inform. Michael Hitz
Duale Hochschule Baden-Württemberg - Stuttgart
Fachbereich Wirtschaftsinfomratik
Paulinenstraße 50
70178 Stuttgart
michael.hitz@dhbw-stuttgart.de

Inhaltsverzeichnis

1 Änderungshistorie

Version — Datum — Änderungen
1.0 — 1.2014 — Initiale Ausgabe
1.1 — 6.2014 — Überarbeitung: Anpassung / Erweiterung Metamodell, Erweiterung *mimesis.ui* DSL.

2 Vorbemerkung

Der vorliegende Projektbericht stellt den aktuellen Stand (6.2014) der Untersuchungen hinsichtlich der technologieneutralen **Beschreibung von Benutzerschnittstellen** im Projekt *mimesis* dar (Teilprojekt *mimesis.ui*).

Die hier dargestellten Teilergebnisse werden im weiteren Projektverlauf vertieft und stellen lediglich den *aktuellen Zustand* der laufenden Arbeiten dar.

Die Arbeiten erfolgen in Kooperation mit einem großen deutschen Versicherungsunternehmen, für welchen die Teilergebnisse erst prototypisch umgesetzt und in einem späteren Schritt produktiv eingesetzt werden.

3 Einleitung

3.1 Motivation

Die fortschreitende Digitalisierung der Geschäftsprozesse in Unternehmen bedarf Benutzerschnittstellen, über welche unterschiedliche Nutzergruppen mit unterschiedlichen Endgeräten mit den Businesssystemen eines Unternehmens in Dialog treten können. Aus Sicht der Businesssysteme sind dies unterschiedliche Zugangskanäle, die bedient werden müssen mit variierenden Anforderungen an die Benutzerschnittstellen aus fachlicher und technischer Sicht.

Unterschiedliche Nutzergruppen haben aufgrund Ihrer Bedürfnisse oder Fähigkeiten spezifische Anforderungen an die zur Interaktion bereitgestellten Frontends - weshalb in den Vergangenen Jahren viel Aufwand in die Erstellung von **nutzergruppenspezifischen Frontendvarianten** für gleichartige Anwendungen floss.

Betrachtet man beispielsweise das Szenario ”Antrag auf eine Lebensversicherung” aus dem Versicherungsbereich, so ergeben sich fachlich motivierte Unterschiede je nach Nutzergruppe (*fachlichem Kanal*) hinsichtlich des Umfangs der vom Anwender erfragten Daten, die zur Stellung eines Antrags benötigt werden. Ein fachlicher Kanal *Vertreter* beispielsweise wird in einem Gespräch mit dem Kunden die Daten erfassen und hier z.B. detailliert Gesundheitsfragen stellen können, die in die Tarifierung des Produktes einfließen können. Ein fachlicher Kanal *Endkunde*, der einen Antrag auf eine Versicherung im Internet stellt, wird allein schon aus Gründen des Datenschutzes solche Fragen nicht über das Internet beantworten wollen. Dies würde nachgelagert in einem persönlichen Gespräch oder per Anschreiben erfolgen, weshalb diese Fragen nicht im Antragsprozess im Internet gestellt werden.

Zu den nutzerspezifischen Anforderungen gesellen sich durch die technischen Entwicklungen der letzten Jahre im Bereich der Endgeräte weitere, **gerätespezifische Anforderungen**. Hier besteht der Bedarf an Varianten für *technische Kanäle* über die der Nutzer auf die Backends zugreift und die z.B. die Nutzungseigenschaften von Geräten oder Restriktionen bei der Darstellung von Inhalten durch variierende Displaygrößen berücksichtigen. Diese Anforderungen resultieren daraus, dass die Anwendung auf unterschiedlichen Ablaufumgebungen mit unterschiedlichen Eigenschaften verwendbar sein sollen und ein dem Ausgabemedium entsprechendes Nutzererlebnis (user experience) bieten müssen. Um dies erreichen zu können, bedarf es der Unterstützung unterschiedlicher Zieltechnologien, mit denen die Dialogschnittstellen umgesetzt werden[1].

[1] Die Untersuchungen fokussieren sich auf **grafische Benutzerschnittstellen**; prinzipiell sind die gefundenen Lösungen jedoch auch auf andere Schnittstellentechnologien anwendbar - z.B. sprachbasierte Dialogsysteme. Im Folgenden werden die Begriffe

Die **nutzerspezifischen und technischen Varianten von Anwendungsfrontends** wurden in der Vergangenheit häufig in eigenen Projekten neu erstellt - von Hand und zu hohen Kosten in der Erstellung und Wartung. Eine Lösung des Problems der Mehrfachentwicklung kann die Modellierung mutikanalfähiger Anwendungen und die automatische Generierung von Benutzerschnittstellenvarianten darstellen.

Um die Mehrfachentwicklung von Oberflächen in unterschiedlichen Technologien zu vermeiden, bietet es sich an, die Oberflächen in einer technologieneutralen Weise zu beschreiben und konkrete Ausprägungen daraus automatisiert herzuleiten. Hierzu existieren bereits einige Ansätze, z.B. Dialog Description Languages (DDL), die jedoch meist schon sehr nah an der Zielumgebung (z.B. webbasierte Frontends) sind und sich dadurch in einem Multikanal-Kontext weniger eignen, der ein hohes Maß an Technologieneutralität aufweisen muss. Aktuelle Ansätze entfernen sich von der reinen Beschreibung von Benutzerschnittstellen und modellieren Anwendungen über Abläufe (Task-/Konversationsbasierete Ansätze) bis hin zur Datenhaltung (z.B. im Webengineering-Umfeld: WebML und UWE). Diese Ansätze abstrahieren weiter von der Zieltechnologie, weisen aber meist eine hohe Komplexität in ihren Modellen auf. Varianten von Frontends lassen sich hier nur schwer umsetzen oder führen zu einer Vielzahl von ähnlichen Modellen, die voneinander abhängen und damit in der Praxis nur schwer verwaltbar sind (z.B. Erhaltung der Konsistenz über Modellvarianten und Abhängigkeiten zwischen Teilmodellen z.B. für Abläufe und Oberflächenbeschreibungen).

Insbesondere die komplexeren Ansätze zielen darauf ab, ein möglichst breites Spektrum an Anwendungsarten zu modellieren - im Falle der Ansätze im Web-Engineering sogar bis hin zur Persistierung der Daten. Die damit einhergehende Komplexität der Modellierung führt dazu, dass die Ansätze in der Praxis nur schwer angenommen werden. Die Fokussierung auf bestimmte Anwendungsarten kann aus unserer Sicht zur Vereinfachung der Modellierung führen und so zu in der Praxis handhabbaren Ergebnissen führen.

Ein in der Praxis häufig vorkommendes Anwendungsmuster sind dialogbasierte Anwendungen, die in einer Art Interview Daten von Nutzern erfragen (auch form filling oder directed dialog, [Chl06]) . Beispiele hierfür sind die Buchung eines Fluges, die Durchführung einer Überweisung oder die Tarifierung eines Versicherungsproduktes.

Diese Anwendungen zeichnen sich dadurch aus, dass sie in einer Folge von Schritten inhaltlich zusammengehörende Informationen im Dialog mit dem Benutzer sammeln (*datensammelnde Anwendungen*). Abhängig von bereits gegebenen Informationen werden ggf. weitere Nachfragen gestellt, die zur Verarbeitung der Informationen in Folgeschritten benötigt werden. Diese Anwendungen treten in Varianten für unterschiedliche Nutzergruppen auf, weisen in den Benutzerschnittstellen einen hohen Standardisierungsgrad auf und verwenden klassifizierbare Interaktionsmuster. Sie eigenen sich daher besonders für die automatische Generierung.

Diese Anwendungen können als Bausteine aufgefasst werden, die in größere Anwendungen aggregiert werden. Dies entspricht dem aktuell vorherrschenden Vorgehen, Funktionalität beispielsweise in Portalen aus kleinen Bausteinen zusammenzusetzen, die im Sinne von Mesh-Ups in den Content eingefügt oder auch in RichClient-Anwendnungen aggregiert werden können. Dadurch kann eine hohe Wiederverwendbarkeit erreicht werden, da die Bausteine in sich geschlossen und unabhängig vom Kontext betrieben werden können.

Da dieses Anwendungsmuster insbesondere im Versicherungsbereich häufig anzutreffen ist, fokussieren sich unsere Untersuchungen auf diese Anwendungskategorie. Unsere Behauptung ist, dass sich hier signifikant vereinfachte Modellierungsformen finden lassen, die eine einfache Erstellung von konsistenten Varianten gestatten.

Oberfläche, Benutzerschnittstelle und User Interface synonym für Dialogschnittstellen verwendet.

3.2 Der Lösungsansatz von mimesis

Die Idee in *mimesis* ist es, sich von der Beschreibung der Benutzerschnittstelle oder der konkreten Abläufe zu entfernen und sich auf eine detailliertere Beschreibung der in einer Anwendung im Dialog verarbeiteten Daten zu stützen. Die durch die Untersuchungen zu verifizierende Annahme ist, dass sich für bestimmte Anwendungskategorien bereits aus einer solchen Beschreibung nicht-triviale Benutzerschnittstellen automatisch herleiten lassen, wenn ausreichend Informationen zur Semantik der enthaltenen Daten vorhanden sind.

Das Ziel des Projektes *mimesis* ist die Entwicklung eines modellgetriebenen Ansatzes zur Generierung für Benutzerschnittstellenvarianten für dialogbasierte Anwendungen, der rein auf den semantisch näher beschriebenen Daten der Anwendung fußt. Der verfolgte Ansatz beruht auf der These, dass sowohl nutzergruppenspezifische als auch technische Varianten von dialogbasierten Anwendungen basierend auf einem solchen Modell automatisch hergeleitet werden können.

Ein Datenmodell im klassischen Sinne (z.B. ER- oder Objektmodell) besitzt jedoch nicht ausreichend Informationen zur Herleitung einer in der Praxis verwendbaren Benutzerschnittstelle, die genügend Variabilität hinsichtlich der technologischen Anforderungen besitzt (kritik: [Mes08], [Dem08], [DMLC08] - in [CP08] sogar als *fast Food UIs* bezeichnet). Die Annahme ist, dass das Datenmodell um Informationen angereichert werden kann, die die enthaltenen Daten rein semantisch/qualitativ näher beschreiben und unabhängig von einer etwaigen Zieltechnologie formuliert werden können (z.B. keinerlei spezifische Informationen für eine HTML- oder RichClient-Oberfläche enthalten)

Das Projekt *mimesis* untersucht daher, welche Informationen in einem datenzentrierten Modell enthalten sein müssen, um daraus praxistaugliche Benutzerschnittstellenvarianten zu generieren.

3.3 Vorgehensweise: der Weg zum datenzentrierten Modell

Die Basis für eine Ableitung der benötigten semantischen Informationen bildet in unserem Vorgehen eine Analyse konkreter Benutzerschnittstellen existierender Anwendungen in unterschiedlichen Technologien (z.B. Web-, RichClient-, mobile-Anwendungen). Es wird untersucht, welche Interaktionsformen und Funktionalitäten/Featuers in den konkreten Anwendungen und Technologien in der Praxis auftreten und untersucht, inwieweit sich diese Eigenschaften von Benutzerschnittstellen technologieneutral beschreiben lassen. Das Ziel dieser Untersuchungen ist es, eine abstrakte Beschreibung von Benutzerschnittstellen zu erhalten, aus denen sich dann wieder konkrete Benutzerschnittstellen in unterschiedlichen Technologien generieren lassen.

In einem weiteren Schritt soll daraufhin untersucht werden, inwiefern sich die im abstrakten Benutzerschnittstellenmodell enthaltenen Eigenschaften aus einem datenzentrierten Modell herleiten lassen, welche Informationen hierzu benötigt werden und wie diese Informationen als semantische Eigenschaften der Daten formuliert werden können (ohne Bezug zur Aufgabe ”Herleitung einer Benutzerschnittstelle” sondern rein qualitativ auf die Daten bezogen).

Dieses Papier befasst sich mit der Analyse der Eigenschaften von Benutzerschnittstellen und Herleitung einer technologieneutralen, abstrakten Beschreibung von Benutzerschnittstellen (*mimesis.ui*) . Die Herleitung der datenzentrierten Beschreibung und Variantenbildung erfolgt gesondert in weiteren Arbeiten.

3.4 Abstrakte Beschreibung von Benutzerschnittstellen im Teilprojekt *mimesis.ui*

Das Teilprojekt *mimesis.ui* befasst sich mit der oben angeführten Analyse der bestehenden Anwendungen und der abstrakten Beschreibung von Benutzerschnittstellen.

Im Rahmen der Arbeiten fokussieren wir uns auf grafische Benutzerschnittstellen für dialogbasierte Anwendungen. Bei den Benutzerschnittstellen für solche Anwendungen kann zwischen nativen und webbasierten Technologien unterschieden werden, welche im Unternehmenskontext Relevanz besitzen. Eine weitere, sinnvolle Kategorisierung kann zudem im Multikanalumfeld nach der Desktop- und mobilen Eignung der Anwendung erfolgen. Beispiele für diese Umgebungen sind:

- **Rich client-Anwendungen (Kat.: nativ + desktop):** z.B. Java-Swing-, Microsoft Windows, Apple Mac OS- und Linux-Anwendungen. Diese Umgebungen zeichnen sich dadurch aus, dass sie über große Darstellungsflächen verfügen. Die Bedienung dieser Geräte erfolgt vorwiegend über Zeigegeräte (z.B. Maus). Die Darstellungen der Eingabecontrols ist nativ und unterscheidet sich zwischen den einzelnen Herstellern.
- **Mobile Endgeräte (Kat.: nativ + mobile):** z.B. Apple iOS, Android, Windows mobile. Die Darstellungsflächen sind auf diesen Geräten tendenziell klein und die Eingabe erfolgt häufig über Gesten auf einem Touchscreen. Die Bedienungskonzepte unterscheiden sich in der Bedienung von Geräten mit Zeigegeräten.
- **Browser-Anwendungen (Kat.: webbasiert + desktop):** z.B. HTML5 / JavaScript, Adobe-Flash. Die Darstellungsflächen sind hier tendenziell groß. Die eingesetzten Technologien sind webbasiert und auf unterschiedlichen Geräten verfügbar.
- **Mobile Browser (Kat.: webbasiert + mobile):** HTML5 / JavaScript. Die Darstellungsflächen sind hier klein, die Technologien nur begrenzt auf die Umgebung angepasst. Da die Geräte touchbasiert sind, müssen hier angepasste Technologien / Frameworks zum Einsatz kommen als dies im desktop-Webbereich der Fall ist.

Jede dieser Dialogkategorien folgt bestimmten Regeln hinsichtlich der *User Experience*. Innerhalb einer Kategorie ist ein einheitliches und konsistentes Aussehen und Verhalten das Ziel - unter anderem aus Gründen der Einhaltung des Corporate Designs, welches eine Anwendung in einem Unternehmenskontext haben muss.

Um zu einer technologieneutralen Beschreibung der Benutzerschnittstellen zu gelangen, wird in *mimesis.ui* untersucht, welche Elemente und Eigenschaften die Oberflächen in den einzelnen Technologien aufweisen und untersucht, welche Informationen notwendig sind, um die vorgefundenen Elemente unabhängig von der umsetzenden Technologie zu beschreiben.

Einerseits werden hierbei statische Eigenschaften der Benutzerschnittstellen betrachtet (z.B. vorhandene Klassen von Eingabeelementen und deren technologieabhängige Varianten) oder strukturelle Aspekte (z.B. die Verteilung von Elementen auf Seiten, Unterseiten oder Zusammenfassung in Gruppen und die Naviagtionsmechanismen zwischen diesen Bereichen), andererseits dynamische Aspekte untersucht (z.B. Nutzeraktionen, Validierung, Ein-/Aus-blenden von Bereichen oder Reaktionen auf Nutzereingaben).

Für die vorgefundenen Muster wird untersucht, welche Informationen notwendig sind, um daraus die technologischen Varianten ableiten zu können und daraus schließlich einen technologieneutrale Beschreibung entwickelt.

Folgende Ergebnisse werden hierbei erstellt:

- Analyse in Benutzerschnittstellen vorkommender Elemente, Features und Eigenschaften
- Analyse der Eigenschaften der gefundenen Elemente und deren technologieneutrale Abstraktion
- Beschreibung der Features in Form eines Metamodells
- technologieneutrale Beschreibung der abstrakten Benutzerschnittstelle in Form einer DSL

Zur technischen Umsetzung wird werden zudem ein Verfahren zur Herleitung konkreter technologischer Varianten aus der Beschreibung entwickelt, und zur Validierung prototypisch umgesetzt.

4 Fragestellungen, Beitrag der Untersuchungen und Abgrenzung

Die im Bereich *mimesis.ui* untersuchten Fragestellungen lassen sich zusammenfassen:

- Was sind Anforderungen, die konkrete Dialogschnittstellen an eine technologieneutrale Beschreibung stellen?
- Wie kann aus diesen Anforderungen eine *technologieneutrale Dialogbeschreibung* hergeleitet werden?
- Kann aus der Dialogbeschreibung für unterschiedliche Geräte und Technologien eine Oberfläche hergeleitet und auf unterschiedlichen Ausgabegeräten ”gerendert” werden?
- Wie kann eine solche Beschreibung *kanalspezifisch* mehreren Geräten zur Verfügung gestellt werden? Hierbei wird insbesondere untersucht, wie eine Beschreibung beschaffen sein soll, damit sie sowohl vor als auch während der Laufzeit der Anwendung auf einem Server oder in einer client-Anwendung in die Zieltechnologie überführt werden kann.

Diese Arbeit liefert einen Beitrag zum Themenfeld der technologieneutralen und multikanalfähigen Beschreibung von Dialogschnittstellen.

Die Untersuchungen erfolgen hierbei anhand des im Umfeldes von Selbstbedienungssystemen häufig auftretenden Szenarios der *datensammelnden Anwendungen*, wodurch eine vereinfachte Erstellung der Dialogbeschreibungen erwartet werden kann.

Es wird eine technologieneutrale Dialogbeschreibung vorgeschlagen, welche anhand der Anforderungen graphischer Benutzeroberflächen aus dem gewählten Anwendungsszenario hergeleitet wird.

Es wird aufgezeigt, wie die Generierung von Varianten für unterschiedliche Umgebungen erreicht und so eine einheitliche Beschreibung für unterschiedliche technische Kanäle erreicht wird.

5 Anforderungen an eine technologieneutrale Dialogschnittstellenbeschreibung

In den folgenden Abschnitten werden die Anforderungen betrachtet, die an eine technologieneutrale und multikanalfähige Dialogschnittstellenbeschreibung zu stellen sind.

Zunächst wird der Betrachtungskontext erläutert, auf dem die folgenden Darstellungen beruhen und die Anwendungskategorie der *datensammelnden Anwendungen* beschrieben, die als Problemraum herangezogen wurde. Daraufhin werden die Anforderungen dargestellt, die grafische Benutzerschnittstellen

an die angestrebte technologieneutrale Beschreibung von Benutzerschnittstellen stellen. Abschließend werden Anforderungen diskutiert, die sich aus der Multikanalfähigkeit und damit verbundener Varianten ergeben. Die Ergebnisse legen die Grundlage für die im darauf folgenden Kapitel beschriebene Umsetzung.

5.1 Ausgangsbasis der Untersuchungen

Um ein möglichst einfaches und doch viele Anwendungsfälle umfassendes Szenario zu erhalten, wurde in Zusammenarbeit mit dem am Projekt beteiligten Versicherer vorab untersucht, welche Arten von Anwendungen häufig auftreten und wo ein Nutzen aus der Möglichkeit der Variantenbildung gezogen werden kann ([Hit13]).

Das Ergebnis dieser Untersuchungen war, dass insbesondere *datensammelnde Anwendungen* (z.B. Tarifierungsrechner für Versicherungsprodukte oder in sich abgeschlossene Dialogfolgen wie Adressänderung, Kontoänderung oder Überweisungen) einen Großteil der Anwendungen in den Portalen darstellen und damit geeignete Kandidaten für eine automatisierte Erstellung sind. Ca. 80% solcher Anwendungen wurden im Portal des Versicherers identifiziert, die zudem häufig als vertreterspezifische Varianten im Vertreter- bzw. Maklerportal auftraten und auch als RichClient-Anwendungen auf den Vertreterrechnern implementiert wurden. Sie stellen somit eine relevante Anwendungskategorie dar.

Diese Anwendungskategorie zeichnet sich zudem dadurch aus, dass sie einfach erfassbare, standardisierte Interaktionstechniken aufweist. Zudem findet sich hier ein hoher Standardisierungsgrad hinsichtlich des Aussehens (definierte Styles über Unternehmensrichtlinien) und ein definierter Umfang an Controls mit standardisiertem Verhalten. Dies spricht für eine besondere Eignung für die automatische Herleitung der Benutzerschnittstellen.

5.2 Charakterisierung: Datensammelnde Anwendungen

Datensammelnde Anwendungen zeichnen sich dadurch aus, dass sie Dialogstrecken besitzen, die Daten sammeln und Punkte im Ablauf besitzen, an denen diese Daten verarbeitet werden (z.B. durch Senden einer Anfrage an den Server und ggf. anschließende Weiterverarbeitung von Ergebnissen der Anfrage) [2].

Der Anwendungsfluss besteht hier aus Teilen, in denen Daten gesammelt werden - unterbrochen von Teilen, in denen Daten serverseitig verarbeitet werden müssen. Abbildung 1 stellt dieses Prinzip grafisch dar. Die Bereiche, die mit *sammeln* überschrieben sind, arbeiten vorwiegend clientseitig, die mit *verarbeiten* überschriebenen Schritte benötigen die Kommunikation mit dem serverseitigen Anteil der Anwendung.

Da in einem *sammelnden Abschnitt* die Oberfläche unabhängig vom Server arbeiten soll[3], muss diese

[2] Eine weitere Anwendungskategorie sind *datendarstellende Anwendungen*, welche hauptsächlich Daten aufbereiten und präsentieren. Diese sind häufig sehr speziell auf den Nutzerkreis angepasst und weisen wenig Interaktion zur Datenerfassung auf. Ein Beispiel hierfür wären Vertragsübersichten, die für unterschiedliche Kanäle sehr unterschiedlich ausfallen, da sich die darzustellenden Daten abhängig vom Kanal stark unterscheiden. Von hier findet häufig der Absprung in *datensammelnde Anwendungen* statt (z.B. vom Vertrag zur Änderung der Adresse des Versicherungsnehmers).

[3] Eine Einschränkung der Serverunabhängigkeit besteht in den Momenten, wo Daten abhängig von Nutzereingaben vom Server nachgeladen werden müssen. Ein triviales Beispiel wäre das Laden von Städtenamen abhängig von der Eingabe einer Postleitzahl in einem Adressdialog oder von Auswahlmöglichkeiten bei der Tarifierung eines Versicherungsvertrages aufgrund der Angabe des Geschlechts.

Abbildung 1: Datensammelnde Anwendungen

einige Funktionalitäten übernehmen, welche von modernen Benutzerschnittstellen erwartet werden. So müssen beispielsweise Eingaben validiert, Bereiche sichtbar / unsichtbar geschaltet werden und eine Navigation innerhalb der Seitenfolge erfolgen können. Auf diese Anforderungen wird im folgenden Abschnitt näher eingegangen.

Die verwendeten Technologien unterscheiden sich hinsichtlich ihrer Mächtigkeit. So kann in nativen Umgebungen mehr Rechenleistung erwartet werden als in Browserbasierten. Dies hat Auswirkungen auf die Überlegung, wo die technologieneutrale Dialogbeschreibung in eine technologieabhängige Oberfläche gewandelt werden kann. Dies kann sowohl client- als auch serverseitig erfolgen, wodurch eine Beschreibungsform gewählt werden muss, die für beide Szenarien geeignet ist. Diese Integrationsoptionen werden im Rahmen des Lösungskonzepts angegeben.

5.2.1 Ein Beispiel: Tarifierungsbaustein für eine private Haftpflichtversicherung

Das Beispiel, welches hier zur Illustration dienen soll, ist ein Tarifierungsbaustein für eine private Haftpflichtversicherung, der in leicht abgewandelter Form von mehreren Nutzergruppen auf unterschiedlichen Geräten verwendet werden kann. Als konkrete Variante dient hier der komplexere Fall für einen Vertreter, der zusammen mit einem Kunden die Informationen zur Tarifierung erfasst.

Abbildung 2 zeigt Auszüge aus dem Frontend exemplarisch. Die dargestellte Anwendung erfasst zwei grundsätzliche Informationsbereiche: zum einen die für die Tarifierung relevanten Daten des Versicherungsnehmers, zum anderen Fragen zur Produktkonfiguration, das für den Versicherungsnehmer zusammengestellt wird (z.B. Art der Versicherung und zusätzliche Komponenten). Die Einstiegsseiten zu den beiden Bereichen sind in Abbildung 2 dargestellt.

Da im Rahmen der Erfassung eine potentiell große Zahl von Eingaben erfolgen muss, sind diese auf unterschiedliche **Seiten** und **Unterseiten** verteilt. Die Seiten sind im Falle der Bedienung durch den Vertreter auf einem Laptop mit einer baumartigen **Navigation** anwählbar (Abbildung 2, links). Der Vertreter kann dadurch *wahlfrei* zu den angebotenen Seiten springen, die für seine Beratung relevant sind. Zudem kann über die Pfeiltasten am oberen Rand *sequentiell* durch die Frageseiten navigiert werden.

Einige der Eingaben bedingen, dass weitere Informationen angegeben werden müssen. Hierdurch ändert sich die Zahl der Felder und Seiten abhängig von den bereits eingegebenen Daten. Damit ändert sich auch die Navigation. Je nach eingegebenen Informationen kommen Seiten hinzu oder entfallen. So führt die Auswahl von *verheiratet* bei der Frage zum *Familienstand* dazu, dass bei der Produktkonfiguration eine *Familienversicherung* gewählt werden kann (**bedingtes Ein-/Ausblenden**)

Die Personendaten (Abbildung 2, oben) umfassen *Namen, Geburtstag, Geschlecht, Familienstand, Anzahl der Kinder* etc. und bilden die Grundlage für das Angebot bestimmter Produktoptionen. Auf den angebotenen Seiten zum Produkt werden sukzessive die Angaben verfeinert. Im ersten Feld (*PH-Art*) wird der Basistarif gewählt. Sofern der Versicherungsnehmer unter 18 Jahre alt ist, kann eine *junge Leute Rabatt* in Anspruch genommen werden (**bedingte Aktivierung**) . Wählt er eine *Familien- oder*

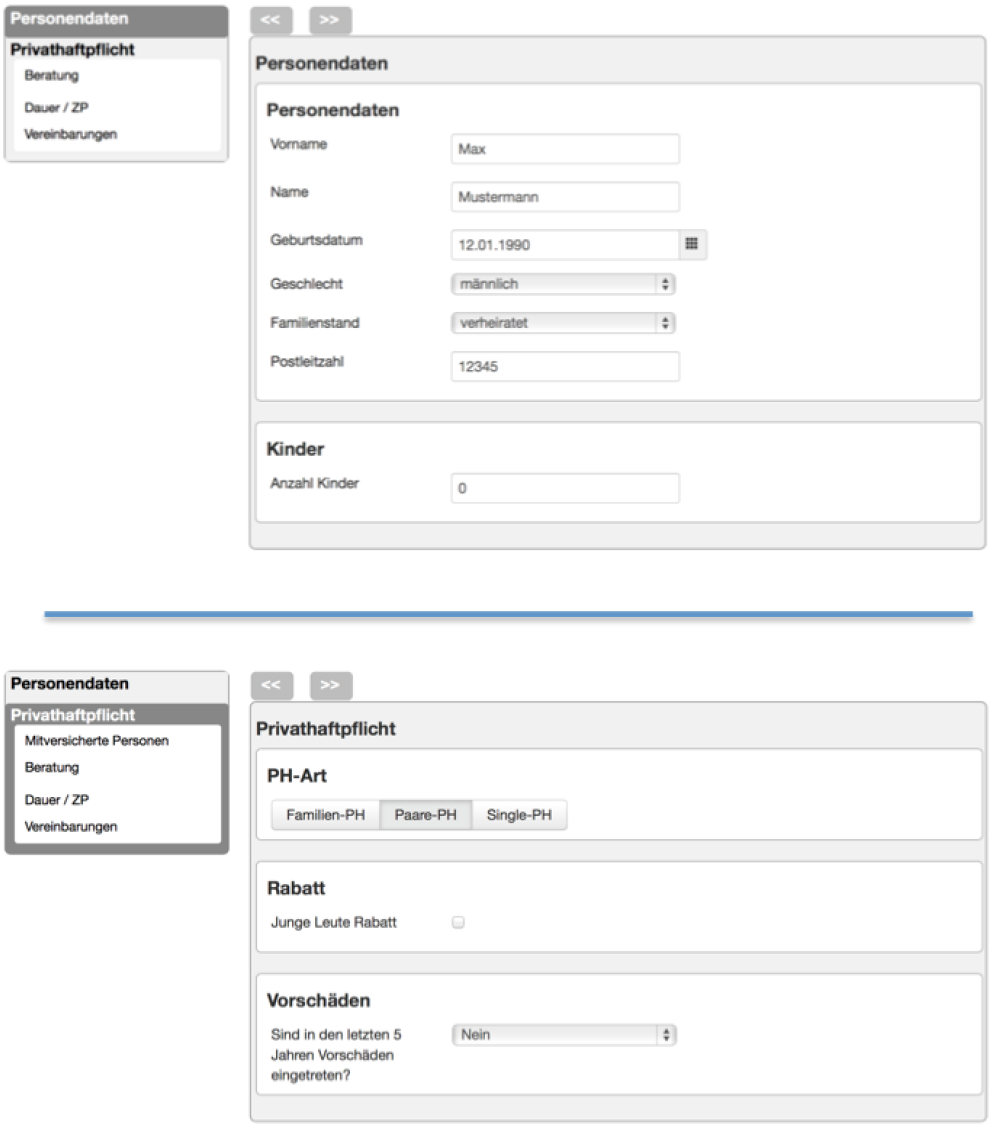

Abbildung 2: Einstieg in die Tarifierungsanwendung

Paare-Haftpflichtversicherung, so müssen auch die Informationen zur *mitversicherten Person* angegebenen werden.

Auf weiteren Seiten (nicht dargestellt) können dann Zusatzbausteine angewählt werden, die ggf. weitere Angaben erfordern. So werden je nach Angaben ganze Seitenbereiche ausgeblendet und Felder deaktiviert.

Die Anwendung folgt dem beschriebenen Muster der *datensammelnden Anwendungen*. Es wird im ersten Schritt eine Reihe von Daten erfasst, die geschlossen zum Server zur Tarifierung gesendet werden (*verarbeitender Schritt*). Das Ergebnis wird dann auf einer Ergebnisseite dargestellt und ggf. weitere *datensammelnde Schritte* gestartet. Im Fall der hier dargestellten Anwendung folgen keine weiteren Schritte. Nachdem der Nutzer alle Informationen angegeben hat und diese **validiert** wurden, erscheint eine Schaltfläche *Übermitteln*, welche die Daten an den Server übermittelt.

In den folgenden Abschnitten werden die Anforderungen, die die dargestellte Anwendungskategorie an die Oberflächen stellt, näher betrachtet.

5.3 Analyse der Anforderungen an die Benutzeroberflächen für datensammelnde Anwendungen

Die Grundlage für die Analyse der Anforderungen waren bereits vorhandene Anwendungen des beteiligten Versicherers ausgewählt, die dem beschriebenen Muster der datensammelnden Anwendungen entsprachen. Die Anwendungen stammten aus den Bereichen der Antragstellung und Tarifierung für Versicherungsprodukte, sowie Self-Services, die über webbasierte Anwendungen im Inter- und Extranet zur Verfügung gestellt werden. Diese Anwendungen existieren häufig für unterschiedliche Nutzergruppen (Endkunden im Internet, Makler und Vertreter im Extranet und auf Vertreterlaptops). Folgende Kategorien wurden dabei betrachtet:

- RichClient-basierte Tarifierungsbausteine für Vertreter
- Webbasierte Tarifierungsbausteine im Vertreter-/Maklerportal sowie
- Varianten dieser Anwendungen für Endkunden im Internet-Auftritt des Unternehmens (im öffentlichen und passwortgeschützten Bereich)
- Weitere Self-Service-Anwendungen für Endkunden im Internet (z.B. Adressänderung, Kontaktformulare, Bankverbindung, Beitragsrechner, Schadenmeldung)

Im Bereich der mobilen Anwendungen wurden lediglich webbasierte Anwendungen des Partners im mobile-Kundenportal untersucht, da zum Zeitpunkt der Untersuchungen keine repräsentativen, nativen Anwendungen zur Verfügung standen. Wir vermuten jedoch, dass sich die abstrahierten Eigenschaften später auch auf mobile Geräte übertragen lassen - dies soll später die Validierungsphase in einer prototypischen Umsetzung nachweisen.

Die datensammelnden Anwendungen im Fokus wurden einer Grobanalyse hinsichtlich ihrer Features und Muster unterzogen und repräsentative Beispiele ausgewählt[4]. Bei der Auswahl lag der Fokus darauf, möglichst komplexe Beispiele zu untersuchen, die ein breites Spektrum an Features aufweisen und hinreichend komplexe Abläufe besitzen, um möglichst viele Eigenschaften/Muster erfassen zu können. Anwendungen, deren Features und Muster bereits offensichtlich durch andere Anwendungen in der Betrachtung abgedeckt waren, konnten dadurch aus der näheren Untersuchung ausgeschlossen werden.

Eine Anwendung, die ein breites Spektrum an Anforderungen abdeckte, war die bereits als Beispiel angeführte Tarifierung einer privaten Haftpflichtversicherung, weshalb diese auch zur Erläuterung im Folgenden primär herangezogen wird.

Zur Absicherung bzw. Ergänzung der vorgefundenen Muster wurde zudem entsprechende Literatur im Themenfeld des Designs von Benutzerschnittstellen und Muster herangezogen (z.B. [Tid06] und

[4] Die Untersuchungen sind aktuell noch *work in progress*. Die finale Zahl der untersuchten Anwendungen wird in einer späteren Version des Papiers nachgereicht. Zudem werden dann auch Beispielabbildungen dieser Anwenundungen eingefügt.

[DFT05]). Diese halfen bei der Strukturierung und fügten weitere gängige Muster den Ergebnissen der eigenen Untersuchungen hinzu, die in den untersuchten Beispielen bisher nicht enthalten sind [5].

Die Themenbereiche, die sich aus der Analyse der Oberflächen ergaben waren einerseits strukturelle, andererseits das Verhalten betreffende Merkmale der Benutzerschnittstellen. Die identifizierten strukturellen Merkmale befassten sich mit der Gruppierung von Informationen und Zusammenfassung von Elementen in Darstellungseinheiten in den Oberflächen und der Navigation zwischen diesen Einheiten. Ein weiteres Themenfeld ist der Bereich der vorgefundenen Varianten von Ein-/Ausgabeekomponenten auf den Seiten (Widgets, Controls) und deren Abstraktionen.

Die folgenden Merkmale wurden dabei von uns identifiziert und werden in den folgenden Abschnitten näher erläutert.

strukturelle Merkmale

- Darstellungseinheiten (Seiten, Unterseiten, Gruppierung)
- Navigation zwischen Darstellungseinheiten
- Ein-/Ausgabekomponenten

verhaltensrelevante Merkmale

- Reaktion auf Änderung: Validierung von Eingaben
- Reaktion auf Änderung: Sichtbarkeitssteuerung
- Reaktion auf Änderung: Dynamische Inhalte
- Reaktion auf Änderung: Anpassung der Navigation
- Benutzeraktionen
- Fehlerbehandlung und Darstellung

Die Untersuchungen brachten auch abgeleitete Anforderungen hervor:

- Datenhaltung und -Bindung
- Initialisierung des Modells

Aus unserer Sicht kann aus einem Modell, das diese Merkmale im Stande ist zu beschreiben eine Benutzschnittstelle automatisiert herleiten. Für diese Merkmale muss somit im nächsten Schritt eine abstrahierte Beschreibung gefunden werden, die dann zur Herleitung ausgewertet wird.

5.3.1 Aufteilung in Darstellungseinheiten (Seitenaufteilung, Gruppierung von Fragen)

Da die Menge der erfassten Daten potentiell hoch sein kann, ist es notwendig, diese in gesonderten Einheiten zu erfassen, die zu unterschiedlichen Zeitpunkten erfragt werden. Bei grafischen Benutzeroberflächen wird dies üblicherweise durch Aufteilung der Einheiten auf **Seiten** und **Unterseiten** umgesetzt. Zwischen diesen *Darstellungseinheiten* muss navigiert werden können (s. Abschnitt "Navigation").

[5] Die Untersuchungen sind aktuell noch *work in progress*. Die finale Zahl der untersuchten Anwendungen wird in einer späteren Version des Papiers nachgereicht. Zudem werden dann auch Beispielabbildungen dieser Anwenundungen eingefügt.

Ein Beispiel dazu ist in Abbildung 2 dargestellt. Hier werden die Personendaten in der Anwendung gesondert von den Vertragsinformationen auf gesonderten Seiten erfragt, die sinnvoll/semantisch zusammenhängende Fragen enthalten.

In Anwendungen, die eine große Darstellungsfläche zur Verfügung haben (Desktop RichClient und Web), werden die Daten in den untersuchten Anwendungen meist über eine hierarchische Navigation zur Auswahl angeboten, sodass beliebig zwischen den Einheiten gesprungen werden kann (*wahlfreier Zugriff*). Die angebotenen Seiten können aufgrund des verfügbaren Platzes üblicherweise relativ viele Daten zur Eingabe anbieten.

In Abbildung 3 ist die hierarchische Seitenaufteilung exemplarisch für den Fall dargestellt, dass der Anwender auf der Seite ”Beratung” einen Zusatzbaustein ”Vermieter” gewählt hat, der zwei Wohnungen im Versicherungsumfang einschließen soll. Die Adressen der Wohnungen werden auf einer gesonderten Seite erfasst, welche hierarchisch jedoch unter der Seite ”Beratung” angesiedelt.

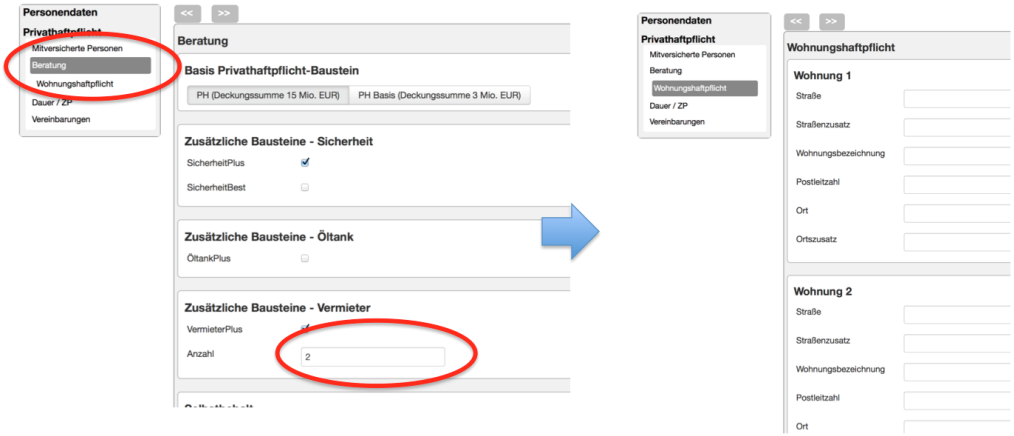

Abbildung 3: Hierarchische Bildung von Untergruppen

Auf Geräten, die über geringen Darstellungsplatz verfügen (mobile Geräte) findet sich selten eine hierarchische Navigation. Auch hier werden Informationen seitenweise erfragt, die Navigation erfolgt aus Platzgründen aber üblicherweise *sequentiell*, indem die Seiten nacheinander durchlaufen werden.

Auch datensammelnde Anwendungen in Portalen besitzen in den meisten Fällen eine *sequentielle* Navigation, da die Anwendungen hier meist eingebettet in den Content der Portalseite ablaufen. Das Vorgehen wird in [Tid06] als *wizard*-Pattern aufgeführt.

Abbildung 4 zeigt die Darstellung der Personendaten in einer Desktop-Anwendung und einer mobilen Anwendung. Während im Desktopbereich große Datenmengen und eine *hierarchische*, baumartige Navigation Platz finden, kann die Darstellung auf mobilen Endgeräten nur spartanisch ausfallen und wird über eine *weiter* -und *zurück*-Navigation umgesetzt (*wizard*-Pattern).

Die vorgefundenen Muster (*hierarchische* und *sequentielle* Abfrage von Darstellungseinheiten) lassen sich nach unseren Erkenntnissen bei datensammelnden Anwendungen auf den hierarchischen Fall reduzieren. Die Anordnung der Frageblöcke in einer hierarchischen Darstellung folgt hier immer der Bedeutung der Daten und hat auch hier eine aus fachlicher Sicht sinnvolle Reihenfolge: die einzelnen Fragen sind auch in der hierarchischen Sicht immer sinnvoll *von oben nach unten* abarbeitbar. Insofern ergibt das Traversieren des Baums nach dem *preorder-Prinzip* (in der Reihenfolge *parent* vor *children*

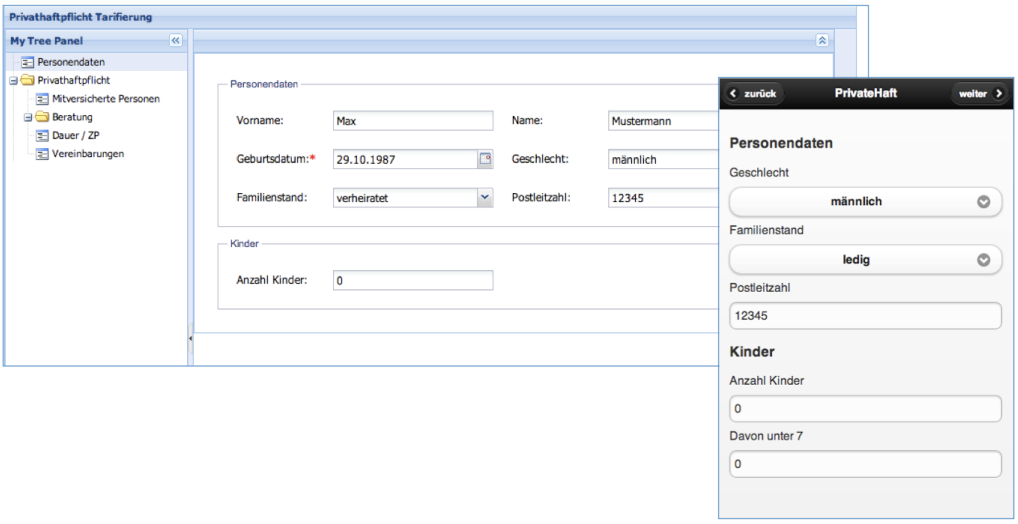

Abbildung 4: Darstellung der Personendaten in einer Desktop- und mobilen Anwendung

durchlaufen, z.B. https://en.wikipedia.org/wiki/Tree_traversal#Pre-order) eine sinnvolle Seitenfolge. Ob wahlfrei auf Seiten/Unterseiten in einer Hierarchie gesprungen wird oder die Hierarchie sequentiell in dieser Reihenfolge durchlaufen wird, spielt in datensammelnden Anwendungen daher keine signifikante Rolle.

Innerhalb einer Darstellungseinheit können die Fragen ebenfalls **gruppiert** werden, um so zusammengehörige Daten hervorheben und als Einheit (z.B. zum ein-/ausblenden) behandeln zu können. So ist in Abbildung 3 für jeden der Zusatzbausteine ein eigener Bereich dargestellt, der abhängig von anderen Eingaben ein- oder ausgeblendet werden kann. Diese Gruppierung tritt in allen betrachteten Szenarien auf und muss ebenfalls modelliert werden können. In Abbildung 4 wird dies zur Abgrenzung der Fragen in den Gruppen ”Personendaten” bzw. ”Kinder” deutlich. Im Falle der Desktop-Variante wird dies zur Darstellung umrahmter Gruppen verwendet, in der mobile-Variante zur Abgrenzung der Gruppen über Zwischenüberschriften.

Als Anforderung zur Modellierung der Seiten und Beziehungen der Seiten zueinander ergibt sich damit:

- Darstellungseinheiten müssen hierarchisch in einer baumartigen Struktur als Seiten und Unterseiten definiert werden können, da dies für eine sinnvolle Eingabestruktur benötigt wird.
- Die Seiten und Untergruppen besitzen eine Reihenfolge, in welcher die enthaltenen Fragen sinnvoll erfragt werden. Die Untergruppen können gesondert präsentiert und damit wahlfrei angesprungen werden, können aber auch sequentiell abgearbeitet werden.

Zudem müssen die Fragen in Gruppen zusammengefasst werden können, um sie an der Oberfläche als sinnvolle Einheiten behandeln zu können.

5.3.2 Navigation zwischen Darstellungseinheiten

Um alle Daten zu können, müssen die im letzten Abschnitt beschriebenen Darstellungseinheiten durchlaufen werden - es muss durch die Darstellungseinheiten *navigiert* werden.

Die technologieneutrale Beschreibung der Navigation stellt eine besondere Herausforderung dar, da unterschiedliche Technologien unterschiedliche Navigationsmöglichkeiten besitzen müssen - dies wurde in Abbildung 4 bereits deutlich. Auf Geräten mit kleinen Darstellungsflächen (z.B. Smartphones und kleinen Tablet-Computern), muss aufgrund des fehlenden Platzes meist eine *sequentielle Navigation* gewählt werden. Hierbei wird eine Startseite angezeigt und nach erfolgter Eingabe der Daten auf der Seite auf eine Folgeseite geleitet. So können sequentiell alle benötigten Daten abgefragt werden. Auf Geräten mit größeren Displays bietet es sich jedoch häufig an, einen *wahlfreien Zugriff* auf die Darstellungseinheiten zu erlauben. Dies ist insbesondere bei der Erfassung großer Datenmengen sinnvoll.

Abbildung 6 und Abbildung 5 zeigen diese beiden Vorgehensweisen im Falle der Tarifierung für die Private Haftpflichtversicherung. Abbildung 5 zeigt eine Variante für einen Privatkunden, der die Anwendung auf einem mobilen Gerät bedient (vereinfachter Ablauf). Abbildung 6 zeigt eine Variante für ein Gerät maximaler Größe. In Abbildung 5 findet die Navigation sequentiell statt (Navigation mit ”Weiter”-und ”Zurück”-Button), im anderen Fall ist eine hierarchische, baumartige Navigation vorhanden, über die der Nutzer der Anwendung frei zwischen den Dialogeinheiten springen kann.

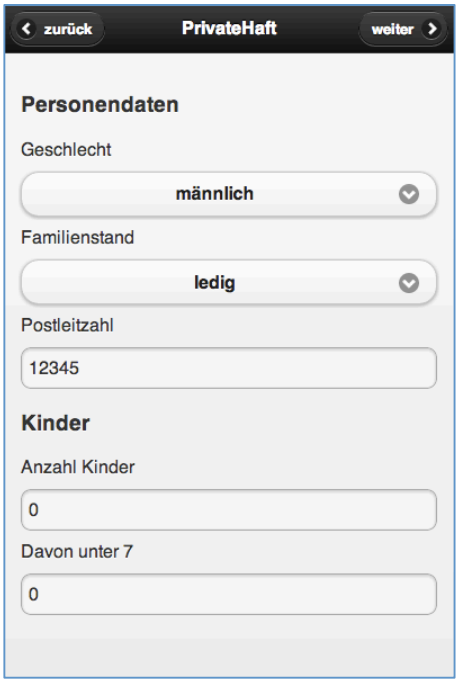

Abbildung 5: Sequentielle Navigation in einer mobilen Anwendung

Die Wahl der Navigation durch die Seiten ist abhängig vom Zielgerät, kann aber auch durch die Menge der darzustellenden Daten bestimmt sein. So kann eine Variante, die wenig Daten erfasst auch auf einem Gerät mit einer großen Ausgabefläche eine sequentielle Navigation nahelegen.

Die Anforderung, die sich hieraus ergibt, ist, dass beide Navigationsarten aus einer technologieneutralen

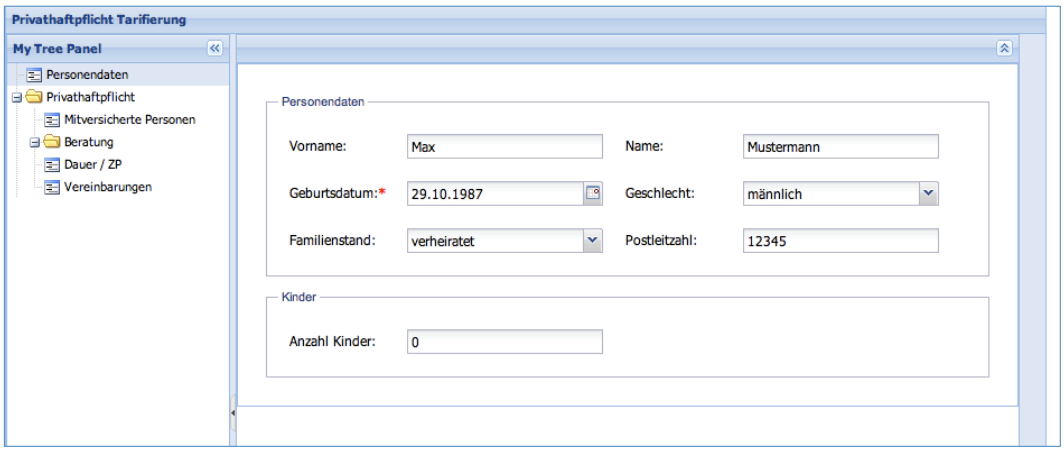

Abbildung 6: Hierarchische Navigation auf Geräten mit größerer Ausgabefläche

Beschreibung herleitbar sein müssen.

In der Literatur existierende Ansätze zur Modellierung webbasierter Anwendungen (z.B. UWE und WebML) modellieren die Navigation in einem gesonderten Modell explizit. Dies erfordert ein Synchronhalten der Seitenbeschreibung und der Navigation. Die gesonderte Modellierung ist dann begründbar, wenn beliebig innerhalb einer großen Anwendung navigiert werden muss und die Navigationspfade nicht herleitbar sind. Da die genannten Ansätze zum Ziel haben, ganze Portale modellieren zu können, ist die explizite Modellierbarkeit der Navigation notwendig.

Im Falle der datensammelnden Anwendungen kann ein einfacherer Weg gewählt werden, der die Navigation automatisch aufgrund struktureller Zusammenhänge herleiten kann. Möglich ist dies dadurch, dass potentiell alle Seiten durchlaufen werden müssen, um zu einem vollständig gefüllten Modell zu gelangen[6]. Alle Seiten müssen dazu durchlaufen werden, die aufgrund der bisher eingegebenen Daten relevant sind.

Es wird in der Arbeit angestrebt, eine automatische Herleitung zu erreichen, da dadurch die Gefahr von Inkonsistenzen zwischen Seiten- und Navigationsmodell vermieden werden kann und zudem der Aufwand einer expliziten Modellierung entfällt.

Die Anforderung, die sich hieraus ergibt, ist, dass ein Navigationsmodell bereits auf der Struktur der zu erfassenden Daten automatisch ableitbar sein soll. Diese Anforderung muss bei der Konzeption einer Lösung für die Modellierung der Darstellungseinheiten berücksichtigt werden (s. Abschnitt 5.3.1).

5.3.3 Ein- / Ausgabekomponenten

Die erfassten Daten sind unterschiedlichen Typs. Die Typinformationen werden in grafischen Benutzeroberflächen für eine Reihe von Aufgaben benötigt, welche die Bedienung der Anwendung vereinfachen. So können für die Eingabe der Daten unterstützende Eingabekomponenten (im Folgenden synonym auch als Widgets, Controls bezeichnet) verwendet werden, auf die jeweiligen Geräte und Interak-

[6]Das Modell ist dann als vollständig zu bezeichnen, wenn alle Informationen enthalten sind, die der nächste verarbeitende Schritt im Anwendungsablauf zur Durchführung benötigt. Abhängig von den bisher eingegebenen Daten kann die Menge variieren. So sind beispielsweise die Daten zu einem optionalen Produktbaustein in der privaten Haftpflichtversicherung erst nach Hinzunahme des Bausteins notwendig.

tionsformen zugeschnitten sind.

Die Funktionsweise und das Aussehen der Eingabekomponenten unterscheidet sich in unterschiedlichen Umgebungen und Technologien z.T. beträchtlich. So verwenden mobile Endgeräte häufig Eingabekomponenten, die auf die Bedienung mit Touchscreens optimiert sind, während im Desktop-Bereich Controls verwendet werden, die eine optimierte Eingabe mit Zeigegeräten ermöglichen. Typische Eingabekomponenten sind im Folgenden mit charakteristischen Spezifika aufgeführt, die bei einer technologieneutralen Beschreibung der Komponenten zu beachten sind. Abbildung 7 zeigt exemplarisch die grafische Darstellung einiger Elemente auf unterschiedlichen Ausgabegeräten.

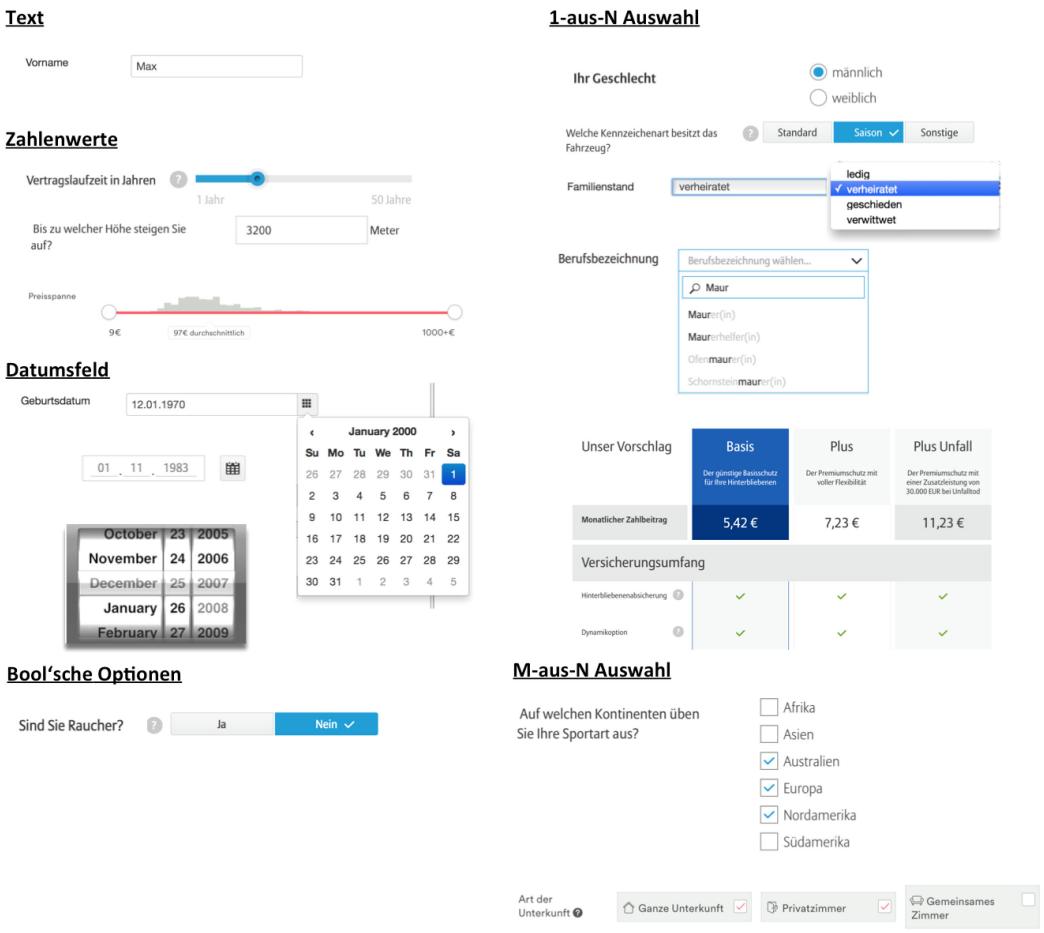

Abbildung 7: Eingabekomponenten auf unterschiedlichen Geräten

In [DFT05] werden grundsätzliche Elemente aufgeführt, die zum Aufbau einer Seite relevant sind. In Abbildung 8 sind Repräsentanten grafisch als Unterelemente von *UI Fragment* dargestellt. In [Tid06] findet sich eine weitergehende Typisierung dieser Elemente, an die wir uns im Folgenden anlehnen.

- **Texteingabe:** Eingabe von alphanumerischen Zeichenketten. In den meisten Fällen besitzen diese Felder eine weitergehende Semantik und damit verbundene Syntax, die bei der Eingabe eingehal-

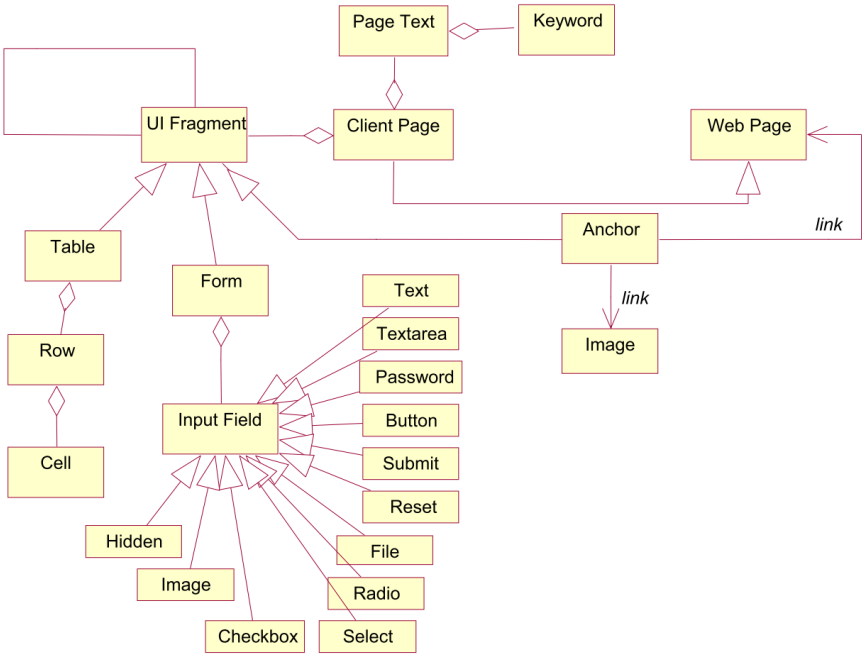

Figure 1: User Interface Fragments Model

Abbildung 8: UserInterface Fragments Model (nach DiLucca2005)

ten werden muss. Ein einfaches Beispiel hierfür ist ein Feld zur Eingabe einer E-Mail-Adresse, welche eine vorgegebene Struktur besitzt oder einer Vertragsnummer, die einen bestimmten Aufbau aufweist (z.B. PAV-263-374-3B2).

- **Texteingabebereiche:** Eingabe von Langtexten, die üblicherweise Freitexte sind und keinen zu prüfenden Regeln folgen.
- **numerische Eingabe:** Eingabe numerischer Informationen. Hier dürfen nur numerische Werte eingegeben werden. Diese Felder besitzen häufig einen eingeschränkten Bereich, aus dem Werte angegeben werden dürfen (value range) oder festgelegte Schrittweiten, in deren Rahmen ein Wert ausgewählt werden darf. Zudem gibt es hier verschiedene Typen (Ganzzahl, Fließkommazahl), für die ggf. weitere Einschränkungen gelten (z.B. Anzahl der Nachkommastellen bei einer Fließkommazahl).
- **binäre Auswahl:** Auswahl einer von zwei Möglichkeiten, z.B. *Ja- / Nein-Werte* oder Auswahl eines Geschlechts.
- **Aktivierungs-Schaltfeld (Checkbox):** Aktivierung eines Wertes i.S. einer Aktivierung.
- **Datumseingabe:** Eingabe von Datumswerten. Je nach Semantik des Datums (Geburtstag, Vertragsbeginn, historische Daten) kann ein umsetzendes Control einen Vorschlag für ein Startdatum machen. Dies dient insbesondere einer schnelleren Erfassung. Ein Geburtstag beispielsweise

liegt üblicherweise für einen Versicherungsnehmer mehr als 20 Jahre in der Vergangenheit; ein Vertragsbeginn hingegen ist ein Datum, das in der Zukunft liegt und mit dem heutigen Datum beginnen kann.

- **Zeiträume:** Verallgemeinerte Form der Datumseingabe, meist über zwei Daten oder einen Zeitraum mit oder ohne Startdatum beschrieben.
- **1-aus-N-Auswahl:** Auswahl eines Wertes aus einer Liste möglicher Ausprägungen. Hierbei müssen die auszuwählenden Werte festgelegt und ggf. ein initialer Wert angegeben werden können. Für diesen Feldtyp existieren meist unterschiedliche Eingabeelemente für einer geringe oder große Zahl möglicher Ausprägungen (z.B. Radiobox-Gurppen vs. Drop-Down-Listen) oder weitere Einschränkungen wie die sortierte Darstellung der Auswahlmöglichkeiten.
- **M-aus-N-Auswahll:** analog der 1-aus-N-Auswahl, jedoch mit der Möglichkeit, mehrere Werte aus dem Wertevorrat auszuwählen.
- **Aktions-Schaltfläche (Button, Link):** Auslösen einer Nutzeraktion. Hierbei muss angegeben werden können, welche Aktion ausgeführt werden soll, und welche Daten aus dem Modell hierzu benötigt werden (s. 5.3.8)

Die aufgeführten sind **grundlegende Typen von Eingabeelementen**, die keine fachlichen Spezifika besitzen und so in allen Anwendungen vorkommen können. In einem fachlichen Kontext werden ggf. zusätzlich Felder benötigt, die eine weitergehende Semantik und Funktionalität besitzen (**fachliche Eingabeelemente**). Dies sind häufig Inhalte, die einer speziellen Syntax folgen bzw. dem Nutzer zu einer vereinfachten Eingabe über eigene Controls angeboten werden. Beispiele hierfür sind Controls zur strukturierten Eingabe von Vertrags- oder Seriennummern in einem bestimmten Format (structured format fields, [Tid06]) oder die Auswahl einer Farbe aus einer Farbpalette. Diese Controls sind meist aus grundlegenden Elementen zusammengesetzt, welche jedoch in einer spezifischen Beziehung zueinander stehen. Diese müssen im Modell ebenfalls als eigene, fachliche Typen beschrieben werden können.

Um eine technologieneutrale Beschreibung der oben genannten Feldtypen zu erreichen, müssen dort alle Informationen enthalten sein, die für eine etwaige Darstellung durch ein Control benötigt werden - eine abstrakierte Beschreibung der Feldtyp-Eigenschaften. In Tabelle 9 sind auszugsweise die grundlegenden Eigenschaften der Feldtypen aus obiger Aufzählung aufgeführt, die wir nach heutigem Stand identifiziert haben und die als Grundlage in der prototypischen Umsetzung verwendet werden sollen.

Aus den vorangegangenen Betrachtungen ergibt sich als Anforderung, dass sowohl für die grundlegenden Feldtypen als auch für etwaige fachlich erweiterte Feldtypen ein abstrakte Beschreibung gefunden werden muss, aus welcher konkrete Eingabeelemente hergeleitet werden können. Hierzu müssen die gefundenen Eigenschaften in technologieneutraler Weise abgebildet werden.

5.3.4 Reaktion auf Änderungen: Validierung von Eingabedaten

Zeitgemäße grafische Oberflächen zeigen dem Nutzer Fehleingaben frühzeitig an. Um dies zu erreichen, benötigt eine Oberfläche möglichst viele Informationen über die einzugebenden Daten, die eine Validierung nach erfolgter Eingabe erlauben.

Die erste Ebene von Prüfungen ist die *syntaktische Validierung*. Hierbei werden die Eingaben auf syntaktisch korrekte Eingabe geprüft. Hierzu müssen Regeln für die Prüfung definiert werden können. Wurden die Daten fehlerhaft eingegeben, muss dem Nutzer eine Fehlermeldung ausgegeben werden.

Beispiel: Eingabe einer ungültigen Postleitzahl.

field type	specification needs	possible controls (samples)	remarks
*= optional information			
textinput	maxlength* format* displayonly*	textfield	might have an optional fixed format constraint. this has to be checked as validation. the content is still a simple text.
longtext	maxlength* size* displayonly*	textarea	usually not checked for validity.
number (integer)	min* max* step* unit (out of n)* displayonly*	numberfield slider knob spinner	numbers might be restricted to certain min./max. values and might just be selected in certain steps. They might have different units (eg. one of 'days', 'month','years' or currencies).
number (float)	min* max* decimals* unit* displayonly*	numberfield with decimals	float numbers might be restricted to certain min./max. values. They might have different units (eg. currencies).
numberrange (integer/float)	min max unit subrange displayonly*	subrange-slider subrange-input	need min/max values. might need steps. The selection contains two values (subrange)
binary input	values (2) displayonly*	checkbox yes/no button switch	
date	day month year displayonly*	textfield (formatted) date picker	usually results in a texfield containing the date
time	timestamp displyonly*		
timerange	startDate (see date) duration (days)* endDate (see date)* displayonly*		
1-of-N	values (N) valuetype* displyonly*	small N: radio group big N: dropdown, SearchBox	values are a fixed list or might get retrieved from a datasink and filtered depending on other values in the model
M-of-N	values(N) valuetype* displayonly*	small N: Checkbox group big N: List	values are a fixed list or might get retrieved from a datasink and filtered depending on other values in the model

Abbildung 9: Eigenschaften grundlegender Eingabeelemente

In Abbildung 10 ist die Reaktion auf die Eingabe einer ungültigen (in diesem Falle zu langen) Postleitzahl dargestellt. Die Reaktion in einer Desktop-spezifischen Variante der Anwendung kann darin bestehen, dass eine Box mit einer Warnmeldung geöffnet wird. Eine nutzerfreundlichere Variante bestünde in der roten Umrahmung des Feldes und Ausgabe der Fehlermeldung direkt beim Eingabefeld.

Die zweite Ebene der Prüfungen findet auf der semantischen Ebene statt. Diese *semantische Validierung* benötigt zur Prüfung weitere Informationen aus dem Kontext und erfordert meist eine programmatische Prüfung. Die *semantische Validierung* kann wiederum in zwei Kategorien eingeteilt werden: lokale

Abbildung 10: Fehlermeldung einer syntaktischen Validierung

Validierungen und solche, die serverseitig ausgeführt werden müssen, da Sie lokal nicht verfügbare Informationen benötigen.

Beispiel: Prüfung der Anzahl mitversicherter Wohnungen beim Zusatzbaustein ”Vermieter Plus”.

In Abbildung 11 ist die Reaktion der Anwendung auf die Angabe einer zu großen Zahl zu versichernder Mietwohnungen dargestellt. Die Validierung beruht auf der im Modell vorhandenen Information, wieviele Wohnungen des Vermieters mitversichert werden sollen.

Abbildung 11: Fehlermeldung einer semantischen Validierung

Diese beiden Formen der Validierung müssen in der technologieneutralen Beschreibung berücksichtigt werden. Es müssen ggf. komplexe Operationen für die Validierung erfolgen, die die Ausführung von Programmcode erfordern. Da dieser Programmcode technologiespezifisch sein wird (insbesondere bei

einer clientseitigen Prüfung), muss eine technologieneutrale Darstellung für die Operationen gefunden werden.

5.3.5 Reaktion auf Änderungen: Sichtbarkeitssteuerung

Soll die Oberfläche der Anwendung in einem Schritt weitgehend autark agieren können, bedeutet dies, dass alle Darstellungseinheiten, Feldgruppen und Felder bereits auf dem Client verfügbar sind. Jedoch sind nicht alle Eingabebereiche zu allen Zeitpunkten relevant. Vielmehr sind manche Bereiche erst sicht- oder editierbar, wenn bestimmte Eingaben getätigt wurden.

Ein Beispiel für die *Editierbarkeit* abhängig von getätigten Eingaben ist die Angabe eines Geburtsdatums, welches mehr als 18 Jahre in der Vergangenheit liegt. In der gewählten Beispielanwendung führt diese Eingabe in den Personendaten dazu, dass auf der Seite zum Versicherungsprodukt kein ”junge Leute Rabatt” ausgewählt werden kann, da dieser Rabatt nur bis zu einem Alter von 18 Jahren gewährt wird. In Abbildung 12 ist diese Auswahl auf der Seite ”Personendaten” dargestellt und die Auswirkung der Auswahl auf der Seite ”Beratung”.

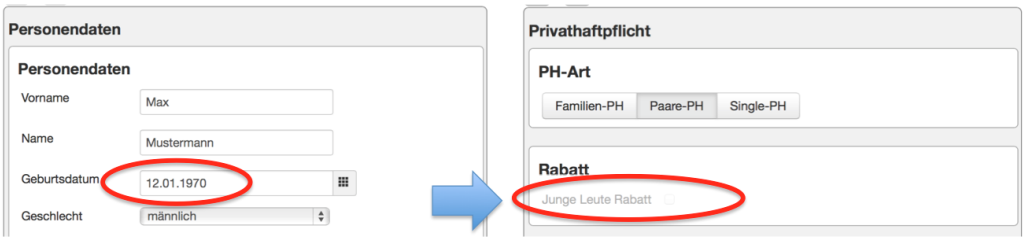

Abbildung 12: Deaktiviertes Feld aufgrund einer Eingabe

Ein Beispiel für *Sichtbarkeit* von Elementen abhängig von getätigten Eingaben ist die Darstellung von optionalen Bausteinen basierend auf der gewählten Produktausprägung. Auf der Seite ”Beratung” kann hierzu ein Basisbaustein der privaten Haftpflichtversicherung ausgewählt werden (siehe Abbildung 13). Wählt der Nutzer den Basisbaustein ”PH Basis”, so werden keine Zusatzbausteine angezeigt. Wählt er hingegen die Variante ”PH mit einer Deckungssumme von 15 Mio. Euro”, stehen anschließend zusätzliche Bausteine zur Auswahl. Die bestehenden Eingabebereiche werden in diesem Fall nach unten verschoben.

Um die Sichtbarkeit und Editierbarkeit von Bereichen steuern zu können, muss in einer technologieneutralen Weise die Beschreibung der hierzu benötigten Regeln erfolgen. Die Formulierung der Regeln muss auf den lokal verfügbaren Daten erfolgen können - analog dem beschriebenen Vorgehen bei der Validierung.

5.3.6 Reaktion auf Änderungen: Dynamische Inhalte

Insbesondere bei Auswahllisten ist eine häufig vorkommende Anforderung, dass die Auswahloptionen abhängig von getätigten Eingaben variieren können. Der Nutzer tätigt eine Eingabe und die Auswahlliste wird inhaltlich modifiziert. Diese Aktualisierung erfordert meist eine Serveranfrage, da die Daten ggf. sehr umfangreich werden können. Es ist jedoch auch möglich, dass die dynamische Befüllung auf lokal

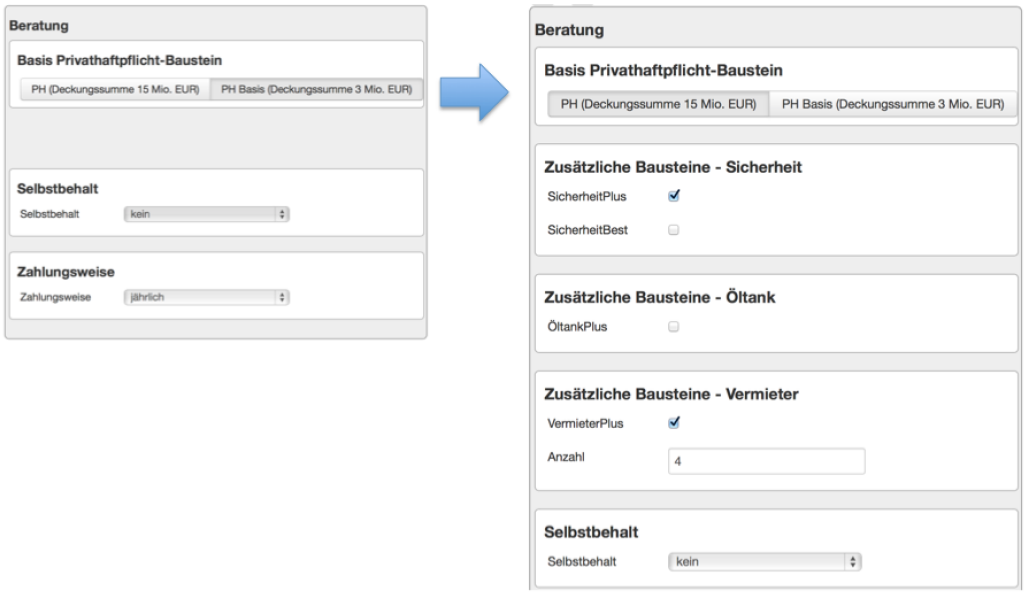

Abbildung 13: Sichtbarkeit von Seiten und Elementen aufgrund einer Eingabe

vorhandenen Daten erfolgen kann, indem eine Filterung der möglichen Ausprägungen vorgenommen wird.

Ein Beispiel für eine solche Abhängigkeit ist die Darstellung möglicher Orte in einem Adresseingabe-Schritt. Der Nutzer beginnt mit der Eingabe einer Postleitzahl und eine Auswahlliste für den Ort wird mit allen Orten befüllt, deren Postleitzahl mit den bereits eingegebenen Ziffern beginnt. Da die Liste aller Städtenamen mit ihren Postleitzahlen zu umfangreich für eine lokale Haltung ist, sollte hier eine Anfrage an den Server erfolgen, welche die Liste der passenden Städtenamen liefert.

Um die Befüllung dynamisch vornehmen zu können, ist in der Dialogbeschreibung eine entsprechende Angabe vorzusehen.

5.3.7 Reaktion auf Änderungen: Anpassung der Navigation

Wie bereits in den Abschnitten 5.3.1 5.3.2 und 5.3.5 beschrieben, ist es notwendig, dass Teile der Oberfläche einer Anwendung basierend auf bereits getätigten Eingaben ein- bzw. ausgeblendet werden. Handelt es sich bei diesen Bereichen um ganze Darstellungseinheiten, hat dies auch Einfluss auf die Navigation. Ausgeblendete Seiten können in einem solchen Fall nicht angesteuert werden und müssen aus der Navigation entfernt werden.

Ein Beispiel für einen solchen Fall findet sich in der Beispielanwendung bei der Angabe zur Art der Privathaftpflicht-Versicherung (siehe Abbildung 14). Hier kann ausgewählt werden, ob es sich um eine Single-, Paare- oder Familien-Haftpflichtversicherung handelt. Im Falle einer Single-Haftpflicht ist die Angabe der mitversicherten Personen nicht relevant. Wird jedoch eine Paare-Haftpflichtversicherung tarifiert, so erscheint in der Navigation ein weiterer Menüpunkt ”Mitversicherte Personen”. Dieser führt auf eine Seite, auf der der ”Lebenspartner in häuslicher Gemeinschaft” mit seinen für die Tarifierung relevanten Daten angegeben werden muss.

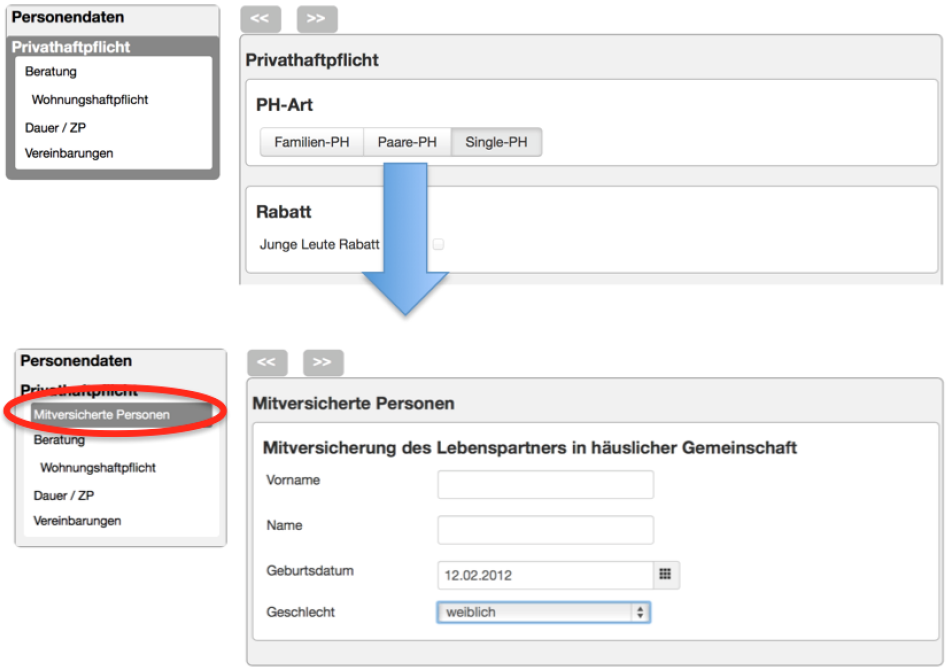

Abbildung 14: Anpassung der Navigation aufgrund einer Eingabe

Um dies abbilden zu können, muss eine hierarchische Navigation auf die Steuerung der Sichtbarkeit reagieren und ggf. Navigationspunkte ein- / ausblenden. Im Fall einer sequentiellen Navigation innerhalb der Anwendung muss die Sichtbarkeit berücksichtigt und unsichtbare Seiten übersprungen werden (zu Navigationsformen in datensammelnden Anwendungen vgl. Abschnitt 5.3.2).

5.3.8 Benutzer- und Elemetaktionen

Unter Benutzeraktionen wird das Auslösen der Ausführung von Programmcode verstanden, wie es beispielsweise durch Betätigen von Menüeinträgen, Buttons oder Links erfolgt. Die Reaktion auf eine solche Aktion ist meist komplexer und fachlicher Natur und benötigt deshalb Programmcode, der bei Aktivierung ausgeführt wird. Neben explizit durch den Benutzer angestoßenen Aktionen existiert zudem der Bedarf an der Ausführung von Programmcode, der durch Ereignisse angestoßen wird, die an einem Element der Benutzerschnittstelle auftreten: ein Beispiel hierfür ist die Darstellung eines Hilfetextes beim überstreichen eines Feldes oder einer Aktion, die bei Änderung eines Feldinhaltes auszuführen ist (z.B. die Beschaffung von Vertragsdaten nach Eingabe einer Vertragsnummer, die in weiteren Feldern zur Vorbefüllung dienen).

Aktionen sind mit Darstellungselementen assoziiert, benötigen ggf. Informationen aus dem Datenmodell der Anwendung und manipulieren ggf. Bereiche des Datenmodells. So können Aktionen existieren, die auf die Inhalte einer ganzen Seite wirken (z.B. die Vorbelegung von Kundendaten aus einer Kundendatenbank, die Gruppen zugeordnet sind (z.B. die Ableitung und Vorbelegung einer IBAN, BIC-Kombination aus der Angabe einer bisherigen Bankverbindung), oder mit einem Feld assoziiert sind.

Zudem können aus Sicht der Anwendung globale Operationen existieren, die analog der Eingabefelder existieren (z.B. der ”Absenden”-Knopf oder das explizite Speichern/Laden bereits eingegebener Daten).

Zur Beschreibung solcher Aktionen in *mimesis.ui* sind folgende Kategorien und Informationen vorzusehen, die später zur Anbindung konkreter Implementierungen für Aktionen verwendet werden sollen. Es sollen folgende Kategorien von Aktionen vorgesehen werden, die aus der bisherigen Analyse hervorgegangen sind und ggf. noch um weitere Kategorien im Verlauf der Arbeit erweitert werden:

Nutzeraktionen: Diese werden als explizit vom Nutzer anzustoßende Aktionen an einem Element (Seite, Unterseite, Gruppe, Element) aufgefasst und daher nahe dem Element explizit angeboten. Die Hierfür benötigten Informationen sind:

- Aktionsname: eindeutige Identifikation für die auszuführende Operation
- Parameter: Referenzen auf Modellinhalte als Parameter der Operation (Ein-/Ausgabe)

Elementaktionen: Diese Aktionen werden über Ereignisse aktiviert, die am Element auftreten können. Hierbei sollen die im Umfeld grafischer Oberflächen üblichen Ereignisse unterstützt werden (z.B. analog der im HTML-Umfeld üblichen Key- und Mouse-Events, vgl. http://www.w3schools.com/tags/ref_eventattributes.asp)

- Aktionsname: eindeutige Identifikation für die auszuführende Operation
- Parameter: Referenzen auf Modellinhalte als Parameter der Operation (Ein-/Ausgabe)
- Art der Aktivierung: z.B. onchange, onselect, onenter, onleave, onkeydown, onkeyup etc.

Featurespezifische Aktionen: (@: benötigt weitere Überarbeitung/Erläuterung). Events, die für Features der Oberfläche verwendet werden. Diese sind optional und werden für ein bestimmtes Feature definiert, welches ein Element besitzen soll. Beispiele hierfür sind das Anbieten des Features ”Hilfe” oder das Angebot möglicher Eingabewerte aus einer serverseitigen Liste basierend auf der bisherigen Eingabe (Autocomplete).

- typ des Features: z.B. *help* oder *autocomplete*
- Aktionsname: eindeutige Identifikation für die auszuführende Operation
- Parameter: Referenzen auf Modellinhalte als Parameter der Operation (Ein-/Ausgabe)
- Art der Aktivierung: z.B. onchange, onselect, onenter, onleave, onkeydown, onkeyup etc.

Nutzer- und Elementaktionen sind dabei Kategorien, die für eine grundsätzliche Funktionsweise der Anwendung notwendig sind und sich auf datenrelevante Operationen beziehen. Die featurespezifischen Aktionen sind für Operationen vorgesehen, die üblicherweise in Oberflächen auftreten, jedoch unterstützenden Charakter haben und damit nicht notwendigerweise in einer finalen Oberfläche umgesetzt werden müssen.

5.3.9 Fehlerbehandlung und Darstellung

Das Thema wurde in der aktuellen Fassung zurückgestellt und wird in späteren Versionen ausgearbeitet.

5.3.10 Datenmodell, Bindung an Eingabefelder und Initialisierung

Die in den letzten Abschnitten formulierten Anforderungen basieren häufig auf der Angabe von Regeln basierend auf bereits erfassten Daten. Diese Daten müssen in einer geeigneten Form vorgehalten werden.

Dazu bedarf es eines lokalen Datenmodells, auf das die Funktionalitäten zugreifen können. Wie dieses Modell umgesetzt wird, ist abhängig von der verwendeten Technologie. Es kann jedoch ein Modell abstrakt definiert werden, auf dem der Anwendungsteil arbeitet.

Abbildung 15 zeigt einen Ausschnitt des fachlichen Modells, auf dem die betrachtete Beispielanwendung arbeitet. Hier gibt es einen Bereich ”Personendaten”, in welchem die Daten des Versicherungsnehmers verzeichnet sind und einen Bereich ”Tarifdaten”, welcher (weiter untergliedert) die Daten des gewählten Produkts abdeckt.

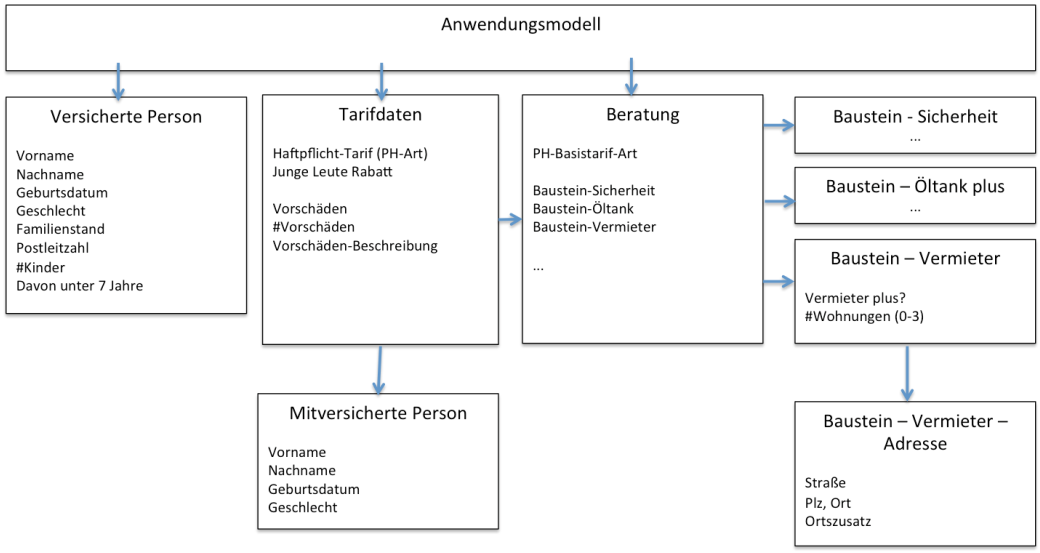

Abbildung 15: Ausschnitt aus dem fachlichen Modell der Beispielanwendung

Die Eingabefelder der Oberfläche korrespondieren mit Datenfeldern in diesem Datenmodell. Um eine eindeutige Korrelation zwischen Eingabefeld und Modelldaten zu ermöglichen, muss diese Zuordnung in der Dialogschnittstellenbeschreibung erfolgen. Theoretisch ließe sich ein solches Modell automatisch herleiten, indem man eine seitenbezogene Speicherung der Daten vornimmt. Da aber üblicherweise das Datenmodell einer Anwendung eine andere Struktur aufweist, als dessen Präsentation an der Oberfläche, ist eine explizite Bindung des Eingabefeldes auf ein von der Dialogschnittstelle unabhängiges Modell zielführender.

Es muss eine technologieneutrale Notation gefunden werden, die eine solche Bindung beschreibt und festlegt, mit welcher Stelle im Datenmodell der Anwendung ein Eingabefeld korrespondiert.

5.4 Betrachtungen zur Multikanalfähigkeit und Bildung von Varianten

Die Multikanalfähigkeit einer Anwendung in ihrer Gesamtheit kann nicht allein durch die Anwendungsoberflächen abgehandelt werden, da hierbei fachliche Faktoren eine Rolle spielen. Die in den bisherigen Veröffentlichungen zum Projekt verwendete Unterscheidung zwischen *fachlichen und technischen Kanälen* ([Hit13]) ist bei der Unterscheidung hilfreich, welche Aspekte von einer Dialogschnittstellenbeschreibung abgedeckt werden können und welche an anderer Stelle behandelt werden müssen.

In bestehenden Ansätzen wird die Multikanalfähigkeit einer Anwendung häufig darauf reduziert, dass Ausgaben für unterschiedliche Geräte erzeugt werden und ggf. abhängig von Ausgabegeräten Daten bei der Erfassung ausgefiltert werden. Diese Auffassung findet man vor allem im Bereich webbasierter Anwendungen, die mit HTML5-Mitteln erstellt werden - vermutlich aufgrund der Möglichkeiten, die HTML5 hier bietet (Stichworte: CSS, responsive design). Es handelt sich um eine von der Fachlichkeit befreite, technisch motivierte Sichtweise.

Dabei wird außer Acht gelassen, dass Multikanalfähigkeit, wie sie im Rahmen von mimesis verstanden wird, auch beinhaltet, dass inhaltlich unterschiedliche Varianten die Multikanalfähigkeit einer Anwendung bedingen.

Dennoch können einige Aspekte insbesondere der *technischen Kanäle* durch eine abstrakte Oberflächenbeschreibung, wie sie hier vorgeschlagen wird, abgehandelt werden

Eine technologieneutrale Beschreibung der Anwendung trägt dazu bei, dass die einmalig erstellten Beschreibungen in unterschiedlichen technologischen Kontexten verwendet werden können. Eine hierarchisch strukturierte Beschreibung gestattet zudem, bei der Transformation der Beschreibung auf Eigenschaften der Ausgabegeräte reagieren zu können. Die abstrakte Beschreibung der Eingabefelder mit den zusätzlichen dynamischen Eigenschaften (Sichtbarkeitssteuerung, Validierung, Aktionenverarbeitung etc.) erlaubt zudem die Erzeugung einer für das Endgerät angemessenen Oberfläche.

Eine abstrakte Dialogschnittstellenbeschreibung eignet sich jedoch nur unzureichend dazu, Granulariätsvarianten automatisch zu erzeugen, die auf fachlichen Anforderungen beruhen - z.B. eine fachlich korrekte Auswahl der zu erfragenden Felder für eine spezifische Nutzergruppe vorzunehmen. Die ließen sich zwar in der Beschreibung unterbringen (z.B. als Regeln zum ein-/ausblenden von Bereichen), würden die Definition bei vielen Varianten jedoch schnell unübersichtlich werden.

Daher ist eine Trennung der fachlichen und technischen Sicht angebracht und wird im Rahmen von mimesis verfolgt. Das hier beschriebene Teilprojekt *mimesis.ui* deckt somit einen Teil der Anforderungen der *technischen* Kanäle ab. Die fachlichen Aspekte werden in einem vorgelagerten Transformationsschritt im Teilprojekt mimesis.model abgehandelt.

6 Technologieneutrale Dialogschnittstellenbeschreibung in *mimesis.ui*

In den folgenden Abschnitten wird eine technologieneutrale Beschreibung für Benutzerschnittstellen vorgestellt, welche anhand der in den letzten Abschnitte gestellten Anforderungen im Projekt *mimesis.ui* erstellt wurde. Hierzu wird zuerst das der Beschreibung zugrundeliegende Metamodell vorgestellt und grundlegende Entscheidungen dargestellt, die bei der Ableitung der in *mimesis.ui* verwendeten Beschreibung getroffen wurden. Die anschließende Darstellung des Aufbaus der Benutzerschnittstellenbeschreibung erfolgt dann entlang der im letzten Kapitel gefundenen Anforderungsbereiche.

6.1 Die Grundlage: mimesis Metamodell

Aus den Anforderungen aus Abschnitt 5.3 wurde ein Metamodell abgeleitet, welches die gefundenen strukturellen und verhaltensrelevanten Merkmale abbildet. Diese sind hier noch einmal aufgezählt:

Strukturelle Merkmale

- Darstellungseinheiten (Seiten, Unterseiten, Gruppierung)
- Navigation zwischen Darstellungseinheiten
- Ein-/Ausgabekomponenten

Verhaltensrelevante Merkmale

- Reaktion auf Änderung: Validierung von Eingaben
- Reaktion auf Änderung: Sichtbarkeitssteuerung
- Reaktion auf Änderung: Dynamische Inhalte
- Reaktion auf Änderung: Anpassung der Navigation
- Benutzeraktionen
- Fehlerbehandlung und Darstellung

Generelle Anforderungen

- Datenhaltung und -Bindung
- Initialisierung des Modells

In Abbildung 16 ist das abgeleitete Metamodell in Form eines UML-Diagramms dargestellt. Die Beschreibung der Benutzerschnittstelle (*uidescription*) besteht dabei aus einer geordneten Liste von **Seiten** (*Pages*), die wiederum eine geordnete Liste von **Seitenelementen** (*PageElements*) beinhaltet. Ein **Seitenelement** kann entweder ein Ein-/Ausgabefeld sein (*Field*) oder eine Gruppe von weiteren Seitenelementen sein (*Fieldgroup*). Zudem können zu einer *Seite* Unterseiten existieren, um die in Abschnitt 5.3.1 angeführten hierarchischen Beziehungen abzubilden. Durch die Schachtelung von Seitenelementen kann innerhalb einer Seite die Gruppierung und Zusammengehörigkeit der Felder erreicht werden und damit die geschlossene Behandlung z.B. für die Sichtbarkeitssteuerung erreicht werden.

Die hierarchische Modellierung von Seiten und Unterseiten gestattet es, aus dem Modell eine **Navigation** zu erzeugen, wie sie in Abschnitt 5.3.2 gefordert wurde. Es kann durch die modellierten Zusammenhänge sowohl ein Baum für eine hierarchische Navigation erzeugt als auch aufgrund der *ordered*-Eigenschaft eine Reihenfolge für eine sequentielle Navigation durch die Seiten und Unterseiten abgeleitet werden. Die **Anpassung der Navigation**, wie sie in Abschnitt 5.3.7 dargestellt wurde, kann über Sichtbarkeitsregeln der *Page*-Elemente gesteuert werden (*exists*).

Die Attribute von *PageElement* und *Field* dienen der Beschreibung der **Ein-Ausgabekomponenten** die in der Benutzerschnittstelle vorkommen. Sie umfassen Informationen zum Typ des Feldes (Details in Abschnitt 6.3.5), als auch der verhaltensrelevanten Merkmale wie **Bedingungen zur Sichtbarkeitssteuerung** (*exists* und *editable*), die Beschreibung der Validierungen und Benutzeraktionen (*actions*) die auf einem Element möglich sind.

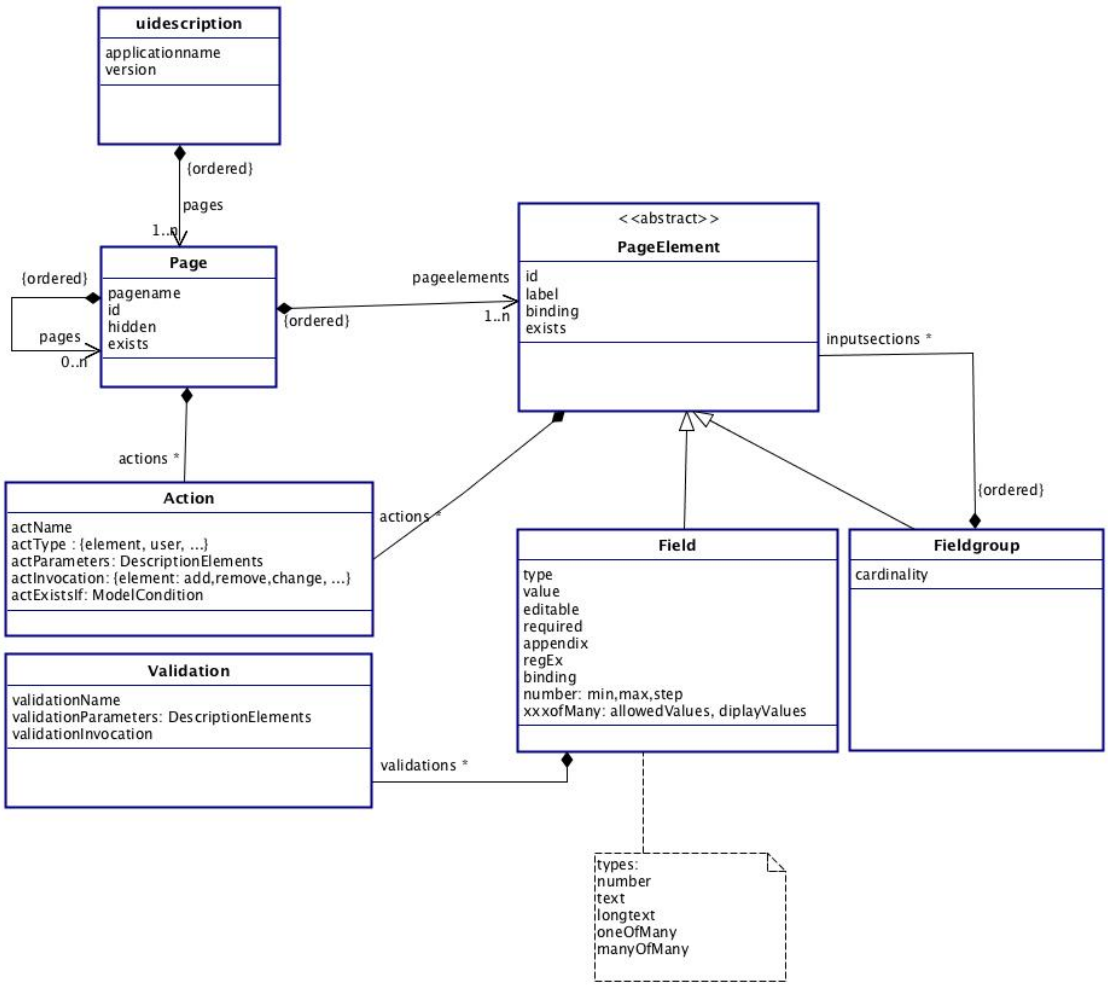

Abbildung 16: mimesis.ui Metamodell

Die **Validierung von Eingaben** kann dabei einerseits syntaktisch über die Angabe regulärer Ausdrücke erfolgen (*regEx*-Attribut) andererseits programmatisch über die Angabe einer Validierungsoperation erfolgen (*Validation* in Abbildung 16). Hierbei kann angegeben werden, welche Operation mit welchen Parametern verwendet werden soll und zu welchem Zeitpunkt die Validierung erfolgen soll (*validationInvocation*; z.B. während der Eingabe oder nach erfolgter Eingabe).

Aktionen (*Action*) und **Reaktionen** auf bestimmte Ereignisse können für Seiten und alle Seitenelemente spezifiziert werden. Hierzu kann ein Typ (nutzer oder element-getriggert) angegeben werden, welche Operation mit welchen Parametern gerufen werden soll und welches Ereignis die Aktion auslöst (z.B. bei Hinzufügen oder Änderung des Elements oder durch Nutzeraktivierung).

Zur Thematik aufrufbarer Operationen vergleiche Abschnitt 6.2.3.

Die Details zu den einzelnen Aspekte und deren konkrete Modellierung werden in den folgenden Abschnitten dargestellt.

6.2 Grundlegende Festlegungen für die Dialogschnittstellenbeschreibung

6.2.1 Ein eigenes Modell und Beschreibungssprache für die Oberflächenbeschreibung

Es existiert eine Reihe von Beschreibungssprachen und Modelle für Benutzerschnittstellen, die sich in Teilen mit den gefundenen Anforderungen decken (vgl. Arbeiten im Umfeld). Häufig sind diese Modelle jedoch auf bestimmte Technologien fokussiert (z.B. auf webbasierte Anwendungen) oder modellieren Aspekte über Konzepte, die sich nicht mit unseren Anforderungen decken. Zudem basieren die meisten Ansätze auf einer technischen Umsetzung mit XML, was die Integration auf unterschiedlichen Plattformen (insbesondere JavaScript-basierte Umgebungen) erschwert und den Einsatz nicht praktikabel erscheinen lässt (vgl. Abschnitt 6.2.2). Auf eine detaillierte Diskussion der Entscheidungskriterien wird hier verzichtet.

Der in *mimesis.ui* verfolgte Ansatz soll zu einer einfachen, auf die wesentlichen Aspekte fokussierten Beschreibung führen, die wir in unseren Arbeiten gefunden haben. Aus diesem Grund wird eine eigene Beschreibungssprache in Form einer DSL (*domain specific language*) verwendet, welche sich auf Eigenschaften des gefundenen Metamodells konzentriert.

6.2.2 JSON als Grundlage für die Beschreibungssprache

Die *mimesis.ui DSL*, die im Folgenden entwickelt wird, beschreibt die Dialogeinheiten, die in einem *datensammelnden Schritt* der Anwendung benötigt werden und deren Abhängigkeiten untereinander. Daraus wird für technische Plattform die endgültige Dialogfolge in einer konkreten Technologie erzeugt.

Die Transformation der technologieneutralen Benutzerschnittstellenbeschreibung in eine konkrete Technologie kann prinzipiell zu unterschiedlichen Zeitpunkten (statisch zum Entwicklungszeitpunkt, dynamisch zur Laufzeit) und auf unterschiedlichen Plattformen erfolgen (server- oder clientseitig)

Da die Transformation an unterschiedlichen Stellen erfolgen kann, muss für die DSL als Basis ein Format gewählt werden, das sowohl server-, als auch clientseitig effizient zu verarbeiten ist. Zudem sollte das Format möglichst wenig Speicherplatz beanspruchen, da ein großer Speicherbedarf insbesondere im mobilen Umfeld kritisch zu sehen ist[7].

Als Basis für *mimesis.ui DSL* bietet sich die Java Script Object Notation (JSON) als Basis an. Eine grundlegende Eigenschaft von JSON ist die beispielsweise im Vergleich zu XML höhere Platzeffizienz[8]. Bei einer clientseitigen Verarbeitung in einem HTML5 basierten Umfeld bietet JSON zudem den Vorteil, dass es in Browsern ein ”first class citizen” ist und sehr einfach in JavaScript-Objekte gewandelt werden kann. Eine Interpretation von XML oder einem eigenen Datenformat wäre im Vergleich deutlich aufwändiger. Durch die einfache Struktur des Formates kann es auf allen Plattformen eingesetzt werden und existieren in allen gängigen Programmiersprachen unterstützende Bibliotheken zur Verarbeitung von JSON.

Die Nachteile von JSON liegen darin, dass die Sprache weniger formal ist als beispielsweise XML. So ist kein *Standardmechanismus* vorgesehen, welcher sich mit der Validierung des Formats befasst. In XML gibt es hierfür das Konzept der XML-Schemata bzw. *document type definitions* (DTD). Für JSON existieren jedoch zusätzliche Bibliotheken, mit denen dies ebenfalls erreicht werden kann (siehe z.B.

[7] Mobile Geräte besitzen einen kleineren Arbeitsspeicher und weniger effiziente Prozessoren als Desktop-Geräte. Schon aus diesem Grund muss auf eine effiziente Speichernutzung geachtet werden. Aber auch die Konnektivität solcher Geräte ist eingeschränkt und variiert. So ist der Verbindungsaufbau zeitintensiv, die Verbindungsgeschwindigkeit variabel und das Volumen ggf. beschränkt.

[8] JSON besitzt eine einfachere Struktur als XML und wirkt lesbarer. Weitere Aspekte sprechen noch dafür, auf die an dieser Stelle aber nichtt eingegangen wird. Für einen Überblick vgl. *http://de.wikipedia.org/wiki/JavaScript_Object_Notation*

http://json-schema.org). Diese können bei Bedarf in einem Projekt eingesetzt werden.

6.2.3 Operationsdefinitionen in *mimesis.ui*

Da an einigen Stellen im Modell der Aufruf von Programmlogik erfolgen muss (z.B. bei Validierungen, Aktionen und Reaktionen, dynamischen Inhalten etc.), soll an dieser Stelle eine grundlegende Entscheidung hinsichtlich des Umgangs mit Programmlogik in mimesis erfolgen.

Es existieren einige Ansätze, Programmcode technologieneutral zu formulieren, u.a. die Verwendung von Scripting-Sprachen, die durch Bereitstellung eines Interpreters für unterschiedliche Ablaufumgebungen die einmalige und technologieneutrale Formulierung des Programmcodes gestatten würden. Eine solche Formulierung führt jedoch zu einer sehr komplexen Definition des Modells und bedingt die Erstellung eines Interpreters oder Compilers, der aus der technologieneutralen Formulierung ausführbaren Code erzeugt. Dies ist in der Praxis nicht mit vertretbarem Aufwand handhabbar.

Die Logik, die in datensammelnden Anwendungen verwendet wird, arbeitet meist auf Elementen, die im Datenmodell der Anwendung / des Clients vorhanden ist. Hier dienen Elemente des Modells einerseits als Eingabe für Operationen (*input Daten*), andererseits werden Elemente des Modells durch die Operationen verändert (*output Daten*). Ggf. werden Aufrufe an Backends vorgenommen, die jedoch auf den Zustand des Frontends keine über die *input* und Manipulation der *output Daten* hinausgehende Wirkung haben.

Im Rahmen von mimesis beschränken wir uns daher auf eine abstrahierte Beschreibung hinsichtlich der Operationen, die in einem Frontend auftreten können. Die Arbeitshypothese ist dabei, dass für die Beschreibung einer Operation lediglich folgende Eigenschaften benötigt werden:

- Eindeutige Bezeichnung der Operation
- Angabe der Eingabeparameter als Referenzen auf das zugrundeliegende Datenmodell
- Angabe der manipulierten Felder des Datenmodells

Es wird davon ausgegangen, dass eine Schnittstellenbeschreibung für alle möglichen Funktionen angegeben werden kann. Dadurch kann von der konkreten Implementierung abstrahiert werden und im Modell der Anwendung eine Referenz auf die Schnittstelle angegeben werden. Eine Prüfung auf Korrektheit der Implementierung ist damit zwar nicht möglich, jedoch kann die Prüfung auf Einhaltung der Schnittstelle erfolgen. Die konkrete Implementierung der Schnittstelle kann dann für die jeweilige Plattform gesondert erfolgen und dort vorhandene Technologien nutzen (z.B. die Implementierung in JavaScript oder Java unter Nutzung dort vorhandener Bibliotheken).

Im Falle eines JavaScript-clients kann dies als JavaScript-Funktion erfolgen. Im Falle von Java in einer Rich Client-Umgebung kann dies per Java-Reflection auf eine Klasse erfolgen, in welcher eine entsprechende Methode definiert ist. Die Beschreibung des Funktionsaufrufs (in diesem Fall die Schnittstelle mit zu übergebenden Parametern) erfolgt in der Beschreibung technologieneutral und muss bei der Transformation in einen entsprechenden Aufruf an die konkrete Implementierung umgesetzt werden.

6.3 Aufbau der Dialogschnittstellenbeschreibung

Die Dialogschnittstellenbeschreibung in *mimesis.ui* folgt dem Metamodell, welches in Abschnitt 6.1 eingeführt wurde. Die hier vorgestellte DSL beschreibt, in welcher hierarchischen Beziehung die Dialogeinheiten zueinander stehen, welche Gruppen von Eingabefeldern existieren und wie diese Felder

beschaffen sind. Zudem werden Informationen für die Typisierung, Validierung und Ansichtensteuerung vorgehalten, die bei einer konkreten Umsetzung der Benutzerschnittstelle ausgewertet werden. Sie setzt damit

Die Beschreibung der Benutzerschnittstelle erfolgt in einer einzigen Datei, welche alle relevanten Informationen zusammenfasst, die zu einer späteren Herleitung der finalen Benutzerschnittstelle in einer bestimmten Technologie verwendet werden kann.

Abbildung 17 zeigt zur Illustration den Auszug aus einer solchen Datei, um einen ersten Eindruck zu erhalten. In den folgenden Abschnitten wird genauer auf die einzelnen Aspekte der Beschreibung eingegangen.

Die Grammatik der DSL findet sich in einer vereinfachten Version im Anhang.

6.3.1 Beschreibung der Darstellungseinheiten (Seiten, Unterseiten, Gruppen)

Wie in den Anforderungen formuliert, müssen die gesammelten Daten strukturiert erfasst und auf Darstellungseinheiten verteilt werden können.

Die Grundstruktur der *mimesis.ui* DSL ist in Abbildung 18 dargestellt. Ein Dokument besteht aus einem *pages*-Bereich, welche die Beschreibung der top-level-Seiten der Benutzerschnittstelle beinhaltet.

6.3.2 Seiten und Unterseiten

Jede hier enthaltene *page* besteht aus einem *fieldgroup*-Bereich, der eine Folge von *fieldgroups* und *inputfields* enthalten kann. Zudem besitzt jeder *page*-Beschreibung einen *pages*-Bereich, in welchem die Unterseiten als weitere *page*-Bereiche abgelegt werden können.

In Abbildung 19 wird die Syntax zur Beschreibung des Dokuments und einer Seite dargestellt. Hier sind auch die Attribute aufgeführt, welche zur Beschreibung des jeweiligen Elements verwendet werden können. Pflichtangaben sind dabei fett hervorgehoben.

Ein Dokument wird durch einen Applikationsnamen und einer Liste von *pages* beschrieben. Jede *page* besitzt einen Page-Namen (*pageName*) und eine eindeutige (alphanumerische) Identifikation / ID (*id*), die später eine eindeutige Identifikation der Seite im Modell gestattet.

Die *page* enthält einen *fieldgroups*-Bereich, in welchem Eingabefelder oder weitere *fieldgroup*-Elemente enthalten sein können. Zudem können zusätzliche *page*-Definitionen im entsprechenden Block angegeben werden. Diese sind als hierarchisch unter dieser Beschreibung angesiedelt zu verstehen.

Für eine Seite können Existenzbedingungen angegeben werden (*exists*), welche die Sichtbarkeit der Seite bestimmen. Hierfür kann ein bool'scher Ausdruck angegeben werden, der sich auf Modellelemente bezieht (vgl. Abschnitte 6.3.7 und **??**). Es existiert zudem ein *actions*-Bereich, in welchem Aktionen spezifiziert werden können, die auf die Seite wirken sollen (vgl. Abschnitt 6.3.11).

6.3.3 Beschreibung von Feldgruppen und Feldern

Eine *fieldgroup* ist eine Sammlung von Feldern und weiteren Feldgruppen. Sie ist durch ein Label und eine eindeutige ID beschrieben. Zudem können Existenzbedingungen angegeben werden (*exists*), welche die Sichtbarkeit der Gruppe bestimmen (vgl. Abschnitte 6.3.7 und **??**) und es existiert ein *actions*-Bereich, in welchem Aktionen spezifiziert werden können, die auf die Feldgruppe wirken sollen (vgl. Abschnitt 6.3.11).

```
{
   "applicationName":"Privathaftpflicht Tarifierung",
   "pages":[
      {
         "pageName":"Personendaten",
         "id":"perdata",
         "fieldgroups":[
            {
               "label":"Personendaten",
               "id":"persDataGroup",
               "inputFields":[
                  {
                     "label":"Vorname",
                     "id":"vname",
                     "type":"text",
                     "binding":"versichertePerson.vorname",
                     "regEx":"^[a-zA-ZäüöÄÜÖßêéèâáà ]{0,}$",
                     "regExText":"Bitte geben Sie den Vornamen ohne Sonderzeichen an.",
                     "value":"Max"
                  },
                  >>>…              ]
            },
            {
               "label":"Kinder",
               "id":"kinderGroup",
               "inputFields":[
                  {
                     "label":"Anzahl Kinder",
                     "id":"anzKinder",
                     "type":"text",
                     "binding":"versichertePerson.anzahlKinder",
                     "regEx":"^[0-9]{0,2}$",
                     "regExText":"Bitte geben Sie die Anzahl Ihrer Kinder ein.",
                     "value":"0"
                  },
                 >>>…               ]
            }
         ]
      },
      {
         "pageName":"Privathaftpflicht",
         "id":"tarifData",
         "fieldgroups":[
           >>> …
         ],
         "pages":[
            {
               "pageName":"Vorschäden",
               "id":"phvPageVorschaeden",
               "hidden": true,
               "fieldgroups":[
                  {
                     "label":"Vorschäden",
                     "id":"phvGroupVorschaeden",
                     "inputFields":[
                        {
                           "label":"Anzahl der Vorschäden",
                           "id":"anzVorschaeden",
                           "type":"text",
                           "binding":"tarifDaten.anzVorschaeden",
                           "regEx":"^[0-9]{0,2}$",
                           "regExText":"Bitte geben Sie die Anzahl der Vorschäden ein.",
```

Abbildung 17: Auszug aus einer mimesis-Seitenbeschreibung

Die Beschreibungen für Felder besitzen eine Reihe von Attributen, die für alle Feldtypen angegeben werden können und Attribute, die *typspezifisch*. Neben einem Label und der ID ist der Typ des Eingabefelds ein Pflichtfeld. Die Attribute der Felder sind im Abschnitt 6.3.5 beschrieben.

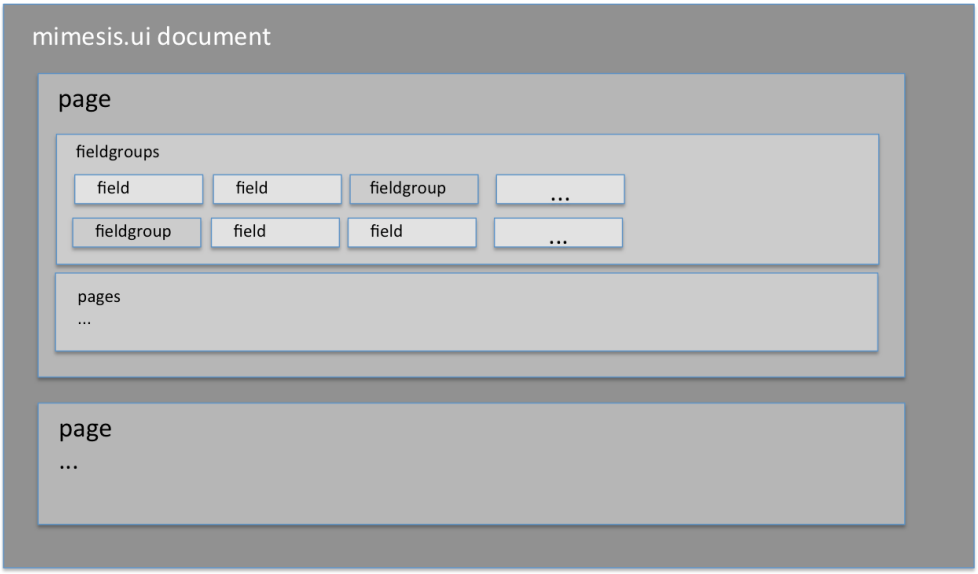

Abbildung 18: Grundsätzliche Struktur der mimesis.ui Dialogbeschreibung

Abbildung 19: Syntax: Dokument und ++display unit / page++

6.3.4 Navigation zwischen Seiteneinheiten

In den Anforderungen wurde formuliert, dass aufgrund der unterschiedlichen Gegebenheiten auf den Zielplattformen sowohl eine sequentielle als auch eine hierarchische Navigation durch die Seiten der Anwendung möglich sein soll.

Die hierarchische Definition der Darstellungseinheiten ermöglicht es im Fall der hier betrachteten Anwendungskategorie, automatisiert eine hierarchische Navigation herzuleiten (vgl. Abschnitt 5.3.2). Dazu muss lediglich die Struktur der Seiten und Unterseiten sinnvoll zur Bildung eines Navigationsbaums ausgewertet werden. Dieser wiederum kann an der Oberfläche zur Ergeugung eines Navigationscontrols verwendet werden, welches die Seiten gezielt anspringt (z.B. über die Sichtbar-Schaltung der gewählten Seite anhand ihrer ID) und so wahlfrei auf die Seiten zugegriffen werden.

Abbildung 20 zeigt diese Herleitung basierend auf der Beschreibung der *pages*-Hierarchie in der an-

geführten Beispielanwendung *Private Haftpflichtversicherung*. Die hier dargestellte Hierarchie kann direkt aus den Seitenbeziehungen hergeleitet werden.

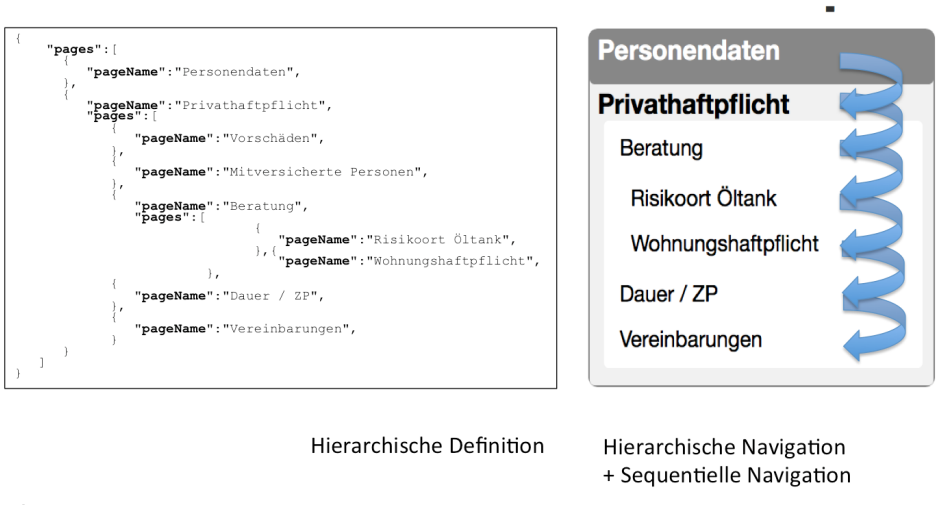

Abbildung 20: Hierarchische und sequentielle Navigation

Auch eine sequentielle Navigation ist für den betrachteten Anwendungsfall sinnvoll herleitbar: hierzu muss der Baum lediglich ”depth-first” traversiert und eine Seitenfolge aufgebaut werden. Wie in Abbildung 20 als Pfeilfolge dargestellt, werden dazu die Seiten in der Reihenfolge angesteuert, in der sie in der hierarchischen Darstellung untereinander stehen.

In beiden Fällen werden durch greifende Existenzbedingungen ausgeblendete Seiten nicht angezeigt und stehen nicht in der Navigation zur Verfügung.

Die Daten, die zur Herleitung der Navigation erforderlich sind, sind also bereits durch die hierarchische Beschreibung der Seiten und Unterseiten in unserem Modell enthalten und können bei der Transformation in eine konkrete Technologie ausgewertet werden.

6.3.5 Ein- / Ausgabekomponenten

Zur Beschreibung der Ein-/Ausgabekomponenten kann zwischen gemeinsamen und typspezifischen Attributen unterschieden werden. Die im folgenden aufgeführten gemeinsamen Attribute sind meist durch die Anforderungen motiviert, die in der Analyse aus den generellen und verhaltensrelevanten Merkmalen abgeleitet wurden, während die typspezifischen Attribute spezifisch für die jeweiligen Feldtypen sind.

In *mimesis.ui* werden in der aktuellen Ausprägung folgende gemeinsame Attribute verwendet:

- **id**: Dient der eindeutigen Identifikation eines Elements in der Benutzerschnittstelle.
- **label**: Anzuzeigender Text in einer grafischen Benutzerschnittstelle.
- **binding**: beschreibt, mit welchem Modellelement im Datenmodell der Anwendung dieses Eingabefeld korreliert. Im Rahmen der vorliegenden Arbeit wird eine Punktnotation verwendet, welche den Pfad durch das Modell hin zum Modelldatenfeld beschreibt (Details in Abschnitt 6.3.10)

- **"value"**: kann zur Vorbelegung eines Feldes mit einem Wert verwendet werden (Details in Abschnitt 6.3.10)
- **"required"**: wird hier *true* angegeben, wird das Feld als notwendig markiert. Bei der Validierung des Dialogs wird geprüft, ob eine Angabe erfolgt ist.
- **exists**: Bedingung für die Existenz des Feldes (Details in Abschnitt 6.3.7).
- **editable**: Bedingung für die Editierbarkeit des Feldes (Details in Abschnitt 6.3.7).
- **actions**: Aktionen, die für das Feld ausgeführt werden können ((Details in Abschnitt 6.3.11)
- **validations**: Spezifikation für die programmatische Validierung des Feldes (Details in Abschnitt 6.3.6)
- **"regEx"** und **"regExText"**: werden zur syntaktischen Validierung verwendet. Hier kann ein regulärer Ausdruck hinterlegt werden, gegen den eine vorgenommene Eingabe geprüft wird. Sollte diese Prüfung fehlschlagen, wird der in *regExText* angegebene Text als Fehlermeldung angezeigt (Details in Abschnitt 6.3.6)

Die typspezifischen Attribute ergeben sich insbesondere aus den in Tabelle 9 (s. Abschnitt 5.3.3) aufgeführten Eigenschaften für die unterstützten Feldtypen. In Abbildung 21 sind für die wichtigsten Typen aus Tabelle 9 Beispiele angegeben. Da die typspezifischen Attribute den Feldern in Tabelle 9 entsprechen, wird auf eine detaillierte Aufzählung an dieser Stelle verzichtet.

Texteingabe

```
{
  "label":"Vorname",
  "id":"vname",
  "type":"text",
  "binding":"versichertePerson.vorname",
  "regEx":"^[a-zA-ZäüöÄÜÖß éèêâàá ]{0,}$",
  "regExText":"Bitte geben Sie den Vornamen
     ohne Sonderzeichen an.",
  "value":"Max"
}
```

Datum

```
{
  "label":"Geburtsdatum",
  "id":"gebdat",
  "type":"date",
  "binding":"versichertePerson.geburtstag",
  "required":true,
  "value":"12.01.1990"
},
```

Numerische Eingabe

```
{
  "label":"Anzahl Kinder",
  "id":"anzKinder",
  "type":"number",
  "binding":"versichertePerson.anzahlKinder",
  "min":0,
  "max":10,
  "value":"0"
}
```

1-aus-N Auswahl

```
{
  "label":"Familienstand",
  "id":"famstand",
  "type":"oneOfMany",
  "binding":"versichertePerson.familienstand",
  "allowedValues":"ledig|verheiratet|geschieden
                   |verwittwet",
  "value":"verheiratet"
  "valuetype":"text"
},
```

Binärwert

```
{
  "label":"Sind in den letzten 5 Jahren Vorschäden
           eingetreten?",
  "id":"phVorschaeden",
  "type":"boolean",
  "value": false,
  "binding":"tarifDaten.vorschaeden"
}
```

Checkbox

```
{
  "label":"Junge Leute Rabatt",
  "id":"jungRab",
  "type":"checkbox",
  "binding":"tarifDaten.jungeLeuteRabatt"
}
```

Abbildung 21: Eingabefeld-Beschreibungen

6.3.6 Reaktion auf Änderungen: Validierung von Eingabedaten

Die Informationen, die zur syntaktischen Validierung der Eingaben benötigt werden, sind in Form von regulären Ausdrücken am Element vermerkt. Hierzu sind die Felder *"regEx" und "regExText"* vorgesehen. Im Attribut *regEx* wird ein regulärer Ausdruck hinterlegt, welcher zur Validierung des Wertes auf der Zielplattform verwendet werden kann.

Eine weitergehende Validierung kann durch Angabe von Validierungsoperationen angegeben werden, die auf dem Element arbeiten. Hierfür ist bei Feldern und Feldgruppen ein *validations*-Bereich angegeben, der die Bescheibung der Operationen enthält, die getriggert werden sollen (vgl. 6.2.3 zu Operationen).

Die Definition einer solchen Validierung ist in Abbildung 22 dargestellt. Dort wird spezifiziert, dass nach Beenden der Eingabe (*onchange*) eine Validierungsfunktion mit dem Namen *checkChildren* mit den Parametern (*numberOfChildren, childrenbelow7* aufgerufen werden soll, die aus dem Modell der Anwendung stammen. Die Funktion muss technologieabhängig zur Verfügung gestellt werden (vgl. Abschnitt 6.2.3). Die aktuell unterstützten Werte für *validationInvocation* sind *onchange* (nach Beenden der Eingabe) und *onedit* (während der Eingabe).

```
"validations":[
   {
      "validatorName":"appvalidations.checkChildren",
      "validatorParams":"children.numberOfChildren, children.childrenbelow7"
      "validationInvocation":"onchange"
   },
   ...
 ]
```

Abbildung 22: Validierung durch Aufruf von Programmcode.

Der Validations-Bereich ist eine Liste; es können somit mehrere Validierungsoperationen angegeben werden, die zu verscheidenen Zeitpunkten getriggert werden können.

6.3.7 Reaktion auf Änderungen: Sichtbarkeitssteuerung

Die Sichtbarkeitssteuerung eines Elements (Seite, Unterseite, Feldgruppe bzw. Feld) kann auf zwei Arten erfolgen:

- ein-/ausblenden eines Elements: hierbei wird das betroffene Element in der Benutzerschnittstelle nicht angezeigt. Die Semantik aus fachlicher Sicht ist hierbei, dass die Daten, die im betroffenen Element enthalten sind in diesem Zustand nicht existieren - also auch nicht Teil des Modells sind.
- deaktivieren eines Elements: hierbei wird das betroffene Element zwar angezeigt, ist aber nicht durch den Nutzer editierbar. Die Semantik aus fachlicher Sicht ist hierbei, dass die enthaltenen Inofrmationen zwar Teil des Modellsaber nicht durch den Benutzer veränderbar sind.

Ob ein Element sicht- bzw. editierbar ist, ergibt sich meist aus den Inhalten, die bereits durch den Benutzer eingegeben wurden - also aus Inhalten des Datenmodells der Anwendung. Hierzu müssen Regeln definiert werden können, die auf diesen Inhalten operieren.

Für diese beiden Eigenschaften können bei den Elementen die Attribute *exists* bzw. *editable* angegeben werden, bei denen in Form von bool'schen Ausdrücken auf Modellinhalten Bedingungen formuliert werden.

Zur Formulierung der Bedingungen wird eine Notation verwendet, wie sie in C, Java bzw. JavaScript üblich ist. Die Referenzierung der Inhalte des Datenmodells der Anwendung erfolgt in einer Punktnotation (vgl. Abschnitt 6.3.10).

Beispiele für diese Regeln sind in Abbildung 23 dargestellt. Für ein Feld *childrenbelow7* wird hier eine Existenzbedingung als bool'scher Ausdruck angegeben, der besagt, dass das Feld nur existieren soll,

wenn der Wert von *numberOfChildren* größer als 0 ist. Im Beispiel darunter ist analog eine Regel für die Editierbarkeit des Feldes *younpeoplerabate* angegeben: das Feld soll nur editierbar sein, wenn das Inhalt von *dateOfBirth*im Modell nach dem *1.1.1990* liegt.

```
{
    "label":"childrenbelow7",
    "id":"liability.customerdata.children.childrenbelow7",
    "binding":"liability.customerdata.children.childrenbelow7",
    "type":"number",
    "exists":"(liability.customerdata.children.numberOfChildren > 0)",
    ...
    "validators":[
      ...
    ]
}
{
    "label":"youngpeoplerabate",
    "id":"liability.contractdata.rabate.youngpeoplerabate",
    "binding":"liability.contractdata.rabate.youngpeoplerabate",
    "type":"boolean",
    "editable":"dateAfter(liability.customerdata.birthdata.dateofbirth, '1990-1-1')"
 }
```

Abbildung 23: Beispiel für die Sichtbarkeitssteuerung

Solche Regeln können bei allen Elementen des Modells angegeben werden, welche *exists* bzw. *editable*-Attribute vorsehen . Aktuell sind dies: Seiten, Unterseiten, Gruppen, Felder.

6.3.8 Reaktion auf Änderungen: Dynamische Inhalte

Dieser Aspekt ist im Konzept bisher lediglich in Grundzügen umgesetzt (Stand 6.2014).

Die Problemstellung hierbei besteht darin, dass auf die Änderung von Inhalten des Modells regiert werden muss, die im Umfeld des betreffenden Elements liegen (z.B. die Reaktion des Feldes *Postleitzahl* durch Änderung des Inhalts *Ort* und daraufhin das Angebot zum Ort passender Postleitzahlen in einer Auswahl).

Der Lösungsansatz besteht darin, an einem betreffenden Element eine ”Reaktions-Operation” zu definieren, die auf Änderungen an bestimmten Modell-Inhalten reagiert und daraufhin eine Operation analog der Aktionen auslöst. Die Notation kann dann analog der in Abbildung 22, Abschnitt 6.3.6 dargestellten Notation für Validierungen gewählt werden unter Angabe der zu beobachtenden Modell-Inhalte.

6.3.9 Reaktion auf Änderungen: Anpassung der Navigation

Im Anforderungsteil wurde beschrieben, dass die Navigation bei Änderung der Sichtbarkeit eines Elements ggf. angepasst werden muss. Für die Navigation bedeutet dies, dass der Zugang zu einer Seite nicht mehr möglich sein soll, sofern deren Sichtbarkeit aufgrund einer angegebenen *exists*-Bedingung ausgesetzt wird (vgl. Abschnitt 6.3.7).

Um dies umsetzen zu können, bedarf es keiner zusätzlichen Informationen im Modell. Durch die Einführung einer eindeutigen ID für *pages*, *fieldgroups* und die Eingabeelemente kann in einer konkreten Umsetzung auf die Änderung des Status einer Seite auch in der Navigation reagiert und ein betroffenes Element aus der Navigation entfernt werden.

6.3.10 Datenmodell, Bindung an Eingabefelder und Initialisierung

Viele Eigenschaften der Oberfläche basieren auf den Inhalten, die in Feldern der Anwendung erfasst wurden. Die erfassten Daten können an unterschiedlichen Stellen in der Benutzeroberfläche dargestellt werden und korrelieren nicht unbedingt mit der Struktur, die der beschriebene hierarchische Seitenaufbau nahelegt. Daher wurde in Abschnitt 5.3.10 als Anforderung formuliert, dass die Daten der Anwendung zur Laufzeit in einem gesonderten Modell gehalten werden sollen, auf das Elemente der Oberfläche im Sinne eines data-binding-Ansatzes (z.B. https://en.wikipedia.org/wiki/UI_data_binding) zugreifen/referenzieren können (Diese Auffassung folgt zudem aktuellen Design-Prinzipien wie dem Model-View-Controller-Paradigma oder dem Model-View-Viewmodel-Ansatz [MÖ93],[ES13]).

In Abschnitt 5.3.10 wurde exemplarisch eine abstrakte Sicht dargestellt, die ein solches Datenmodell aus Sicht einer Benutzerschnittstelle in einer datensammelnden Anwendung hat. Es besteht aus einer Reihe, miteinander in hierarchischer Beziehung zueinander stehender Objekte. Die einzelnen Elemente können eindeutig durch ihre Position in der Hierarchie identifiziert werden. Hierfür kann eine Punktnotation verwendet werden, wie sie in vielen Technologien um UI-Umfeld bereits gängige Praxis ist und kann daher auch als Grundlage für die Zwecke in *mimesis.ui* wiederverwendet werden. So bedienen sich z.B. AngularJS (JavaScript-Umfeld) sowie Java Server Faces (JSF) eines solchen Ansatzes. In JSF wird hierfür beispielsweise die *Unified Expression Language (UEL)* zur Beschreibung der Korrelation verwendet.

Die Bindung eines Feldes an ein Modellelement erfolgt daher in *mimesis.ui* über das *binding*-Attribut, welches den Bezug eines Eingabefeldes zu einem Element im Datenmodell der Anwendung herstellt. Hier wird ein Modellelement eindeutig mit dem Eingabefeld verknüpft. In Abbildung 21 sind hierfür Beispiele angegeben. Hiermit und mit der Typinformation stehen alle Informationen für eine Bindung während der Laufzeit zur Verfügung, die der Programmcode zum Befüllen des Datenmodells anhand der Eingaben benötigt.

Durch die Angabe eines *value*-Attributs kann zusätzlich die Vorbelegung eines Eingabefeldes mit einem Wert vorgenommen werden (vgl. Abbildung 21). Das Datenmodell kann hiermit in Verbindung mit Angabe der *binding*-Informationen mit Werten vorbelegt werden.

6.3.11 Benutzer- und Elementaktionen

In Abschnitt 5.3.8 wurden die Anforderungen hinsichtlich der Anbindung von Nutzer- und Elementaktionen skizziert. Im *mimesis.ui*-Metamodell wurde eine Modellierung gewählt, welche die dort aufgeführten Aktionskategorien

- Nutzeraktionen
- Elementaktionen
- Feature-Aktionen

abbildet. Hierzu kann bei der Beschreibung eines Elements (Seite, Untereite, Gruppe, Feld) ein Block angegeben werden, der eine Liste von Aktionsbeschreibungen enthält. Zur Definition einer Aktion werden folgende Informationen angegeben (vgl. 6.2.3 zur Definition von Operationen in *mimesis.ui*)

- *actionId*: Eindeutige ID für die Aktion
- *actionLabel*: Text zur Darstellung an der Oberfläche

- *actionType*: Kategorie der Operation (*user, element,* bzw. Feature-Name)
- *actionName*: eindeutiger Name der zu rufenden Implementierung der Aktion
- *actionParams*: beteiligte Elemente des Datenmodells in Punktnotation (s. 6.3.10)
- *actionInvoke*: Ereignis zur Aktivierung der Operation.

Abbildung 24 zeigt ein Beispiel einer solche Aktionsbeschreibung für eine Gruppe (*Kundendaten*). Hier werden zwei Aktionen angeboten: das *Öffnen der Kundendatenbank* und die *automatische Befüllung* der Kundendaten anhand des angegebenen Nachnamens.

```
"actions":[
    {
      "actionId":"appactions.openCustomerDB_id",
      "actionLabel":"Öffne Kundendatenbank ...",
      "actionName":"appactions.openCustomerDB",
      "actionParams":"liability.customerdata",
      "actionType":"user",
      "actionInvoke":"select"
    },
    {
      "actionId":"appactions.autofillCustomer_id",
      "actionLabel":"Kundendaten automatisch füllen",
      "actionName":"appactions.autofillCustomer",
      "actionParams":"liability.customerdata.lastname, liability.customerdata",
      "actionType":"user",
      "actionInvoke":"select"
    }
  ]
```

Abbildung 24: Beispiel für eine Aktionsbeschreibung (Gruppe)

Abbildung 24 zeigt die Definition einer *element action*, die einem Button zugeordnet ist und die aktuellen Falldaten aus einem Zwischenspeicher läd und die Definition einer *feature action* des Typs *info*, die auf die bei betreten des Feldes aktiviert wird.

7 Arbeiten und Technologien im Umfeld - eine Sammlung

Zur modellgetrienbenen Entwicklung von Benutzerschnittstellen wurden in den vergangenen Jahren unterschiedliche Ansätze entwickelt, die sich in der Fokussierung auf Aspekte von Benutzerschnittstellen unterscheiden. Hinsichtlich dem Themenbereich *mimesis.ui* lassen sich diese Arbeiten in diese Grundkategorien einteilen, die sich einerseits mit der Beschreibung von Benutzerschnittstellen, andererseits mit der Beschreibung der Abläufe in Anwendungen befassen:

User Interface Description Languages (UIDL) beschreiben die konkrete Ausprägung eines UIs unabhängig von einer spezifischen Technologie. Beispiele sind **JavaFX** ([Fed11]), **UIML** ([APB+99]), **UsiXML** ([Lim04]) und **XForms** ([DKMR03]). Hier werden technologieneutral Ein-/Ausgabeelemente, Beziehungen zwischen Elementen und das Verhalten innerhalb der UI (z.B. Sichtbarkeitsregeln) beschrieben ([NM09]). Teilweise beinhalten die Ansätze bereits grundlegende Task-Konzepte (z.B. UsiXML).

Task-/konversationsbasierte Ansätze beschreiben die Anwendung über Ausprägungen der Dialogflüsse, abgeleitet aus Taskmodellen. Zur Ableitung von UIs werden Tasks mit technologieneutralen

UI-Beschreibungen assoziiert - z.B. **CAP3** [VLC11], **MARIA** [PSS09] und die Arbeiten zu Konversationen von Popp et al. (z.B. [PFA+09], [RKP+14]). Der Fokus liegt hier auf der expliziten Modellierung der Abläufe und Varianten (z.B. **MANTRA**, [Bot06]), es existieren jedoch auch erste Ansätze zur Modellierung von Varianten und Produktfamilien ([PWB13], [TVTB12]).

Im folgenden sind weitergefasste Literaturreferenzen angegeben, die im weiteren Verlauf der Arbeit weitere Erkenntnisse beitragen werden. Eine detaillierte Betrachtung der Inhalte erfolgt im Rahmen der mit mimesis verbundenen Dissertation bzw. in weiteren Veröffentlichungen. Sie dienen an dieser Stelle als Sammlung möglicher Referenzen.

Grundlagen zum Thema UI-Design

- [BATO+10]: Simple but crucial user interfaces in the world wide web: introducing 20 guidelines for usable web form design
- [DFT05]: Recovering Interaction Design Patterns in Web Applications
- [Gal07]: The Essential Guide to User Interface Design: An Introduction to GUI Design
- [Mye04]: Graphical User Interface Programming
- [Nil09]: Design patterns for user interface for mobile applications
- [Tid06]: Designing Interfaces: Patterns for effective Interaction Design
- [BATO+10]

Arbeiten zu Oberflächen und Regeln

- Adaptive-UI: [BS02]: [MP02]: [Neb12]: [NN12]: [YN08]:
- Pagination: [FSVM06]: Splitting rules for graceful degradation of user interfaces
- Reasoning / Validation: [CG04]: Representing and reasoning on XForms document

User Interface Description Languages

- Sprachen/Notationen: z.B. XForms, XUL, XIML, ...
- UI-Beschreibungen:
  - [VS10]: Generative pattern-based design of user interfaces
  - [LjMR09]: Designing User Interface Adaptation Rules with T : XML
  - [NM09]: Creating a Lightweight User Interface Description Language: An overview of the Personal Universal Controller Project
  - [dSL06]: Using IMML and XICL components to develop multi-device web-based user interfaces
  - [Lim04]: USIXML: A User Interface Description Language Supporting Multiple Levels of Independence.
  - [DLW06]: Graphical user interfaces as documents
  - [Trz10]: GUIs without Pain – the Declarative Way
  - [PE01]: XIML: A Universal Language for User Interfaces

  - [PSS09]: Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environment
  - [TZ03]: Abstract User Interface Representations : How Well do they Support Universal Access ?
  - [Bot06]: A Model-Driven Approach to the Engineering of Multiple User Interfaces
  - [MF10]: Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models

- Dissertationen, Diplomarbeiten, . . . : [Wol11], [PS09], [Mor08],
- Conference Papers: [FDRS04] , [BGL06]

8 Validierung des Ansatzes

Zur Validierung des Ansatzes wurde eine prototypische Implementierung erstellt. Das Ziel war der Nachweis, dass die aus der Analyse bestehender Benutzerschnittstellen abgeleiteten Eigenschaften zur technologieneutralen Beschreibung von Oberflächen den realen Anforderungen genügen.

Bei der Implementierung lag der Fokus auf webbasierten Benutzerschnittstellen für unterschiedliche Gerätekategorien (Desktop-, Tablet- und mobile-Devices), für welche unter Verwendung geeigneter Frameworks unterschiedliche technologische Varianten der abstrakten Dialogbeschreibung hergeleitet wurden.

Hierzu wurden folgende Schritte umgesetzt:

- Implementierung der auf dem entwickelten Metamodell fußenden *mimesis.ui* DSL
- Implementierung der Transformation von der abstrakten Benutzerschnittstellenbeschreibung in unterschiedliche Zieltechnologien
- Festlegung einer für webbasierte Anwendungen geeigneten Architektur

8.1 Prototypische Implementierung

8.1.1 Lösungsansatz für die Implementierung der Herleitung der Benutzerschnittstelle aus der technologieneutralen Dialogbeschreibung

Der in *mimesis.ui* verfolgte Lösungsansatz besteht aus der Definition und dem Konzept zur Transformation einer technologieneutralen Dialogbeschreibung in ein konkretes Ausgabeformat, welches je nach gefordertem fachlichem und technischem Kanal variieren kann.

In Abbildung 25 ist das Vorgehen dargestellt, welches aus der technologieneutralen Beschreibung eine konkrete Benutzerschnittstelle herleitet.

Das Kernelement der Implementierung bildet dabei ein Transformator, welcher eine Dialogbeschreibung und die Zielplattform als *Eingabeinformationen* erhält. Die Dialogbeschreibung enthält die Informationen des in Abschnitt **??** beschriebenen Metamodells (in Form der in den letzten Abschnitten vorgestellten *mimesis.ui* DSL). Das enthaltene Modell beschreibt die fachliche Variante, für welche eine

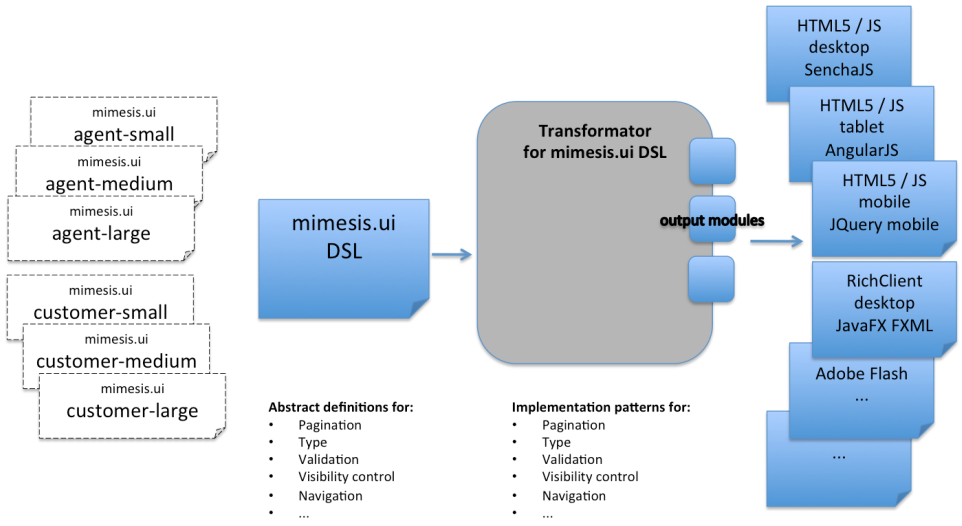

Abbildung 25: Transformation der Seitenbeschreibung in die Zielsprache

Benutzerschnittstelle hergeleitet werden soll (Abbildung 25 zeigt exemplarisch die fachlichen Varianten *agent* und *customer* in unterschiedlichen Granularitätsstufen[9]).

Die Schritte zur Transformation lassen sich wie folgt zusammenfassen:

- Lesen der DSL und Aufbau eines internen Modells gemäß *mimesis.ui* Metamodell
- Transformation in unterschiedliche Technologien; Mapping der enthaltenen Konzepte auf technologische Lösungen
- Bereitstellung des Ergebnises : lauffähige Anwendung zu Integration in unterschiedliche Portale

Nach dem Einlesen der im Modell enthaltenen Informationen generiert der Transformator die für die konkret geforderte Zielplattform benötigte Dialogschnittstelle. Hierzu werden unterschiedliche *output*-Module verwendet, welche die technologieneutrale Beschreibung der einzelnen Eigenschaften in konkrete Implementierungen für die verwendete Zielplattform umsetzen (z.B. das HTML-Markup und JavaScriptcode für eine HTML-Oberfläche mit dem Framework AngularJS oder den Programmcode zur Erzeugung der Oberfläche in einer nativen Umgebung wie JavaFX).

8.1.2 Architektur / Integration des Prototypen

Bereits vor Beginn des mimesis-Projektes wurde in einem ersten Schritt zusammen mit dem Partner aus dem Versicherungsbereich eine rudimentäre Dialogschnittstellenbeschreibung erstellt und i.S. eines Proof-of-concepts umgesetzt. Die Beschreibung wurde in einer JavaScrip-basierten Webanwendung

[9]Die hier angeführten Granularitätsstufen sind als Beispiel zu sehen. Welche Ausprägungen existieren können, hängt davon ab, welche Stufen für den Anwendungsfall relevant sind. Es könnten ebenso Granularitätsstufen *internet, desktop* oder Kombinationen verwendet werden.

clientseitig im Browser interpretiert und unterschiedliche in JavaScript erstellte Generatoren für unterschiedliche Gerätetypen prototypisch umgesetzt. Damit konnten erste Aussagen zur Machbarkeit getroffen werden. Dieser Ansatz wird im Rahmen des mimesis-Projektes nicht weiterverfolgt, da er aufwändig und fehleranfällig in der Erstellung und Wartung ist. Der clientseitige Ansatz birgt zudem die Unsicherheit der Zielumgebung: in browserbasierten Umgebungen ist ein Hauptproblem die Anzahl der unterschiedlichen Browser-Implementierungen. Es existieren viele Versionen verschiedener Produkte, die unterschiedlichen Leistungsumfang - und leider auch Insuffizienzen - aufweisen (vgl. z.B. http://www.webdevout.net/browser-support). Hinsichtlich der Ablaufumgebung ist nicht bekannt, wie leistungsfähig die Prozessoren der Produkte sind, von denen die Anwendung gerufen wird, was eine clientseitige Verarbeitung nicht praktikabel erscheinen lässt.

Für das mimesis-Projekt wurde daher eine serverseitige Implementierung der Transformation mittels Java als Programmiersprache verfolgt. Diese technologische Basis wurde gewählt, da Java im Enterprise-Umfeld weit verbreitet ist und dadurch die Lösung einfach in eine bestehende Landschaft integriert werden kann.

Der im letzten Abschnitt beschriebenen Transformator wurde als WebService (*mimesis.ui Server*) bereitgestellt, welcher von einer Client-Anwendung mit der Erstellung eines UIs beauftragt werden kann (s. Abbildung 26). Der webbasierte Client (z.B. ein in ein Portal eingebetteter Applikationsrahmen) fordert vom *mimesis.ui* Server das UI für eine bestimmte Anwendung (*application*) für eine bestimmte Zieltechnologie (*technology*) an, die im Umfeld des Clients eingebettet werden kann. Der *mimesis.ui* Server beschafft sich aus einer angeschlossenen Datenbank die Benutzerstellenbeschreibung für diese Anwendung und reicht sie an den Transformator weiter. Der Transformator wählt aufgrund der angeforderten *technology* ein passendes *output module* und transformiert die Schnittstellenbeschreibung in das entsprechende Ausgabeformat. Der *mimesis.ui* Server liefert das Ergebnis und die zur Ausführung benötigten Artefakte (z.B. JavaScript-Dateien) an den Client aus, der das Ergebnis in sein Umfeld einbettet und ausführt.

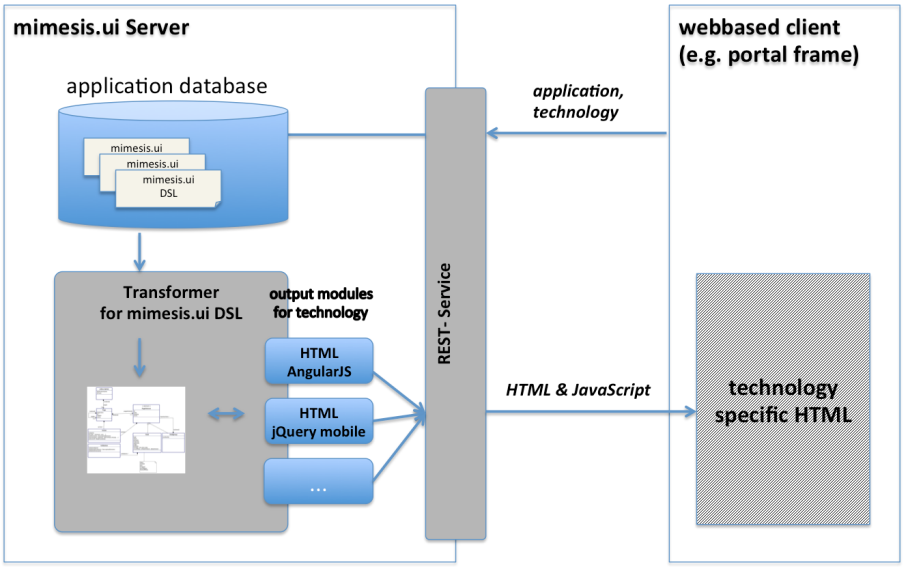

Abbildung 26: Architektur des Prototyps

8.1.3 Verwendete Technologien

Der aktuelle Prototyp für den Transformator ist ein in Java erstellter Compiler, der serverseitig - aber auch in Java RichClient Anwendungen - eingesetzt werden kann. Die technische Grundlage für den Compiler ist mit dem Compiler-Generator ANTLR4 basierend auf einer Grammatik für *mimesis.ui DSL* erstellt worden.

Die Ausgabe, die der Transformator erzeugt, ist aktuell für ein HTML5- und JavaScript-Umfeld bestimmt. Zur Erstellung der *output*-Module wurde das Templating-Framework *Freemarker* verwendet, welches die Erstellung unterschiedlicher HTML5-Varianten für unterschiedliche Kanäle vereinfacht. Dazu müssen im Idealfall lediglich die verwendeten Templates ausgetauscht werden um eine entsprechende Ausgabe zu erhalten.

Variante 1

Die aktuell erstellten Templates erzeugen eine Variante, die für mittlere Devices (Desktop und Tablets) geeignet ist, welche eine hierarchische Navigation zulassen. Diese Darstellung kann auch auf mobilen Geräten verwendet werden, wenn statt der hierarchischen eine zur Verfügung gestellte sequentielle Navigation verwendet wird. Die meisten Screenshots in diesem Dokument wurden aufgrund dieser Variante erstellt.

- jQuery als Basisframework
- Twitter Bootstrap für die Oberflächenelemente
- Eingabekomponenten (z.B. Datumskomponente)
- eine eigene Modell-Implementierung mit databinding-Mechanismus, welche das Datenmodell der Anwendung dynamisch aufbaut. Das Datenmodell muss nicht vorher in Form von Klassen instantiiert sein, sondern wird anhand der *data-bindings* bei Eingabe der Werte zur Laufzeit automatisch angelegt und befüllt.
- eine eigene Implementierung des Sichtbarkeitssteuerung
- eine eigene Implementierung der Navigationskontrolle

Variante 2

Zusätzlich wurde eine zweite Variante erstellt, anhand der auch grundsätzlich die Eignung für mobile Geräte demonstriert werden konnte.

- AngularJS als Basisframework
- Twitter Bootstrap für die Oberflächenelemente
- mobile Angular UI (http://mobileangularui.com) als responsive Framework, das mobile-Aspekte abdeckt
- weitere Open Source verfügbare Eingabekomponenten
- Nachbildung der für *mimesis.ui* benötigten Features mit AngularJS

Für beide Varianten wurden entsprechende *output*-Module für den in Abschnitt 8.1.1 beschriebenen Transformator erstellt und entsprechende JavaScript-Bibliotheken erstellt.

Die verwendeten Technologien können zwar bereits in der vorliegenden Form im mobilen Umfeld eingesetzt werden, jedoch bietet sich hierfür zukünftig die Verwendung eines spezialisierteren Frameworks an, welches eher den Vorgehensweisen auf mobilen Endgeräten entspricht und entsprechende Controls dafür anbietet (z.B. jQuery mobile oder SenchaTouch). Ein jQuery mobile-Prototyp ist dank des Templating-Ansatzes einfach zu erstellen, da er strukturell ähnlich der bisherigen Implementierung ist.

9 Stand und Ausblick zu *mimesis.ui*

Als erster Schritt zur Validierung der Qualität der erzeugten Ausgaben wurden die bereits bei der Analyse verwendeten bestehenden Anwendungsfrontends nachmodelliert und die generierten Ergebnisse hinsichtlich ihrer Funktionalität verglichen. Da die Eigenschaften der technologieneutralen Beschreibung auf diesen Anwendungen aufgebaut wurden, konnten hier alle im Modell berücksichtigten Anforderungen erfüllt werden - dies liegt in der Natur der Sache und ist weniger verwunderlich.

Darüber hinaus konnte jedoch gezeigt werden, dass aus einer einzigen Benutzerschnittstellenbeschreibung das UI für unterschiedliche Geräte hergeleitet werden kann und die Oberfläche für die untersuchten Plattformen nicht - wie bisher - neu erstellt werden muss.

Die Implementierung konnte dabei allerdings aufgrund des prototypischen Charakters (Unvollständigkeit insbesondere bei Zahl der konkreten Implementierung ”spezialisierter” Controls) noch nicht endgültig die Belastbarkeit des Ansatzes für neu zu erstellende Anwendungen nachweisen. Jedoch konnten bisher alle an das Projekt herangetragenen Anforderungen konzeptionell abgebildet werden - was ein Indiz für die generelle Nutzbarkeit darstellt.

Daher wird in den nächsten Monaten das hier vorgestellte Konzept von *mimesis.ui* und die Ergebnisse der prototypischen Implementierung in einem konkreten Projekt bei dem am Projekt partizipierenden Versicherungsunternehmen eingesetzt. Es handelt sich hierbei um dynamische Dialoge für die Erfassung von Risikofragen, welche für unterschiedliche Plattformen und Portale bereitgestellt werden müssen, die mir unterschiedlichen Technologien umgesetzt wurden. Hierfür wird eine Reihe komplexer Controls benötigt, die sich zielumgebungsspezifisch verhalten. Die Anforderungen aus diesem Projekt werden die bisher erreichten Ergebnisse vervollständigen und zur Validierung des Ansatzes in der Praxis beitragen.

10 Anhang: ANTLR4-Grammatik *mimesis.ui DSL*

Die in Listing 1 dargestellte Grammatik wurde im Rahmen der Erstellung des Compilers zur Erzeugung der HTML5-Benutzerschnittstelle verwendet. Sie beschreibt nicht vollständig die Möglichkeiten der Sprache, soll aber den aktuellen Stand illustrieren.

Listing 1: mimesis.ui DSL

```
grammar UIDescriptionGrammar;

// Grammar to interpret page descriptions in the
// mimesis.pagedescription language
// (c) michael.hitz − source is part of the mimesis project. all rights reserved.

uidescription:
        '{' applicationdescription ',' pages '}'
        ;

applicationdescription:
        APPLICATIONNAME ':' JSTRING
        ;

pages:
        PAGES ':' '[' page* ']'
        ;

page:
        '{' pagedescription '}' (',' page)*
        ;

pagedescription:
          pagedescriptionattributes ',' inputsectionspage (',' elementactions)? (',' pages)?
        ;

pagedescriptionattributes:
        |pagedescriptionattribute (',' pagedescriptionattribute)*
        ;

inputsectionspage:
        FIELDGROUPS ':' '[' groupelement? (',' groupelement)* ']'
        ;

inputsections:
        INPUTFIELDS ':' '[' groupelement? (',' groupelement)* ']'
        ;

groupelement:
        inputfield
        | fieldgroup
        ;

fieldgroup:
        '{' fieldgroupdescription '}'
        ;

fieldgroupdescription:
        fieldgroupattributes ',' inputsections (',' elementactions)?
    ;

fieldgroupattributes:
```

```
53          fieldgroupattribute (',' fieldgroupattribute)*
54          ;
55
56  inputfield:
57          '{' inputfielddescription '}'
58          ;
59
60  inputfielddescription:
61           fieldattributes (',' elementactions)? (',' validators)?
62
63          ;
64
65  fieldattributes:
66                  fieldattribute (',' fieldattribute)*
67          ;
68
69  fieldattribute:
70          | LABEL   ':' JSTRING
71          | APPEND   ':' JSTRING
72          | ID     ':' JSTRING
73          | TYPE    ':' JSTRING
74          | BINDING   ':' JSTRING
75          | REGEX    ':' JSTRING
76          | REGEXTEXT   ':' JSTRING
77          | VALUE   ':' value
78          | REQUIRED   ':' BOOL
79          | ALLOWEDVALUES  ':' JSTRING
80          | DISPLAYVALUES  ':' JSTRING
81          | EXISTANCE   ':' JSTRING
82          | EDITABLE   ':' JSTRING
83           | MINVALUE   ':' JSTRING
84           | MAXVALUE   ':' JSTRING
85           | STEPVALUE  ':' JSTRING
86           | pair // emergency exit for custom attributes
87      ;
88
89   fieldgroupattribute:
90          | LABEL   ':' JSTRING
91          | ID     ':' JSTRING
92          | EXISTANCE  ':' JSTRING
93          | EDITABLE  ':' JSTRING
94           | pair // emergency exit for custom attributes
95      ;
96
97    pagedescriptionattribute:
98          | PAGENAME   ':' JSTRING
99          | ID     ':' JSTRING
100         | EXISTANCE  ':' JSTRING
101          | pair // emergency exit for custom attributes
102     ;
103
104 elementactions:
105         ELEMENTACTIONS ':' actionlist
```

```
106             ;
107
108     actionlist:
109             '[' (actionobject (',' actionobject)*)? ']'
110             ;
111     actionobject:
112             '{' actionattribute (',' actionattribute)* '}'
113             ;
114     actionattribute:
115             pair // temporary. explicit later
116             ;
117     validators:
118             ELEMENTVALIDATORS ':' validatorlist
119             ;
120     validatorlist:
121             '[' (validatorobject (',' validatorobject)*)? ']'
122             ;
123     validatorobject:
124             '{' validatorattribute (',' validatorattribute)* '}'
125             ;
126     validatorattribute:
127             pair // temporary. explicit later
128             ;
129
130     // random extra data. used for
131     // development of new features
132     pair: JSTRING ':' value;
133
134     // basic values
135     value: JSTRING
136         | NUMBER
137         | BOOL
138         | 'null'
139         | object
140         |  valuearray
141         ;
142
143     valuearray:
144             '[' ((value) (',' value)*)? ']'
145             ;
146
147     // embedded object and list
148     object:
149             '{' (pair (',' pair)*)? '}';
150
151     // Terminals
152     APPLICATIONNAME:  '"'applicationName'"';
153     PAGES:    '"'pages'"';
154     FIELDGROUPS:  '"'fieldgroups'"';
155     INPUTFIELDS:   '"'inputFields'"';
156
157     PAGENAME:  '"'pageName'"';
158     LABEL:    '"'label'"';
```

```
APPEND:  '"'append"';
ID :  '"'id"';
TYPE:  '"'type"';
BINDING:  '"'binding"';
VALUE:  '"'value"' ;
REGEX:  '"'regEx"';
REGEXTEXT:  '"'regExText"';
REQUIRED:  '"'required"';
HIDDEN:  '"'hidden"';
DISABLED:  '"'disabled"';
EXISTANCE:  '"'exists"';
EDITABLE:  '"'editable"';
ALLOWEDVALUES: '"'allowedValues"';
DISPLAYVALUES:  '"'valueLabels"';
MINVALUE:  '"'min"';
MAXVALUE:  '"'max"';
STEPVALUE:  '"'steps"';

ELEMENTACTIONS:  '"'actions"';
ELEMENTVALIDATORS: '"'validators"';

VALIDATION:  '"'validation"';
VALIDATIONERROR:  '"'validationError"';

LCURLY : '{' ;
RCURLY : '}' ;
LBRACK : '[' ;
RBRACK : ']' ;

JSTRING : '"' (ESC | ~["'\\])* '"' ;

fragment ESC : '\\' (["'\\/bfnrt] | UNICODE) ;
fragment UNICODE : 'u' HEX HEX HEX HEX ;
fragment HEX : [0-9a-fA-F] ;

NUMBER
    : '-'? INT '.' INT EXP? // 1.35, 1.35E-9, 0.3, -4.5
    | '-'? INT EXP // 1e10 -3e4
    | '-'? INT // -3, 45
    ;
BOOL : 'true'|'false';

fragment INT : '0' | '1'..'9' '0'..'9'* ; // no leading zeros
fragment EXP : [Ee] [+\-]? INT ;

LINE_COMMENT : '//' .*? '\r'? '\n' -$>$ skip ; // Match "//" stuff '\n'
COMMENT : '/*' .*? '*/' -$>$ skip ; // Match "/*" stuff "*/"
WS : [ \t\n\r]+ -$>$ skip ;
```

Literatur

[APB+99] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams und Jonathan E. Shuster. UIML: An appliance-independent XML user interface language. In *WWW '99 Proceedings of the eighth international conference on World Wide Web*, Seiten 1695–1708, 1999.

[BATO+10] J.A. Bargas-Avila, A.N. Tuch, K. Opwis, S.P Roth, O. Brenzikofer und S. Orsini. Simple but crucial user interfaces in the world wide web: introducing 20 guidelines for usable web form design. In Rita Matrai, Hrsg., *User Interfaces*, number May, Kapitel 1, Seiten 1–11. InTech, 2010.

[BGL06] Matthias Book, Volker Gruhn und Matthias Lehmann. Automatic dialog mask generation for device-independent web applications. In *Proceedings of the 6th international conference on Web engineering - ICWE '06*, Seiten 209–216, New York, USA, 2006. ACM Press.

[Bot06] Goetz Botterweck. A Model-Driven Approach to the Engineering of Multiple User Interfaces. In *Proceeding MoDELS'06 Proceedings of the 2006 international conference on Models in software engineering*, Seiten 106–115, 2006.

[BS02] Eric Blechschmitt und Christoph Strödecke. An architecture to provide adaptive, synchronized and multimodal human computer interaction. In *MULTIMEDIA '02 Proceedings of the tenth ACM international conference on Multimedia*, Seiten 287–290, 2002.

[CG04] PY Cheow und Guido Governatori. Representing and reasoning on XForms document. In *ADC '04 Proceedings of the 15th Australasian database conference*, Seiten 141–150, 2004.

[Chl06] Paul Chlebek. *User Interface-orientierte Softwarearchitektur*. Vieweg & Sohn Verlag, Wiesbaden, 1. auflage. Auflage, 2006.

[CP08] Gaëlle Calvary und Anne-marie Pinna. Lessons of Experience in Model-Driven Engineering of Interactive Systems : Grand challenges for MDE ? 2 Where we are in MDE for HCI. In *First International Workshop on Challenges in Model-Driven Software Engineering (ChaMDE), MODELS'08*, 2008.

[Dem08] Alexandre Demeure. Requirements and models for next generation UI languages. In *WorkShop Üser Interface Description Languages for Next Generation User Interfaces", CHI 2008*, Seiten 1–4, 2008.

[DFT05] Giuseppe Antonio Di Lucca, Anna Rita Fasolino und Porfirio Tramontana. Recovering Interaction Design Patterns in Web Applications. *Ninth European Conference on Software Maintenance and Reengineering*, Seiten 366–374, 2005.

[DKMR03] Micah Dubinko, Leigh Klotz, Roland Merrik und T. Raman. XForms 1.0 W3C Recommendation - http://www.w3.org/TR/xforms, 2003.

[DLW06] Dirk Draheim, Christof Lutteroth und Gerald Weber. Graphical user interfaces as documents. In *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction design centered HCI - CHINZ '06*, Seiten 67–74, New York, New York, USA, 2006. ACM Press.

[DMLC08] Alexandre Demeure, Jan Meskens, Kris Luyten und Karin Coninx. Design by Example of Graphical User Interfaces adapting to available screen size. In *Proceedings of CADUI 2008, the 7th International Conference on Computer-Aided Design of User Interfaces.*, 2008.

[dSL06] Lirisnei Gomes de Sousa und Jair Cavalcanti Leite. Using IMML and XICL components to develop multi-device web-based user interfaces. In *Proceedings of VII Brazilian symposium on Human factors in computing systems - IHC '06*, Seite 138, New York, New York, USA, 2006. ACM Press.

[ES13] Karl Eilebrecht und Gernot Starke. *Patterns Kompakt - Entwurfsmuster für effektive Software-Entwicklung*. Springer Vieweg, Berlin Heidelberg, it-kompakt. Auflage, 2013.

[FDRS04] Peter Forbrig, Anke Dittmar, Daniel Reichart und Daniel Sinnig. From Models to Interactive Systems Tool Support and XIML. In *In (IUI-CADUI 2004) Workshop: Making Model-based UI Design Practical: Usable and Open Methods and Tool*, 2004.

[Fed11] Irina Fedortsova. Mastering FXML, 2011.

[FSVM06] Murielle Florins, Francisco Montero Simarro, Jean Vanderdonckt und Benjamin Michotte. Splitting rules for graceful degradation of user interfaces. *Proceedings of the 11th international conference on Intelligent user interfaces - IUI '06*, Seite 264, 2006.

[Gal07] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design*. Wiley & Sons, 2007.

[Hit13] Michael Hitz. Eine Multikanal-Architektur für Frontendsysteme und deren Erweiterbarkeit durch Variantenbildung. In *Proceedings, LNI, GI Software Engineering 2013, Workshopband*, Seiten 583–589. GI, Köllen Druck+Verlag GmbH, Bonn., 2013.

[Lim04] Quentin Limbourg. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In Maristella Matera und Sara Comai, Hrsg., *ICWE Workshops*, Seiten 325–338. Rinton Press, 2004.

[LjMR09] Víctor López-jaquero, Francisco Montero und Fernando Real. Designing User Interface Adaptation Rules with T : XML. In *IUI '09 Proceedings of the 14th international conference on Intelligent user interfaces*, Seiten 383–387, 2009.

[M$\ddot{9}$3] Hanspeter Mössenböck. *Objektorientierte Programmierung*. Springer Verlag, 1993.

[Mes08] Jan Meskens. Gummy for Multi-Platform User Interface Designs : Shape me , Multiply me , Fix me , Use me. In *AVI '08 Proceedings of the working conference on Advanced visual interfaces*, Seiten 233–240, 2008.

[MF10] António Miguel und João Pascoal Faria. Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models. In Rita Matrai, Hrsg., *User Interfaces*, number May 2010. InTech, 2010.

[Mor08] Tatjana Morozova. *Diplomarbeit Modellierung und Generierung von Web*. Dissertation, 2008.

[MP02] Guido Menkhaus und Wolfgang Pree. A Hybrid Approach to Adaptive User Interface Generation. In *Proceedings of the 24th International Conference on Information Technology Interfaces, 2002. ITI 2002.*, Seiten 185–190, 2002.

[Mye04] Brad a. Myers. Graphical User Interface Programming. *Tribology and Interface Engineering Series*, 44(1):1–12, 2004.

[Neb12] Michael Nebeling. *Lightweight Informed Adaptation: Methods and Tools for Responsive Design and Development of Very Flexible, Highly Adaptive Web Interfaces. PhD Thesis, ETH Zurich.* Dissertation, ETH Zürich, 2012.

[Nil09] Erik G. Nilsson. Design patterns for user interface for mobile applications. *Advances in Engineering Software*, 40(12):1318–1328, 2009.

[NM09] Jeffrey Nichols und Brad a. Myers. Creating a Lightweight User Interface Description Language: An overview of the Personal Universal Controller Project. *ACM Transactions on Computer-Human Interaction*, 16(4), November 2009.

[NN12] Michael Nebeling und Moira C Norrie. jQMultiTouch : Lightweight Toolkit and Development Framework for Multi-touch / Multi-device Web Interfaces. In *EICS '12 Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, Seiten 61–70, 2012.

[PE01] Angel Puerta und Jacob Eisenstein. XIML: A Universal Language for User Interfaces - http://www.ximl.org/documents/XimlWhitePaper.pdf. *White Paper, RedWhale Software*, 2001.

[PFA+09] Roman Popp, Jürgen Falb, Edin Arnautovic, Hermann Kaindl, Sevan Kavaldjian, Dominik Ertl, Helmut Horacek und Cristian Bogdan. Automatic generation of the behavior of a user interface from a high-level discourse model. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*, Seiten 1–10, 2009.

[PS09] Plamen Paskalev und Ilka Serafimova. Runtime generation of a user interface, described in a database. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing - CompSysTech '09*, Seite 1, New York, New York, USA, 2009. ACM Press.

[PSS09] Fabio Paterno, Carmen Santoro und Lucio Davide Spano. Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environment. *ACM Transactions on Computer-Human Interaction*, 16(4), November 2009.

[PWB13] Andreas Pleuss, Stefan Wollny und Goetz Botterweck. Model-driven development and evolution of customized user interfaces. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '13*, Seite 13, New York, New York, USA, 2013. ACM Press.

[RKP+14] David Raneburger, Hermann Kaindl, Roman Popp, Vedran Šajatovi und Alexander Armbruster. A Process for Facilitating Interaction Design through Automated GUI Generation. In *SAC '14 Proceedings of the 29th Annual ACM Symposium on Applied Computing*, Seiten 1324–1330. ACM Press, New York, 2014.

[Tid06] Jenifer Tidwell. *Designing Interfaces: Patterns for effective Interaction Design*. O'Reilly Media, 2006.

[Trz10] Mariusz Trzaska. GUIs without Pain – the Declarative Way. In Rita Matrai, Hrsg., *User Interfaces*, number May. InTech, 2010.

[TVTB12] Vi Tran, Jean Vanderdonckt, Ricardo Tesoriero und François Beuvens. Systematic generation of abstract user interfaces. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12*, Seite 101, New York, New York, USA, 2012. ACM Press.

[TZ03] Shari Trewin und Gottfried Zimmermann. Abstract User Interface Representations : How Well do they Support Universal Access ? In *CUU '03: Proceedings of the 2003 conference on Universal usability*, Seiten 77–84, 2003.

[VLC11] Jan Van den Bergh, Kris Luyten und Karin Coninx. CAP3: Context-Sensitive Abstract User Interface Specification. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '11*, Seiten 31–40, 2011.

[VS10] Jean Vanderdonckt und Francisco Montero Simarro. Generative pattern-based design of user interfaces. *Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems - PEICS '10*, Seiten 12–19, 2010.

[Wol11] Andreas Wolff. *Modellbasierte Generierung von Benutzungsoberflächen*. Dissertation, 2011.

[YN08] Takuto Yanagida und Hidetoshi Nonaka. Architecture for Migratory Adaptive User Interfaces. In *8th IEEE International Conference on Computer and Information Technology, 2008. CIT 2008.*, Seiten 450–455, 2008.