Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3225

Ein semantisches Interaktionsmodell für Usability Patterns

Simon Brodtmann

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Jochen Ludewig
Betreuer:	Dipl.-Inf. Holger Röder
Begonnen am:	13. Juli 2011
Beendet am:	12. Januar 2011
CR-Klassifikation:	D.2.1, D.2.2

Zusammenfassung

Obwohl die Usability in der Softwareentwicklung in den letzten Jahren immer mehr an Bedeutung gewonnen hat, fehlt es in der Praxis noch an geeigneten Methoden und Werkzeugen, welche die Entwickler bei diesem Thema ausreichend unterstützen. Unkonkrete Anforderungen, unübersichtliche Pattern-Kataloge und die fehlende Prüfung am fertigen Produkt sind der aktuelle Stand der Praxis. Mit einem Katalog von Usability Patterns, welche funktionale Anforderungen an das Produkt definieren und in Use-Case-Spezifikationen integriert werden, wurde von Holger Röder bereits eine Methode entwickelt, welche die Usability einer Software mit System verbessern soll. Meine Vorgängerin Ruslana Brull erarbeitete darauf basierend eine Werkzeugunterstützung. Dieses Werkzeug erlaubt die Erstellung einer funktionalen Use-Case-Spezifikation mit eingebetteten Usability Patterns.

Das Ziel dieser Arbeit ist es, die Verwendung der Usability Patterns weiter zu vereinfachen, indem der Benutzer bei der Auswahl der Patterns unterstützt wird. Das geschieht einerseits durch Veränderung der Arbeitsweise mit den Patterns: Ein sehr abstraktes semantisches Interaktionsmodell ermöglicht es, Eigenschaften von Interaktionsschritten mit sinnvollen Vorschlägen für Usability Patterns zu verknüpfen. Dadurch muss der Benutzer nicht direkt geeignete Patterns auswählen, sondern beschreibt seine Interaktionsschritte und erhält daraufhin Vorschläge für Patterns.

Im zweiten Schritt wird versucht, die abstrakte Beschreibung der Interaktionsschritte zu automatisieren, um dem Benutzer auch diese Arbeit abzunehmen. Das geschieht durch die Analyse von vergangenen Beschreibungen von Interaktionsschritten, ähnlich wie es bei der Erkennung von Spam-Mail funktioniert. Eine anschließende Evaluierung dieses Verfahrens schließt die Arbeit ab.

Inhaltsverzeichnis

1. Einleitung

1.1. Aufgabenstellung

Es folgt ein vollständiges Zitat der Aufgabenstellung:

Hintergrund

Usability-Aspekte werden in Software-Projekten selten systematisch berücksichtigt. Zudem wird das Thema Usability häufig auf die grafische Gestaltung der Benutzungsschnittstelle reduziert, ohne etwa den Einfluss funktionaler Aspekte auf die Usability von Software zu beachten. Dies wird der Bedeutung guter Usability für den Erfolg einer Software in vielen Fällen nicht gerecht. Das Konzept «Usability Patterns» sieht aus diesem Grund vor, dass funktionale Merkmale, die die Usability der Software verbessern können, bereits früh im Entwicklungsprozess ausgewählt und – eng verzahnt mit den weiteren Anforderungen an die Software – explizit spezifiziert werden. Ein einzelnes Usability Pattern beschreibt dabei in strukturierter Form ein bewährtes funktionales Usability-Merkmal. Entwickler können auf einen Katalog von Usability Patterns zurückgreifen, um für den Nutzungskontext der Software geeignete Merkmale auszuwählen.

Die Anwendung von Usability Patterns wird durch die Annotierung von Use Cases spezifiziert. Use Cases einer vorhandenen Anforderungsspezifikation werden dabei speziell gekennzeichnet und um zusätzliche Anforderungen zur Integration der jeweiligen Usability-Merkmale ergänzt. Als Vorgabe für den Usability-Ingenieur definiert jedes Usability Pattern Spezifikationsschablonen, die beschreiben, wie und an welcher Stelle das beschriebene Usability-Merkmal spezifiziert werden kann. Am Institut für Softwaretechnologie wird aktuell das Java-basierte Use-Case-Werkzeug Tulip entwickelt, dass auf Grundlage dieser Spezifikationsschablonen die Annotierung von Use Cases erlaubt.

In den Spezifikationsschablonen wird ein einfaches syntaktisches Modell von Use Cases verwendet (Use Case – Ablauf – Schritt). Inhaltliche Aspekte der beschriebenen Interaktion zwischen System und Benutzern werden dabei momentan nicht berücksichtigt. So kann etwa formal nicht ausgedrückt werden, dass ein Usability Pattern nur auf Eingabeschritte des Benutzers, nicht auf Ausgabeschritte des Systems angewendet werden kann und dass somit auch nur entsprechende Eingabeschritte in den Use Cases annotiert werden dürfen.

Eine entsprechende Erweiterung des Modells um inhaltliche Aspekte erscheint geeignet, die Auswahl von Usability Patterns und die Spezifizierung ihrer Anwendung zu erleichtern.

Aufgabenstellung

Ziel der Diplomarbeit ist die Entwicklung eines semantischen Interaktionsmodells für Use Cases und die Integration dieses Modells in das Use-Case-Werkzeug Tulip.

Im ersten Schritt soll untersucht werden, welche Interaktionselemente auf Grundlage der Interaktionsbeschreibungen in Use Cases – ggf. auf unterschiedlichen Abstraktionsstufen – inhaltlich unterschieden werden können. Daraus soll ein semantisches Interaktionsmodell entwickelt werden.

Anschließend sollen die vorhandenen Spezifikationsschablonen für Usability Patterns auf Grundlage des entwickelten Modells präzisiert werden. Tulip soll um die Unterstützung für das semantische Modell und die präzisierten Spezifikationsschablonen erweitert werden. Das erweiterte Werkzeug soll im Rahmen des Softwarepraktikums (ab November 2011) evaluiert werden.

Schließlich soll untersucht werden, inwieweit eine (teil-)automatische inhaltliche Klassifizierung einzelner Interaktionselemente in Interaktionsbeschreibungen, also eine Abbildung auf das semantische Modell, möglich ist. Hier soll insbesondere die Möglichkeit geprüft werden, die Abbildung anhand von Schlüsselwörtern in den Interaktionsbeschreibungen durchzuführen. Falls möglich, soll eine entsprechende Funktionalität prototypisch in Tulip integriert werden.

Teilaufgaben

- Erstellung eines Projektplans mit Angabe von Meilensteinen
- Einarbeitung in die Themen Usability, Interaktionsmodellierung und Usability Patterns
- Entwicklung eines semantischen Modells für Interaktionselemente in Use Cases
- Präzisierung der Spezifikationsschablonen auf Grundlage des Modells
- Erweiterung des Use-Case-Werkzeugs Tulip um Unterstützung für das Modell
- Evaluation des erweiterten Werkzeugs
- Untersuchung der Möglichkeit einer automatischen inhaltlichen Klassifizierung von Interaktionselementen in Use Cases, ggf. prototypische Implementierung in Tulip
- Dokumentation des Vorgehens und der Ergebnisse in einem Bericht; zusätzlich Gestaltung eines Produktplakats im Format DIN A2

- Präsentation der Ergebnisse in einem Zwischen- und einem Endvortrag

1.2. Ziel der Diplomarbeit

Mit den Usability Patterns (Röder, 2011) und deren Integration in dem Use Case Editor Tulip (Brodtmann, et al., 2011), steht dem Softwareentwickler ein Werkzeug zur Verfügung, um die funktionalen Anforderungen mit weiteren, Usability fördernden, funktionalen Anforderungen zu ergänzen. Problematisch bei dieser Vorgehensweise ist, wie auch bei der manuellen Verwendung von Patterns jeglicher Art, dass der Softwareentwickler bei der Auswahl der Patterns den gesamten Katalog im Kopf haben muss. Er wählt aus dem Katalog jeweils die Patterns aus, die zum aktuellen Use Case passen. Das Ziel dieser Diplomarbeit ist es, die Auswahl an Patterns zu erleichtern, indem das notwendige Expertenwissen für den Benutzer reduziert wird und, indem die Auswahl an Patterns für einen Use Case sinnvoll – d. h. an den Kontext angepasst – reduziert wird.

2. Usability und Usability Patterns

2.1. Usability

Diese Arbeit befasst sich mit den Thema Usability in der Softwareentwicklung. Daher muss zunächst dieser Begriff definiert werden. Der englische Begriff Usability (im Deutschen Gebrauchstauglichkeit oder umgangssprachlich Benutzerfreundlichkeit) wird in der ISO-Norm 9241-11 (ISO 9241-11, 1998) definiert als:

> „Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen."

Die Usability ergibt sich also aus dem Produkt, dem aktuellen Benutzer und dem Nutzungskontext. Je nach Benutzer und Motivation des Benutzers ergibt sich also eine andere Usability. In dieser Arbeit gehe ich davon aus, dass der Entwickler, welcher im Begriff ist eine Software zu erstellen, seine Zielgruppe und den geeigneten Nutzungskontext bereits bestimmt hat, bzw. die erarbeiteten Konzepte lassen dem Entwickler die Freiheit, die Zielgruppe und den Nutzungskontext frei zu wählen und jederzeit zu ändern.

In der Praxis wird die Usability zwar bereits oft besprochen, findet meist aber keine Integration in den Entwicklungsprozess. Die Standardliteratur wie z. B. (Shneiderman, 1998) beschreibt zwar im Detail, wie man gute Usability erkennt und, wie man einzelne Elemente einer Software in dieser Hinsicht verbessern kann. Auf die Integration in den Entwicklungsprozess geht sie aber nicht ausreichend ein. Um die Usability zu integrieren, müssen Anforderungen an diese erhoben und spezifiziert werden. Anschließend muss sie in die Entwurfsphase integriert werden. Am Ende der Entwicklung muss die Usability gemäß den zuvor erhobenen Anforderungen überprüft werden. Bisher existierende nicht-funktionale Formulierungen wie „die Software sollte gut benutzbar sein" können weder in eine konkrete Umsetzung überführt, noch systematisch überprüft werden. Die Standardliteratur, aber auch Kataloge mit Lösungsmustern für gängige Probleme wie (Tidwell, 2005) und (Welie, 2008), richten sich an den Programmierer, welcher die Richtlinien und Lösungsvorschläge dieser Literatur nach eigenem Ermessen umsetzt. Das ist zwar eine brauchbare Basis für gute Usability, reicht aber mangels Systemantik und Prüfbarkeit nicht aus. Zudem ist nicht garantiert, dass der Programmierer in diesem Gebiet ausgebildet ist und es beachtet.

Mit einem gesteigerten Interesse an guter Usability, wächst auch der Bedarf an Methoden für den Entwicklungsprozess. Nachdem sich der Gedanke der Usability etabliert hatte, wurden Methoden für den Entwicklungsprozess quasi rückwärts entwickelt. In den neunziger Jahren des letzten Jahrhunderts wurden diverse Methoden zur Beurteilung der Usability einer Software, wie sie in (Holzinger, 2005) übersichtlich vorgestellt werden, entwickelt. Think aloud tests z. B. begleiten einen Benutzer bei der Verwendung einer Software und werten dessen laut ausgesprochenen Gedanken aus. Solche Verfahren eignen sich gut, um die Usability der Software zu bewerten und mit Anforderungen zu vergleichen. Dafür muss aber wenigstens ein Teil der Software bereits existieren.

Später stellten (Folmer, et al., 2003) und (Juristo, et al., 2004) das Konzept der „architecture-sensitive usability mechanisms" vor, was einem Katalog von Lösungsmustern entspricht. Diese Lösungsmuster enthalten explizit die Information, dass (und wie) sie in der Architektur der Software mit eingeplant werden müssen und bieten somit eine Möglichkeit, Usability teilweise in die Entwurfsphase einer Software zu integrieren.

Einen Schritt weiter gehen Usability Patterns wie sie in (Röder, 2011) beschrieben sind. Sie bilden einen Katalog von Lösungsmustern und werden als funktionale Anforderungen ausformuliert, in die Anforderungsspezifikation einer Software integriert. Damit ist sowohl eine Grundlage für die systematische Berücksichtigung der Usability in der Entwurfsphase, als auch eine Möglichkeit der systematischen Überprüfung der Usability in den Tests geschaffen. Zwar ist damit noch nicht automatisch ein Entwicklungsprozess mit integrierter Unterstützung für Usability geschaffen, aber es steht nun ebenfalls ein Werkzeug für die Anforderungserhebung zur Verfügung, sodass die Integration der Usability in den Entwicklungsprozess von Anforderung bis hin zu den Abnahmetests möglich ist.

2.2. Usability Patterns

Usability Patterns sind gängige Lösungsmuster für Probleme in Software, welche durch ihre Umsetzung nachweislich die Usability verbessern. Die Verwendung eines Usability Patterns resultiert nicht aus einer funktionalen Anforderung an das Produkt, schließlich möchte niemand eine Software kaufen, welche ihm „Undo" als alleinstehendes Feature anbietet. Die Verwendung eines Usability Patterns führt aber sehr wohl zu einer oder mehreren funktionalen Anforderungen. Soll z. B. eine Aktion rückgängig gemacht werden können, so wird bei der Verwendung des Usability Patterns „Undo", dieses als funktionale Anforderung ausformuliert – ebenso detailliert, wie die Anforderung an die eigentliche Funktion. Es wird also beschrieben,

dass eine Aktion rückgängig gemacht werden können soll und, wie diese Software das aus Benutzersicht umsetzen soll. Um die Übersicht zu bewahren, führt ein verwendetes Usability Pattern in einem Projekt lediglich zu einer einzigen Ausformulierung in der Anforderungsspezifikation. Diese Ausformulierung ist spezifisch für eine konkrete Umsetzung und wird ab diesem Zeitpunkt auch Usability Feature genannt.

Usability Patterns ergänzen die Use Case Spezifikation – ein Teil der Anforderungsspezifikation. Die Verknüpfung zwischen Usability Patterns und Use Cases geschieht mittels Annotationen. Eine Annotation stellt die konkrete Verwendung eines Patterns an einer bestimmten Stelle in der Software dar. Beschreibt das Pattern „Undo“ wie die Software dieses grundsätzlich umsetzen soll und wie die Interaktion mit dem Benutzer gestaltet werden soll, so steht die Annotation, verknüpft mit einem Interaktionsschritt in der Use Case Spezifikation, für eine konkrete Stelle an der ein Undo möglich sein soll.

Zusammengefasst gilt: Ein Usability Pattern wird je Anforderungsspezifikation einmalig als Usability Feature ausformuliert. Anschließend werden beliebig viele Interaktionsschritte mit einer Annotation dieses Patterns verknüpft.

3. Semantisches Interaktionsmodell

Wie bereits in „1.2 Ziel der Diplomarbeit“ erwähnt, ist ein Ziel dieser Arbeit, Benutzern der Usability Patterns die Arbeit mit den Patterns zu erleichtern. In der Aufgabenstellung der Diplomarbeit wird der Begriff „semantisches Interaktionsmodell“ bereits als mögliche Lösung für das Problem verwendet. Im Folgenden werde ich diesen Begriff für den weiteren Verlauf der Arbeit definieren.

Der Begriff semantisches Interaktionsmodell besteht aus zwei Wörtern. Es ist ein Interaktionsmodell, da es Interaktionsschritte zwischen Benutzern einer Software und der Software selbst in Form von natürlicher Sprache als Abbildungsmerkmal[1] hat. Die Semantik dieser Interaktionsschritte soll, im Gegensatz zu vielen existierenden Interaktionsmodellen, durch dieses Modell soweit abstrahiert werden, sodass mit einem reduzierten Vokabular automatisiert Aussagen über die Interaktion getroffen werden können, ohne dass dabei für die Usability Patterns wesentliche Informationen verloren gehen. Mit Hilfe dieser Aussagen kann eine Verknüpfung zwischen den Interaktionsschritten und den Patterns hergestellt werden. Bestehende Interaktionsmodelle wie z. B. UML-Aktivitätsdiagramme und Use Case Beschreibungen abstrahieren zwar ebenfalls Interaktionsschritte, allerdings werden dort die semantischen Aussagen nach wie vor in natürlicher Sprache ohne beschränktes Vokabular formuliert, was eine automatisierte Auswertung aufwändig und fehleranfällig macht.

Zur Veranschaulichung: Das Usability Pattern „direkte Validierung“ ist ausschließlich auf Interaktionsschritte anwendbar, in denen eine textuelle Eingabe von einem menschlichen Benutzer getätigt wird. Das semantische Interaktionsmodell soll in diesem Beispiel eine Verbindung zwischen den Informationen „textuelle Eingabe“ und „menschlicher Benutzer“, sowie dem dafür geeigneten Pattern „direkte Validierung“ ermöglichen.

Zweck des Interaktionsmodells ist, dem Benutzer der Usability Patterns die Auswahl aus dem Katalog zu erleichtern, indem das notwendige Expertenwissen über die Patterns und die Auswahl der Patterns für einen Use Case, dem jeweiligen Kontext entsprechend, auf eine überschaubare Zahl reduziert wird. Das Interaktionsmodell muss dazu selbst das

[1] Nach der Definition von „Modell“ in (Ludewig & Lichter, 2007)

Expertenwissen enthalten um eine Verbindung zwischen den Interaktionsschritten und den Usability Patterns herstellen zu können.

3.1. Literatur

Zuerst stellt sich die Frage, ob es schon Überlegungen gibt, die die gleiche oder eine ähnliche Fragestellung wie die aus der Aufgabenstellung dieser Diplomarbeit zur Grundlage haben.

Usability Patterns tauchen in einer ähnlichen Form bereits 2004 in diversen Arbeiten auf. In (Juristo, et al., 2004) werden mit „Architecture-sensitive usability mechanisms" Architekturmuster vorgestellt, mit denen die Usability bereits während der Erstellung der Softwarearchitektur verbessert werden kann. Ebenso wie Pattern-Kataloge im Bereich Interface Design – z. B. (Tidwell, 2005) und (Welie, 2008) – überlassen die Autoren die Auswahl der Patterns aber dem Benutzer. Zwar gibt es bereits Ansätze zur Auswahl von Patterns aus großen Katalogen – (Zdun, 2006) beschreibt ein solches Verfahren – allerdings geht es dabei um die Auswahl von Patterns für ein Projekt. Für die Auswahl von Patterns auf Use-Case-Ebene konnte ich keine Ansätze finden.

Ein semantisches Interaktionsmodell soll die Use-Case-Beschreibungen in natürlicher Sprache abstrahieren. Daher folgt eine Literaturrecherche zu dem Thema Interaktionsmodellierung.

Es gibt zahlreiche Techniken, welche sich mit der Modellierung von Mensch-Maschine-Interaktionen befassen. Repräsentanten dieser Techniken sind GOMS (Goals Operators Methods and Selection rules) (Card, et al., 1983) und UAN (User Action Notation) (Hix & Hartson, 1993). Diese Techniken haben gemeinsam, dass sie eine *Zerlegung* von Interaktionen in atomare Schritte, wie z. B. einzelne Klicks und Mausbewegungen, vornehmen. Ziel dieser Diplomarbeit ist aber eine *Abstrahierung* von Interaktionen um semantische Aussagen darüber treffen zu können. Im Bereich der Interaktionsmodellierung bin ich daher ebenfalls nicht fündig geworden.

Die Forschung im Bereich der künstlichen Intelligenz beschäftigt sich mit ähnlichen Aufgaben wie die dieser Arbeit. In (Feigenbaum, 1977) wird im Bereich der Wissensmodellierung, ein Teilbereich der künstlichen Intelligenz, das Konzept der Expertensysteme vorgestellt. Ein Expertensystem verkörpert das Wissen eines Experten auf einem bestimmten Wissensgebiet in Form einer Software. Da der Pattern Katalog mit den Usability Patterns bereits einen Teil des Expertenwissens beinhaltet, ist es naheliegend in diese Richtung zu gehen. Da ein Expertensystem immer spezifisch für ein Wissensgebiet ist, gibt es keine fertige Lösung für die

Aufgabenstellung dieser Arbeit. Es wird daher eine eigene Lösung für ein Expertensystem vorgestellt.

Auch Aspekte aus der Linguistik sollten beachtet werden, denn letztendlich ist eine Aufgabe dieser Arbeit die Abstrahierung von Informationen aus Use-Case-Beschreibungen in natürlicher Sprache. Darauf werde ich in „4. Automatische Klassifizierung von Use Cases“ näher eingehen.

3.2. Entwicklung des Modells

Das zu entwickelnde Modell soll Beschreibungen von Interaktionsschritten in natürlicher Sprache soweit abstrahieren, sodass eine automatisierte Abbildung auf die Usability Patterns möglich ist. Ich suche daher als erstes nach einer Klassifizierung von Interaktionsschritten, welche abstrakt genug ist, um einen Großteil aller möglichen Interaktionsschritte mit dem Pattern-Katalog verbinden zu können. Das fertige Interaktionsmodell bietet eine geeignete Klassifizierung für beliebige Interaktionsschritte und verbindet diese Klassifizierung mit dem momentanen Stand des Pattern-Katalogs.

3.2.1. Anforderungen an das Modell

Umfang

Im ersten Schritt wird die Klassifizierung von Hand vorgenommen. Use-Case-Beschreibungen müssen also von Hand klassifiziert werden. Daraus ergibt sich die Anforderung, dass die Klassifizierung überschaubar sein muss. Je nach gewählter Technik, ergeben sich daraus unterschiedliche Einschränkungen für die Klassifizierung: In einer flachen Klassifizierung sollte die Zahl der Klassen das Fassungsvermögen des Kurzzeitgedächtnisses nicht überschreiten. In einer hierarchischen Klassenstruktur gilt es zusätzlich die Tiefe stark zu beschränken.

Abstraktionsgrad

Die Klassifizierung muss abstrakt genug sein, um die Einschränkungen einhalten zu können. Gleichzeitig muss sie abstrakt genug sein, sodass einzelne Klassen eine ausreichende Aussagekraft aus semantischer Sicht darstellen. So lassen sich die Klassen „Mausbewegung über ein Textfeld“, „Klick“ und „Drücken von alphanumerischen Tasten“ nicht eindeutig mit einzelnen (wenigen) Patterns verbinden. Kombiniert man diese drei Klassen allerdings zu „textuelle Eingabe“ (die Information, dass die Eingabe in einem Textfeld geschieht und nicht „ins leere“, ist darin enthalten), dann macht diese Klasse für sich genommen eine Aussage, die eindeutig mit bestimmten Patterns verbunden werden kann. Die Patterns „direkte Validierung“ und „Auto-

Vervollständigung“ können z. B. mit dieser Klasse verknüpft werden. Im Folgenden fasst der Begriff „Interaktionsschritt“ diese beiden Ebenen zusammen.

Vollständigkeit

Einen Anspruch auf Vollständigkeit gibt es bei dem Interaktionsmodell nicht. Es müssen weder alle Interaktionsaktionsschritte Klassifiziert werden können, noch müssen alle möglichen Klassen gefunden werden. Es reicht bereits wenn der Großteil aller Interaktionsschritte klassifizierbar ist und wenn das resultierende Interaktionsmodell in der Praxis einen Vorteil bringt.

Erweiterbarkeit

Da sich die Softwareentwicklung stetig weiterentwickelt, wird sich in Zukunft sowohl der Pattern Katalog als auch die Notwendigkeit für bestimmte Klassen im Interaktionsmodell ändern. Es muss daher möglich sein, Änderungen am Interaktionsmodell ohne großen Aufwand umzusetzen.

Nutzungskontext

Usability Patterns können auf mehreren Ebenen einer Use-Case-Spezifikation angewendet werden. Die Spezifikation als Ganzes beinhaltet die Anforderungen an ein gesamtes Softwareprojekt. Es gibt Usability Patterns, welche auf dieser Ebene angewendet werden, allerdings gibt es hier nur ein Element, eben dieses eine Projekt. Der Nutzen des Interaktionsmodells ist auf dieses eine Element beschränkt. Es wird daher auf eine Unterstützung auf Projektebene verzichtet. Einzelne Use-Case-Spezifikationen bestehen aus Abläufen: Einem Normalablauf und optional Alternativabläufe für Sonderfälle. Einige der Usability Patterns (z. B. „globales Undo“) werden mit Abläufen verknüpft. Abläufe bestehen wiederum aus Schritten. Ein Schritt setzt sich zusammen aus einer Position innerhalb des Ablaufes, einem Akteur und einer Beschreibung. Zusätzlich kann ein Alternativablauf mit einem Schritt verknüpft werden, wodurch man Verzweigungen der Abläufe markiert. Einige Usability Patterns (z. B. „direkte Validierung“) werden mit Schritten verknüpft. Das Interaktionsmodell muss Abläufe und Schritte unterstützen. Diese unterschiedlichen Ebenen in der Use-Case-Beschreibung sind in dieser Arbeit nicht immer relevant und werden daher oft nicht explizit unterschieden. Um das Modell zu optimieren, werden die Informationen aus der Klassifizierung der einzelnen Schritte verwendet, um die Abläufe teilweise automatisiert zu klassifizieren. Wird ein Schritt als „irreversibel“ klassifiziert, dann kann diese Information auf den Ablauf übertragen werden und es kann daraus gefolgert werden, dass das Pattern „globales Undo“ darauf nicht anwendbar ist.

3.2.2. Klassifizierungsstrategien

Im Folgenden stelle ich zwei mögliche Klassifizierungsstrategien vor, aus denen ich eine geeignete auswähle.

Taxonomie

Eine Taxonomie zerteilt ein Wissensgebiet in Partitionen und stellt Beziehungen zwischen ihnen her[2]. Das ist auch rekursiv möglich in einem hierarchischen Taxonomiebaum. So können Klassen mit jedem Schritt weiter verfeinert und unterteilt werden, bis die gewünschte Granularität erreicht ist.

Für die Einordnung eines Interaktionsschrittes in den Taxonomiebaum, wählt man einen Pfad durch den Baum. Mit jedem Schritt der Klassifizierung trifft man eine weitere Aussage über die Interaktion und erhält, mit Erreichen eines Blattes im Baum, eine – innerhalb des Interaktionsmodells – vollständige Klassifizierung.

In einem Baum ist der Pfad von der Wurzel zu einem Blatt stets eindeutig. Es reicht daher aus die Blätter des Baumes mit Aussagen über die Usability Patterns zu verknüpfen.

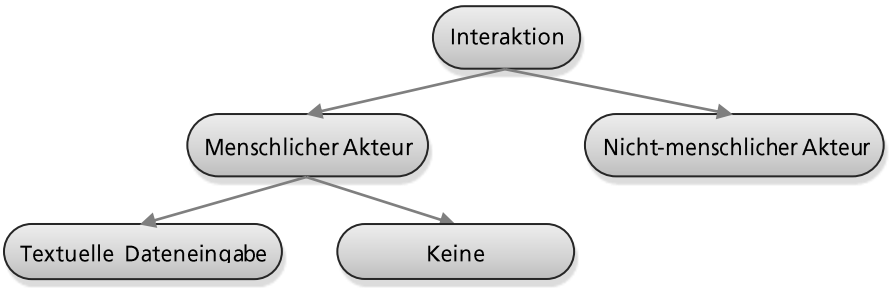

Abbildung 1: Beispiel eines Taxonomiebaumes

In Abbildung 1 ist ein mögliches Beispiel für einen Taxonomiebaum dargestellt. Wählt man für einen Interaktionsschritt den Pfad von der Wurzel über „Menschlicher Akteur“ zu „Textuelle Dateneingabe“, dann trifft man damit zwei Aussagen. Das Modell, welches das nötige Expertenwissen enthält, verbindet nun das Blatt „Textuelle Dateneingabe“ mit der Aussage, dass das Usability Pattern „direkte Validierung“ mit einer hohen Wahrscheinlichkeit sinnvoll für diesen Interaktionsschritt verwendet werden kann.

Vorteile

- Die Klassifizierung eines Aktionsschrittes ist sehr intuitiv, denn auf dem Weg von der Wurzel bis zu einem Blatt gibt es bei jedem Schritt nur eine Frage, welche mit der Wahl

[2] Vergleich (Ludewig & Lichter, 2007) bzw. (IEEE Standard 610.12, 1990)

aus wenigen Alternativen Kindknoten beantwortet werden kann. Es handelt sich sozusagen um eine geführte Befragung des Benutzers zu dem Interaktionsschritt.

- Eine Partitionierung ist per Definition eine Einteilung oder Zerlegung[3] und damit vollständig. Somit trifft stets eine der Alternativen für den Interaktionsschritt zu. Stellt es sich bei der Erstellung des Modells als unmöglich heraus alle Elemente einer Partition zu definieren oder scheint es nicht sinnvoll, dann kann z. B. mit einem Element „Sonstiges" die Vollständigkeit erreicht werden. Somit ist es für jeden beliebigen Interaktionsschritt möglich ein Blatt im Taxonomiebaum zu erreichen.

Nachteile

- Sind zwei Fragen unabhängig voneinander, dann müssen beide Fragen beantwortet werden können. Kann es sich in dem obigen Beispiel auch um eine textuelle oder keine Dateneingabe handeln wenn es sich um einen nicht-menschlichen Akteur handelt, dann muss der Taxonomiebaum zwangsweise Knoten mehrfach enthalten. Mit zunehmender Zahl an unabhängigen Fragen müssen immer mehr Kopien eines Knotens erstellt werden.

Attribute

Ein Attribut beschreibt eine Eigenschaft eines Interaktionsschrittes und stellt somit eine Klasse im Interaktionsmodell dar. Zur detaillierten Klassifizierung kann aus einer definierten Menge von Attributen eine beliebige Teilmenge ausgewählt und dem Interaktionsschritt zugeordnet werden. Die Attribute haben weder eine Reihenfolge noch eine Hierarchie. Jedes Attribut trifft Aussagen über beliebig viele Usability Patterns.

Überträgt man nun das obige Beispiel aus Abbildung 1 auf die attributbasierte Klassifizierung, dann stellt jeder Knoten ein Attribut dar. Alle Knoten auf einem Pfad im Taxonomiebaum entsprechen den Attributen, die gewählt werden müssen, um eine äquivalente Aussage zu treffen.

Vorteile

- Da die Attribute unabhängig voneinander sind, müssen keine Attribute mehrfach vorkommen. Man hat für jeden Interaktionsschritt die Auswahl aus allen vorhandenen Attributen die nicht bereits zugewiesen sind. Das Interaktionsmodell wird dadurch wesentlich kompakter als mit einem Taxonomiebaum.

[3] Laut Duden

Nachteile

- Die Attribute haben weder eine Ordnung noch eine Hierarchie. Daher werden sie bei der Verwendung linear aufgelistet. Um die Auswahl an Attributen übersichtlich zu halten, sollte die Zahl daher stark begrenzt bleiben.
- Attribute besitzen in der Form keine Abhängigkeiten. Da bestimmte Aussagen sich aber widersprechen können, muss eine Abhängigkeit möglich sein.

Fazit

Die bisherigen Informationen zu den beiden vorgestellten Strategien reichen nicht aus, um eine endgültige Entscheidung treffen zu können. Dazu müssen zuerst der Umfang der Klassen sowie die Abhängigkeiten untereinander bekannt sein. Anschließend kann mit einer beispielhaften Umsetzung veranschaulicht werden, welche der Strategien besser geeignet ist und ob Anpassungen der Grundidee notwendig sind.

3.2.3. Datengrundlage

Die Klassifizierung von Interaktionsschritten soll helfen, Usability Patterns den Interaktionsschritten aus beliebigen in der Praxis vorkommenden Use Cases zuzuordnen. Es gibt folgende Quellen, welche zur Ermittlung von Klassen verwenden werden können:

- Existierende Software: Software aus dem Alltagsgebrauch bietet eine gute Grundlage um Klassen zu finden und eine mögliche Klassifizierung zu testen. Mit der richtigen Auswahl an Software wird bereits ein Großteil aller möglichen Interaktionen und damit Klassen abgedeckt.
- Use Cases: Bestehende Spezifikationen – wie sie von Tulip z. B. bereits vorliegt – enthalten Interaktionsschritte bereits in einer abstrakteren Form als die Anwendung selbst und sind daher einfacher zu verwerten. Allerdings ist es schwieriger an eine Spezifikation zu kommen als an die dazugehörige Software. Ein Nachteil von Use-Case-Spezifikationen ist, dass sie aus Sicht des Softwareentwicklers und nicht aus Sicht eines echten Benutzers formuliert sind.
- Usability Patterns: Da die Klassen des Interaktionsmodells auf die Usability Patterns abgebildet werden, ist es sinnvoll die Klassen aus Sicht der Patterns zu erstellen. So ergeben sich keine Klassen die nutzlos sind, also auf kein Pattern abgebildet werden. Andersherum können so Patterns gefunden werden, die keine passende Klasse besitzen.

Mit Hilfe dieser Datengrundlage ist es möglich eine praxisrelevante Menge an Interaktionsschritten zu ermitteln und Klassen dafür zu bilden. Zudem können Klassen gefunden

werden, für die es noch kein Usability Pattern gibt (Hinweis auf den Bedarf von weiteren Patterns) und Patterns, für die es noch keine Klassen gibt.

3.2.4. Aussagen über Interaktionsschritte

Da der Katalog mit den Usability Patterns für diese Arbeit vorgegeben ist, stelle ich die Sicht der Patterns in den Vordergrund. Für jedes Pattern ist es notwendig einige Fragen zu den Interaktionsschritten zu beantworten, um eine Empfehlung machen zu können. Im Folgenden werde ich diese Fragen ausformulieren, um damit anschließend die Klassifizierung durchzuführen.

Wie bereits in „3.2.1 Anforderungen“ → „Nutzungskontext“ erwähnt, ist die Unterscheidung zwischen Ablauf und Schritt nicht immer relevant. Auch bei den Fragestellungen und anschließend in den Klassen wird dies zunächst nicht unterschieden, denn die Klassen werden nur mit relevanten Usability Patterns verknüpft, welche bereits eine Anwendbarkeit auf Ablauf oder Schritt definiert haben. Diese Anwendbarkeit ist daher implizit auch für die Klassen gegeben.

Um welche Art von Akteur handelt es sich?

Usability ist per Definition nur relevant für menschliche Akteure, also für Use Cases mit mindestens einem Schritt, der von einem Menschen durchgeführt wird. Maschinengesteuerten Aktionen führen daher nur dann zu einem Pattern-Vorschlag, wenn der menschliche Akteur davon betroffen ist z.B. durch eine lange Ausführungsdauer bei einer Aktion, die sonstige Arbeiten mit der Anwendung verhindert.

Inwiefern wird mit Datensätzen gearbeitet?

Für viele Usability Patterns ist es wichtig zu wissen, ob überhaupt auf einem Datensatz gearbeitet wird oder nicht. Mit Datensatz ist z. B. eine Datei auf der Festplatte oder eine Datenbank gemeint.

Es wird daher unterschieden:

- Eine Aktion verändert einen Datensatz (schreibender Datenzugriff).
- Eine Aktion zeigt Daten an oder navigiert, ohne einen Datensatz zu verändern (lesender Datenzugriff). Ansichts- und Tabwechsel gehören z. B. in diese Kategorie.

- Eine Aktion führt eine Konfiguration der Anwendung durch (kein Datenzugriff). Die Arbeit mit der Anwendung wird dadurch verändert (in der Regel erleichtert), aber es wird kein Datensatz verwendet.

Um welche Art von Datenveränderung handelt es sich?

Um eine datenverändernde Aktion weiter zu spezifizieren, stellt sich die Frage nach der Art der Datenveränderung. Die Usability Patterns, welche bei einer Datenveränderung verwendet werden können, legen folgende Antworten auf diese Frage nahe:

- Die Aktion verändert Daten in einem geöffneten Datensatz.
- Die Aktion öffnet einen bestehenden Datensatz, legt einen neuen Datensatz an, speichert oder schließt einen geöffneten Datensatz. Diese Aktionen werden nicht unterteilt, da die Unterteilung für die vorhandenen Patterns bereits feingranular genug ist und weil einige dieser Aktionen oft in Kombination aufgerufen werden. Datensatz öffnen schließt z. B. häufig den aktuell offenen Datensatz.

Handelt es sich um eine Dateneingabe?

Für die Eingabe von Daten gibt es diverse Möglichkeiten der Unterstützung durch eine Software und entsprechend einige Usability Patterns. Daher wird unterschieden zwischen textuellen Dateneingaben, nicht-textuellen Dateneingaben (z. B. per Maus) und sonstigen Aktionen (keine Dateneingabe). Nicht-textuelle Dateneingaben geschehen z. B. über Formularfelder jeglicher Art, die mit der Maus bedient werden. Sie werden von den textuellen Eingaben unterschieden, da sie nur stark begrenzte Wertmengen zulassen und daher andere Unterstützungen für den Benutzer benötigen.

Kann die Aktion rückgängig gemacht werden?

Einige Patterns beschäftigen sich mit irreversiblen Aktionen. Aus Benutzersicht können irreversible Aktionen fatale Folgen haben und müssen daher besonders beachtet werden.

Findet diese Aktion in einem modalen Dialog statt?

Modale Dialoge haben einen besonderen Status dadurch, dass sie die Aufmerksamkeit des Benutzers auf diese eine Aufgabe lenken und weitere Aktionen ausschließen.

Wie lang ist die Reaktionszeit des Systems nach Abschluss der Aktion?

Aktionen jeder Art können langwierig sein. Viele Programme gehen mit solchen Aktionen falsch um und verwirren den Benutzer. Gemeint ist explizit nicht die Dauer die der Benutzer zur

Eingabe oder zum Auslösen benötigt, sondern die Dauer die der Benutzer auf die Fertigstellung der Aktion wartet. Eine Reaktionszeit von bis zu einer Sekunde wird dabei als "sofort" (kurz) akzeptiert (Shneiderman, 1998). Längere Wartezeiten stören den Arbeitsfluss und sollten von Usability Patterns unterstützt werden.

Aus wie vielen Eingabeschritten besteht der Ablauf?

Ab einer gewissen Zahl von Eingabeschritten für einen Ablauf wird dieser zunehmend unübersichtlich und damit schlechter benutzbar. Diese Frage ist für Abläufe, aber nicht für Schritte beantwortbar. Wurde zudem die Frage „Handelt es sich um eine Dateneingabe?“ für alle Schritte eines Ablaufs beantwortet, dann kann die Zahl der Eingabeschritte automatisch ermittelt werden.

3.2.5. Klassifizierungen

Mit Hilfe der obigen Fragen ist es nun möglich eine einfache Klassifizierung für das Interaktionsmodell zu erstellen.

Taxonomie

Bei einer Taxonomie werden obige Frage nacheinander alle beantwortet. Jede mögliche Antwort auf eine Frage stellt ein Kindknoten des aktuellen Knotens dar. Begonnen wird mit der Wurzel, welche selbst keine Aussage darstellt. Ein Interaktionsschritt ist fertig klassifiziert, wenn ein Blatt im Taxonomiebaum erreicht ist.

Fragen, die unabhängig von allen anderen Fragen gestellt werden können, müssen in jedem möglichen Pfad vorkommen und entsprechend häufig im Baum vorkommen. Es fällt auf, dass die Fragen nur wenige Abhängigkeiten besitzen. Daher wird der Taxonomiebaum relativ groß. Konkret: Gibt es keine Abhängigkeiten der Fragen und es gibt n unterschiedliche Fragen, dann sind insgesamt n! Fragen im Baum untergebracht, jede mit zwei bis drei Antwortmöglichkeiten.

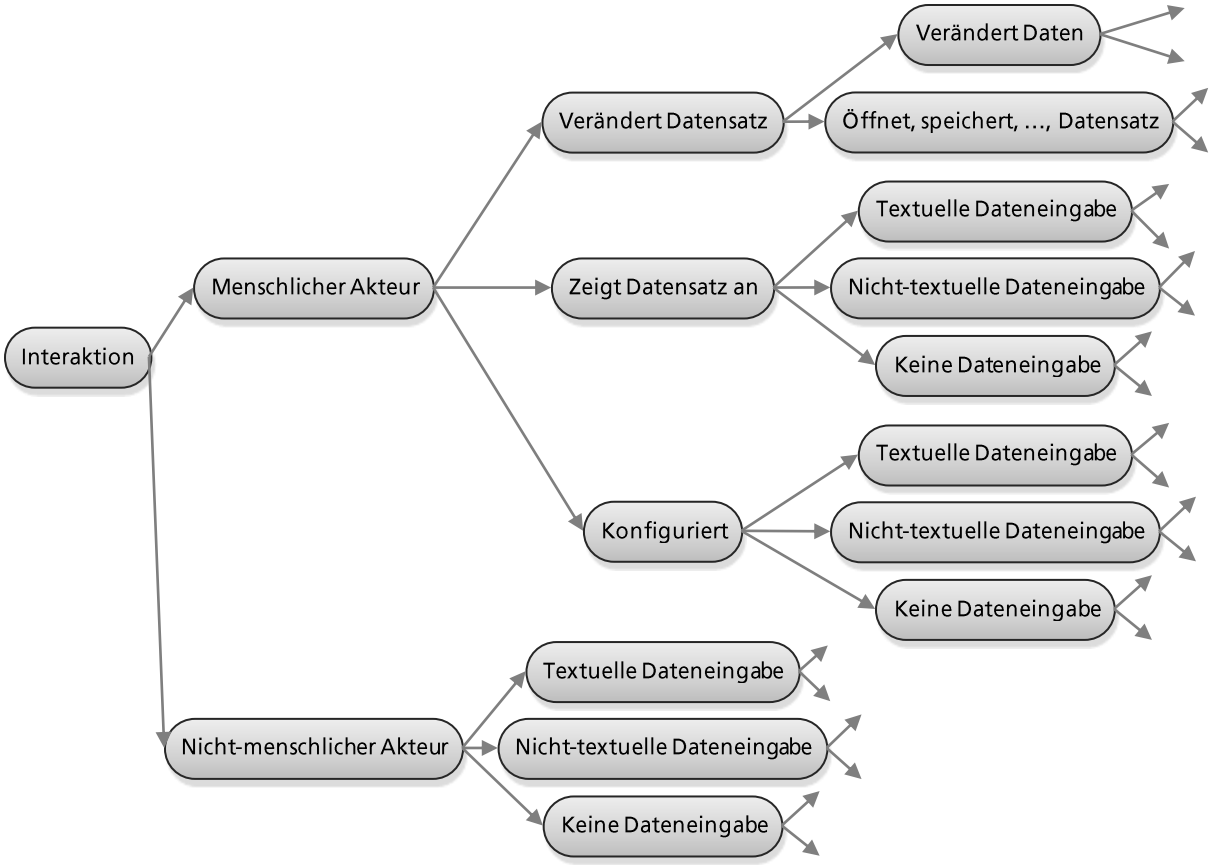

Abbildung 2: Teil einer Klassifizierung mit Hilfe eines Taxonomiebaumes

Abbildung 2 zeigt zur Übersicht nur einen Teil des Taxonomiebaumes. Obwohl erst drei Fragen beantwortet wurden, ist der der Baum schon relativ umfangreich.

Fazit

Folgende Gründe sprechen gegen eine Taxonomie:

- In jedem Schritt gilt es eine Frage zu beantworten. Sollen alle Fragen beantwortet werden, dann erhält man einen sehr tiefen Baum. Der Pfad von der Wurzel zu einem Blatt wird daher sehr lang. Gleichzeitig gibt es je Schritt nur wenige Alternativen, da eine Frage nur zwei bis drei Antwortmöglichkeiten bietet.
- Ziel bei der Klassifizierung eines Interaktionsschrittes ist es, anfangs alle Usability Patterns vorzuschlagen und mit jedem Schritt im Taxonomiebaum die Auswahl zu verkleinern. Da einige Patterns in keine Kategorie passen, würden sie in jedem Fall bis zum letzten Schritt vorgeschlagen werden. Die Auswahl an Patterns würde daher und wegen der hohen Tiefe sehr langsam abnehmen und vermutlich niemals die gewünschte Größe erreichen.

- Es gibt sehr wenige Abhängigkeiten zwischen den Fragen. Daher müssen sehr viele Knoten im Baum mehrfach vorkommen. Eine Taxonomie wäre besser geeignet, wenn es mehr Abhängigkeiten gäbe.

Attribute

Ein Attribut stellt eine Antwort auf eine der obigen Frage dar und ist mit Usability Patterns über eine Empfehlung verknüpft. Es kann – unter Beachtung definierter Abhängigkeiten – eine beliebige Teilmenge der Attribute einem Interaktionsschritt zugewiesen werden. Durch verrechnen der einzelnen Empfehlungen für ein bestimmtes Usability Pattern, kann die endgültige Empfehlung für dieses Pattern bestimmt werden. Empfehlen zwei gesetzte Attribute das gleiche Pattern, dann ergibt sich daraus eine stärkere Empfehlung, als durch nur eines der Attribute. Während ein Interaktionsschritt mit Attributen versehen wird, verbessert sich so mit jedem Schritt die Auswahl an Usability Patterns. Die Attributierung für einen Interaktionsschritt wird beendet, wenn alle Fragen beantwortet sind oder, wenn die Auswahl an Patterns ausreichend übersichtlich ist.

Um Abhängigkeiten zwischen den Antworten einer Frage zu vereinfachen (jede Frage darf für einen Interaktionsschritt nur einmal beantwortet werden), gibt es zu jeder Frage genau ein Attribut. Ein Attribut hat stets einen Wert, welcher der Antwort auf die Frage entspricht. Abbildung 3 zeigt ein kleines Beispiel, um die Funktionsweise der Attribute zu veranschaulichen.

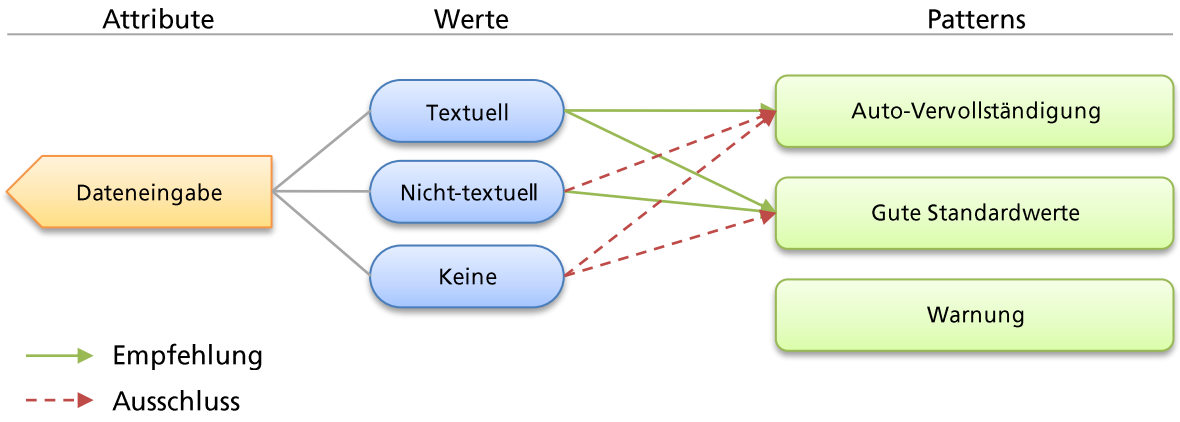

Abbildung 3: Teil einer Klassifizierung mit Hilfe von Attributen

Fazit

Die Nachteile der Taxonomie hat die Attributierung nicht. Es spricht auch sonst nichts gegen diese Klassifizierungsstrategie. Aus den acht Fragen mit je zwei bis drei möglichen Antworten erhält man ein überschaubares Modell. Bei der Klassifizierung eines Interaktionsschrittes stehen anfangs alle Usability Patterns zur Auswahl. Wurden Attribute zugewiesen, so werden einzelne

Patterns empfohlen und von anderen wird abgeraten. Man muss nicht alle Fragen beantworten, sondern bricht ab, sobald die Empfehlungen für eine Entscheidung ausreichen.

Positiv ist zudem, dass manche Attribute automatisch aus dem Kontext ermittelt werden können und daher nicht vom Benutzer selbst gesetzt werden müssen. Die Frage „Um welche Art von Akteur handelt es sich?“ kann z. B. einmalig je Akteur beantwortet werden.

3.3. Das Attributbasierte Interaktionsmodell

Die beispielhafte Umsetzung der Lösungsvorschläge hat ergeben, dass eine Attributbasierte Klassifizierung eine geeignete Lösung ist. Im Folgenden wird das Attributbasierte Interaktionsmodell ausführlich beschrieben. Es wird dazu in drei Abstraktionseben unterteilt.

3.3.1. Metamodell

Wie bereits beschrieben, besteht das Metamodell aus einer Menge von Attributen. Ein Attribut hat einen Namen und entspricht einer Fragestellung aus „**Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**“. Zudem hat ein Attribut mindestens einen Wert.

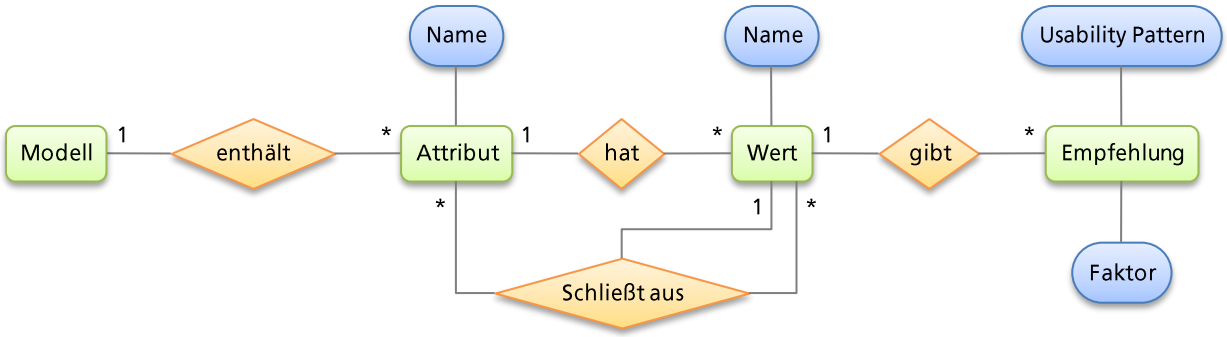

Abbildung 4: ER-Diagramm des Metamodells

Ein Wert hat einen Namen und entspricht einer Antwort auf die Fragestellung des Attributs. Ein Wert kann andere Werte oder Attribute ausschließen. Wurde z. B. festgelegt, dass es sich bei einem Interaktionsschritt nicht um eine datenverändernde Aktion handelt, dann ist es nicht sinnvoll die Art der Datenveränderung festzulegen. Außerdem gibt ein Wert beliebig viele Empfehlungen.

Eine Empfehlung besteht aus einem Usability Pattern (bzw. aus dessen eindeutigen ID) und einem Faktor. Die Empfehlung wird im folgenden Abschnitt genauer erläutert.

Empfehlung

Eine Empfehlung macht eine Aussage darüber, ob ein Usability Pattern für den verknüpften Interaktionsschritt verwendet werden sollte oder nicht. Da Werte von Attributen beliebige Empfehlungen machen können, ist es möglich, dass zwei oder mehr Attribute je eine Empfehlung für dasselbe Usability Pattern machen. Die Empfehlungen müssen daher verrechnet werden können.

Eine einfache Möglichkeit ist, positive Empfehlungen und Ausschlüsse zu zählen und auf einer Ordinalskala zu vergleichen. Dafür spricht, dass über den Wert an sich keine Aussage getroffen werden kann. Lediglich die Relation zu anderen Werten ist relevant. So kann gesagt werden: Ein Pattern A wird durch zwei Attributwerte empfohlen; Pattern B nur durch einen. Pattern A wird daher stärker empfohlen als Pattern B. Über den absoluten Wert der Empfehlung lässt sich keine Aussage treffen, da die Grundlage für jegliche Aussage fehlt.

Gegen die Ordinalskala spricht die Möglichkeit der späteren Justierung des Modells durch Erfahrungswerte. Führt in dem vorherigen Beispiel die Empfehlung von Pattern B viel häufiger zu einer tatsächlichen Verwendung des Patterns, als bei Pattern A, dann muss die Empfehlung von Pattern B stärker gewichtet werden, als die von Pattern A. Es spricht ebenfalls gegen die Ordinalskala, dass ein sicherer Ausschluss eines Patterns durch eine weitere Empfehlung nicht ausgeglichen werden kann. Dazu ist eine Multiplikation mit 0 notwendig.

Es wird daher eine Rationalskala benötigt. Sie lässt nicht nur die Justierung der einzelnen Wert-Pattern-Relationen zu, sondern ermöglicht die Erweiterung des Modells um weitere Aussagen über die Attributierung der Umgebung (z. B. Schritte im selben Ablauf).

Auf dieser Rationalskala hat der Nullpunkt die Bedeutung, dass ein Pattern nicht sinnvoll anwendbar ist (Ausschluss). Der Wert 1 stellt eine neutrale Aussage für ein Pattern dar (es kann keine Aussage getroffen werden). Werte größer als 1 drücken eine positive Empfehlung aus und Werte zwischen 0 und 1 eine negative Empfehlung.

3.3.2. Modell

Das Modell beschreibt die konkreten Attribute mit ihren Werten und Empfehlungen. Zwar können die Empfehlungen durch eine beliebige rationale Zahl dargestellt werden, allerdings beschränke ich mich in diesem Modell auf Ausschlüsse (Wert 0) und Empfehlungen mit dem Wert 2. Usability Patterns für die keine Aussage gemacht wird, erhalten den Wert 1. Sämtliche Empfehlungen sind damit gleich stark. Für eine feinere Justierung der Werte wären empirische

Daten notwendig, welche mir nicht zur Verfügung stehen. Zudem reicht es, um zu zeigen ob dieses Modell grundsätzlich funktioniert, mit diesen drei Werten zu arbeiten.

Abläufe und Schritte eines Use Cases müssen im Modell unterschieden werden. Nicht jedes Attribut ist auf beide anwendbar. Teilweise können Attribute automatisch aus dem Kontext heraus gesetzt werden. Eine Beschreibung eines Attributs enthält daher für Abläufe und Schritte die Information, wie diese jeweils gesetzt werden.

Es folgt eine Auflistung aller Attribute.

Akteur

Dieses Attribut wird automatisch anhand der Wahl des Akteurs ausgewählt. Ob ein Akteur menschlich ist oder nicht, wird einmalig für jeden Akteur festgelegt.

Werte:

- Menschlich
- Nicht menschlich

Der Wert „Menschlich" schließt folgende Patterns aus: Keine

Der Wert „Nicht menschlich" schließt folgende Patterns aus:

- Wiederholung
- Warnung
- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format
- Filter
- Vorschau
- Assistent

Der Wert „Nicht menschlich" schließt folgende Attribute aus:

- Art der Aktion
- Art der Datenveränderung

Schritt: Der Wert ergibt sich aus dem, dem Schritt zugeordneten Akteur.

Ablauf: Es gilt der Wert „Menschlich" wenn mindestens ein Schritt im Ablauf diesen Wert hat.

Art der Aktion

Werte:

- Datenverändernd
- Datenanzeigend oder navigierend
- Konfigurierend

Der Wert „Datenverändernd" schließt folgende Patterns aus:

- Filter

Der Wert „Datenverändernd" schließt folgende Attribute aus:

- Akteur: Nicht menschlich

Der Wert „Datenanzeigend oder navigierend" schließt folgende Patterns aus:

- Globales Undo
- Objektbezogenes Undo
- Wiederholung
- Warnung
- Gute Standardwerte
- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format
- Papierkorb
- Dokumentwiederherstellung
- Automatisches Speichern
- Sicherheitskopie

Der Wert „Datenanzeigend oder navigierend" schließt folgende Attribute aus:

- Akteur: Nicht menschlich
- Art der Datenveränderung
- Reversibilität: Irreversibel

Der Wert „Konfigurierend" schließt folgende Patterns aus:

- Globales Undo
- Objektbezogenes Undo
- Wiederholung
- Warnung

- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format
- Filter
- Papierkorb
- Dokumentwiederherstellung
- Automatisches Speichern
- Sicherheitskopie

Der Wert „Konfigurierend" schließt folgende Attribute aus:

- Akteur: Nicht menschlich
- Art der Datenveränderung
- Reversibilität: Irreversibel

Schritt: Der Wert wird manuell gesetzt.

Ablauf: Der Wert wird manuell gesetzt.

Art der Datenveränderung

Werte:

- Verändert Daten im aktuellen Datensatz
- Datensatz speichern, öffnen, schließen, neu erstellen

Alle Werte schließen folgende Attribute aus:

- Akteur: Nicht menschlich
- Art der Aktion: Datenanzeigend oder navigierend

Der Wert „Verändert Daten im aktuellen Datensatz" schließt folgende Patterns aus:

- Sicherheitskopie
- Filter

Der Wert „Datensatz speichern, öffnen, schließen, neu erstellen" schließt folgende Patterns aus:

- Globales Undo
- Objektbezogenes Undo
- Wiederholung

- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format
- Papierkorb
- Dokumentwiederherstellung
- Automatisches Speichern

Der Wert „Datensatz speichern, öffnen, schließen, neu erstellen“ hebt folgende Patterns hervor:

- Sicherheitskopie

Schritt: Der Wert wird manuell gesetzt.

Ablauf: Der Wert wird manuell gesetzt.

Dateneingabe

Werte:

- Textuell
- Nicht textuell
- Keine

Der Wert „Textuell“ hebt folgende Patterns hervor:

- Gute Standardwerte
- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format

Der Wert „Textuell“ schließt folgende Attribute aus:

- Dauer: Lang

Der Wert „Nicht textuell“ hebt folgende Patterns hervor:

- Gute Standardwerte

Der Wert „Nicht textuell“ schließt folgende Patterns aus:

- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format

Der Wert „Nicht textuell“ schließt folgende Attribute aus:

Dauer: Lang

Der Wert „Keine“ schließt folgende Patterns aus:

- Gute Standardwerte
- Direkte Validierung
- Auto-Vervollständigung
- Nachsichtiges Format

Schritt: Der Wert wird manuell gesetzt.

Ablauf: Nicht zutreffend.

Reversibilität

Werte:

- Reversibel
- Irreversibel

Der Wert „Reversibel“ schließt folgende Patterns aus:

- Warnung

*Der Wert „Irreversibel“ schließt folgende Patterns **für den Ablauf** aus:*

- Globales Undo
- Objektbezogenes Undo
- Automatisches Speichern
- Papierkorb

*Der Wert „Irreversibel“ schließt folgende Patterns **für den Schritt** aus:*

- Globales Undo
- Objektbezogenes Undo
- Automatisches Speichern
- Papierkorb
- Dokumentwiederherstellung
- Sicherheitskopie

Der Wert „Irreversibel“ hebt folgende Patterns hervor:

- Warnung

Der Wert „Irreversibel“ schließt folgende Attribute aus:

- Art der Aktion: Datenanzeigend oder navigierend

Schritt: Der Wert wird manuell gesetzt.

Ablauf: Es gilt der Wert „Irreversibel“, wenn mindestens ein Schritt im Ablauf diesen Wert hat.

Modalität

Werte:

- Modal
- Nicht modal

Der Wert "Modal" hebt folgende Patterns hervor:

- Abbruch

Der Wert "Nicht modal" schließt folgende Patterns aus:

- Abbruch

Schritt: Der Wert wird manuell gesetzt.

Ablauf: Es gilt der Wert „Modal“, wenn mindestens ein Schritt im Ablauf diesen Wert hat.

Dauer

Werte:

- Kurz
- Lang

Der Wert „Kurz“ schließt folgende Patterns aus:

- Fortschrittsanzeige
- Verarbeitungsanzeige
- Ausführung im Hintergrund

Der Wert „Lang“ hebt folgende Patterns hervor:

- Fortschrittsanzeige
- Verarbeitungsanzeige
- Ausführung im Hintergrund

Der Wert „Lang“ schließt folgende Attribute aus:

- Dateneingabe: Textuell
- Dateneingabe: Nicht textuell

Schritt: Der Wert wird manuell gesetzt.

Ablauf: Nicht zutreffend.

Zahl der Eingabeschritte

Werte:

- Wenige
- Viele

Der Wert "Wenige" schließt folgende Patterns aus:

- Assistent

Der Wert "Viele" hebt folgende Patterns hervor:

- Assistent

Schritt: Nicht zutreffend.

Ablauf: Es gilt der Wert „Viele“ wenn der Ablauf mehr als zwei Schritte mit dem Attribut „Dateneingabe: Textuell“ oder „Dateneingabe: Nicht textuell“ hat.

3.3.3. Instanz

Das Modell wird auf jeden Interaktionsschritt in einem Projekt separat angewendet. Einem solchen Interaktionsschritt wird eine beliebige Teilmenge von Attributen mit jeweils einem Wert zugewiesen. Jedes Attribut darf nur einmal gesetzt werden. Abbildung 5 zeigt mit Hilfe eines ER-Diagramms, wie eine Instanz des Modells in ein Projekt eingebunden wird. Attribut und Wert der Attributierung sind über eine eindeutige ID mit dem Modell verknüpft.

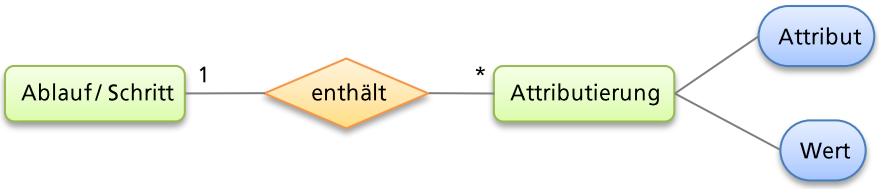

Abbildung 5: ER-Diagramm der Instanziierung des Modells

Empfehlung

Spätestens bei der Verwendung einer Instanz dieses Modells ist es notwendig Empfehlungen aufgrund von Attributen miteinander zu verrechnen. Nachdem die Skala und der Wertebereich,

sowie die Bedeutung dazu, bereits festgelegt sind, möchte ich nun die mathematische Verrechnung der Empfehlungen erläutern.

Je Interaktionsschritt wird jedem Pattern aus dem Katalog für den Empfehlungswert der Initialwert 1 zugewiesen. Jedes verwendete Attribut multipliziert den aktuellen Faktor zu jedem Pattern mit einem definierten Wert, entsprechend seiner Empfehlung für das jeweilige Pattern. Für die Empfehlungen kann eine m x n Matrix verwendet werden, wobei m alle Patterns enthält und n alle zulässigen Attribut-Wert-Kombinationen. Die Werte der Matrix stellen die jeweilige Empfehlung dar. Bei der Anwendung des Modells auf einen Interaktionsschritt wird eine n x 1 Matrix erstellt. Mit den Werten 0 und 1 wird ausgedrückt, ob eine Attribut-Wert-Kombination verwendet wird (1) oder nicht (0). Die beiden Matrizen werden nun miteinander verrechnet.

Es gilt:

- Die Empfehlungsmatrix E ist eine m x n Matrix. e_{ij} ist der Wert der Zeile i und der Spalte j.
- Die Verwendungsmatrix V_a ist eine 1 x n Matrix. v_i ist der Wert der Spalte i. a ist der Interaktionsschritt, für den die Verwendungsmatrix gilt.
- Die Ergebnismatrix R_a ist eine m x 1 Matrix. r_i ist der Wert der Zeile i.

Für jede Zeile i aus R_a ergibt sich folgender Wert:

$$r_i = \prod_{1 \le j \le n : v_j = 1} e_{ij}$$

Oder veranschaulicht: Je Zeile in der Ergebnismatrix wird das Produkt über eine Zeile der Empfehlungsmatrix gebildet, wobei aber nur die Spalten in das Produkt einfließen, die in der Verwendungsmatrix den Wert 1 haben.

Die Beispielmatrix E_1 (analog zu Abbildung 4) enthält Zeilen für folgende Patterns: Auto-Vervollständigung, Gute Standardwerte und Warnung. Und sie enthält die Spalten für die Attribut-Werte-Kombinationen jeweils Dateneingabe mit Textuell, Nicht-textuell und keine. In diesem Beispiel wird das Attribut Dateneingabe mit dem Wert Nicht-Textuell verwendet (V_1). Daraus resultiert das Ergebnis R_1:

$$E_1 = \begin{pmatrix} 2 & 0 & 0 \\ 2 & 2 & 0 \\ 1 & 1 & 1 \end{pmatrix}; V_1 = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}; R_1 = \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}$$

Das Ergebnis R_1 sagt nun aus, dass das Pattern „Auto-Vervollständigung“ nicht verwendet werden sollte; das Pattern „Gute Standardwerte“ hingegen schon.

In der Umsetzung wird die globale Matrix E nicht als solche abgespeichert werden können, da sowohl der Patternkatalog, als auch die Attributauswahl flexibel geändert werden können. Um die Flexibilität zu erhalten, gibt es ein Mapping zwischen Patternkatalog und Attributen. Dieses Mapping enthält eine Liste aller Attribute und kann beliebig erweitert werden. Jeder Eintrag eines Attributs enthält die Empfehlungen für alle Patterns die für das Attribut relevant sind. Alle Patterns mit neutraler Bewertung sowie alle im Mapping fehlenden Patterns werden zur Laufzeit um die neutrale Empfehlung 1 ergänzt. Das Mapping funktioniert daher auch noch wenn der Patternkatalog erweitert, das Mapping aber nicht aktualisiert wird.

Beispiel

Zur Veranschaulichung des Modells und seiner Instanziierung folgt ein fiktives Beispiel eines Use Cases, welcher attributiert und anschließend mit Annotationen versehen werden soll.

Das Projekt enthält folgende Auswahl aus dem Usability Pattern Katalog:

- Wiederholung
- Direkte Validierung
- Gute Standardwerte
- Warnung
- Abbruch
- Globales Undo

Folgender Use Case ist definiert:

Projekt speichern

Ziel	Das Projekt soll im Dateisystem gespeichert werden.
Beschreibung	Mit Hilfe eines Auswahldialoges wählt der Benutzer, in welcher Datei das Projekt gespeichert werden soll.
Akteure	*Benutzer*
Normalablauf	
Vorbedingung	Es ist ein nicht leeres Projekt offen.
1 Benutzer	wählt im Menü den Befehl "Projekt speichern".
2 System	zeigt einen Dialog zur Auswahl der Datei, in der das Projekt gespeichert werden soll.
3 Benutzer	wählt den Pfad über die Anzeige des Dateisystem-Baumes oder per Texteingabe und gibt den gewünschten Dateinamen in einem Textfeld ein.
4 Benutzer	bestätigt die Aktion.
5 System	speichert das Projekt in der ausgewählten Datei und schließt den

	Dialog.
Nachbedingung	Das Projekt ist in der ausgewählten Datei gespeichert.

Ich wende nun das semantische Interaktionsmodell auf diesen Use Case an und weise Attribute zu. Für Empfehlungen und Abraten von Patterns werden folgende Symbole verwendet:

- Empfehlung eines Patterns im Projekt.

  Neutrale Anzeige eines Patterns (kein Symbol)

- Abraten eines Patterns im Projekt.

- Empfehlung eines Patterns, welches nicht dem Projekt zugewiesen ist.

Schritt 1:

- Akteur: Menschlich (automatisch gesetzt)

Die Attributierung hat keine Auswirkung.

Es wird kein Pattern verwendet.

Schritt 2:

- Akteur: Nicht menschlich (automatisch gesetzt)
- Modalität: Modal

Die Patternauswahl ändert sich folgendermaßen:

Gute Standardwerte

- Direkte Validierung

- Warnung

Die Auswahl wurde damit von drei auf ein Pattern reduziert.

Es wird kein Pattern verwendet.

Schritt 3:

- Akteur: Menschlich (automatisch gesetzt)
- Dateneingabe: Textuell

Die Patternauswahl ändert sich folgendermaßen:

- Direkte Validierung

- 👍 Gute Standardwerte

  Warnung

- 👍 Auto-Vervollständigung

- 👍 Nachsichtiges Format

Die Auswahl wurde nicht reduziert, aber mit zwei Vorschlägen verbessert.

Es wird das vorgeschlagene Pattern „Gute Standardwerte verwendet“.

Schritt 4:

- Akteur: Menschlich (automatisch gesetzt)

Die Attributierung hat keine Auswirkung.

Es wird kein Pattern verwendet.

Schritt 5:

- Akteur: Nicht menschlich (automatisch gesetzt)

Die Patternauswahl ändert sich folgendermaßen:

  Gute Standardwerte

- 👎 Direkte Validierung

- 👎 Warnung

Die Auswahl wurde damit von drei auf ein Pattern reduziert.

Es wird kein Pattern verwendet.

Normalablauf:

- Akteur: Menschlich (automatisch gesetzt)
- Art der Aktion: Datenverändernd
- Art der Datenveränderung: Datensatz speichern, öffnen, schließen, neu erstellen
- Modalität: Modal (automatisch gesetzt)
- Reversibilität: Reversibel (automatisch gesetzt)
- Zahl der Eingabeschritte: Wenige (automatisch gesetzt)

Die Patternauswahl ändert sich folgendermaßen:

- 👍 Abbruch

- Globales Undo
- Wiederholung

Die Auswahl wurde von drei auf eins reduziert und dieses eine Pattern wird vorgeschlagen.

Es wird das vorgeschlagene Pattern „Abbruch“ verwendet.

Das Ergebnis sieht folgendermaßen aus (Die Attributierung gehört nicht zur fertigen Use Case Spezifikation und wird daher nicht angezeigt; Annotationen der Patterns sind mit gekennzeichnet):

Projekt speichern

Ziel	Das Projekt soll im Dateisystem gespeichert werden.
Beschreibung	Mit Hilfe eines Auswahldialoges wählt der Benutzer, in welcher Datei das Projekt gespeichert werden soll.
Akteure	*Benutzer*
Normalablauf	
Vorbedingung	Es ist ein nicht leeres Projekt offen.
1 Benutzer	wählt im Menü den Befehl "Projekt speichern".
2 System	zeigt einen Dialog zur Auswahl der Datei, in der das Projekt gespeichert werden soll.
3 Benutzer	wählt den Pfad über die Anzeige des Dateisystem-Baumes oder per Texteingabe und gibt den gewünschten Dateinamen in einem Textfeld ein.
	Standardwerte
4 Benutzer	bestätigt die Aktion.
5 System	speichert das Projekt in der ausgewählten Datei und schließt den Dialog.
Nachbedingung	Das Projekt ist in der ausgewählten Datei gespeichert.
Abbruch	

3.4. Implementierung

Das semantische Interaktionsmodell wurde so, wie es hier vorgestellt wurde in Tulip integriert. Während alle vorherigen Versionen von Tulip von mir – mit Fehlerbehebungen und anderen neuen Funktionen – die Hauptversionsnummer 1 tragen, definiert Tulip mit der Umsetzung des

semantischen Interaktionsmodells die Hauptversionsnummer 2. Die beiden Versionen unterscheiden sich ansonsten nicht.

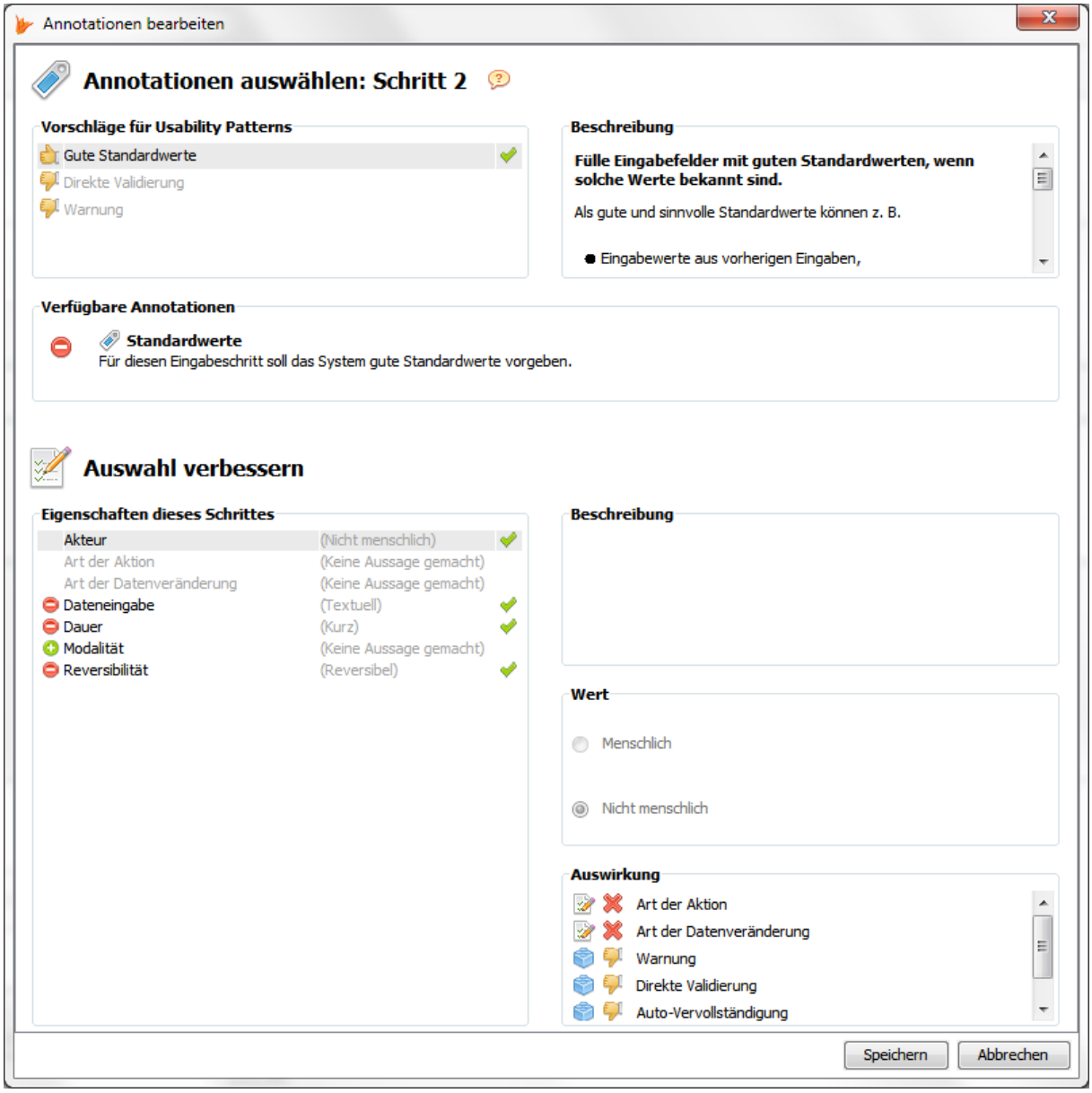

Abbildung 6: Umsetzung des semantischen Interaktionsmodells

In Abbildung 6 ist die für das Modell relevante Ansicht zu sehen. Es handelt sich um den Dialog „Annotationen bearbeiten“, welcher den Dialog „Annotation hinzufügen“ aus Version 1 von Tulip ersetzt und für jeden Schritt und jeden Ablauf in einem Use Case aufrufbar ist.

„Vorschläge für Usability Patterns“ listet alle syntaktisch passenden Usability Patterns für Schritt 2 auf, also alle Patterns im Projekt, welche auf Schritte anwendbare Annotationen enthalten. Mit dem Daumensymbol wird die Empfehlung für ein Pattern angezeigt. Patterns mit dem Empfehlungswert 0 werden zudem ausgegraut, können aber trotzdem verwendet werden.

Das Element „Eigenschaften dieses Schrittes“ im Abschnitt „Auswahl verbessern“ listet alle Attribute auf. Rechts daneben kann der Wert ausgewählt werden. Das Attribut „Akteur“ wird automatisch gesetzt und kann daher an dieser Stelle nicht bearbeitet werden. Zur Verbesserung der Transparenz, werden zudem die Auswirkungen des aktuell gesetzten Wertes angezeigt.

3.4.1. Metamodell

Das Metamodell wurde, gemeinsam mit dem Modell und einigen Hilfsklassen, in ein eigenes Eclipse-Projekt namens „AttributeModel“ ausgelagert. Um das Metamodell zu spezifizieren wurde XSD[4] verwendet. Es folgt ein Ausschnitt, welcher die Definition des Attributs zeigt.

```
<xsd:complexType name="attribute">
	<xsd:sequence>
		<xsd:element name="name" type="xsd:string" />
		<xsd:element name="description" type="xsd:string" />
		<xsd:element name="values">
			<xsd:complexType>
				<xsd:sequence>
					<xsd:element maxOccurs="unbounded" name="value"
							type="value" />
				</xsd:sequence>
			</xsd:complexType>
		</xsd:element>
	</xsd:sequence>
	<xsd:attribute name="id" type="xsd:string" />
	<xsd:attribute name="editable" type="xsd:string" default="sequence, step" />
	<xsd:attribute name="scope" type="xsd:string" default="sequence, step" />
</xsd:complexType>
```

Mit dem Attribut „ID“ ist eine eindeutige Zuordnung möglich. Mit „editable“ kann festgelegt werden, ob der Wert manuell oder automatisch gesetzt wird. „scope“ beschränkt den Kontext ggf. auf Schritte oder Abläufe.

Als Kindelemente enthält ein Attribut den Namen, einen Beschreibungstext, sowie die möglichen Werte, welche über einen eigenen Datentypen verfügen.

Mit Hilfe eines Skriptes wird aus dem XSD-Dokument Java-Code generiert, sodass eine äquivalente Klassenstruktur in Java zur Verfügung steht.

3.4.2. Modell

Das Modell befindet sich ebenfalls im Projekt „AttributeModel“ und besteht aus folgenden Komponenten:

- Die Definition der Attribute und Werte mit deren Empfehlungen.

[4] XML Schema Definitions (W3C, 2010)

- Die Prüfung der Abhängigkeiten zwischen den Attributen.
- Die Berechnung des Empfehlungswertes für ein Pattern.
- Automatisches Setzen von Attributen die das zulassen (wie z. B. „Akteur“ und „Zahl der Schritte“).

Die Darstellung in der Oberfläche wird vollständig von Tulip selbst übernommen und ist daher auch dort im Quellcode integiert.

Die Attribute und Werte sind in einem XML-Dokument definiert, welches den Regeln des XSD-Dokuments mit dem Metamodell entspricht. Aus dem XML-Dokument wird zur Laufzeit eine Java-Objektstruktur erstellt. Alle anderen Komponenten sind in Java implementiert.

Es folgt ein Ausschnitt des XML-Dokuments, welcher die Definition des Attributs „Akteur“ zeigt:

```
<attribute id="actor" editable="false">
	<name>Akteur</name>
	<values>
		<value id="human" name="Menschlich" default="true" />
		<value id="not human" name="Nicht menschlich">
			<suggestion>
				<pattern id="wiederholung" suggestion="0" />
				<pattern id="warnung" suggestion="0" />
				<pattern id="direktevalidierung" suggestion="0" />
				<pattern id="autovervollstaendigung" suggestion="0" />
				<pattern id="nachsichtigesformat" suggestion="0" />
				<pattern id="filter" suggestion="0" />
				<pattern id="vorschau" suggestion="0" />
				<pattern id="assistent" suggestion="0" />
			</suggestion>
			<exclusion>
				<attribute id="action type" />
				<attribute id="data manipulation" />
			</exclusion>
		</value>
	</values>
</attribute>
```

Zu sehen ist, dass dieses Attribut nicht manuell gesetzt werden kann. Es sind die beiden Werte „Menschlich“ und „Nicht menschlich“ definiert, sowie einige Ausschlüsse von Usability Patterns und Attributen.

3.4.3. Instanz

Bei der Instanziierung des Modells muss unterschieden werden zwischen der Instanz zur Laufzeit von Tulip und dessen Persistierung im gespeichert Projekt. In beiden Fällen müssen lediglich Attribut-Wert-Paare mit den jeweiligen Schritten und Abläufen verknüpft werden. Alle weiteren Informationen können aus dem Modell hergeleitet werden.

Zur Laufzeit werden die Attribut-Wert-Paare in den Objekten der Schritte und Abläufe als Attribut (im Sinne der Objektorientierten Programmierung) gespeichert:

```
private HashMap<Attribute, Value> attributes;
```

Im gespeicherten Projekt, welches eine XML-Datei darstellt (hier wird ohne XSD gearbeitet), werden die Attribut-Wert-Paare entsprechend der Instanz des Modells bei den jeweiligen Schritten und Abläufen gespeichert. Folgender Abschnitt aus einem Tulip-Projekt zeigt die Attributierung aus Abbildung 6:

```
<step>
	<stepNumber>2</stepNumber>
	<actorId />
	<description>fügt ein neues UC-Diagramm in den Baum ein.</description>
	<references />
	<attributes>
		<attribute id="input" valueId="textual" />
		<attribute id="actor" valueId="not human" />
		<attribute id="duration" valueId="short" />
		<attribute id="reversibility" valueId="reversible" />
	</attributes>
</step>
```

Die „actorId“ ohne weitere Eigenschaften bezeichnet das System als Akteur, welches immer als nicht menschlich definiert ist.

3.5. Auswertung

Ziel des semantischen Interaktionsmodells ist eine Erleichterung bei der Zuweisung von Usability Patterns – bzw. deren Annotationen – zu Interaktionsschritten. Mit Hilfe der Attribute ist es möglich, das notwendige Fachwissen über die Patterns zu reduzieren. Man muss lediglich seine Use Cases und die Attribute kennen. Die Attribute sind in der Zahl weniger und leichter zu verstehen als die Patterns. Zusätzlich mit dem integrierten Fachwissen im Modell wird die Zuweisung der Patterns erheblich erleichtert. Vor allem mit steigender Patternzahl im Katalog wird das Modell zunehmend nützlicher, da die Zahl der Attribute gleich bleibt.

Selbst wenn der Nutzen für kleine Projekte nicht signifikant erscheint, durch weitere Unterstützung auch bei der Attributierung kann der Nutzen weiter erhöht werden. Das wird im nächsten Kapitel näher besprochen.

Ich werte nun noch die Anforderungen an das Modell aus:

Umfang: Das Modell enthält momentan sechs manuell zu setzende Attribute und ist damit überschaubar im Umfang.

Abstraktionsgrad: Der Abstraktionsgrad ist für die Verknüpfung zwischen Usability Pattern und Beschreibung von Interaktionsschritten geeignet.

Vollständigkeit: Es gibt keinen Anspruch auf Vollständigkeit.

Erweiterbarkeit: Das Modell ist leicht erweiterbar und kann an neue Anforderungen angepasst werden.

Nutzungskontext: Das Modell berücksichtigt, wie vorher festgelegt, Abläufe und Schritte.

4. Automatische Klassifizierung von Use Cases

Das semantische Interaktionsmodell erleichtert zwar die Arbeit mit den Usability Patterns, bringt aber Projekten mit wenig Patterns keinen großen Nutzen. Benutzer, die den Pattern-Katalog bereits sehr gut kennen, haben von diesem Modell ebenfalls nur einen geringen bis keinen Nutzen.

Der nächste Schritt besteht daher darin, ein Verfahren zu entwickeln, um die Klassifizierung weiter zu erleichtern oder sogar zu automatisieren. In diesem Kapitel werden verschiedene Ansätze vorgestellt und verglichen. Einer dieser Ansätze wird ausgewählt und prototypisch umgesetzt um sein Funktionieren zu bestätigen oder zu widerlegen.

4.1. Entwicklung des Modells

Es gibt viele Möglichkeiten um an die Aufgabenstellung heran zu gehen. Nachdem ich die Rahmenbedingungen definiert habe, werde ich einige dieser Möglichkeiten vorstellen.

Erwähnt werden sollte, dass die Lösungen sich an Tulip orientieren, da sie letztendlich in Tulip umgesetzt werden und die Voraussetzungen dafür vorhanden sein müssen. Das wirkt sich vor allem auf die Datengrundlage aus.

4.1.1. Anforderungen

Ziel

Die gewählte Lösung muss den Aufwand der Zuordnung von Usability Patterns zu Interaktionsschritten weiter reduzieren, als es das semantische Interaktionsmodell alleine kann.

Aufwand für den Benutzer

Im Idealfall werden Patterns für Interaktionsschritte vorgeschlagen, ohne dass der Benutzer etwas dafür tun muss. Ebenfalls gut ist, wenn auf Wunsch eine automatische Attributierung oder Patternzuweisung vorgenommen wird. Es reicht aber bereits, wenn der Aufwand geringer ist, als mit dem semantischen Interaktionsmodell alleine oder ohne Hilfestellung.

Justierbarkeit

Die fertige Lösung muss Möglichkeiten der Justierung bieten. Da die Lösung nur Vorschläge machen kann, die mit hoher Wahrscheinlichkeit, aber niemals sicher sinnvoll sind, gilt es diese Wahrscheinlichkeit zu maximieren. Es muss daher Parameter geben, die diese Wahrscheinlichkeit beeinflussen.

4.1.2. Datengrundlage

Folgende Quellen stehen als Datengrundlage für eine mögliche Lösung zur Verfügung:

Interaktionsschritt

Ein Ablauf enthält folgende Informationen zur Verwertung:

- Vorbedingung
- Nachbedingung
- Zahl der Schritte

Ein Schritt enthält folgende Informationen zur Verwertung:

- Position im Ablauf
- Akteur
- Beschreibung

Kontext

Folgende Informationen können den Kontext von einem Interaktionsschritt definieren:

- Use Case, der den Interaktionsschritt enthält
- Umgebende Interaktionsschritte im selben Use Case
- Umgebende Use Cases

Ein Use Case enthält folgende Informationen zur Verwertung:

- Name
- Ziel
- Beschreibung
- Hauptakteur
- Priorität
- Ebene

4.1.3. Datenauswertung

Es gibt folgende Möglichkeiten die vorhandenen Daten auszuwerten:

Expertenwissen

Die Daten können aus Sicht eines Experten ausgewertet werden, um eine fundierte Aussage über deren Inhalt machen zu können.

Erfahrung

Eine Lösung kann sich Informationen aus vergangenen Klassifizierungen und Pattern-Zuweisungen zu Nutze machen und damit zukünftige Ergebnisse vorhersagen.

4.1.4. Nutzungskontext

Für die Automatisierung gibt es verschiedene Kontexte, welche sehr unterschiedliche Ergebnisse liefern können. Ich werde sie daher aufzählen und Unterschiede aufzeigen.

Interaktionsschritt → Usability Patterns

Es erfolgt eine automatische Klassifizierung des Interaktionsschrittes mit Hilfe seiner textuellen Beschreibung und nach Bedarf seines Kontextes. Anhand dieser Klassifizierung werden Usability Patterns vorgeschlagen, oder wahlweise direkt ausgewählt.

Beruht diese Klassifizierung auf Erfahrungswerten von vergangenen Pattern-Zuweisungen, dann verwendet sie ausschließlich das Verhalten des Benutzers, indem sie sein Verhalten analysiert und für zukünftige Pattern-Zuweisungen vorhersagt. Dabei wird kein Expertenwissen – wie es im semantischen Interaktionsmodell enthalten ist – verwendet. Das semantische Interaktionsmodell wird damit verworfen.

Alternativ kann der Interaktionsschritt aus Expertensicht analysiert werden, ohne vergangene Benutzeraktionen zu berücksichtigen. So kann man z. B. definieren, dass die Wörter bzw. Wortfolgen „Eingabe“, „gibt ein“ und „tippt“ in einem Beschreibungstext für eine Eingabe steht und daher u. a. das Pattern „Direkte Validierung“ verwendet werden sollte.

Interaktionsschritt → Attribute

Im Unterschied zur vorherigen Möglichkeit, werden hierbei keine Usability Patterns vorgeschlagen oder ausgewählt, sondern Attribute des semantischen Interaktionsmodells.

Beruht diese Klassifizierung auf Erfahrungswerten von vergangenen Pattern-Zuweisungen, dann wird zwar kein Expertenwissen für die Analyse der Beschreibung verwendet, aber durch die Verknüpfung mit dem semantischen Interaktionsmodell, wird trotzdem ein Pattern-Vorschlag aus Expertensicht gemacht. So wird das Verhalten des Benutzers mit dem Expertenwissen kombiniert.

Bei dieser Methode die Beschreibung aus Expertensicht zu analysieren, ist nicht sinnvoll, da so in zwei Schritten (Beschreibungsanalyse und Attributanalyse) jeweils dasselbe Expertenwissen integriert ist.

Attribute → Usability Patterns

Unabhängig vom semantischen Interaktionsmodell können Erfahrungswerte von Benutzeraktionen verwendet werden. Führt eine bestimmte Kombination von Attributen immer zur Verwendung eines bestimmten Patterns, dann kann dieser Erfahrungswert verwendet werden um dieses Pattern in Zukunft zu empfehlen oder direkt zuzuweisen, wenn diese Kombination von Attributen gewählt wurde.

Die Auswertung der Attribute aus Expertensicht ist bereits im semantischen Interaktionsmodell enthalten und wird hier daher nicht erneut behandelt.

4.1.5. Kombination mit dem semantischen Interaktionsmodell

Alle Lösungen, welche Usability Patterns unabhängig vom semantischen Interaktionsmodell vorschlagen, haben das gleiche Ziel wie das Interaktionsmodell und resultieren in einem zweiten unabhängigen Ergebnis für Vorschläge von Usability Patterns. Es muss daher bestimmt werden, wie diese beiden Ergebnisse in Relation zu einander stehen.

Es gibt dafür mehrere Möglichkeiten:

- Erfahrungswerte werden nach einer Lernphase verwendet, um die Expertenvorschläge aus dem semantischen Interaktionsmodell zu ersetzen. Man geht dabei davon aus, dass der Benutzer nach der Lernphase selbst Experte genug ist, sodass Vorschläge aufgrund seines Verhaltens nicht nur die Qualität der Expertenvorschläge erreicht, sondern zusätzlich auf seine Bedürfnisse zugeschnitten sind.
- Erfahrungswerte werden mit den Expertenvorschlägen verrechnet. Es wird nach wie vor das Expertenwissen verwendet, erweitert um eine zusätzliche personalisierte Justierung.
- Erfahrungswerte werden verwendet um das semantische Interaktionsmodell zu justieren. Bisher enthält das Modell das Expertenwissen einer einzigen Person. Es kann

nun um das Wissen von beliebig vielen Personen erweitert werden, indem Justierungen z. B. online ausgetauscht werden. Im Unterschied zum vorherigen Punkt bleibt das semantische Interaktionsmodell selbst *nicht* unberührt. Es gibt zudem nach wie vor nur eine Quelle für Pattern-Vorschläge, nämlich das semantische Interaktionsmodell.

4.1.6. Mögliche Lösungen

Es werden nun einige Lösungsvorschläge gemacht, welche sich teilweise auch sehr gut ergänzen können.

Erweiterung des semantischen Interaktionsmodells

Mit den Attributen „Akteur“ und „Zahl der Eingabeschritte“ kennt das Interaktionsmodell bereits zwei Attribute, welche nicht im Interaktionsschritt selbst gesetzt werden, sondern aus dem Kontext ermittelt werden können. So wird ein Akteur z. B. einmalig als menschlich oder nicht menschlich deklariert um diese Informationen für alle Schritte und Abläufe zu verwenden. Das Attribut „Zahl der Eingabeschritte“ kann für einen Ablauf aus den Schritten mit dem gesetzten Attribut „Dateneingabe“ ermittelt werden.

Diese Lösung kann die Zuweisung der bisher manuell zu setzenden Attribute nicht automatisieren, sondern zielt darauf ab weitere Attribute zu definieren, dessen Wert aus dem vorhandenen Kontext ermittelt werden kann.

Vorteile

- Neue Attribute können in die Bereits vorhandene Infrastruktur integriert werden. Lediglich der Teil zum automatischen Setzen muss für jedes Attribut dieser Art implementiert werden. Der Aufwand der Umsetzung ist daher sehr gering.

Nachteile

- Die bisherigen Attribute müssen weiter von Hand gesetzt werden.

Fazit

Durch die bereits vorhandene Infrastruktur und den geringen Aufwand der Umsetzung, kann die Erweiterung des semantischen Interaktionsmodells in beliebigem Umfang zusätzlich zu anderen Lösungen umgesetzt werden. Als alleinige Lösung müsste es möglichst umfassend umgesetzt werden, damit die fehlende Automatische Zuweisung der bisherigen Attribute kompensiert werden kann.

Einfache Textanalyse

Für jedes Attribut und jeden Attributwert können Signalwörter (oder Wortfolgen) definiert werden, welche eine automatisierte Zuweisung des Attributs erlauben. So sind z. B. die Wortfolgen „gibt ein“ und „tippt“ ein starkes Indiz für das Attribut „Dateneingabe: Textuell“. Es muss also ein Katalog von Wortfolgen, verknüpft mit jeweils einem Attribut und einem Wert des Attributs, erstellt werden. Zusätzlich kann eine Gewichtung definiert werden, welche angibt, mit welcher Wahrscheinlichkeit das Attribut tatsächlich anwendbar ist.

Vorteile

- Diese Lösung knüpft direkt an das semantische Interaktionsmodell an und automatisiert die dadurch notwendig gewordene Attributierung. Der Benutzer bekommt damit also kontextabhängige Vorschläge für Patterns, ohne etwas dafür tun zu müssen.
- Dieser Algorithmus ist sehr einfach zu verstehen und auch sehr einfach umzusetzen.
- Es ist sehr einfach den Katalog im Nachhinein zu ergänzen.
- Es besteht die Möglichkeit, neben der Implementierung des Algorithmus in Tulip, ebenfalls einen Editor zu implementieren, um den Benutzer selbst den Katalog anpassen zu lassen. Wird der Katalog in einem simplen Format in einer Textdatei gespeichert, dann ist die Bearbeitung mit einem Texteditor aber ebenfalls ausreichend einfach.

Nachteile

- Die Erstellung des Katalogs ist sehr aufwändig und die Qualität ist abhängig vom Wortschatz und dem Schreibstil des Erstellers.
- Der Katalog muss dem Stand der Technik immer wieder angeglichen werden. Modische Begriffe wie z. B. „wischen“ aus dem Bereich der Smartphones und Tablets sorgen für eine Verschlechterung des Katalogs, wenn dieser nicht um die neuen Begrifflichkeiten ergänzt wird.
- Stellt eine Wortfolge kein sicheres Indiz für ein Attribut dar, sondern besteht lediglich eine Wahrscheinlichkeit zwischen 0 und 100%, dann ergibt sich das Problem der Datengrundlage. Wie hoch ist die Wahrscheinlichkeit, dass es sich um eine Aktion mit dem Attribut „Dauer: Lang“ handelt, wenn im Text das Wort „wartet“ steht? Woher bekommt man diese Prozentzahl? Möchte man das statistisch ermitteln, dann steigt der Aufwand für diesen Algorithmus stark an.
- Diese Lösung ist Sprachabhängig und kann Sprachen nur soweit unterstützen wie sie von den Erstellern integriert wurden.

Fazit

Zwar ist die Umsetzung dieser Lösung relativ einfach, aber die Erstellung des Katalogs ist sehr aufwändig. Ein großer Nachteil ist die Tatsache, dass der Katalog kontinuierlich gepflegt werden muss. Diese Lösung scheint daher nicht optimal.

Semantische Textanalyse

Durch die semantische Analyse und Abstrahierung von Beschreibungstexten ist es möglich, einen Gegenpart zum bisherigen semantischen Interaktionsmodell zu erstellen und vollautomatisiert eine Verbindung zwischen den Texten und Attributen herstellen zu können. Mit Hilfe von Ontologien (Buitelaar, et al., 2005) kann die Abstrahierung verbessert werden, da auch den direkten Treffern ähnliche Begriffe verwertet werden können.

Vorteile

- Diese Lösung knüpft, wie die einfache Textanalyse, direkt an das semantische Interaktionsmodell an und automatisiert die dadurch notwendig gewordene Attributierung. Die Ergebnisse sind dabei vermutlich deutlich besser als bei der einfachen Textanalyse.

Nachteile

- Es muss ein Abstrahierungsmodell der natürlichen Sprache erstellt werden, welches mit dem Interaktionsmodell verknüpft werden kann. Dieses wird auf Grund der Vielfalt der natürlichen Sprache deutlich umfangreicher sein als das Interaktionsmodell.
- Die Erstellung einer Ontologie kommt hinzu und erhöht somit den Aufwand weiter.
- Diese Lösung ist Sprachabhängig und kann Sprachen nur soweit unterstützen wie sie von den Erstellern integriert wurden.
- Mit Veränderung der natürlichen Sprache, sowie der Technologischen Entwicklungen, muss auch das Modell angepasst werden.

Fazit

Obwohl diese Lösung direkt an die bisherigen Ergebnisse der Diplomarbeit anknüpft und somit konzeptionell die ideale Lösung wäre, so scheint der Aufwand für die Umsetzung dem Nutzen nicht ganz gerecht zu werden.

Analyse des Benutzerverhaltens

Annahme: Die Kombination bestimmter Attribute führt häufig zur gleichen Wahl von Patterns. Man speichert daher für jedes Attribut, mit welcher Wahrscheinlichkeit dieses Attribut in der Vergangenheit zur Wahl eines bestimmten Patterns geführt hat. Anhand dieser Datenbasis kann

für neue Schritte und Abläufe im Use Case nach der Attributierung ein besserer Vorschlag für Patterns gemacht werden als bisher.

Vorteile

- Im besten Fall erhält der Benutzer nach einer Justierungsphase genau die Patterns vorgeschlagen, die er auch tatsächlich verwenden möchte.
- Durch die Anpassung des Algorithmus an den Benutzer werden schlechte Vorschläge mit der Zeit stark reduziert.
- Im Gegensatz zum Bayes-Theorem (siehe unten), können hierbei die Daten der Einlernphase auch sehr gut auf andere Benutzer übertragen werden, da lediglich abstrahierte Daten verwendet werden.

Nachteile

- Die Zuweisung der Attribute wird damit nicht automatisiert. Es werden lediglich die Pattern-Vorschläge verbessert.

Fazit

Die Umsetzung dieser Lösung scheint ähnlich aufwändig wie die des Bayes-Theorems, bietet aber weniger Vorteile und Möglichkeiten.

Bayes-Theorem (Carlin & Louis, 2000)

Die Beschreibung der Schritte und Abläufe dient als Eingabe für den Algorithmus. Ein Benutzer wählt zuerst selbst Attribute für einen Interaktionsschritt aus und justiert dadurch den Algorithmus (Einlernphase). Im Laufe der Zeit können mit Hilfe der gesammelten Daten automatisiert Vorschläge für Attribute gemacht werden, die dem bisherigen Verhalten des Benutzers entsprechen. Korrekturen des Benutzers (Attributwahl entgegen der Vorschläge) verbessern die Vorschläge immer weiter. Analog kann diese Lösung auch direkt für Pattern-Vorschläge umgesetzt werden. Dabei werden dann nicht Attribute, sondern direkt Patterns vorgeschlagen.

Dieses Verfahren ist Benutzerabhängig und benötigt eine Einlernphase in der keine qualitativ hochwertigen Vorschläge gemacht werden können.

Zusätzlich zur Verwendung der Beschreibung des Schrittes oder des Ablaufs, kann es eine Verbesserung des Algorithmus geben, wenn der Kontext ebenfalls mit einbezogen wird.

Vorteile

- Einmal implementiert, benötigt dieser Algorithmus keine regelmäßige Pflege.
- Durch häufiges Verwenden des Algorithmus werden die Ergebnisse immer besser.

- Es handelt sich um ein rein statistisches Verfahren. Die Verwendung von bestimmten Wörtern in den Texten wird automatisch mit den Patterns in Verbindung gebracht.
- Der Benutzer kann den Algorithmus beeinflussen. Durch korrigieren der Ergebnisse verbessert der Benutzer zukünftige Ergebnisse selbst.
- Der Algorithmus kann mit allen drei Möglichkeiten aus „4.1.4 Nutzungskontext“ umgesetzt werden.
- Der Algorithmus arbeitet in seinem Kontext vollständig. Es gibt kein Ergebnis welches vergessen werden kann. Die Herausforderung liegt also lediglich in der Definition des Kontextes, also die Bestimmung der Eingabedaten.
- Der Algorithmus ist nicht abhängig von der verwendeten natürlichen Sprache.

Nachteile

- In der Einlernphase liefert der Algorithmus keine guten Ergebnisse.
- Gesammelte Daten können nur bedingt von anderen Benutzern verwendet werden. Die Justierung basiert auf dem Wortschatz und dem Schreibstil einer bestimmten Person. Es kann sein, dass ein bereits Justierter Algorithmus für einen anderen Benutzer schlechtere oder sogar verwirrende Ergebnisse liefert. Andererseits kann ein von mehreren Personen justierter Algorithmus evtl. sogar bessere und allgemeingültige Ergebnisse liefern.

Fazit

Das Bayes-Theorem sieht nach einer sehr vielversprechenden Grundlage für eine automatische Zuordnung von Patterns bzw. Attributen aus. Die Justierung benötigt zwar einige Durchläufe, anschließend liefert der Algorithmus aber voraussichtlich sehr gute Ergebnisse. Dass der Algorithmus grundsätzlich funktioniert, zeigt der erfolgreiche Einsatz bei Spam-Filtern. Diese Lösung beinhaltet zudem die Textanalyse und die Analyse des Benutzerverhaltens und stellt daher die umfassendste aller vorgestellten Lösungen dar, ohne dabei den Aufwand der Umsetzung zu stark anzuheben.

4.2. Bayes-Theorem

Da das Bayes-Theorem am meisten Möglichkeiten bei vertretbarem Aufwand verspricht, entschied ich mich dieses umzusetzen, es prototypisch in Tulip zu integrieren und anschließend auszuwerten.

Bei Spam-Filtern wird diese mathematische Grundlage ebenfalls verwendet. Für eine E-Mail entscheidet der Benutzer zunächst, ob diese Spam ist oder nicht. Handelt es sich um eine Spam-

Mail, dann werden alle Wörter in dieser Mail als Positivfund gespeichert, andernfalls als Negativfund. Die Funde werden je Wort aufaddiert. Kommt das Wort „Viagra“ in zehn Spam-Mails vor und in drei Nicht-Spam-Mails, dann wird dieses Wort entsprechend mit zehn Positivfunden und drei Negativfunden abgespeichert. Nach einer gewissen Einlernphase kann dann ohne Benutzeraktion anhand der gespeicherten Daten entschieden werden, ob es sich bei einer Mail um Spam handelt oder nicht. Dazu werden alle Wörter einer Mail in der vorhandenen Datenbank gesucht und ihre Wahrscheinlichkeitswerte für das Indiz einer Spam-Mail multipliziert. Automatisch kategorisierte Mails können ebenfalls verwendet werden, um den Algorithmus weiter zu justieren. Wird eine Mail falsch kategorisiert, dann kann der Benutzer dies von Hand korrigieren und damit die Daten für den Algorithmus verbessern. Analog kann dieses Verfahren auf Usability Patterns oder Attribute des semantischen Interaktionsmodells übertragen werden. Der Benutzer wählt für einen Interaktionsschritt, ob ein bestimmtes Pattern oder Attribut verwendet wird. Der Bayes-Algorithmus wird anschließend mit dieser Information und dem Beschreibungstext des Interaktionsschrittes befüllt. Nach einigen Schritten des Einlernens kann, anhand der vorhandenen Daten, mit diesem Algorithmus entschieden werden, ob jenes Pattern oder Attribut verwendet werden sollte oder nicht. Für jedes Pattern bzw. für jedes Attribut muss eine unabhängige Datenbasis abgespeichert werden, da das Ergebnis des Algorithmus nur als „ja“ oder „nein“ interpretiert werden kann.

4.2.1. Mathematische Grundlage

Sei $P(A)$ die Wahrscheinlichkeit für ein Ereignis A und $P(A|B)$ die Wahrscheinlichkeit für ein Ereignis A unter der Bedingung, dass B eingetreten ist (bedingte Wahrscheinlichkeit). Für das gleichzeitige Eintreten zweier Ereignisse gilt nach der Wahrscheinlichkeitstheorie: $P(A \cap B) = P(A) \cdot P(B|A) = P(B) \cdot P(A|B)$. Daraus folgt:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{P(A \cap B)}{P(A)} \cdot P(A)}{P(B)} = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Wird nun diese Formel auf das semantische Interaktionsmodell angewendet, dann ergibt sich folgende Bedeutung: $P(A|B)$ ist die Wahrscheinlichkeit, dass eine Attribut-Wert-Kombination A zu einer Beschreibung eines Interaktionsschrittes mit dem enthaltenden Wort B passt. Es berechnen sich:

$$P(A) = \frac{Zahl\ der\ Interaktionsschritte\ mit\ \text{Attribut} - \text{Wert} - \text{Kombination}\ A}{Zahl\ der\ Interaktionsschritte}$$

$$P(B) = \frac{Zahl\ der\ Interaktionsschritte\ mit\ dem\ Wort\ B}{Zahl\ der\ Interaktionsschritte}$$

$$P(B|A) = \frac{Zahl\ der\ Interaktionsschritte\ mit\ Attribut - Wert - Kombination\ A\ und\ Wort\ B}{Zahl\ der\ Interaktionsschritte\ mit\ Attribut - Wert - Kombination\ A}$$

Da P eine Wahrscheinlichkeit als Ergebnis hat, handelt es sich hierbei um eine reelle Zahl zwischen 0 und 1. 0,5 bedeutet, dass keine Aussage getroffen werden kann und stellt daher die neutrale Aussage dar.

Die Wahrscheinlichkeit dafür, dass ein Interaktionsschritt in eine bestimmte Kategorie passt, wird im Folgenden auch als Vorhersage oder Empfehlung des Algorithmus bezeichnet.

4.3. Implementierung

Wie bereits erwähnt, wird diese Mathematische Grundlage bereits für die Erkennung von Spam-Mails verwendet. Es liegt daher nahe, zuerst nach fertigen Bibliotheken zu suchen, welche einen passenden Algorithmus bereits integriert haben. Tatsächlich gibt es eine große Auswahl an Bibliotheken, welche auf Java basieren. *Classifier4J* (Lothian, 2005) ist eine der kleineren Bibliotheken und überzeugt durch eine sehr einfache Verwendung und ausreichende Dokumentation. Andere Bibliotheken enttäuschen gerade bei der Dokumentation für den Einstieg, wodurch die Entscheidung relativ leicht war. Weitere Vorteile von *Classifier4J* sind die geringe Dateigröße, sowie die einfache Möglichkeit, den Algorithmus selbst anzupassen (was aber vermieden wurde).

Die Bibliothek bietet mit *IWordsDataSource* eine Schnittstelle zur Verwaltung der Wörter mit Ihren Wahrscheinlichkeiten und mit *BayesianClassifier* den eigentlichen Algorithmus. Die Hilfsklasse *BayesClassifier* in Tulip bildet eine zusätzliche Schicht zur Verwendung des Algorithmus innerhalb von Tulip. Wegen der vielen unabhängigen Kategorien, die Unterschieden werden, müssen ebenso viele Instanzen des Klassifizierers verwendet werden, jeweils mit einer eigenen Datenbasis:

```
private Map<Object, SimpleWordsDataSource>;
private Map<Object, BayesianClassifier>;
```

Object wird deswegen als Klasse für den Schlüssel der *Map* verwendet, damit sowohl die Verknüpfung mit den Usability Patterns, als auch mit den Attributen aus dem semantischen Interaktionsmodell möglich ist. Für eine finale Umsetzung müsste man sich für eine der beiden Möglichkeiten entscheiden, aber zu Vergleichszwecken sind hier beide sinnvoll. Ziel dieser Arbeit ist es, ein Konzept zu erarbeiten und auf seine Tauglichkeit zu prüfen. Dafür reicht eine prototypische Implementierung, welche beide Möglichkeiten der Datenverknüpfung vergleicht.

4.3.1. Persistierung der Daten

Der Algorithmus wird durch einen Benutzer mit diversen Tulip-Projekten befüllt. Um die gewonnenen Daten auch bei weiteren Starts von Tulip verwenden zu können, müssen diese, unabhängig von den Projekten, abgespeichert werden. Ich habe mich für ein sehr einfaches Textformat entschieden: Je SimpleWordsDataSource (also je Kategorie) wird eine Textdatei angelegt. Sie erhält eine eindeutige ID der Kategorie als Dateinamen. So heißt zum Beispiel die Datei für das Pattern „Abbruch“ „Pattern-ID-abbruch“. Jede Zeile in der Datei enthält maximal einen Eintrag. Ein Eintrag besteht aus dem Wort, den positiven Treffern und den negativen Treffern, jeweils durch ein Trennzeichen getrennt. Ein Beispiel für einen Eintrag ist „akteur:0,2“. Das Wort „akteur“ (Klein-/Großschreibung spielt keine Rolle) kommt in keinem Interaktionsschritt vor, welcher das Pattern Abbruch verwendet und in zwei Schritten, welche das Pattern nicht verwenden. Alle so angelegten Dateien werden gemeinsam in ein einziges ZIP-Archiv gepackt und auf der Festplatte gespeichert.

4.3.2. Integration in die Oberfläche

Natürlich muss der Algorithmus auch grafisch verwendet werden können. Einerseits muss das Einlernen kontrolliert werden können, andererseits muss es eine Ausgabe für Vorschläge anhand des Algorithmus geben. Anders als bei E-Mails, ist es in Tulip nicht trivial zu entscheiden, ob Daten neu sind oder nicht, ob der Algorithmus also mit einem bestimmten Datensatz befüllt werden soll oder nicht. Ein Datensatz ist neu, wenn er über „Use Case hinzufügen“ angelegt wurde. Was aber gilt für einen bearbeiteten Datensatz? Um diese Frage die Beurteilung des Algorithmus nicht beinflussen zu lassen, entschied ich mich dafür, das Einlernen des Algorithmus zunächst vollständig manuell zu gestalten. Dafür wurde ein Dialog in Tulip integriert, mit dem beliebig viele Tulip-Projektdateien ausgewählt werden können, welche anschließend verwendet werden um den Algorithmus einzulernen. Vor jedem Einlernen werden alle alten Daten gelöscht. Dieses Vorgehen macht die Verwendung des Algorithmus für den Alltag nicht sinnvoll, erleichtert aber dessen Auswertung und die Beurteilung für dessen generelle Tauglichkeit.

Die Ausgabe der errechneten Wahrscheinlichkeiten für die Patterns bzw. Attribut-Wert-Kombinationen erfolgt im Dialog „Annotationen bearbeiten“ an der jeweiligen Stelle, wo diese Patterns und Werte angezeigt werden.

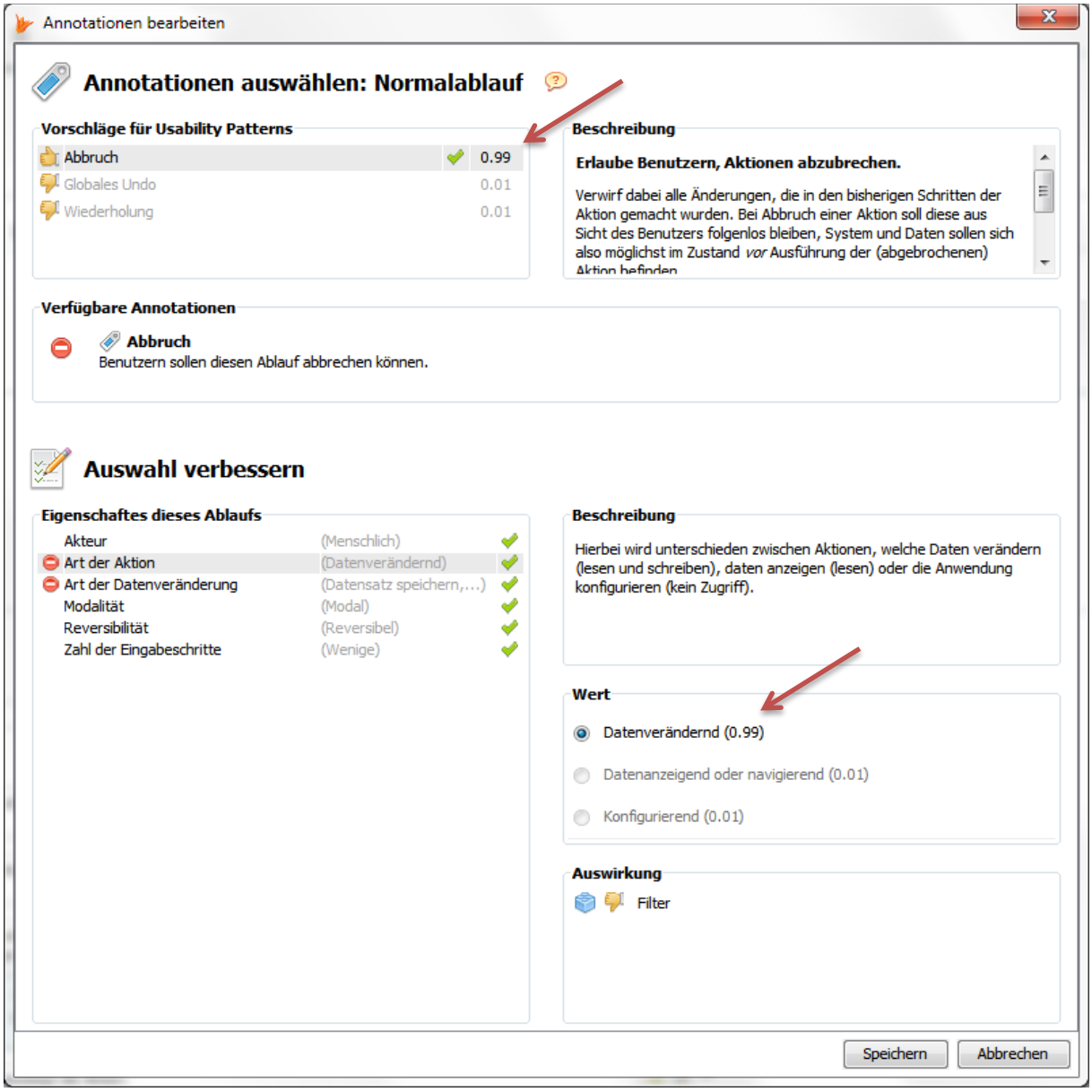

Abbildung 7: Integration des Bayes-Algorithmus in die grafische Oberfläche von Tulip

4.3.3. Justierung der Umsetzung

Schon am Anfang der Implementierung hat sich gezeigt, dass Möglichkeiten zur Justierung benötigt werden. So überwiegen z. B. die Interaktionsschritte, in denen ein Pattern oder Attribut nicht verwendet wird, so stark über die, in denen es verwendet wird, dass niemals ein Vorschlag gemacht wird.

Gewichtung

Positive Treffer werden stärker gewichtet als negative Treffer. Dafür habe ich eine Konstante mit einem Faktor in die Hilfsklasse integriert. Jeder positive Treffer wird dadurch nicht einfach,

sondern eben mehrfach registriert. Der Faktor muss so gewählt werden, dass die Trainingsdaten bestätigt werden (verwendete Usability Patterns bzw. Attribute werden vorgeschlagen). Gleichzeitig sollen negative Treffer aber nicht gänzlich ihre Auswirkung verlieren. Es gilt daher: Der Faktor soll so klein wie möglich und so groß wie nötig sein. Erst ab einem Faktor von ca. 8 für positive Treffer, werden die Trainingsdaten bestätigt.

Obergrenze für negative Treffer

Da die Ergebnisse der Wörter aus einem Text verrechnet werden, kann ein einzelnes Wort mit einem sehr hohen Wert an negativen Treffern, das Gesamtergebnis sehr stark beeinflussen. Um das zu vermeiden, wurde eine Obergrenze eingebaut.

Stoppwörter filtern

Stoppwörter sind solche Wörter, die sehr häufig vorkommen, für die Bedeutung des Textes innerhalb der aktuellen Aufgabe aber nicht relevant sind. Ich habe eine nicht zu umfangreiche Liste aus dem Internet herausgesucht, auf Plausibilität geprüft und einen Filter in die Hilfsklasse für den Algorithmus integriert. Es geht dabei nicht darum, eine vollständige Liste von nicht relevanten Wörtern zu erhalten und damit den Einfluss dieser Wörter auszuschließen, sondern lediglich deren Einfluss zu reduzieren. Dafür ist diese Liste ausreichend.

Interpretation des Ergebnisses

Das Ergebnis des Algorithmus ist eine Wahrscheinlichkeit, also eine reelle Zahl zwischen 0 und 1. Zwar wird mit der prototypischen Implementierung diese Zahl einfach als solche ausgegeben, allerdings ist es für eine finale Implementierung durchaus denkbar, die Interpretation des Ergebnisses anzupassen.

4.4. Evaluierung

Mit einer grafischen Oberfläche zur Anbindung an die Funktion des Algorithmus und der Möglichkeit der Justierung, kann der Algorithmus nun evaluiert werden.

4.4.1. Ideale Daten

Zunächst wird, mit Hilfe von selbst erstellten Testdaten, geprüft, ob der Algorithmus grundsätzlich funktioniert. Die Datenmenge wird gering gehalten, um Zusammenhänge besser erkennen zu können. Es werden bewusst Signalwörter für gewisse Kategorien von Interaktionsschritten verwendet. Die Testdaten bestehen aus dem Beispiel in „3.3.3 Instanz“ und

befinden sich ebenfalls auf der beiliegenden CD unter „Projekte/Modell Instanz Beispiel.tulip“. Diese Datei kann mit Tulip Version 2 und Version 3 verwendet werden. Da der Algorithmus sowohl mit den Usability Patterns, als auch mit den Attributen des semantischen Interaktionsmodells verknüpft ist, wird die attributierte Version des Beispiels verwendet. Das Projekt liegt mit nur diesem einen Use Case als Tulip-Datei vor und wird verwendet, um den Bayes-Algorithmus zu befüllen. Wie bereits erwähnt, werden dabei alle alten Daten des Algorithmus gelöscht. Es gibt also keine Einflüsse von früheren Tests. Anschließend muss der Algorithmus die Testdaten bestätigen, sprich, alle tatsächlichen verwendeten Patterns und Attribute werden zur Verwendung vorgeschlagen. Umgekehrt werden nicht verwendete Patterns und Attribute nicht vorgeschlagen. Durch die geringe Datenmenge gibt es sehr wenige Wörter, welche nicht eindeutig einem Pattern oder Attribut zugeordnet werden können. Daher sollten die Ergebnisse sehr klar sein. Die ausgegebenen Wahrscheinlichkeiten für die Zuordnung zu den jeweiligen Klassen sollten daher nahe 0 oder 1 liegen, nicht aber nahe der neutralen Wahrscheinlichkeit von 0,5. Abbildung 7 zeigt den Dialog „Annotationen bearbeiten“ für den Normalablauf aus dem hier verwendete Beispielprojekt. Dort ist bereits zu sehen, dass die Lerndaten durch den Algorithmus bestätigt werden. Im gesamten Use Case gibt es nur sehr wenige Abweichungen von 0,01 und 0,99 (der Algorithmus gibt übrigens keine 0 und 1 aus, da es sich immer nur um Vorhersagungen handelt und daher keine sicheren Aussagen gemacht werden können).

Fazit

Mit von mir konstruierten Daten funktioniert der Algorithmus, denn er bestätigt die Daten, mit welchen er eingelernt wurde. Er kann also im nächsten Schritt mit realen Daten getestet werden. Es wurde hier bereits eine Justierung des Algorithmus vorgenommen. Als Faktor für Positivfunde wurde 10 gewählt, als Obergrenze für Negativfunde ebenfalls 10.

4.4.2. Reale Daten

Tulip wurde im Rahmen des Software Praktikums des Studiengangs Softwaretechnik im Wintersemester 2011/2012 von 23 Teams à drei Personen eingesetzt. Dabei handelte es sich um Version 1 von Tulip. Die Attributierung, welche erst mit Version 2 eingeführt wurde, wurde daher von mir nachgeholt. Dieser Einsatz von Tulip kann als Feldversuch gesehen werden, denn die Probanden (Teams) wussten bei der Erstellung ihrer Use-Case-Spezifikationen nichts von einer späteren Auswertung der Daten. Von den 23 Use-Case-Spezifikationen stehen mir 20 zur Verfügung.

Eine Projektdatei

Zuerst teste ich mit einer Projektdatei eines Teams („Projekte/SoPra/G01.tulip“ auf der CD). Es gibt höchstens drei Autoren, welche an diesem Projekt gearbeitet haben. Vermutlich haben sie es gemeinsam getan (so sah es die Aufgabenstellung vor). Damit erhalte ich also praxisnahe Daten, welche stark abweichende Schreibstile von unterschiedlichen Autoren ausschließen. Es folgt eine Kopie (inklusive Rechtschreibfehler) des Use Cases „Projekt anlegen“ aus diesem Projekt:

Ziel	Ein neues Projekt des Scrum Prozesses in der SprintNotes Verwaltung anlegen.
Beschreibung	Der Administrator legt über die Administrationsoberfläche ein neues Scrum Projekt an.
Akteure	Administrator
Normalablauf	
Vorbedingung	Benutzer identifiziert sich als Administrator.
1 Administrator	Administrator erstellt ein neues Scrum Projekt in der Datenbank.
	Bedingung für Sonderfall Das Scrum Projekt existiert schon. — Alternativablauf 1a
	Direkte Validierung Gültige Werte: Noch nicht verwendete Projektnamen.
	Standardwerte Werte: Neuste SB- und NE-Versionsnummer
Nachbedingung	Scrum Projekt wird in der Datenbank angelegt.
Speicherung von Wiederherstellungsdaten	
Abbruch	
Alternativablauf 1a	
Vorbedingung	Das Scrum Projekt existiert schon.
1a1 Administrator	Hinweis an Administrator vom System, dass ein Projekt mit diesem Namen bereits existiert. Administrator wird aufgefordert, einen neuen Namen einzugeben oder die Projekterstellung abzubrechen.
Nachbedingung	Administrator kann einen neuen Namen eingeben und kehrt zum Normalablauf zurück. Administrator kann die Projekterstellung abbrechen und kehrt anschließend in die Administrationsebene zurück.

Zusätzlich sind folgende Attribute gesetzt:

- Normalablauf:
  - Art der Aktion: Datenverändernd
  - Art der Datenveränderung: Datensatz speichern, öffnen, schließen, neu erstellen

- Schritt 1:
  - Art der Aktion: Datenverändernd
  - Dateneingabe: Textuell

Auch diese Projektdatei verwende ich, um den Algorithmus einzulernen und überprüfe das Ergebnis anhand derselben Datei. Wie bei der selbst erstellten Projektdatei, werden die Lerndaten durch den Algorithmus weitestgehend bestätigt. In dem obigen Use Case wurden alle tatsächlich verwendeten Usability Patterns und Attribute mit dem Wert 0,99 bewertet und alle nicht verwendeten mit 0,01. In anderen Use Cases gibt es aber bereits Vorschläge, welche von den Lerndaten abweichen. Dabei handelt es sich ausschließlich um nicht gesetzte Attribute. Es fällt auf, dass diese Vorschläge nicht immer sinnvoll sind. Weiter fällt auf, dass ich die vorgeschlagenen Werte für die Attribute im Projekt sehr häufig gesetzt habe, im Gegensatz zu den anderen möglichen Werten. Die Häufigkeit der Verwendung von Attributen und Patterns spielt also maßgeblich eine Rolle bei den Vorschlägen. Dafür gibt es eine einfache und plausible Erklärung: Durch die häufige Verwendung eines Patterns oder eines Attributs gibt es in der Datenbank entsprechend viele positive Treffer, aber sehr wenige negative Treffer. Analog gibt es für Patterns und Attribute, welche in der Vergangenheit nicht verwendet wurden, keine positiven Treffer und es erfolgen keine Vorschläge.

Öffne ich eine Projektdatei eines anderen Teams ohne den Algorithmus neu einzulernen, dann erhalte ich sehr wenige Pattern-Vorschläge, aber vor allem werden zugewiesene Patterns selten bestätigt. Daraus kann gefolgert werden, dass entweder die Daten des Algorithmus wegen zu unterschiedlichem Vokabular nicht zwischen verschiedenen Autoren ausgetauscht werden können, oder dass eine Projektdatei nicht ausreicht um den Algorithmus zufriedenstellend einzulernen.

Viele Projektdateien

Um von den soeben erwähnten zwei möglichen Ursachen eine ausschließen zu können, brauche ich viele Projektdateien eines einzigen Autors. Diese Daten liegen mir nicht vor, da Tulip noch nicht sehr lange existiert und daher noch nicht von einer Person für viele verschiedene Projekte eingesetzt wurde. Ich kann als Ersteller des semantischen Interaktionsmodells und des Algorithmus zur automatisierten Kategorisierung selbst keine geeigneten Testdaten erstellen, da ich die Funktionsweise der Mechanismen zu gut kenne und von meinem Hintergrundwissen beeinflusst bin. Es bleibt nur die Möglichkeit, den Algorithmus mit allen Projektdateien verschiedener Autoren einzulernen und das Ergebnis zu beurteilen.

Zuerst lerne ich den Algorithmus mit allen 20 Use-Case-Spezifikationen ein und überprüfe, ob die verwendeten Usability Patterns und Attribute vom Algorithmus bestätigt werden und, ob sinnvolle neue Vorschläge gemacht werden. Ebenfalls interessant ist, ob der Algorithmus eine falsche Verwendung von Patterns aufzeigen kann (es sind mir bereits einige Fehlverwendungen aufgefallen). Tatsächlich werden verwendete Patterns und Attribute bestätigt. Für das obige Beispiel trifft das zu und zusätzlich erhalten alle nicht verwendeten Patterns und Attribute sehr niedrige Werte (deutlich unter 0,5). Bei ein paar anderen Use Cases werden zusätzliche Patterns vorgeschlagen. Die Vorschläge sind nicht immer passend, aber auch nicht gänzlich falsch. Eine Tendenz kann dabei aber nicht festgestellt werden.

Anschließend lerne ich den Algorithmus mit nur 19 Projekten ein und verwende das 20. Projekt, um den Algorithmus auf seine Praxistauglichkeit zu prüfen. Ich wähle das Projekt aus dem vorherigen Beispiel zur Überprüfung aus. Die meisten tatsächlich verwendeten Patterns und Attribute werden anschließend auch empfohlen. Es fällt auf, dass die Patterns „Dokumentwiederherstellung“ und „Papierkorb“ wesentlich schlechter vorhergesagt werden, als die anderen. Andere Patterns hingegen werden fast zu 100% bestätigt. Eine Prüfung der anderen Projekte erklärt dies: Gerade das Pattern „Dokumentwiederherstellung“ wird in deutlich weniger Projekten verwendet als z. B. „Direkte Validierung“. Teilweise werden die Patterns von den verschiedenen Teams auch unterschiedlich interpretiert und verwendet, was jeweils zu anderen Signalwörtern führt. Die Vermutung ist also, dass die vorhandene Datenmenge zum Einlernen für seltener verwendete Patterns nicht ausreicht. Häufig verwendete Patterns hingegen werden mit den vorhandenen Daten bereits sehr gut vorhergesagt bzw. empfohlen. Zusätzlich zu den bereits verwendeten Patterns und Attributen werden auch andere vorgeschlagen. Auch hier gilt: Die Vorschläge sind teilweise sehr gut, aber es gibt auch einige unpassende Vorschläge. Es kann sich also auf keinen Fall um eine zuverlässige Vorhersage von passenden Patterns und Attributen handeln.

Fazit

Im Rahmen einer Diplomarbeit ist es sehr schwierig an gute Daten zur Evaluierung zu kommen. Um herauszufinden, ob der verwendete Algorithmus für einen einzelnen Benutzer funktioniert, wäre eine Langzeitstudie mit einem Benutzer, welcher mehrere Projekte durchläuft, optimal. Ebenfalls interessant ist, ob der Algorithmus mit einer großen Masse an Daten von verschiedenen Autoren funktioniert. Denkbar ist eine vernetzte Datenbank, welche von hunderten Benutzern simultan verwendet wird. Gerade seltener verwendete Usability Patterns könnten so sinnvoller Empfohlen werden als mit wenigen Daten.

4.5. Auswertung

Die Evaluierung hat gezeigt, dass der Algorithmus grundsätzlich funktioniert und bereits gewählte Patterns und Attribute durch ein hohes Ergebnis bestätigen kann. Umgekehrt sind Vorschläge des Algorithmus aber nicht so gut, als dass man diese für eine vollautomatisierte Zuweisung verwenden könnte. Selbst als unterstützende Funktionalität würden die Ergebnisse den Benutzer häufig mit unsinnigen Vorschlägen verwirren. Gerade durch die falschen Vorschläge ist es nicht gegeben, dass der Benutzer ein reduziertes Wissen über die Usability Patterns haben darf. Da mit den vorhandenen Daten aber nicht alle wichtigen Fragen beantwortet konnten, möchte ich diese Lösung auch nicht als unbrauchbar deklarieren. Es ist gut möglich, dass der Bayes-Algorithmus wesentlich mehr Daten benötigt um brauchbare Resultate zu liefern. Auch konnte die Abhängigkeit vom Sprachstil des Autors nicht gezeigt oder widerlegt werden.

5. Zusammenfassung

Im Rahmen dieser Arbeit konnte ein Modell entwickelt werden, welches Interaktionsschritte soweit abstrahiert, sodass eine automatisierte Verknüpfung zwischen der textuellen Beschreibung und den Usability Patterns möglich ist. Anstatt für jeden Schritt Patterns aus dem Katalog auszuwählen, kann der Benutzer nun die Interaktionsschritte beschreiben und erhält Vorschläge für die Verwendung von ein paar wenigen Patterns. Zwar kann das Modell seine Stärken in den bisher betrachteten kleinen Projekten und mit dem aktuell sehr überschaubaren Katalog von Usability Patterns nicht ausspielen, aber spätestens mit einem deutlich umfangreicheren Pattern-Katalog und größeren Projekten dürfte der Nutzen die Kosten übersteigen.

Die Arbeit mit den Usability Patterns wurde anfangs bereits mit Informationen aus dem Kontext eines Interaktionsschrittes unterstützt. So ist mit der Wahl des Benutzers bereits bekannt, ob es sich um einen menschlichen Benutzer handelt. Zusätzlich zu den Informationen aus dem Kontext wurde eine Methode erarbeitet und evaluiert, um, mit Hilfe der Beschreibungstexte der Interaktionsschritte, dem Benutzer den Aufwand der abstrakten Beschreibung der Interaktionsschritte abzunehmen. Das Ergebnis der Evaluation ist, dass der entwickelte Algorithmus mit der vorhandenen Datenbasis keine gute Grundlage für automatisierte Vorschläge von Usability Patterns darstellt. Ob der Algorithmus, wie bei der Spam-Erkennung, deutlich mehr Daten für gute Ergebnisse benötigt, oder ob er gänzlich ungeeignet ist, kann aber nicht final entschieden werden.

Zusätzlich zur konzeptionellen Arbeit und dessen Umsetzung, wurde Tulip deutlich in seiner Bedienbarkeit verbessert. Viele Details, die den Rahmen der ersten Diplomarbeit gesprengt hätten, konnten nun beachtet werden. Außerdem wurden einige neue Funktionalitäten eingebaut.

Ausblick

Mit einem wachsenden Katalog an Usability Patterns, muss das semantische Interaktionsmodell ebenfalls aktualisiert werden. Durch heuristische Daten können die Vorschläge für Patterns noch deutlich verbessert werden. Erst wenn der Katalog unüberschaubar groß geworden ist, gibt es eine echte Notwendigkeit für die entwickelte Unterstützung und bietet sie einen echten Vorteil.

Der Algorithmus zur automatisierten Beurteilung von Beschreibungstexten kann den Erwartungen nicht gerecht werden und muss entweder für ein finales Urteil mit deutlich mehr Daten evaluiert, oder aber verworfen werden.

Literaturverzeichnis

Brodtmann, S., Brull, R., Kuhn, T. & Röder, H., 2011. *Tulip.* [Online] Verfügbar unter: http://www.iste.uni-stuttgart.de/se/werkzeuge/tulip.html [Zugriff am 1. Dezember 2011].

Buitelaar, P., Cimiano, P. & Magnini, B., 2005. *Ontology learning from text: methods, evaluation and applications.* 1. Hrsg. Amsterdam: IOS Press.

Card, S. K., Moran, T. P. & Newell, A., 1983. *The psychology of human-computer interaction.* 1. Hrsg. Hillsdale: Lawrence Erlbaum Associates.

Carlin, B. P. & Louis, T. A., 2000. *Bayes and empirical Bayes methods for data analysis.* 2. Hrsg. s.l.:Chapman & Hall.

Feigenbaum, E. A., 1977. *The art of artificial intelligence: Themes and case studies of knowledge engineering.* s.l., s.n.

Folmer, E., Gurp, J. v. & Bosch, J., 2003. *A Framework for Capturing the Relationship between Usability and Software Architecture.* s.l., John Wiley & Sons, Ltd..

Hix, D. & Hartson, H. R., 1993. *Developing User Interfaces: Ensuring Usability Through Product & Process.* 1. Hrsg. New York: Wiley.

Holzinger, A., 2005. *Usability engineering methods for software developers.* New York, ACM.

IEEE Standard 610.12, 1990. *IEEE Standard Glossary of Software Engineering Technology.* s.l.:IEEE Standards Association.

ISO 9241-11, 1998. *Anforderungen an die Gebrauchstauglichkeit - Leitsätze.* s.l.:s.n.

Juristo, N., Moreno, A. M. & Sánchez, M. I., 2004. *Clarifying the Relationship between Software Architecture and Usability.* s.l., s.n.

Juristo, N., Moreno, A. & Sánchez, M., 2004. *Clarifying the Relationship between Software Architecture and Usability.* s.l., Sixteenth International Conference on Software Engineering and Knowledge Engineering.

Lothian, N., 2005. *Classifier4J.* [Online]
Verfügbar unter: http://classifier4j.sourceforge.net
[Zugriff am 2 Januar 2012].

Ludewig, J. & Lichter, H., 2007. *Software Engineering.* 1. Hrsg. Heidelberg: dpunkt.verlag.

Röder, H., 2011. *A Pattern Approach to Specifying Usability Features in Use Cases.* Pisa, Italy, s.n.

Shneiderman, B., 1998. *Designing the user interface.* 3. Hrsg. s.l.:Addison-Wesley Longman Inc..

Tidwell, J., 2005. *Designing Interfaces.* 1. Hrsg. Sebastopol: O'Reilly.

W3C, 2010. *W3C - XML Technology - Schema.* [Online]
Verfügbar unter: http://www.w3.org/standards/xml/schema
[Zugriff am 12 Dezember 2011].

Welie, M. v., 2008. *Welie.com.* [Online]
Verfügbar unter: http://www.welie.com/patterns
[Zugriff am 6. Dezember 2011].

Zdun, U., 2006. *Systematic pattern selection using pattern language grammars and design space analysis.* [Online]
Verfügbar unter: http://onlinelibrary.wiley.com/doi/10.1002/spe.799/abstract
[Zugriff am 6 Dezember 2011].

Abbildungsverzeichnis

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, den 11. Januar 2012

_____

(Simon Brodtmann)