Usability von grafischen Editoren auf dem Prüfstand

Erfahrungsbericht über die Zusammenführung heterogener Benutzergruppen am Beispiel eines Werkzeugs für das Requirements Engineering

Katharina Juhnke

itemis AG
Niederlassung Leipzig
Dohnanyistraße 15
04103 Leipzig

katharina.juhnke@itemis.de

Abstract

Das Verbinden unterschiedlicher Gedankenwelten und die Stärkung der Zusammenarbeit heterogener Benutzergruppen durch EIN Werkzeug – Traum oder bereits Realität? Dieser Beitrag zeigt am Beispiel eines bestehenden textuellen Editors und einem dazu korrespondierend entwickelten grafischen Editor, wie Teamwork zwischen grafikaffinen und textaffinen Benutzergruppen im Requirements Engineering gelingen kann. Im Rahmen der benutzerzentrierten Konzeption und der Usability Evaluation des grafischen Editors zur Erstellung und Bearbeitung formaler Anforderungsmodelle kamen verschiedene Methoden des Usability Engineerings zum Einsatz. Die dabei gesammelten Erfahrungen und Erkenntnisse werden durch diesen Beitrag vorgestellt. Die Ergebnisse der Usability Evaluation zeigen, dass grafische Editoren und der Einsatz von Layout-Algorithmen, individuell anpassbaren grafischen Repräsentationen (sog. Sichten), Modellvalidierung oder Restriktionen für die intuitive Erstellung syntaktisch korrekter Anforderungsmodelle die Arbeit für bestimmte Benutzergruppen zum Kinderspiel macht.

Keywords

Requirements Engineering, Anforderungsspezifikation, grafische Editoren, heterogene Benutzergruppen, Usability Evaluation

1. Einleitung

Damit der Einsatz von Werkzeugen für das Requirements Engineering auch einen wirklichen Nutzen erzielt, spielen Aspekte wie die einfache Bedienbarkeit, benutzerspezifische Anpassung des Werkzeugs und die mögliche Integration in den bestehenden Entwicklungsprozess eine entscheidende Rolle. Benutzern sollte ein Werkzeug in die Hand gegeben werden, welches hinsichtlich ihrer Kenntnisse und Fähigkeiten angemessen ist und mit dem sie souverän umgehen können. Werkzeughersteller spezialisieren sich in diesem Kontext zumeist auf eine Benutzergruppe und unterstützen diese durch abgestimmte Eingabemasken und Expertensichten. Wenn jedoch heterogene Benutzergruppen aufeinander treffen, stellt sich die Frage, wie diese mit dem gleichen Werkzeug produktiv zusammenarbeiten können. Muss sich eine der Benutzergruppen zwangsläufig

an die durch ein Werkzeug vorgegebene und für eine andere Benutzergruppe optimierte Arbeitsweise anpassen? Wie können Alternativen aussehen und welche Herausforderungen gilt es zu bewältigen, wenn heterogene Benutzergruppen zusammengeführt werden sollen?

Der folgende Erfahrungsbericht behandelt diese Fragestellungen und zeigt, wie eine benutzerfreundliche Lösung für die Zusammenführung heterogener Benutzergruppen am Beispiel eines Werkzeugs für das Requirements Engineering konzipiert wurde. Im Zuge dessen entstand ein grafischer Editor, welcher die grafische Modellierung formaler Anforderungsmodelle im Requirements Engineering ermöglicht und zudem als Schnittstelle zwischen verschiedenen Benutzergruppen agiert. Für einen ersten Prototypen wurde im Anschluss eine Usability Evaluation konzipiert, durchgeführt und ausgewertet.

2. Kontext

Das Erstellen einer konsistenten und verständlichen Spezifikation im Rahmen des Requirements Engineerings ist längst nicht mehr alleinige Aufgabe von Requirements Engineers. Daher besteht die besondere Herausforderung in der interdisziplinären Zusammenführung verschiedener Expertengruppen (z. B. Projektmanager, Usability Professionals, Requirements Engineers, Stakeholder, u.v.m.), die unterschiedliche Vorlieben und Kenntnisse im Umgang mit Werkzeugen besitzen.

Im Kontext dieses Beitrags wird das Werkzeug YAKINDU Requirements (www.yakindu.de) betrachtet (Geyer, 2013). Dieses Spezifikationswerkzeug für die Erfassung, Dokumentation und Verfolgung von Anforderungen steht im Kontrast zu typischen grafikbasierten Anforderungswerkzeugen, wie zum Beispiel dem Enterprise Architect (www.sparxsystems.de) oder MagicDraw (www.nomagic.com), da es einerseits einen textbasierten und andererseits einen am Konzept des „Use Case Modeling“ (Bittner & Spence, 2002) orientierten Ansatz verfolgt. Anstelle der grafischen Modellierung werden Anforderungsmodelle mit einer benutzerspezifisch anpassbaren textuellen Notation formal beschrieben und automatisiert in grafische Repräsentationen weiterverarbeitet. Zudem kann der Benutzer anhand der erfassten textuellen Anforderungsmodelle individuelle Spezifikationsdokumente für unterschiedliche Zielgruppen generieren. Abbildung 1 zeigt den grundlegenden Aufbau von YAKINDU Requirements mit dem Navigationsbereich (1), einem textuellen Editor (2) und der grafischen Repräsentation in der sogenannten Diagram Preview (3). Die grafische Repräsentation wird abhängig von der Selektion im textuellen Editor generiert und dem Benutzer angezeigt. Werden am textuellen Anforderungsmodell Änderungen vorgenommen, wird die grafische Repräsentation automatisch aktualisiert.

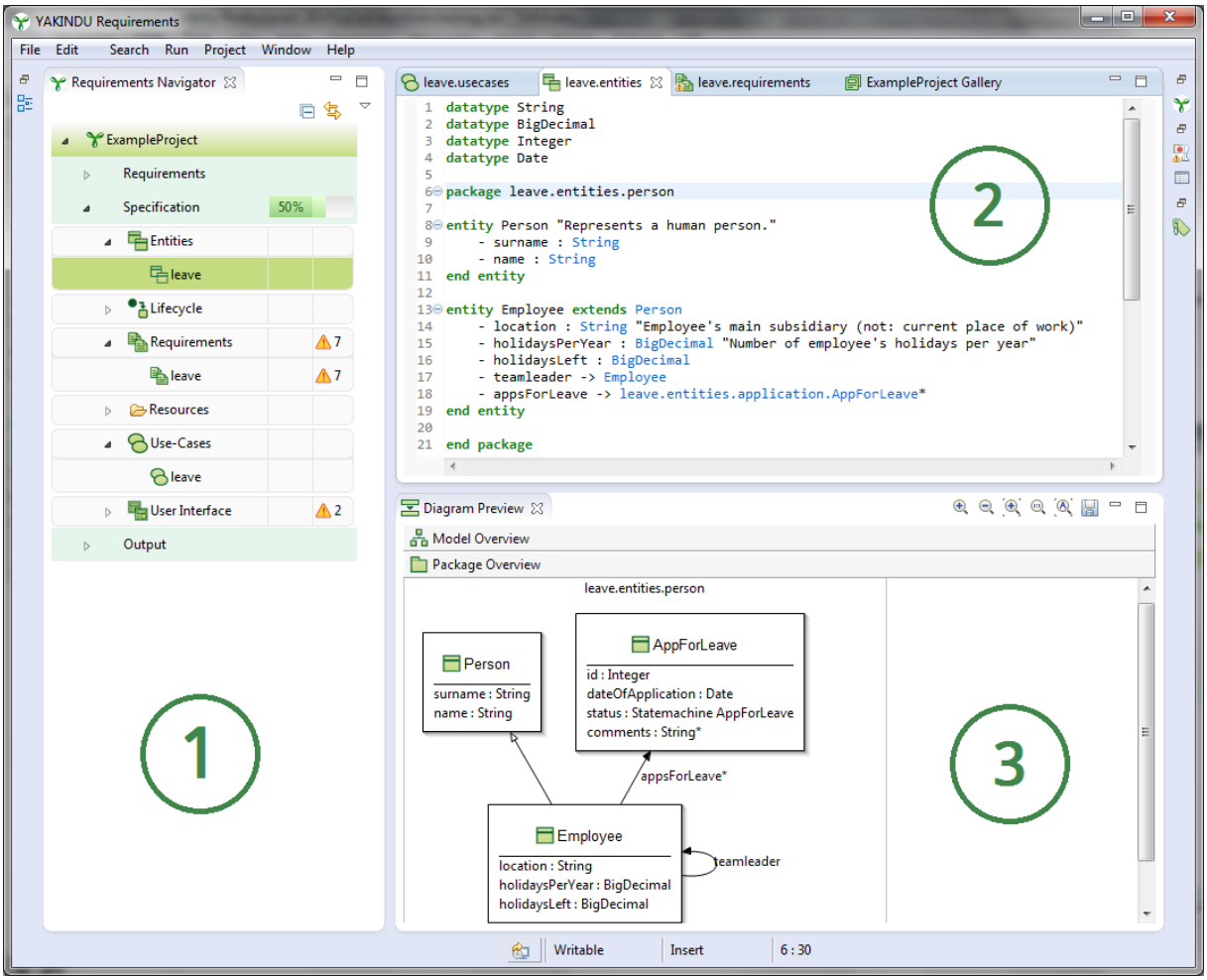

Abb. 1: Benutzeroberfläche des Werkzeugs YAKINDU Requirements

3. Zielsetzung

Aufgrund der zwei komplementären Ansätze erreicht YAKINDU Requirements insbesondere textaffine Benutzergruppen mit einem technischen Hintergrundwissen (z. B. Programmierer). Mithilfe eines mächtigen textuellen Editors werden Benutzer mittels diverser Assistenten, Modellvalidierung, Syntaxhervorhebung, Textvervollständigung oder Templates darin unterstützt, formale Anforderungsmodelle schnell und einfach zu erstellen. Die daraus generierten grafischen Repräsentationen können einzeln exportiert oder für die Generierung individueller Spezifikationsdokumente verwendet werden. In dieser Form dienen sie als Kommunikationsgrundlage für den Wissensaustausch zwischen textaffinen und grafikaffinen Benutzergruppen und tragen somit zu einem gemeinsamen Verständnis über das zu entwickelnde System bei (Balzert, 2001). Allerdings wird eine werkzeuggestützte Zusammenarbeit dieser beiden Benutzergruppen nicht unterstützt, da grafikaffine Benutzergruppen die Arbeit mit einem textbasierten Werkzeug meiden.

Das Ziel der Studie bestand darin, ein Konzept und einen Prototypen zu implementieren, welcher die Erstellung und den Umgang mit formalen Anforderungsmodellen in YAKINDU Requirements vereinfacht und benutzerfreundlicher gestaltet. Dadurch sollen die verschiedenen Benutzergruppen angesprochen und deren jeweiliger intuitiver Arbeitsprozess unterstützt werden. Grafikaffine Benutzer sollen nicht länger nur die Ergebnisse des Werkzeugs benutzen, sondern zu einer aktiven

Arbeit mit der Anwendung motiviert werden. Zudem soll eine Möglichkeit geschaffen werden die Arbeitsergebnisse effizient zusammenzuführen, sodass Vorteile aus beiden Gedankenwelten verknüpft werden.

Basierend auf der Arbeitsweise grafikaffiner Benutzer, welche in der Regel grafische UML-Modellierungswerkzeuge benutzen, stellt ein grafischer Editor den Ausgangspunkt für das zu entwickelnde Konzept dar. Um beide Benutzergruppen zu unterstützen und Arbeitsergebnisse zusammenzuführen reicht dies allein jedoch nicht aus. Daher muss eine Verbindung zwischen dem existierenden textuellen Editor und dem grafischen Editor geschaffen werden, um grafisch modellierte formale Anforderungsmodelle auf die bisherige textuelle Repräsentation abzubilden und umgekehrt.

4. Analyse

Aufgrund der gesammelten Erfahrungen aus dem Praxiseinsatz von YAKINDU Requirements in Kundenprojekten sowie im akademischen Umfeld und der Durchführung von Usability Evaluationen (u. a. mittels Usability-Tests, Thinking Aloud, Heuristische Evaluation, AttrakDiff und System Usability Scale (SUS) Fragebogen) standen zu Beginn der Studie bereits Informationen zur Verfügung. Daraus begründete sich ein anfänglicher Erweiterungsbedarf, welcher die Unterstützung verschiedener Benutzergruppen fordert.

Ergänzend dazu wurden im Rahmen eines Cognitive Walkthrough zwei grundlegende Schwachstellen im bisherigen Entwicklungsstand von YAKINDU Requirements kategorisiert, zu denen nachfolgend die identifizierten Aspekte dargestellt werden:

Schwächen der textuellen Notationen

Benutzer mit technischem Hintergrundwissen bevorzugen die textuellen Notationen. Diesen textuellen Ansatz galt es in der Analysephase aus Sicht eines grafikaffinen Benutzers zu betrachten, dessen Ziel die Erstellung einer grafischen Repräsentation eines formalen Anforderungsmodells ist, welche als Kommunikationsgrundlage im Entwicklungsprozess eines Systems dient. Eines der hauptsächlichen Hindernisse für diese Benutzergruppe besteht darin, dass die jeweiligen grafischen Repräsentationen ausschließlich mithilfe der textuellen Editoren generiert und editiert werden können. Die in Abbildung 1 dargestellte Diagram Preview ist lediglich eine Sicht und kein Editor. Zwar bieten die textuellen Editoren ein hohes Maß an Unterstützung aufgrund verschiedener Assistenten, jedoch müssen diese Mechanismen dem Benutzer bekannt sein. Ist dies nicht der Fall, stehen insbesondere grafikaffine Benutzer vor dem Problem, die Syntax und Semantik der bereitgestellten textuellen Notation zunächst verstehen zu müssen, ehe sie effektiv damit arbeiten können.

Schwächen der generierten grafischen Repräsentationen

Die zweite grundlegende Schwachstelle besteht in Bezug auf die Diagram Preview und die generierten grafischen Repräsentationen eines Anforderungsmodells. Diesbezüglich konnten verschiedene Mängel hinsichtlich der verwendeten Layout-Algorithmen und der nicht vorhandenen Interaktionsmöglichkeiten mit den grafischen Repräsentationen identifiziert werden, die unter den folgenden Punkten zusammengefasst wurden:

1. Das Layout der grafischen Repräsentationen (z. B. Position und Größe von Elementen) kann nicht manuell verändert werden.

2. Die Berechnungen der Layout-Algorithmen sind teilweise nicht optimal, sodass Elemente sich beispielsweise überschneiden (vgl. Abbildung 2).
3. Die Sichten auf ein Anforderungsmodell sind in der Diagram Preview begrenzt und nicht individuell anpassbar. Der Benutzer kann nicht auswählen, welche Elemente in einer Sicht dargestellt werden sollen.
4. Der Detaillierungsgrad der dargestellten Informationen ist nicht individuell anpassbar, wodurch immer alle Eigenschaften eines Elementes angezeigt werden.

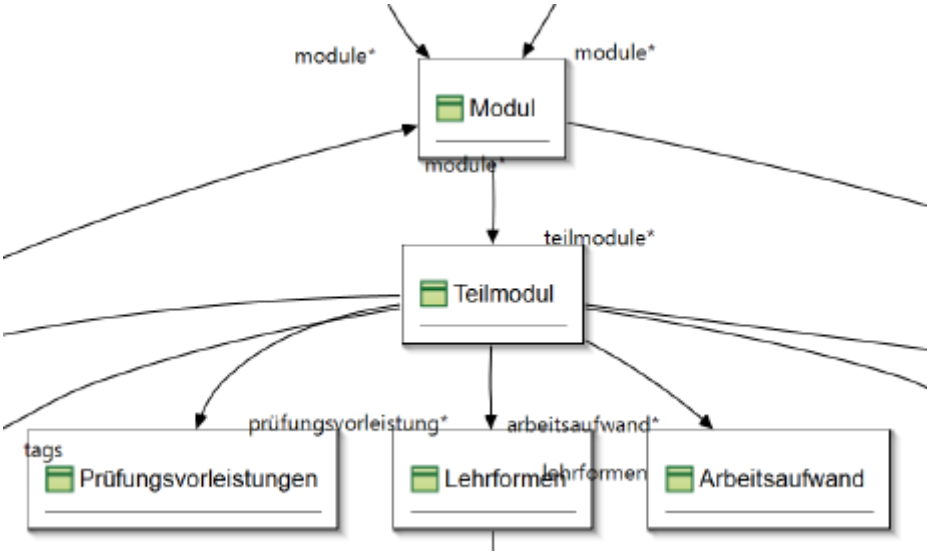

Abb. 2: Ausschnitt aus einem Anforderungsmodell mit fehlerhaftem Layout

Zusätzlich zum Cognitive Walkthrough wurde im Vorfeld der Implementierung des Prototyps eine Existenzstudie über eine Auswahl grafischer Editoren im Umfeld von UML-Modellierungswerkzeugen durchgeführt (Eichelberger, 2011). Zusätzlich zielten Kontextinterviews auf den Informationsgewinn bezüglich des Zusammenspiels von textuellem und grafischem Editor sowie auf die beiderseitige Unterstützung der verschiedenen Benutzergruppen ab. Diese Maßnahmen bildeten zusammen die Grundlage für die Erhebung von Anforderungen an den zu entwickelnden Prototypen.

5. Ergebnisse

Basierend auf den durchgeführten Untersuchungen konnten einerseits Anforderungen ermittelt werden, die für die Usability eines grafischen Editors im Kontext formaler Anforderungsmodelle und für die angestrebte Unterstützung grafikaffiner Benutzergruppen relevant sind. Andererseits resultierten Anforderungen, die sich auf Aspekte der beiderseitigen Unterstützung verschiedener Benutzergruppen und deren Zusammenführung beziehen. Zusammenfassend wurden die ermittelten Anforderungen in folgende Schwerpunkte kategorisiert:

- (E) Entwicklung eines grafischen Editors zur Erstellung und Bearbeitung formaler Anforderungsmodelle unter Berücksichtigung benutzerzentrierter Anforderungen.
- (V) Übertragung der Modellvalidierungsregeln vom textuellen Editor auf den grafischen Editor, um auf semantische Fehler eines grafisch modellierten Anforderungsmodells hinzuweisen.

- (L) Integration eines automatischen Layout-Algorithmus und Bereitstellung von Mechanismen für die manuelle Beeinflussung des Layouts eines grafischen Anforderungsmodells.
- (X) Realisierung eines Sichtenkonzepts zur individuellen Erstellung grafischer Repräsentationen (Sichten) eines formalen Anforderungsmodells und Bereitstellung von Mechanismen zur Bearbeitung des Inhalts dieser Sichten.
- (S) Synchronisation zwischen textuellem und grafischem Editor in denen ein Anforderungsmodell unterschiedlich repräsentiert wird.
- (N) Navigation zwischen Elementen des Anforderungsmodells im textuellen und grafischen Editor.

Diese Schwerpunkte muss ein integrierter grafischer Editor im Zusammenspiel mit einer textuellen Notation, wie sie in YAKINDU Requirements gegeben ist, realisieren, um textaffine und grafikaffine Benutzergruppen gleichermaßen anzusprechen und zu unterstützen. Die Schwerpunkte (E) und (V) tragen dazu bei, dass grafikaffinen Benutzern Mechanismen für die grafische Modellierung zur Verfügung gestellt werden, welche die Erstellung korrekter formaler Anforderungsmodelle unterstützen. Schwerpunkt (L) greift die Problematik auf, dass insbesondere grafikaffine Benutzer dazu neigen, sich mit Fragen bzgl. des Layouts abzulenken anstelle sich mit der eigentlichen inhaltlichen Erstellung eines Anforderungsmodells zu beschäftigen. Diesbezüglich besteht auch für die Umsetzung von Schwerpunkt (E) eine Prämisse darin, die Funktionalitäten möglichst einfach und deren Umfang überschaubar zu halten. Das Erstellen benutzerspezifischer Sichten auf Grundlage des Schwerpunktes (X) kommt sowohl textaffinen als auch grafikaffinen Benutzern zugute, da somit keine Einschränkungen mehr durch die vordefinierten grafischen Repräsentationen der Diagram Preview bestehen. Um die Erlernbarkeit der textuellen Notationen und eine Zusammenführung der Arbeitsergebnisse beider Benutzergruppen zu erreichen, muss eine Synchronisation zwischen textuellem und grafischem Editor realisiert werden. D. h., dass Modifikationen im grafischen Editor Auswirkungen auf die textuelle Repräsentation eines Anforderungsmodells haben müssen und umgekehrt. Dieses Erfordernis wird durch den Schwerpunkt (S) aufgegriffen, wobei die Navigation im Schwerpunkt (N) ebenfalls der Zuordnung zwischen grafischer und textueller Repräsentation dient.

Eine detaillierte Aufführung der für eine textuelle Notation aus YAKINDU Requirements beispielgebend erfassten Anforderungen und deren Zuordnung zu den aufgezeigten Schwerpunkten würde den Rahmen dieses Erfahrungsberichtes sprengen und ist daher unter (Juhnke, 2014) dokumentiert.

6. Prototyp

Am Beispiel des entwickelten Prototyps sollen nachfolgend die realisierten Konzepte und ein Ausschnitt der umgesetzten Anforderungen dargestellt werden. Dies erfolgt beispielgebend für die textuelle Notation für Geschäftsmodelle in YAKINDU Requirements und anhand der im vorherigen Abschnitt aufgeführten Schwerpunkte.

(E) Grafischer Editor

Der entwickelte grafische Editor ist in Abbildung 3 dargestellt. Zu sehen ist eine Zeichenfläche, welche verschiedene Elemente eines Anforderungsmodells enthält. Im Beispiel sind dies Elemente zur Modellierung von Geschäftsobjekten und deren Beziehungen (Datentypen, Packages, Entitäten, Attribute, Referenzen und Supertype-Beziehungen). Auf der rechten Seite enthält der grafische

Editor eine Palette, welche lediglich Modellelemente enthält, die zum jeweiligen Kontext des dargestellten Anforderungsmodells passen.

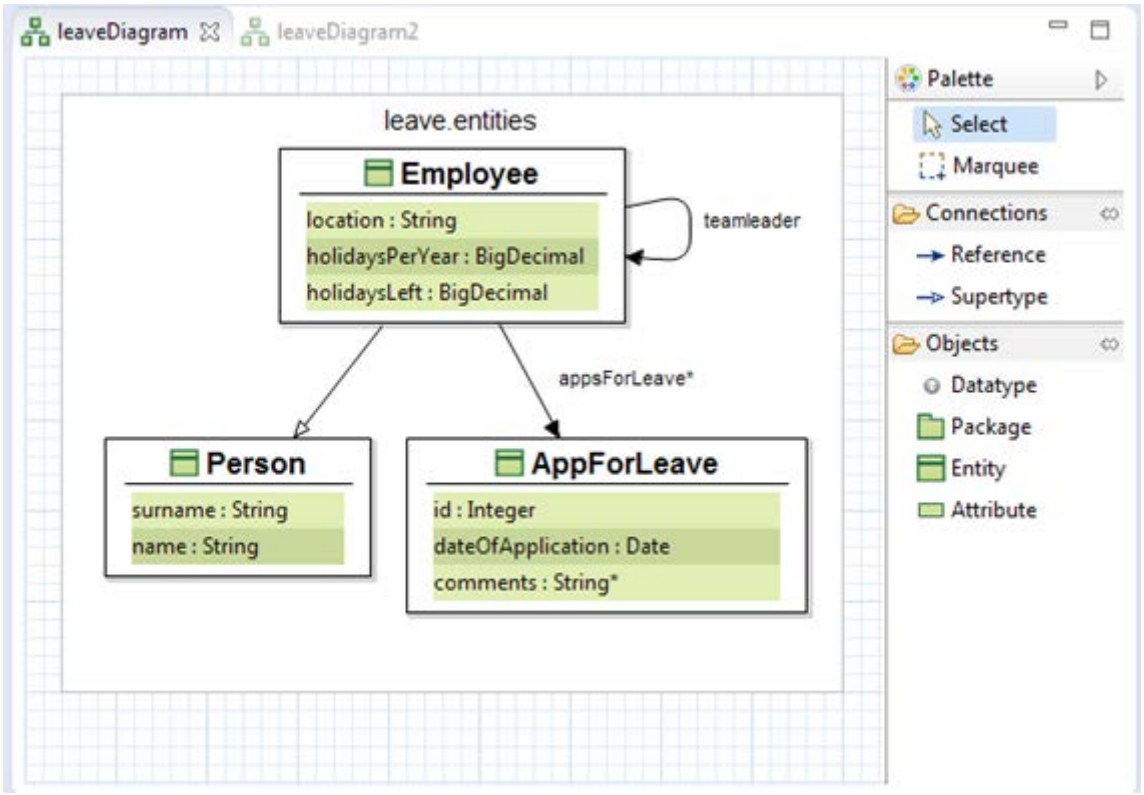

Abb. 3: Prototyp des grafischen Editors in YAKINDU Requirements

Der grafische Editor wurde mithilfe von Graphiti (www.eclipse.org/graphiti), einem Eclipse-basierten Framework entwickelt. Ziel des Frameworks ist es, die Entwicklung benutzerfreundlicher homogener grafischer Editoren zu erleichtern und zu beschleunigen. Die realisierten Benutzerinteraktionen in Graphiti entstanden in enger Zusammenarbeit mit Usability-Experten, sodass bereits mit der Verwendung der Standardimplementierungen benutzerfreundliche grafische Editoren entwickelt werden können (Brand et al., 2011).

Darüber hinaus wurden weitere benutzerzentrierte Konzepte im Kontext von YAKINDU Requirements umgesetzt, welche die grafische Erstellung und Bearbeitung von Anforderungsmodellen so beeinflussen, dass diese syntaktisch und semantisch korrekt sind. Dazu gehören die realisierten Restriktionen im grafischen Editor, die in Abhängigkeit der zugehörigen textuellen Notation ermittelt wurden. Zum Beispiel können Attribute lediglich innerhalb einer Entität angelegt werden und Entitäten müssen einem Package zugeordnet sein. Abbildung 4 zeigt, dass ein Attribut nur in eine andere Entität verschoben und nicht auf der Zeichenfläche abgelegt werden kann. Ist eine Interaktion nicht erlaubt, wird dies dem Benutzer anhand eines Verbotszeichens verdeutlicht. Die Prüfung, ob eine Interaktion erlaubt ist, erfolgt auch wenn Elemente aus der Palette auf der Zeichenfläche angeordnet und wenn Beziehungen zwischen Elementen erstellt oder modifiziert werden.

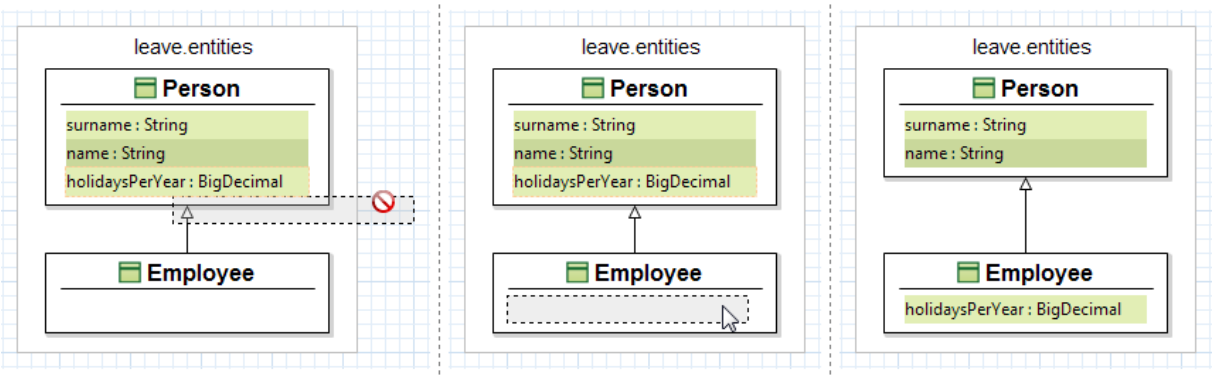

Abb. 4: Umsetzung von Restriktionen im grafischen Editor

Eine weitere Besonderheit wurde in Bezug auf das Erstellen von Beziehungen zwischen Elementen umgesetzt. Abbildung 5 zeigt, dass ausgehend von einem existierenden grafischen Element mithilfe eines Context Buttons eine Beziehung zu einem anderen Element mittels Drag & Drop Support gezogen werden kann. Entspricht der Endpunkt der gezogenen Beziehung keiner anderen Entität, sondern einem leeren Bereich innerhalb eines Packages, so kann der Benutzer an dieser Stelle eine neue Entität erzeugen. Dafür wird ihm ein Menü angezeigt, anhand dessen er die Art der zu erstellenden Beziehung auswählen kann.

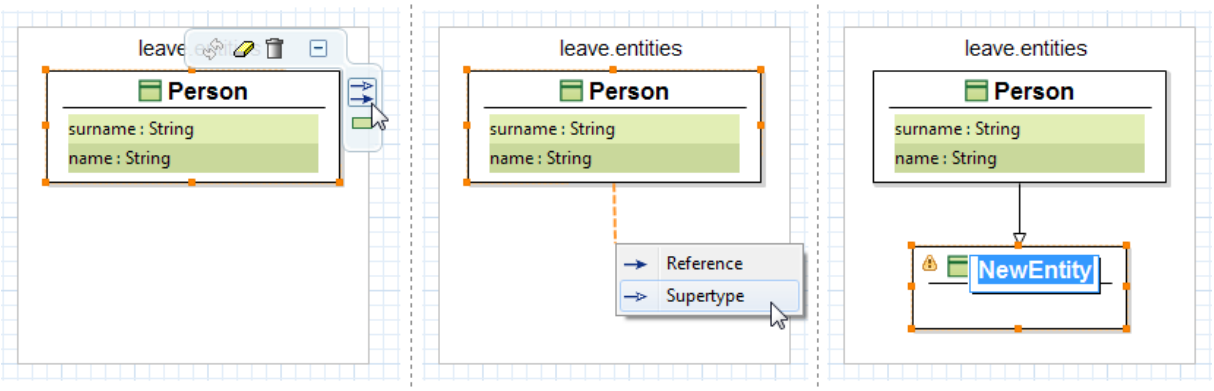

Abb. 5: Erstellen von Beziehungen mittels Drag & Drop Support

Die erzeugte Entität kann anschließend aufgrund des automatisch aktivierten Editiermodus sofort editiert werden. Hierfür stehen Eingabehilfen zur Verfügung, die aus den bestehenden textuellen Editoren übernommen wurden. Abbildung 6 zeigt den aktivierten Editiermodus für ein Attribut und den mittels Tastenkombination Strg + Leertaste aufgerufenen Content Assist. Dieser zeigt für die Bearbeitung eines Attributs alle verfügbaren Datentypen samt zusätzlichen Informationen an, die im Anforderungsmodell existieren.

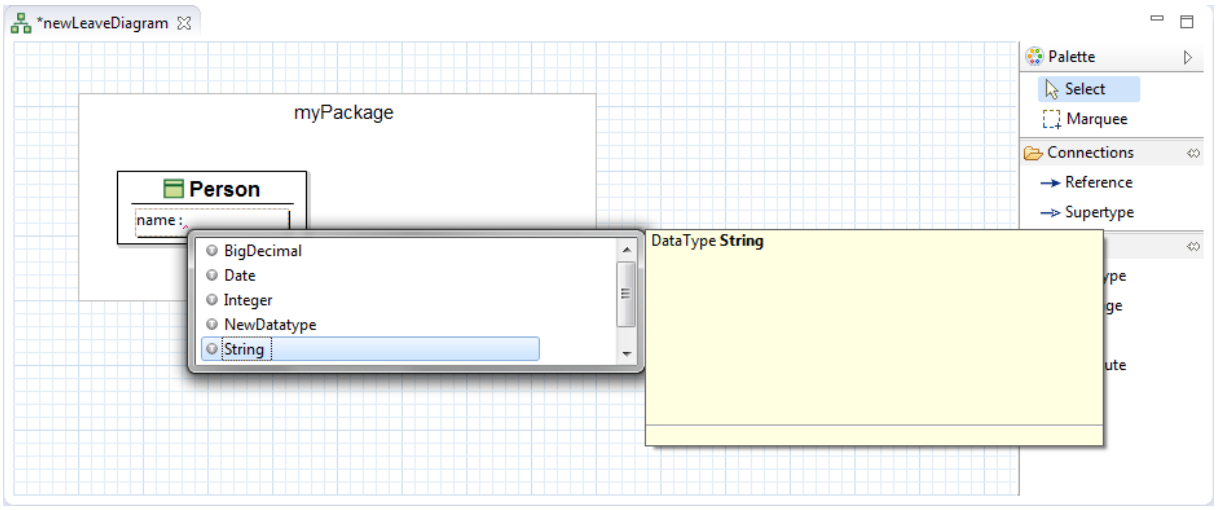

Abb. 6: Editiermodus für das Editieren eines Attributs mit dem integrierten Content Assist

Mithilfe des integrierten Content Assists werden Funktionalitäten aus dem bestehenden textuellen Editor übernommen und in den grafischen Editor integriert. Dadurch können Benutzer mit technischem Hintergrundwissen einerseits bekannte Funktionen im grafischen Editor verwenden und andererseits werden grafikaffine Benutzer darin unterstützt, korrekte Werte für verschiedene Eigenschaften zu setzen. Dabei zeigt der Content Assist lediglich für den aktuellen Kontext relevante Informationen an (z. B. mögliche Datentypen für ein Attribut). D. h., dem Benutzer wird lediglich ein Ausschnitt aus der jeweils korrespondierenden textuellen Notation angezeigt, wodurch er diese indirekt erlernt. Fehleingaben werden dem Benutzer entweder bereits während der Eingabe verdeutlicht, indem sich der Hintergrund eines Textfeldes rot einfärbt oder nach Bestätigung einer fehlerhaften Eingabe ein Fehlerdialog mit entsprechenden Hinweisen erscheint. Des Weiteren unterstützen die Tooltips den Benutzer beim Erlernen der unterschiedlichen Funktionalitäten des grafischen Editors.

Die Konzepte für die Bedienung und Modifikation der grafischen Elemente wurden an dieser Stelle bewusst einfach gehalten, sodass der Benutzer für das Editieren eines Elements nicht auf umfangreiche Dialoge angewiesen ist.

(X) Sichtenkonzept

Die Realisierung des Sichtenkonzepts beinhaltet die Erstellung und Modifikation einer Sicht, basierend auf einem bestehenden textuellen Anforderungsmodell. Das bedeutet, dass mehrere individuelle Sichten mit dem grafischen Editor erstellt werden können. Hierzu wurde ein Wizard implementiert, welcher einerseits den Zusammenhang zwischen dem textuellen Anforderungsmodell und der jeweiligen Sicht im grafischen Editor herstellt und mit dem sich andererseits der initiale Inhalt einer Sicht beeinflussen lässt. Abbildung 7 zeigt diesen Wizard, in dem der Name der Sicht, das zu verlinkende textuelle Anforderungsmodell und die enthaltenen Elemente ausgewählt werden können.

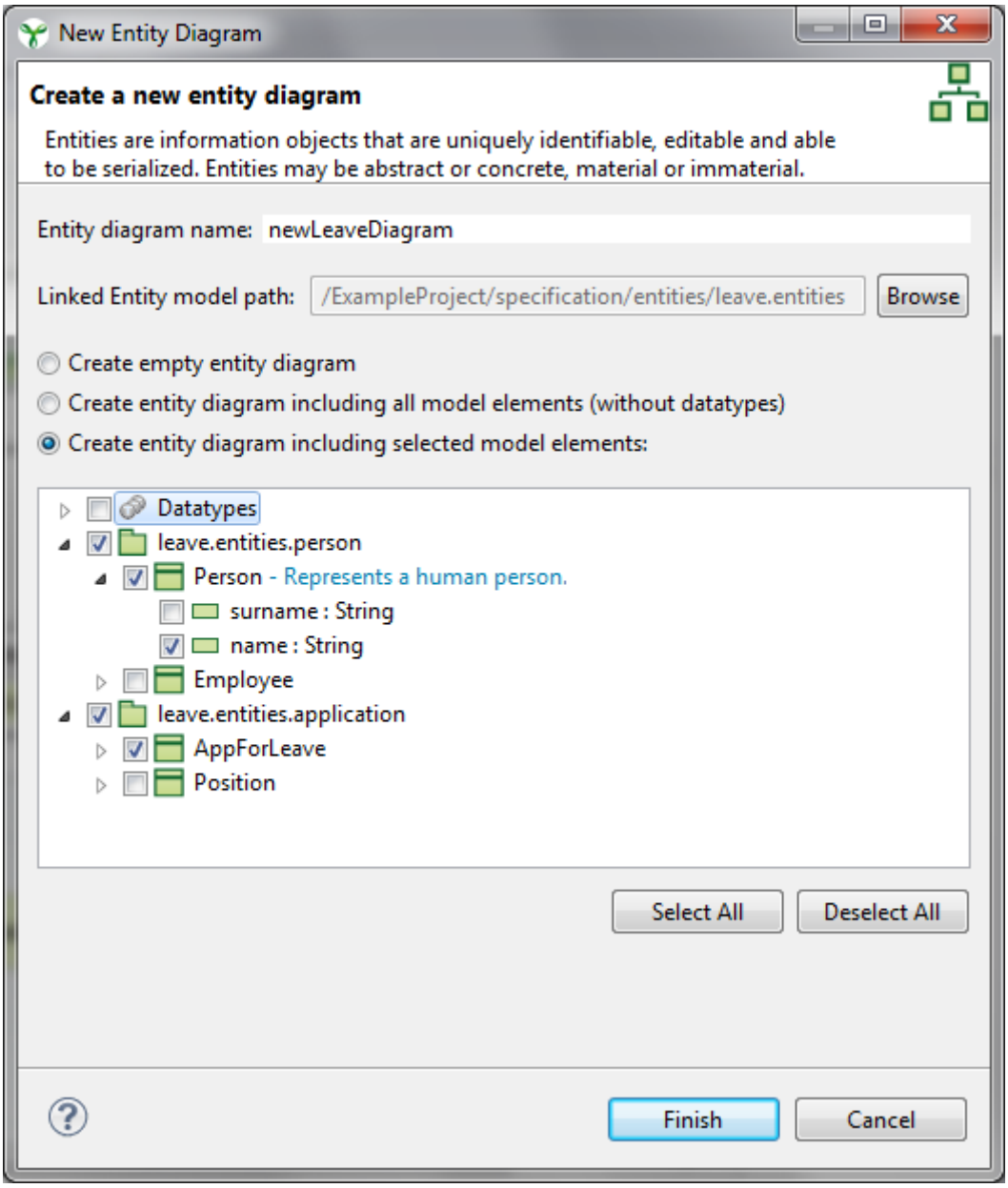

Abb. 7: Wizard zur Erstellung einer Sicht

Der Inhalt einer Sicht kann modifiziert werden, indem neue Elemente mithilfe der Palette hinzugefügt, bestehende Elemente über Context Buttons gelöscht oder direkt editiert werden. Die jeweilige Sicht ist dabei ein individuelles Abbild des korrespondierenden textuellen Anforderungsmodells, sodass nicht immer alle Elemente visualisiert sein müssen, die im textuellen Anforderungsmodell enthalten sind. Aus technischer Sicht wird dies realisiert, indem grafische Elemente jeweils auf Modellelemente aus der textuellen Repräsentation verweisen. Daher ist es möglich Elemente aus der Sicht zu entfernen, ohne sie aus dem textuellen Anforderungsmodell zu löschen (remove vs. delete). Um einer Sicht existierende Modellelemente hinzuzufügen, wurde eine Outline View implementiert, die immer alle Elemente des korrespondierenden textuellen Anforderungsmodells enthält (vgl. Abb. 8). Die in der Outline View enthaltenen Elemente sind zusätzlich markiert, sofern für diese bereits eine Repräsentation im grafischen Editor existiert. Dies wird dem Benutzer durch ein orangefarbenes Häkchen am Symbol des betreffenden Elementes angezeigt (vgl. Element ’Person’ in Abb. 8). Bereits in einer Sicht enthaltene Elemente können der

Sicht nicht mehrfach hinzugefügt werden. Dadurch wird das Erzeugen semantisch fehlerhafter Anforderungsmodelle vermieden.

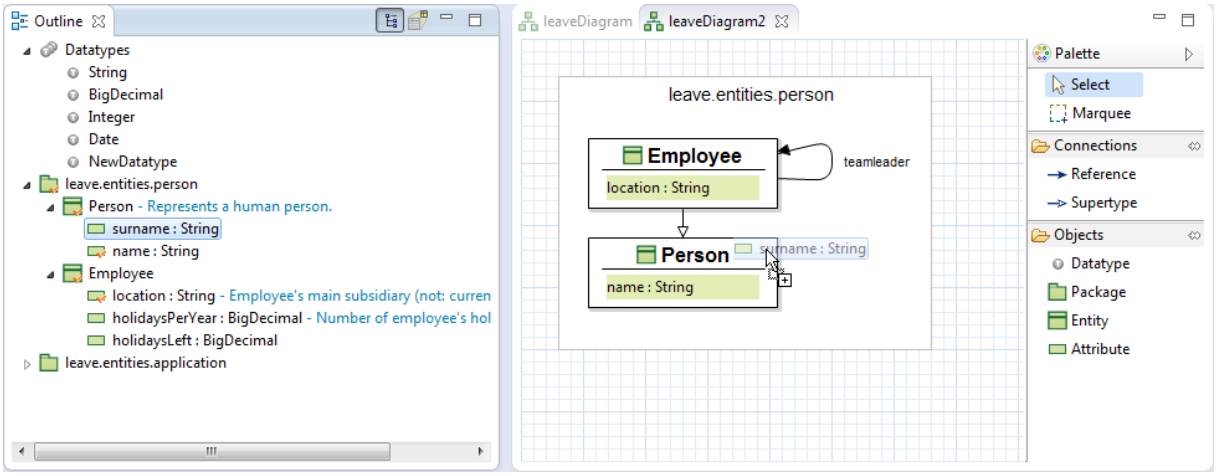

Abb. 8: Drag & Drop Funktionalität der Outline View

Weiterhin werden über das Kontextmenü des grafischen Editors weitere Funktionalitäten bereitgestellt, um den Inhalt einer Sicht zu modifizieren. Hierzu gehören das Hinzufügen aller Attribute oder aller Beziehungen zu einer ausgewählten Entität und das Hinzufügen aller Elemente zu einer Sicht. Außerdem ist es möglich den Detaillierungsgrad der dargestellten Informationen anzupassen, indem zum Beispiel Attribute einer Entität aus- bzw. eingeblendet werden können (Collapse & Expand).

(V) Modellvalidierung

Die Übertragung von Modellvalidierungsregeln vom textuellen auf den grafischen Editor ist relevant, um auf die grafische Erstellung semantisch fehlerhafter Anforderungsmodelle hinzuweisen und diese somit zu verhindern. Daher werden äquivalent zum textuellen Editor Fehler im grafischen Editor angezeigt, sobald eine Änderung zu einem fehlerhaften Anforderungsmodell führt (vgl. Abb. 9). Dabei werden sogenannte Image Decorator eingesetzt, die einen Fehler oder eine Warnung in Bezug auf ein betroffenes Modellelement markieren. Dem Benutzer werden nähere Informationen hinsichtlich der Ursache eines Fehlers anhand eines Tooltips präsentiert.

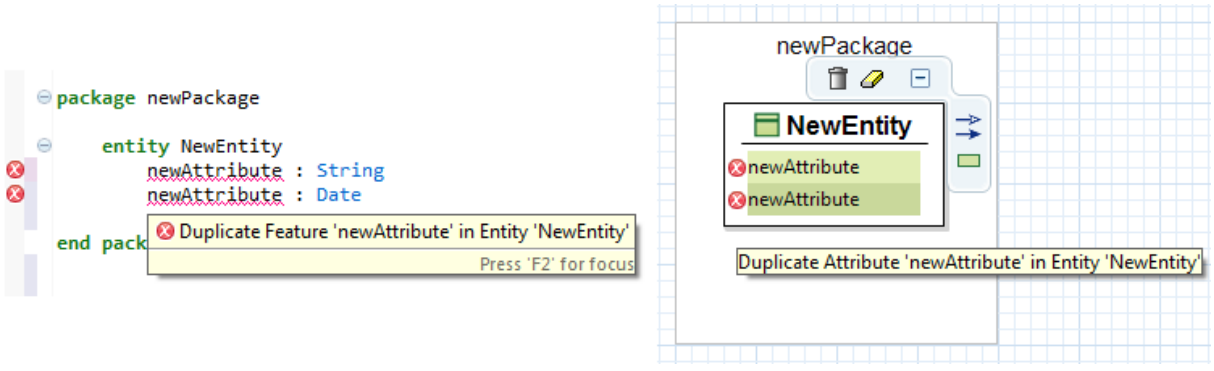

Abb. 9: Anzeige von Modellvalidierungsergebnissen im textuellen und grafischen Editor

Die Anzeige eines Fehlers ist für den Fall der Vergabe gleicher Namen und für das Erzeugen von Kreisbeziehungen in Vererbungshierarchien realisiert. Des Weiteren werden Warnungen angezeigt, wenn ein Package keine Entitäten und eine Entität keine Attribute beinhalten. In Bezug auf das Sichtenkonzept erfolgt die Anzeige von Modellvalidierungsergebnissen im grafischen Editor stets

basierend auf dem korrespondierenden textuellen Anforderungsmodell, da nicht immer alle Modellelemente auch im grafischen Editor enthalten sein müssen. Wird beispielsweise in einer Sicht eine Entität angelegt und ihr ein Name gegeben, unter welchem bereits eine andere Entität im textuellen Anforderungsmodell existiert, so wird an der grafisch neu erzeugten Entität ein Fehler angezeigt.

(L) Automatisches und manuelles Layout

Die Anbindung eines Layout-Algorithmus erfolgt aus zweierlei Gründen. Einerseits soll das automatische Layout einer Sicht vom Benutzer ausgelöst werden können, um den Aufwand für die Anordnung und Darstellung der grafischen Elemente zu verringern. Andererseits wird das automatische Layout während des Erstellungsprozesses einer Sicht verwendet, wenn initial bereits Elemente in einer Sicht enthalten sein sollen. Dafür wurde das Framework KIELER (http://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Home) eingesetzt, mithilfe dessen für eine erstellte Sicht im grafischen Editor auf Knopfdruck ein Layout berechnet wird. Weiterhin passt sich das Layout einer Entität automatisch an, wenn dieser Attribute hinzugefügt oder von dieser entfernt werden.

Weiterhin können Größe und Position von Elementen im grafischen Editor manuell vom Benutzer angepasst werden.

(S) Synchronisation

Im Rahmen der Konzeption und Entwicklung des Prototyps wurde zunächst eine Synchronisation zwischen textuellem und grafischem Editor in Echtzeit angestrebt. D. h. sobald ein Element im grafischen Editor hinzugefügt, gelöscht oder modifiziert wird, sollte diese Änderung auch ohne dem Erfordernis die Sicht zu speichern im korrespondierenden textuellen Editor angezeigt werden. Dadurch sollte dem Benutzer die Möglichkeit geboten werden die Auswirkungen seiner Interaktionen im grafischen Editor unmittelbar im textuellen Editor nachzuvollziehen und umgekehrt. Im Laufe der Arbeit stellte sich jedoch heraus, dass eine Realisierung dieses Vorhabens aufgrund der verwendeten Frameworks und deren unausgereiften Implementierungen zum Zeitpunkt der Entwicklung noch nicht möglich war. Daher wurde eine Synchronisation in Abhängigkeit des Speichermechanismus realisiert. Wird das Anforderungsmodell im textuellen Editor modifiziert und diese Änderungen gespeichert, so werden die Änderungen im grafischen Editor visualisiert und umgekehrt. Dadurch hat ein grafikaffiner Benutzer die Möglichkeit die Änderungen am textuellen Anforderungsmodell nachzuvollziehen und die jeweils davon betroffene Sicht zu aktualisieren.

(N) Navigation

Die Navigation zwischen der Outline View und dem grafischen Editor wurde innerhalb der Implementierung eines ersten Prototyps realisiert. D. h., dass selektierte Elemente in der Outline View auch im grafischen Editor selektiert werden. In einer weiteren Iterationsphase gilt es die Navigation zwischen dem textuellen und dem grafischen Editor umzusetzen, sodass die Selektion eines grafischen Modellelements die korrespondierende Zeile im textuellen Editor selektiert.

7. Usability Evaluation

Zu dem entwickelten Prototyp wurde abschließend eine Usability Evaluation durchgeführt. Das Evaluationsziel bestand dabei in der Bewertung des Prototyps zum Ende des Entwicklungsprozesses, entsprechend einer summativen Evaluation (Sarodnick & Brau, 2011). Neben der Bewertung galt es

zudem Usability-Probleme zu identifizieren, die in der Bedienung des Prototyps während einer typischen Nutzungssituation auftreten können und Empfehlungen für mögliche Verbesserungen zu erfassen.

Als grundlegende Funktionalitäten und Aspekte, die getestet werden sollten, wurden folgende Punkte betrachtet:

- Anlegen von Sichten zu einem bestehenden textuellen Anforderungsmodell.
- Erstellen und Bearbeiten von Elementen innerhalb des grafischen Editors.
- Bearbeiten von Sichten auf ein Anforderungsmodell mithilfe der verschiedenen bereitgestellten Mechanismen (Wizard, Outline View, Kontextmenü, etc.).
- Beurteilung des Synchronisationsmechanismus zwischen textuellem und grafischem Editor.

Für die Usability Evaluation des entwickelten Prototyps wurde eine Kombination aus verschiedenen empirischen Evaluationsmethoden eingesetzt, indem ein Usability-Test sowie die Fragebogen- und Interviewmethode aufeinander abgestimmt wurden.

Das Skript für den Usability-Test wurde basierend auf den in der Analysephase identifizierten Aufgaben erarbeitet. Es enthält ein Aufgabenszenario und verschiedene Aufgabenstellungen. Diese wurden durch den Testbenutzer in einer Laborstudie bearbeitet, wobei er von einem Testleiter begleitet wurde. Die Testsitzungen wurden mithilfe des Werkzeugs Morae (www.techsmith.de/morae.html) für spätere Auswertungszwecke aufgezeichnet.

Zur quantitativen Erhebung von Daten wurden zudem zwei Fragebögen eingesetzt. Es erfolgte im Anschluss an jede bearbeitete Aufgabe eine Befragung, um differenziertere Aussagen über den entwickelten Prototypen zu erhalten. Dafür wurden die in Abbildung 10 dargestellten Fragen ausgewählt, die sich an den After Scenario Questionnaire (ASQ) von James Lewis anlehnen (Lewis, 1991). Damit kann die empfundene Komplexität einer Aufgabe (Frage 1), das Lernverhalten bzw. die Lernkurve des Testbenutzers (Frage 2) und das Maß der bereitgestellten Hilfefunktionalitäten des Systems (Frage 3) beurteilt werden. Die Bewertung aller drei Fragen zusammengefasst, ist als Maß der Zufriedenstellung eines Testbenutzers mit der Bearbeitung einer Aufgabe aufzufassen.

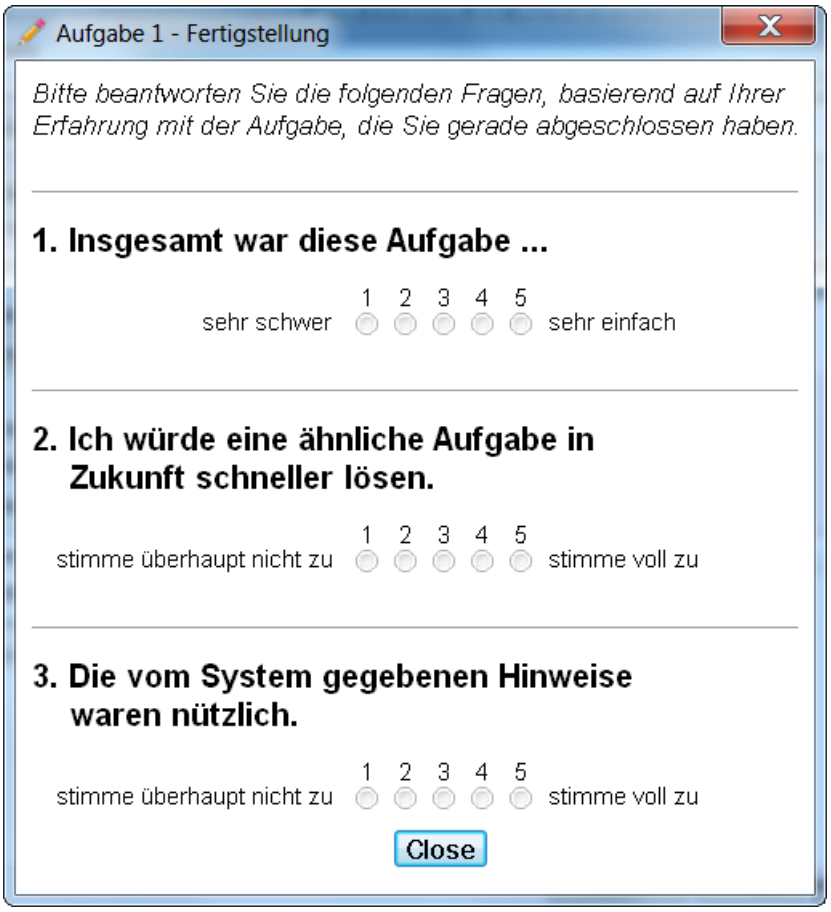

Abb. 10: Fragen an den Testbenutzer nach Fertigstellung einer Aufgabe des Usability-Tests

Für die subjektive Bewertung des entwickelten Prototyps, im Anschluss an den durchgeführten Usability-Test, wurde der standardisierte SUS-Fragebogen ausgewählt (Brooke, 1996). Der konzipierte Ablauf der durchgeführten Usability-Tests ist in Abbildung 11 schematisch dargestellt.

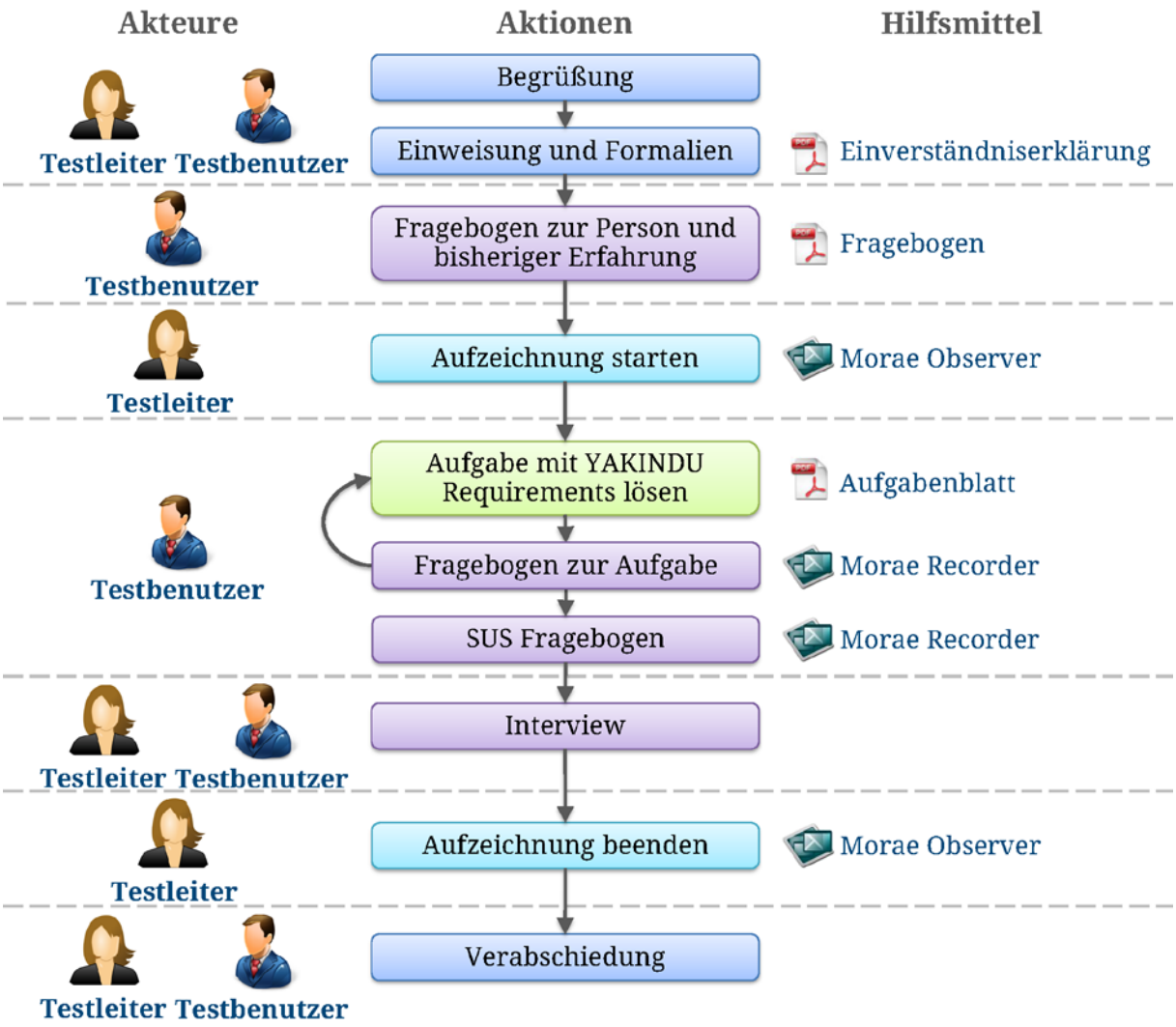

Abb. 11: Schematischer Ablauf des Usability-Tests

Der Einsatz des konzipierten Usability-Tests in Verbindung mit den anschließenden Interviews und dem SUS-Fragebogen erwies sich als geeignete Kombination von Methoden für die durchgeführte Usability Evaluation eines grafischen Editors.

Der entwickelte Prototyp erreichte einen SUS Score von 76,6%, was einer guten Usability entspricht. Übersetzt in andere Bewertungsskalen ist die Usability des Prototyps mit der Note C zu bewerten oder mit dem Adjektiv 'akzeptabel' zu charakterisieren (Bangor, 2009). Unter den insgesamt 27 identifizierten Usability-Problemen befindet sich lediglich ein schwerwiegendes Problem. Mit der durchgeführten Usability Evaluation konnte eine positive und umfassende Bewertung des entwickelten Prototyps aufgezeigt werden, die in ausführlicher Form in (Juhnke, 2014) dokumentiert ist.

8. Fazit

Das Ergebnis der Studie ist eine prototypische Umsetzung eines grafischen Editors, welcher mit seinem Funktionsumfang grafikaffinen Benutzern eine effiziente Möglichkeit zur Erstellung syntaktisch fehlerfreier formaler Anforderungsmodelle bietet. Dabei wird eine Brücke zu bestehenden und rein textuellen Mechanismen geschlagen, welche ein interdisziplinäres Zusammenarbeiten und die Kommunikation zwischen verschiedenen Benutzergruppen fördert. Die

Usability Evaluation zeigte zudem, dass grafische Editoren Benutzern ohne technisches Hintergrundwissen einen einfachen Umgang mit Anforderungsmodellen und dem Werkzeug YAKINDU Requirements ermöglichen.

Zusätzlich profitieren textaffine Benutzer von der Realisierung des Sichtenkonzepts, wenn es darum geht benutzerspezifische Repräsentationen zu generieren. Anhand der erhobenen Anforderungen konnten Schwerpunkte definiert werden, deren Beachtung bei der Entwicklung grafischer Editoren die Usability erhöht. Zusammen mit der Vorstellung einer geeigneten Methodik zur Usability Evaluation grafischer Editoren wurde somit ein umfassendes Szenario von der Konzeption, über die Entwicklung bis hin zur Usability Evaluation einer benutzerzentrierten Lösung im Requirements Engineering gezeichnet.

9. Ausblick

Als ein nächster Schritt ist die Einarbeitung der Ergebnisse aus der durchgeführten Usability Evaluation in den entwickelten Prototypen umzusetzen. Dabei ist u. a. der Aspekt der Synchronisation in Echtzeit erneut zu betrachten, da diese Funktonalität maßgeblich dazu beiträgt Zusammenhänge und Auswirkungen von Interaktionen zwischen dem textuellen und grafischen Editor nachzuvollziehen. Neben der textuellen Notation für Geschäftsmodelle sind das entwickelte Konzept sowie die Usability Evaluation auf weitere textuelle Notationen in YAKINDU Requirements übertragbar. Dafür müssen die jeweiligen Restriktionen einer Notation extrahiert und auf den grafischen Editor übertragen werden. Das Skript für einen Usability-Test gilt es entsprechend des jeweiligen Typs von Anforderungsmodell individuell anzupassen. Daraus sollen Schablonen entstehen, die für die Usability Evaluation grafischer Editoren zur Erstellung von Anforderungsmodellen im Kontext des Requirements Engineerings eingesetzt werden können. Außerdem gilt es zu evaluieren, inwiefern die ermittelten Schwerpunkte außerhalb des Requirements Engineerings für die Kombination textueller und grafischer Editoren anwendbar und gültig sind.

Literatur

1. Balzert, H. (2001). „Lehrbuch der Software-Technik“. Heidelberg: Spektrum
2. Bangor, A., Kortum, P. & Miller, J. (2009). „Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale”. JUS Journal of Usability Studies 4, 3, (S. 114-123)
3. Bittner, K. & Spence, I. (2002). „Use Case Modeling”. Addison-Wesley.
4. Brand, C., Gorning, M., Kaiser, T., Pasch, J. & Wenz, M. (2011). „Graphiti Entwicklung hochwertiger grafischer Modelleditoren“. Eclipse Magazin, 1, (S. 67-72)
5. Brooke, J. (1996). „SUS - A quick and dirty usability scale”. Usability evaluation in industry. (S. 189-194)
6. Eichelberger, H., Schmid, K. & Eldofan, Y. (2011). „A comprehensive analysis of UML tools, their capabilities and their compliance”. Forschungsbericht, Universität Hildesheim, Institut für Informatik.
7. Geyer, F., Trompeter, J. & Jendryschik, M. (2013). „Von der Nutzungsanforderung zur formalen Softwarespezifikation”. Tagungsband Usability Professionals 2013, German UPA, (S. 54-59)

8. Juhnke, K. (2014). „Konzeption und prototypische Entwicklung eines grafischen Editors zur Darstellung und Bearbeitung einer textuellen domänenspezifischen Sprache zur Anforderungsmodellierung“. Masterarbeit, HTWK Leipzig.
9. Lewis, J. R. (1991). „An After-Szenario Questionnaire for Usability Studies”. Psychometric Evaluation Over Three Trials. 4, (S. 79ff)
10. Sarodnick, F. & Brau, H. (2011). „Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendung“. Bern: Verlag Hans Huber.

Vita

Katharina Juhnke studierte Informatik an der HTWK Leipzig und ist IT-Beraterin sowie Usability Engineer bei der itemis AG. Dort ist sie seit 2011 in den Bereichen Usability Engineering und Requirements Engineering tätig. Schwerpunkte ihrer Arbeit liegen in der Konzeption und Entwicklung Eclipse-basierter Anwendungen im Kontext des Requirements Engineerings sowie der modellbasierten Softwareentwicklung, der Durchführung von Usability Evaluationen und dem IT-Marketing. Seit 2012 ist sie zudem als Referentin und Coach für werkzeuggestütztes Requirements Engineering tätig.