UML4ALL: Gemeinsam in Diversity Teams Software modellieren

Für Menschen mit und ohne Seheinschränkung

Karin Müller [1], Vanessa Petrausch [1], Gerhard Jaworek [1], Jörg Henß [2], Stephan Seifermann [2], Claudia Loitsch [1], Rainer Stiefelhagen [1]
[1] Studienzentrum für Sehgeschädigte (SZS) am Karlsruher Institut für Technologie (KIT) und [2] FZI Forschungszentrum Informatik, Karlsruhe
karin.e.mueller@kit.edu

Zusammenfassung

Das gemeinsame Entwerfen von UML-Modellen durch Personen mit und ohne Seheinschränkung ist durch die aktuell vorhandenen Werkzeuge und Modellierungssprachen nur eingeschränkt möglich. Grund hierfür sind unter anderem Werkzeuge wie Eclipse, die nur bedingt barrierefrei bedienbar sind und Modellierungssprachen wie z.B. PlantUML, die die Anforderungen für eine barrierefreie Nutzung nur teilweise erfüllen. So können blinde Software-Entwicklerinnen und –Entwickler zwar mit Eclipse und den existierenden UML-Plugins grafische Modelle erzeugen, diese jedoch nicht mit einem Screenreader lesen. Im Rahmen des Cooperate-Projektes wurde ein Kooperationswerkzeug entwickelt, das je nach Arbeitsweise eine textuelle oder grafische Modellierung erlaubt. Änderungen werden synchron gehalten. Für die textuelle Modellierung von UML-Diagrammen wurde mit UML4ALL eine textuelle Syntax entworfen, die ohne die Nachahmung von visuellen Elementen durch ASCII-Zeichen auskommt und der Arbeitsweise von Menschen mit Blindheit beispielsweise durch die Verwendung einer Präfixnotation entgegenkommt. Diese neue Sprache hilft nicht nur Menschen mit Seheinschränkung, sondern allen, die Modelle textuell entwerfen möchten. Das beschriebene Vorgehen lässt sich auch auf andere grafische Notationen übertragen. Um den Einstieg in die UML, in UML4ALL, in PlantUML und in die Entwicklungsumgebung Eclipse zu erleichtern, wurden Schulungsmaterialien erstellt, die mit Hilfe von taktilen Grafiken, textuellen Beschreibungen und Einführungstexten die unterschiedlichen Themen näher erläutern. Alle bisher entwickelten Inhalte inklusive einer ersten Version des Kooperationswerkzeugs wurden von Personen mit Seheinschränkung getestet und zur Verbesserung der Materialien verwendet.

Einleitung

2008 trat die UN-Konvention [1] über die Rechte von Menschen mit Behinderung in Deutschland in Kraft. Sie sichert Menschen mit Behinderung das Recht auf Arbeit in einem offenen, inklusiven und zugänglichen Arbeitsumfeld zu. Dies umfasst auch die Verpflichtung der Vertragsstaaten, angemessene Strukturen zu schaffen, um möglichst alle Arbeitsgebiete - auch den Bereich der Informationstechnologie - zugänglich zu machen. In der Informationstechnologie (IT)-Branche ist der Bedarf an Fachkräften hoch und es herrscht Konsens darüber, dass es insbesondere in der Softwareentwicklung, einen Fachkräfte-Engpass gibt. In einer Analyse aus dem Jahr 2013 wurde “die durchschnittlich abgeschlossene Vakanzzeit (Niveau und Veränderung) als auch das Verhältnis der Zahl der Arbeitslosen zur Zahl der offenen Stellen” herangezogen [4]. Dabei ist im Bereich der Informatik und Softwareentwicklung eine Tendenz einer steigenden Vakanzzeit von 2012 auf 2013 erkennbar [2]. Generell wurde die Bedeutung der IT für weitere Branchen und somit ein steigender Bedarf an Fachkräften, beispielsweise zur Datenanalyse mittels Big Data Technologien und dem Aufbau von Know-How zur Stärkung der eigenen Wettbewerbsfähigkeit und Effizienz, bereits branchenübergreifend im Jahr 2012 belegt [3].

Im IT-Bereich bietet insbesondere die Software-Entwicklungsbranche für Menschen mit Sehschädigung gute Beschäftigungsmöglichkeiten, da viele Teile der Softwareentwicklung wie das traditionelle Programmieren mittels Editoren, Debuggern und Compilern prinzipiell textbasiert erfolgen können und damit durch die Entwicklung der Screenreader-Technologien zugänglich sind. Screenreader sorgen dafür, dass Inhalte auf einem Bildschirm entweder vorgelesen oder über die Braillezeile erfassbar werden. Seit vielen Jahren kommen in der Softwareentwicklung jedoch verstärkt grafische Modellierungswerkzeuge zum Einsatz, die den Standards der Barrierefreiheit nicht genügen und mit bisherigen Methoden von Menschen mit Seheinschränkung nicht sinnvoll

verwendet werden können. Damit ist die Voraussetzung nicht mehr gegeben, dass Fachkräfte mit Sehschädigung beim Entwurf von Softwaresystemen mit Modellierungswerkzeugen ohne Unterstützung mitarbeiten können.

Für eine reibungslose Zusammenarbeit in Entwicklungsteams, die aus Menschen mit und ohne Seheinschränkung bestehen, sind daher Werkzeuge notwendig, welche barrierefrei bedienbar sind und Lösungen, die verschiedene Sichtweisen auf ein Modell erlauben und somit eine Kooperation ermöglichen. Ein solches Werkzeug ist häufig nicht nur für Menschen mit Seheinschränkung in Diversity Teams, sondern auch für Personen interessant, die aus Gründen der Produktivität eine textuelle Darstellung zum Modellieren bevorzugen. Gerade in der Entwicklung werden häufig Arbeitstechniken ohne Mauseinsatz, die auch Menschen mit Seheinschränkung nutzen, angewendet, um das Arbeitstempo zu erhöhen. Daher muss ein inklusiv nutzbares Modellierungswerkzeug die Möglichkeit bieten, die Darstellung zu wählen, die der eigenen Arbeitsweise entspricht: eine textuelle Darstellung, die von Menschen mit Blindheit oder hochgradiger Sehbehinderung und eine visuelle bzw. textuelle Alternative, die von sehenden Teammitgliedern genutzt werden kann. Es bedarf auch einer Software, die in Echtzeit alle Varianten synchronisiert und verwaltet.

Die in diesem Artikel beschriebenen Arbeiten wurden im Rahmen des vom Bundesministerium für Arbeit und Soziales geförderten dreijährigen Projektes Cooperate durchgeführt. Ziel der Arbeit ist, existierende Ansätze im Bereich der textuellen Notationen für Modellierungssprachen zu vergleichen, deren Barrierefreiheit und Nutzbarkeit für Diversity Teams zu untersuchen und neue softwarebasierte Lösungen für die Zusammenarbeit von Menschen mit und ohne Seheinschränkung für den Software-Entwicklungsbereich zu entwickeln. Untersucht wird dabei exemplarisch die Modellierung mittels der weit verbreiteten Unified Modeling Language (UML). Im Einzelnen entwickeln wir folgende Komponenten: (1) eine Eclipse Erweiterung, welche UML Modellierungsaufgaben in Softwareentwicklungsteams unter Berücksichtigung der genannten Ziele unterstützt, (2) eine neue textuelle UML Syntax, UML4ALL, sowie (3) Schulungen, welche von Multiplikatoren in der IT-Ausbildung genutzt werden können.

Der Artikel gliedert sich in folgende Abschnitte: Im ersten Abschnitt werden Anforderungen und Umsetzungsansätze für die Entwicklung eines Kooperationswerkzeugs vorgestellt. Diese sind größtenteils schon in der aktuellen Version der Software enthalten. Im zweiten Abschnitt werden verschiedene textuelle Modellierungssprachen untersucht. Dabei werden Richtlinien für den Entwurf von barrierefreien textuellen Modellierungssprachen formuliert, die dann in der Cooperate eigenen Syntax, UML4ALL, umgesetzt wurden. Der dritte Abschnitt beschreibt die im Cooperate-Projekt entstandenen Schulungsunterlagen. Im letzten Abschnitt erfolgt ein Ausblick auf zukünftige Arbeiten.

Entwicklung eines Kooperationswerkzeugs

Modellierungsumgebungen werden in Unternehmen und Bildungseinrichtungen bereits eingesetzt, um alleine oder gemeinsam einfach und schnell Modelle zu erstellen, zu bearbeiten und zu verstehen. Leider sind diese Werkzeuge nicht oder nur eingeschränkt auch für Menschen mit Seheinschränkung barrierefrei nutzbar. Das im Rahmen von Cooperate entwickelte Kooperationswerkzeug ist eine Modellierungsumgebung, die einen neuen, inklusiven Ansatz verfolgt, damit Softwareentwicklerinnen und Softwareentwickler mit oder ohne Seheinschränkung gemeinsam Modelle entwerfen können. Abbildung 1 veranschaulicht das Ziel-Szenario, in dem eine blinde Person (1) mit Sprachausgabe und Braillezeile an der textuellen Variante der Modelle arbeitet, eine normal sehende Person (2), welche die visuelle Variante modifiziert und eine Person mit Sehbehinderung (3), welche die stark vergrößerte textuelle Variante bearbeitet. Das Kooperationswerkzeug ist die Voraussetzung dafür, dass die Projektergebnisse, wie z. B. die barrierefreie textuelle Notation für UML-Diagramme auch in der Praxis eingesetzt werden können.

Bei einer Voruntersuchung der Anforderungen an eine solche Modellierungsumgebung [11] durch Befragungen, sowie Nachforschungen in Diskussionsgruppen und Webseiten wurden sechs zu adressierende Aspekte ermittelt:

1. Zunächst muss die verwendete Modellierungssprache und die dafür verwendeten Notationen (z.B. grafische und textuelle) den Mitgliedern des Teams bekannt sein. Dies stellt eine Grundvoraussetzung dar, da eine Notation bei falscher Verwendung von Elementen die Kommunikation im Team nicht mehr erleichtert, sondern mehrdeutig macht und damit erschwert.
2. Mindestens eine Notation für ein UML-Diagramm muss dabei barrierefrei sein, da ansonsten Teile des Teams ausgeschlossen werden.
3. Beim Bearbeiten von UML-Diagrammen ist eine dem Stand der Technik entsprechende Editierunterstützung erforderlich, um die gewohnte Modellierungsgeschwindigkeit zu erhalten und Fehler zu ver-

meiden. Dies bedeutet auch, dass ggf. technische Notwendigkeiten hinter gut zu bedienenden Editor-Funktionen versteckt werden müssen.

4. Wesentlich für die Arbeit im Team ist nach dem Editieren eines UML-Diagramms in einer Notation die Erhaltung der Konsistenz zwischen verschiedenen Notationen. Dabei dürfen keine Informationen verloren gehen. Dies beinhaltet vor allem Darstellungsinformationen und Querverweise auf Elemente innerhalb der Notationen. Die Konsistenzerhaltung muss zudem automatisch erfolgen.
5. Für einen praktischen Einsatz vor allem in Unternehmen muss neben diesen Punkten sichergestellt werden, dass die Modellierungsumgebung in deren üblichem Umfeld einsetzbar ist. Entscheidend hierfür ist, dass der Einsatz der Modellierungsumgebung nicht zwingende Voraussetzung für jedes Teammitglied ist, sondern stattdessen eine Integration mit bestehenden Werkzeugen vorgenommen werden kann. Dies ermöglicht eine schrittweise Migration bei gleichzeitiger Erhaltung der Arbeitsfähigkeit des Teams.
6. Gerade auch für Unternehmen ist die Nachhaltigkeit der Umgebung wichtig. Es muss daher sichergestellt werden, dass die Konzepte zur barrierefreien Umsetzung von Modellierungssprachen dokumentiert werden, um bspw. nach Änderungen in der UML die Modellierungsumgebung entsprechend anpassen zu können.

Diese Anforderungen werden durch das Cooperate-Projekt teilweise technisch, teilweise organisatorisch adressiert. Ein Überblick über die Lösungsansätze findet sich in Tabelle 1. Organisatorisch wird bspw. die Verständlichkeit der Notationen durch Schulungen gewährleistet, die im Abschnitt „Entwicklung von Schulungsunterlagen“ beschrieben werden. Die barrierefreien Notationen werden durch eine Sprachdefinition nach dem Vorgehen aus dem Abschnitt „Textuelle Modellierungssprachen“ erstellt. Ebenso wird die Nachhaltigkeit der Modellierungsumgebung durch einen Leitfaden zur Entwicklung barrierefreier Modellierungsumgebungen und Anwendungen im Allgemeinen sichergestellt werden. Hier fließen vor allem auch die Gestaltungsrichtlinien für textuelle Notationen ein.

Tabelle 1 Überblick über Anforderungen und zugehörige wesentliche Lösungsansätze in Cooperate

Anforderungen	**Wesentlicher Lösungsansatz**
R1 Verständnis von Notationen	S1 Lehrmaterial zu Beschreibungssprachen
R2 Zugänglichkeit von Notationen	S2 Textuelle UML-Notation
R3 Editierunterstützung	S3 Unterstützungsfunktionen im textuellen Editor
R4 Konsistenz von Notationen	S4 Echtzeitsynchronisation von Notationen
R5 Anwendbarkeit in der Praxis	S5 Integration kommerzielle Modellierungsumgebung
	S6 Schulungsunterlagen Kooperationswerkzeug
R6 Nachhaltigkeit	S7 Richtlinien zu barrierefreier Entwicklung

Technisch werden vor allem die Forderungen nach Editierunterstützung, Konsistenz und praktischer Einsetzbarkeit in Unternehmen erfüllt. Ein grober Überblick, wo diese Forderungen im Kooperationswerkzeug verortet sind, findet sich in Abbildung 1. Dabei ist der Aufbau so, dass es verschiedene Arbeitsplätze gibt, die jeweils eine andere Notation für das zu bearbeitende UML-Diagramm nutzen. Im konkreten Fall sind textuelle und grafische Darstellungen vorhanden. Zusätzlich kann jeder Nutzer seine gewünschten Assistenzlösungen (Screenreader mit Sprachausgabe und Braillezeile, Vergrößerungssoftware) nutzen. Die Editierunterstützung findet an den jeweiligen Arbeitsplätzen statt. Alle Nutzenden greifen dabei auf einen zentralen Modellspeicher zu. Beim Übertragen von Änderungen in diesen Modellspeicher werden die Änderungen auch für die anderen Notationen übernommen, sodass im Modellspeicher die Notationen stets konsistent sind. Die einzelnen Arbeitsplätze fragen dann die jeweils aktuellste Version ab, was dort für Konsistenz sorgt. Zur Einsetzbarkeit in Unternehmen wird beispielhaft eine Anbindung an ein kommerzielles Modellierungswerkzeug entwickelt. Dieses gliedert sich wie ein weiterer Arbeitsplatz in das Kooperationswerkzeug ein.

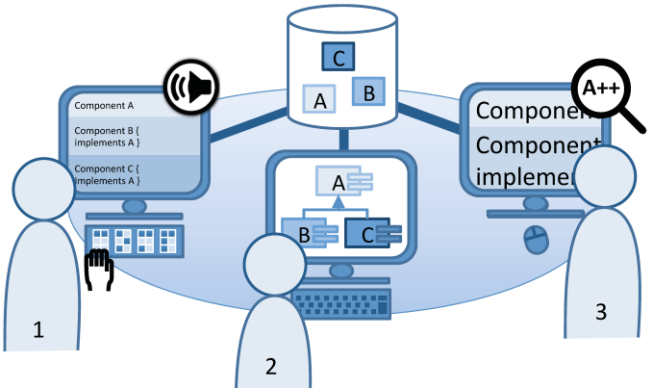

Abb. 1 Aufbau des Kooperationswerkzeugs im Diversity Team bestehend aus blinden (1) Menschen, Menschen ohne Seheinschränkung (2) und Menschen mit Sehbehinderung (3). Dabei nutzen alle einen gemeinsamen Modellspeicher (oben Mitte), auf dem das Modell gespeichert wird.

Die Editier-Unterstützung ist Grundvoraussetzung für eine barrierefreie Interaktion mit UML-Diagrammen [11]. Unterstützt wird sowohl das vollständige Neuanlegen eines Diagramms, als auch das Editieren eines bestehenden Diagramms. Für Ersteres ist vor allem das Neuanlegen von Elementen relevant. Der textuelle Editor unterstützt dies durch Vervollständigung von Quelltext, Fehlerbenachrichtigung und Refaktorisierungen. Die Vervollständigung bezieht sich hier sowohl auf Schlüsselwörter, als auch auf Querverweise. Ein Nutzer muss dadurch nicht alle Details der Notation kennen und muss sich nicht das vollständige Modell merken, was eine enorme Erleichterung zu bisherigen Verfahren wie dem Arbeiten mit reinen Texteditoren darstellt. Fehlerbenachrichtigungen sind nach der Anforderungserhebung [10] auch in verbreiteten Editoren noch problematisch. In der Modellierungsumgebung erfolgen sie daher in Echtzeit und ermöglichen es, direkt zur betroffenen Stelle im Texteditor zu springen. Auf einen Fehler wird mit einer nicht-invasiven akustischen Benachrichtigung hingewiesen. Mittels Refaktorisierungen können komplexe Änderungen in einem einfachen Schritt durchgeführt werden. Beispielsweise kann ein Element umbenannt werden, wodurch auch alle dieses Element referenzierende Elemente automatisch angepasst werden. Für das Editieren bestehender Diagramme bietet der Editor Unterstützung beim Explorieren und Verstehen des existierenden Diagramms. Dazu ist es möglich, von Querverweisen direkt zur Definition des Elements zu springen. Die Darstellung kann über das Ausblenden von Quelltextbestandteilen das Querlesen unterstützen. Häufig genutzte Explorationsaufgaben sollen mit Hilfe von vordefinierten Suchanfragen besser beantwortbar werden.

Kollaboration innerhalb des Kooperationswerkzeugs ist nur möglich, wenn die UML-Diagramme in den verschiedenen Notationen die gleichen Informationen darstellen und nicht widersprüchlich zueinander sind. Im Kooperationswerkzeug werden dafür die verschiedenen Darstellungen miteinander synchronisiert. Dies geschieht über bidirektionale und inkrementelle Transformationen. Bidirektional bedeutet hier, dass eine Änderung in beide Richtungen (Text zu Grafik bzw. Grafik zu Text) übertragen werden kann. Inkrementell heißt in diesem Kontext, dass nur die von der Änderung betroffenen Bestandteile geändert werden. Letzteres ist vor allem von Bedeutung, wenn nicht alle Informationen in beiden Notationen vorhanden sind, da diese sonst bei vollständig neuem Erzeugen verloren gehen würden. Ein prominentes Beispiel dafür sind Layoutinformationen. Diese werden nicht zwischen verschiedenen Notationen synchronisiert, sollen aber auch nach Änderungen erhalten bleiben. Verschiedene Editor-Technologien tragen ebenfalls zur Komplexität der Konsistenzhaltung bei. Für eine Erläuterung der Problematik sei auf einen initialen Vorschlag zur Konsistenzhaltung [10] verwiesen. Technisch erfolgt die Umsetzung über bidirektionale, inkrementelle Modelltransformationen [13].

Um die Einsetzbarkeit des Kollaborationswerkzeugs in Unternehmen zu demonstrieren, wird im Rahmen des Cooperate-Projekts die im industriellen Umfeld weit verbreitete UML-Modellierungsumgebung Enterprise Architect angebunden. Es ist keine vollständige Migration notwendig. Die Integration erfolgt durch Abbildung der internen Repräsentation des UML-Modells von Enterprise Architect in die Repräsentation des grafischen Editors des Kooperationswerkzeugs. Die Lösung wird, wie das Kooperationswerkzeug selbst, als Open-Source auf Github [12] bereitgestellt.

Textuelle Modellierungssprachen

Textuelle Notationen von UML Modellen bieten viele Vorteile, u.a. ein effizientes Bearbeiten sowie eine präzise Darstellung. Zudem verbessern textuelle Notationen die Zugänglichkeit für Menschen mit Seheinschränkung. Textuelle Repräsentationen können von einem Screenreader vorgelesen, auf einem Brailledisplay ausgegeben

oder stark vergrößert mit einer Bildschirmlupe dargestellt werden. Aktuell gibt es jedoch keine standardisierte textuelle Notation für UML. Existierende Ansätze sind weitestgehend domänenspezifisch und für individuelle Anforderungen und Nutzungskontexte entwickelt. Eine ausführliche Gegenüberstellung von 31 textuellen UML Notationen ist in einer aktuellen Erhebung [9] publiziert. Basierend auf dieser Untersuchung bestehen die wesentlichen Unterschiede in den unterstützten Diagrammtypen und Elementen (UML 2.5 Abdeckung), Art der Syntax, Möglichkeiten des Editierens sowie Anwendbarkeit in Entwicklerteams in Bezug auf Konsistenz, Lizenzen und Entwicklungsprozesse. Eine wesentliche Voraussetzung für die kollaborative Arbeit zwischen Menschen mit und ohne Seheinschränkung ist die Bereitstellung einer textuellen und grafischen Variante. 15 der 31 untersuchten Notationen fokussieren jedoch den reinen Export von UML-Grafiken aus dem Quelltext heraus. Diese unidirektionale Generierung von Grafiken, welche meist Dokumentationszwecken dient, schließt das kollaborative Bearbeiten in grafischer und/oder textueller Form aus. Nur 5 Notationen (PlantUML, TCD, TextUML, tUML, txtUML) erlauben den erforderlichen modellbasierten und UML-kompatiblen Export. Allerdings ist die Abdeckung der Diagrammtypen bei diesen 5 Notation nicht ausreichend: TCD unterstützt ausschließlich Klassendiagramme; TextUML Klassen- und Zustandsdiagramme; tUML Klassen-, Zustands- und Kompositionsstrukturdiagramme und txtUML Klassen-, Zustands- und Aktivitätsdiagramme. PlantUML bietet mit Klassen-, Objekt-, Sequenz-, Zustands-, Aktivitäts-, Anwendungsfall-, Komponenten- und Verteilungsdiagrammen die größte Abdeckung, erweist sich aber hinsichtlich der Editoren-Charakteristik sowie der Syntax als nicht optimal. Beispielsweise bietet PlantUML keine Syntaxhervorhebung oder Autovervollständigung - beide Funktionen sind jedoch im Rahmen der effizienten Softwareentwicklung inzwischen Standard. Hinsichtlich der Syntax weist PlantUML für Menschen, die auf einen Screenreader angewiesen sind, keine gute Usability auf. So werden beispielsweise alle Diagramme in die gleichen Start- und Endtags eingebettet: *@startuml* und *@enduml*. Somit ist nicht sofort ersichtlich, um welche Diagrammart es sich handelt und es muss aus der Syntax auf den Diagrammtyp geschlossen werden. In Tabelle 1 werden drei PlantUML Beispiele gezeigt, die sich in der Syntax nur minimal unterscheiden, jedoch einen erheblichen Unterschied im Typ aufweisen. Wird in PlantUML kein Schlüsselwort verwendet, handelt es sich immer um ein Sequenzdiagramm. Wird dagegen eine ID mit dem vorangestellten Schlüsselwort “class” definiert, handelt es sich um ein Klassendiagramm. Dabei muss nur eine Klasse im gesamten Diagramm durch das Schlüsselwort “class” definiert werden, um das gesamte Diagramm als Klassendiagramm zu definieren. Dies kann bei der Nutzung eines Screenreaders leicht übersehen werden, sodass es mühsam ist, den Typ des Diagramms zu bestimmen, da aufwändig nach eventuellen Schlüsselwörtern gesucht werden muss. Anwendungsfälle können durch runde Klammern um die ID definiert werden. Im Beispiel des Anwendungsfalldiagramms in Tabelle 1 ist “Alice” damit ein Anwendungsfall, “Bob” ist jedoch ein Akteur. Bei diesen Beispielen ist leicht zu sehen, dass minimale Änderungen, welche teilweise leicht bei der Benutzung eines Screenreaders übersehen werden, einen Unterschied in der Bedeutung nach sich ziehen. Darüber hinaus werden in textuellen Notationen häufig visuelle Elemente nachgeahmt, was im Sinne eines universellen Designs nicht der geforderten Kompatibilität mit Screenreadern entspricht. Beispielsweise repräsentiert PlantUML eine Assoziation zwischen zwei Elementen als Zeichen *Bindestrich* gefolgt von *Größer Als, was visuell einen Pfeil darstellt, aber für einen Screenreader, d.h. auf der Braillezeile oder über Sprachausgabe so nicht erkennbar ist.*

Tabelle 2 Gegenüberstellung verschiedener PlantUML Notationen.

Sequenzdiagramm	**Klassendiagramm**	**Anwendungsfalldiagramm**
@startuml Alice -> Bob @enduml	@startuml class Alice -> Bob @enduml	@startuml (Alice) -> Bob @enduml

Richtlinien für den Entwurf von barrierefreien textuellen Notationen

Wie im vorangegangenen Abschnitt erläutert, gibt es viele textuelle Modellierungssprachen, die für sehr unterschiedliche Zwecke und Zielgruppen entworfen wurden. Die Vielfalt an textuellen Notationen zeigt, dass es ein Bedarf auch von Sehenden gibt, Modelle über Text zu entwickeln. Einige der Notationen werden auch von Menschen mit Seheinschränkung genutzt, jedoch wurden diese Sprachen nie auf deren Barrierefreiheit hin untersucht. Ausgehend von der Analyse verschiedener textueller Modellierungssprachen wurde ein Fragebogen entworfen, in welchem vier textuelle Sprachen (PlantUML [7], Umple[1], Earl Grey [6] und yUML[2]) von sechs Expertinnen und Experten bewertet wurden. Die Sprachen wurden stellvertretend so ausgewählt, dass sie unterschiedliche Konzepte repräsentieren. Tabelle 3 zeigt, wie eine Assoziation zwischen einer Klasse “Lehrer” und einer Klasse “Schüler” in den jeweiligen Sprachen beschrieben wird. yUML repräsentiert beispielsweise die Sprachen, welche Klassen und Assoziationen durch grafische ASCII-Zeichen darstellt, während Earl Grey für

[1] http://cruise.eecs.uottawa.ca/umple/
[2] http://yuml.me/

Sprachen steht, welche hauptsächlich Schlüsselwörter verwendet (siehe Tabelle 3). Eine ausführliche Begründung zur Auswahl der Sprachen ist in der zugrundeliegenden Untersuchung [5] zu finden.

Tabelle 3 Beispiel eines Klassendiagramms in 4 textuellen Notationen [5]

PlantUML	**Umple**	**Earl Grey**	**yUML**

class Lehrer {
fach: string
}
Lehrer -> Schüler
	class Lehrer {
	String fach;
	1 -> 1 Schüler;
	}
		class Lehrer
		fach: String
		end
		association
		Lehrer[1]
		Schüler[1]
		end
			[Lehrer|fach: String]
			[Lehrer]->[Schüler]

Insgesamt wurden 29 Aspekte des Klassendiagramms von zwei blinden Programmierern, zwei Accessibility-Expertinnen und zwei Informatikern durchgeführt, die alle Sprachelemente bezüglich der Barrierefreiheit, der Verständlichkeit und der technischen Machbarkeit bewerteten. Basierend auf der Analyse des Fragebogens und einer ausführlichen Untersuchung der relevanten Literatur, wurden fünf allgemeine Richtlinien mit insgesamt 18 Prinzipien zusammengestellt. Diese Richtlinien [5] sollen Softwareentwicklerinnen und -entwicklern beim Entwurf von neuen barrierefreien textuellen Repräsentationen für Software-Artefakte unterstützen. Sie befassen sich mit Aspekten der Nutzbarkeit der textuellen Notation (Usability), der Verbesserung der Zugänglichkeit, der Umsetzung der Notation, der konkreten Syntax und der Interaktion zwischen Notation und Werkzeug:

1. Usability / Nutzbarkeit der textuellen Notation
    a. Halte die Sprache einfach und konsistent
    b. Entwickle eine intuitive Sprache
    c. Einfache Erlernbarkeit
    d. Garantiere durchgängige Konzepte
2. Unterstützung der Zugänglichkeit für Menschen mit Seheinschränkung
    a. Erzwinge unterstützende allgemeine Strukturen
    b. Verwende Überschriften für Hauptstrukturen
    c. Ermögliche Transformations-Mechanismen (Braille, Sprachausgabe, Text)
3. Umsetzung der Notation
    a. Minimiere die Nachahmung grafischer Elemente
    b. Verwende natürliche Sprache
    c. Verwende beschreibende Notationen (Schlüsselwörter)
    d. Übernehme existierende Notationselemente
4. Konkrete Syntax der Sprache
    a. Verwende Sonderzeichen für die Strukturierung des Aufbaus
    b. Verwende einen einheitlichen Stil auf jedem Abstraktionsgrad
    c. Verwende Programmiersyntax für Programmierstrukturen
    d. Verwende eindeutige Ausdrücke
5. Interaktion zwischen der Notation und dem Werkzeug
    a. Versichere die Unabhängigkeit von einer bestimmten Plattform
    b. Unterstütze Schreib-Mechanismen für Arbeitsweisen von Menschen mit und ohne Seheinschränkung
    c. Stelle Mechanismen zur Versionskontrolle bereit

Entwurf einer eigenen Notation: die UML4ALL Sprache

Textuelle Notationen werden von Menschen mit und ohne Seheinschränkung gleichermaßen verwendet. Die Analyse der unterschiedlichen textuellen Notationen und insbesondere der PlantUML-Syntax [7], die bereits von Personen mit Blindheit genutzt wird, ergab, dass diese nur bedingt barrierefrei nutzbar sind. In PlantUML sind viele Layout-Informationen enthalten und grafische UML-Konstrukte werden mit ASCII-Zeichen nachgeahmt. Dies widerspricht den Richtlinien 3a (Minimiere die Nachahmung grafischer Elemente) und auch 1b (Entwickle eine intuitive Sprache). Ein weiterer Nachteil von PlantUML ist, dass zwar einzelne Diagramme modelliert werden können, jedoch die Wiederverwendung von bereits erstellten Elementen nicht möglich ist. Auch können mit PlantUML Diagramme erzeugt werden, jedoch ist die andere Richtung, das heißt von der Grafik zum Text, nicht

möglich. Aus diesen Gründen wurde im Cooperate-Projekte UML4ALL, eine eigene UML Syntax, entworfen, die möglichst viele Elemente aus PlantUML wiederverwendet, um einen einfachen Einstieg in unsere Sprache zu gewährleisten, jedoch da Änderungen vorsieht, wo die Barrierefreiheit betroffen ist:
Benennung der Diagrammart: Beim Entwurf der UML4ALL Sprache wird eine eindeutige Benennung der Diagrammart gefordert. So beginnt ein Diagramm immer mit @start-x und schließt mit @end-x. Beispielsweise steht @start-clsd für den Anfang eines Klassendiagramms (**cl**a**ss** **d**iagram), @start-ucd für das Anwendungsfalldiagramm (**us**e **c**ase **d**iagram), @start-actd für das Aktivitätsdiagramm (**act**ivity **d**iagram) und @start-stmd für das Zustandsdiagramm (**st**ate **m**achine **d**iagram).
Nutzung von Schlüsselworten: Relationen zwischen Klassen werden nicht mit ASCII-Zeichen nachgeahmt, sondern durch abgekürzte englische Schlüsselworte ersetzt, wie zum Beispiel asc für eine Assoziation, agg für eine Aggregation, com für eine Komposition oder isa für eine ist-Beziehung.
Klammerstrukturen: Mithilfe von Klammerstrukturen werden zusammenhängende Konstrukte definiert wie z.B. Blöcke um Klasseninhalte zu spezifizieren oder eckige Klammern für Zusatzinformationen bei Assoziationen. Runde Klammern werden verwendet, um Methoden zu definieren. Dies lehnt sich an den vertrauten Kontext von Programmiersprachen an, sodass der Einstieg in die UML4ALL Syntax erleichtert wird.
Attribute, Methoden und Parameter werden in der UML-Schreibweise angegeben, wie z.B. “name: string” oder “method (a:int, s:string)”.
Anordnung der Argumente: Es wurden verschiedene Anordnungen von Argumenten mit zwei blinden Informatikern getestet, um herauszufinden, welche Anordnung eine rasche Aufnahme der Inhalte gewährleistet. Es stellte sich heraus, dass die sogenannte „Polnische Notation“ bzw. Präfixnotation am ehesten der Arbeitstechnik von Menschen mit Blindheit entspricht. Präfixnotationen haben den Vorteil, dass sie die Hauptinformation zuerst nennen und damit durch die sequenzielle Lesetechnik von Menschen mit Blindheit zuerst erfasst werden.

Das in Abbildung 2 dargestellte Klassendiagramm wird textuell in Tabelle 3 in der UML4ALL-Sprache und zum Vergleich in der PlantUML Notation beschrieben. Dargestellt sind drei Klassen “Student”, “Person” und “Informatik”. Die Klasse Student erbt dabei von der Klasse Person und besitzt eine Assoziation “studiert” zur Klasse Informatik. Tabelle 3 verdeutlicht den Unterschied zwischen den beiden Sprachen und zeigt den Vorteil einer barrierefrei nutzbaren Sprache wie UML4ALL, die Schlüsselworte anstelle von ASCII-Nachahmungen nutzt.

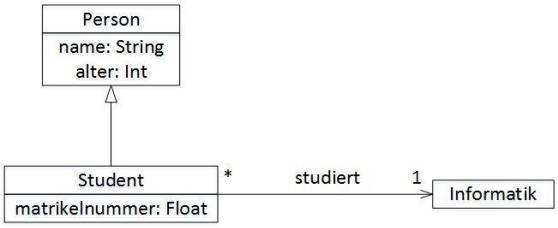

Abb. 2 Grafische Darstellung eines Klassendiagramms mit drei Klassen (Person, Student, Informatik)

Tabelle 3 Gegenüberstellung der UML4ALL und PlantUML-Syntax anhand eines Beispiels

UML4ALL	**PlantUML**
@start-clsd “Studium”	@startuml

class Student {
  matrikelnummer: float
}
class Person {
  name: string
  alter: int
}
class Informatik
	title "Studium"
	
	class Student {
	  matrikelnummer: float
	}
	class Person {
	  name: string
	  alter: int
	}
	class Informatik

isa (Student, Person) asc studiert (Student, Informatik) card[*:1]	Student -|> Person Student "*" --> "1"Informatik: studiert
@end-clsd	@enduml

Entwicklung von Schulungsunterlagen

Der Einstieg in die Modellierung mit UML ist für Softwareentwicklerinnen und -entwickler mit Seheinschränkung sehr schwierig, da bereits in der Ausbildung und im Studium Konzepte mit visuellen Mitteln erklärt werden und diese nicht zugänglich sind [8]. Erschwerend kommt hinzu, dass die Modelle mit grafischen Programmieroberflächen entworfen werden, die häufig die Nutzung der benötigten Hilfsmittel nicht unterstützen.
Im Rahmen des Cooperate-Projektes wurde eigens dafür ein inklusives Schulungskonzept entwickelt, das den Einstieg in die Modellierung von Softwareprojekten fördert. Insgesamt werden drei Schulungen entwickelt, welche verschiedene Aspekte bei der Modellierung von Software fokussieren:

1. Erlernen der Unified Modeling Language (UML) und deren textuelle Repräsentationen
2. Einstieg in die Entwicklungsumgebung Eclipse
3. Zusammenarbeit im Diversity Team mit dem Kooperationswerkzeug

Die Schulungen zur UML und Eclipse sind bereits abgeschlossen. Die Einführung in das Arbeiten im Diversity Team befindet sich aktuell in der Planung und Entwicklung. Nachfolgend werden die ersten beiden Schulungen detaillierter beschrieben.

UML Schulung

Das Ziel der UML Schulung ist, Personen mit Sehschädigung den Zugang zur UML zu ermöglichen, sodass sie UML-Diagramme verstehen, diese modellieren lernen und sich bei Diskussionen über Diagramme aktiv beteiligen können. Die Schulung gliedert sich in die folgenden drei Module und ist zielgruppengerecht aufbereitet:

1. Das erste Modul enthält Materialien für das Selbststudium der Grundlagen der UML.
2. Das zweite Modul beinhaltet Zusatzmaterial in Form von UML-Diagrammen mit PlantUML
3. Und das letzte Modul enthält Materialien für eine Präsenzschulung.

Das Hauptdokument der Schulung ist für das Selbststudium geeignet und erklärt schrittweise textuell und grafisch die Hauptelemente der UML anhand von fünf Diagrammtypen (Klassen-, Anwendungsfall-, Aktivitäts-, Zustandsübergangs- und Sequenzdiagramm). Alle Grafiken sind sowohl als Vektorgrafik, um bei einer hohen Auflösung eine pixelfreie stark vergrößerte Darstellung zu ermöglichen als auch als taktile Grafik eingebunden, um eine Grundvorstellung der grafischen Elemente von UML zu gewährleisten. Ist ein mentales Modell der UML vorhanden, vereinfacht es eine Diskussion über UML Diagramme mit Personen ohne Seheinschränkung. Als Abschluss eines jeden Kapitels sind fünf Verständnisfragen und eine praktische Modellierungsaufgabe enthalten, um das Gelernte zu vertiefen. Die Lösungen sind im Anhang aufgeführt. Zusätzlich zu den genannten Materialien gibt es eine Einführung in die textuelle Notation PlantUML, die zukünftig durch die UML4ALL-Syntax inklusive einer Einführung in die neue Syntax abgelöst wird. Ebenso sind alle Grafiken der Schulung in PlantUML-Notation umgesetzt. Als letztes Modul gibt es eine Folienpräsentation, welche in der Aus- und Weiterbildung für Menschen mit Seheinschränkung oder als Leitfaden für die Durchführung einer Schulung verwendet werden kann.

Eclipse Schulung

Als Vorarbeit zur Eclipse-Schulung wurde eine Online-Umfrage durchgeführt, um zu untersuchen, wie Personen mit Sehschädigung Eclipse nutzen und ob Eclipse barrierefrei nutzbar ist. Die Umfrage wurde von insgesamt 14 Personen, die in der Mehrheit Informatik-Studierende oder Angestellte in der IT-Branche sind, beantwortet. Die Auswertung zeigt, dass die Barrierefreiheit von Eclipse im Allgemeinen schon recht gut ist und viele blinde Entwicklerinnen und Entwickler mit Eclipse programmieren. In der Umfrage wurden zwei Hauptproblemfelder identifiziert: (a) der effiziente Umgang mit Fehlern und Warnungen und (b) die Installation von Plug-Ins. Fehler und Warnungen werden direkt während der Programmierung visuell angezeigt, sind aber erst nach dem Speichern auch für Menschen mit Seheinschränkung zugänglich. Die Installation von Plug-Ins war vor dem Release von Eclipse Neon nicht ohne sehende Hilfe möglich. Mit der neuesten Version ist jedoch auch dieser Dialog barrierefrei. Die Informationen aus der Umfrage flossen direkt in die Entwicklung der Schulung mit ein. So ist beispielsweise ein Kapitel entstanden, welches die oben genannten Probleme beschreibt und Lösungen anbietet.

Die Eclipse-Schulung beschreibt unter anderem die Einrichtung von Eclipse bis hin zum ersten eigenen Programm in Java. Dabei wird vor allem auf die notwendigen Accessibility-Anpassungen eingegangen, z. B. die Aktivierung der Java Access Bridge, die Einstellung des Screenreaders zum Vorlesen von Punktationen und wie Tabulatoren durch Leerzeichen ersetzt werden können. Danach wird der Aufbau von Eclipse anhand eines importierten Programms erklärt. Um Personen mit Blindheit eine Vorstellung über den Aufbau eines Eclipse-Startbildschirms zu vermitteln, wurde dazu eine taktile Grafik erstellt. Die Schulung beschreibt danach das Erstellen eines eigenen Java-Programms. Während der gesamten Schulung wird besonders auf die Verwendung

von Shortcuts für die einzelnen Aktionen Wert gelegt, da Personen mit Blindheit auf diese angewiesen sind, um effizient arbeiten zu können. Die Eclipse-Schulung ist für das Selbststudium konzipiert, da die Nutzung von Programmierumgebungen stark von persönlichen Präferenzen und dem genutzten Betriebssystem abhängt.

Auswertung der Präsenz-Schulungen – „Lessons learned“

Die erstellten Schulungsmaterialien zu UML und Eclipse wurden bereits bei Schulungen von sehgeschädigten Personen eingesetzt. Das Material zum Einstieg in UML wurde im Vorfeld durch zwei blinde Personen, welche Erfahrung in UML haben, getestet. Anhand des erhaltenen Feedbacks wurden die taktilen Grafiken verbessert und einige Inhalte verständlicher formuliert. Die Rückmeldungen zeigten, dass es hilfreich ist, den Anhang der Schulung mit den Wissensfragen zur Überprüfung des Gelernten als eigenständiges Dokument anzubieten. Der Vortest der Schulungsmaterialien ergab außerdem, dass eine kurze Einführung in die PlantUML Syntax für Personen ohne Vorkenntnisse den Einstieg in PlantUML erleichtert. Diese Einführung ergänzt nun die PlantUML Notation der taktilen Grafiken.
Die UML Schulung wurde mit insgesamt vier Studierenden durchgeführt. Drei der Studierenden hatten bereits Vorerfahrung mit UML und PlantUML, der vierte Student hatte keine Vorerfahrung, benötigte UML jedoch für seine aktuelle Vorlesung. Im letzten Fall wurden mithilfe der taktilen Grafiken die Bedeutung der einzelnen Elemente erklärt. Der Student erarbeitete sich die PlantUML-Syntax im Selbststudium und konnte sein neu erworbenes Wissen direkt bei der Abgabe der Modellierungsaufgaben einsetzen. Dadurch konnte er die Vorlesung ohne Verzögerung erfolgreich absolvieren.
Bei der zweiten Schulung, welche zusätzlich PlantUML enthielt, gab es keine Verständnisschwierigkeiten, es wurden jedoch Verbesserungen des Layouts (taktile Grafiken und die Beschreibung der Grafiken in PlantUML-Notation in Braille) angeregt. Zusätzlich wurde um die Aufnahme des Sequenzdiagramms in das Schulungsmaterial gebeten. Beide Vorschläge wurden bereits umgesetzt und sind in der aktuellen Version der Schulungsmaterialien enthalten.

Die Eclipse Schulung ist konzeptuell für das Selbststudium ausgelegt und nicht als Präsenzschulung gedacht. Dennoch wurde diese Schulung ebenfalls mit einem Studenten durchgeführt, da dieser Eclipse aktuell in seinem Studium benötigte. Inhaltlich wurde zuerst Eclipse mit allen benötigten Komponenten und Einstellungen installiert, der Aufbau von Eclipse anhand einer taktilen Grafik veranschaulicht und erste kleine Programme mit den benötigten Schritten erstellt und bearbeitet. Die Programmieraufgaben konnten danach problemlos für die entsprechende Vorlesung durchgeführt werden.

Ausblick

Die neu entstandene Notation, UML4ALL, wird in den nächsten Monaten auf der Homepage www.uml4all.net veröffentlicht, um sie einer breiteren Zielgruppe zur Verfügung zu stellen. Der letzte Schritt, die Erprobung des Kooperationswerkzeugs in einem Diversity Team und die Erweiterung der Syntax auf weitere Diagrammtypen, stehen noch aus und werden im letzten Projektjahr durchgeführt, ausgewertet und publiziert. Die bisherigen Tests zeigen, dass Menschen mit Seheinschränkung mithilfe der Schulungsmaterialien in Kombination mit den taktilen Grafiken schneller einen Einstieg in UML schaffen und selbst mit UML modellieren können, was die Voraussetzung für eine erfolgreiche Zusammenarbeit in einem Diversity Team ist. Die aktuelle Version des Kooperationswerkzeugs wurde bereits von mehreren Personen getestet und das Feedback der Nutzer floss in die Weiterentwicklung mit ein. Der aktuelle Stand der Entwicklungen wird auf der Projekt-Webseite http://www.cooperate-project.de/ dokumentiert. Dort sind auch alle Informationen zum Werkzeug und zu den Schulungsunterlagen zu finden.

Literatur

[1] Bundesgesetzblatt (2008). Gesetz zu dem Übereinkommen der Vereinten Nationen vom 13. Dezember 2006 über die Rechte von Menschen mit Behinderungen. *Bundesgesetzblatt*. Bonn

[2] Bundesagentur für Arbeit (2013). Der Arbeitsmarkt in Deutschland – Fachkräfteengpassanalyse Juni 2013. *Arbeitsmarktberichterstattung*. Nürnberg.

[3] Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS (2012). Big Data - Vorsprung durch Wissen - Innovationspotenzialanalyse. *Fraunhofer Gesellschaft, Institut für Intelligente Analyse- und Informationssysteme IAIS* , Sankt Augustin.

[4] Neuberger N. (2014). Die Debatte über den Fachkräftemangel. *Deutsches Institut für Wirtschaftsforschung e.V.* [abgerufen am 01.12.2016] http://www.diw.de/documents/publikationen/73/diw_01.c.435016.de/diw_roundup_4_de.pdf

[5] Petrausch V., Seifermann S., Müller K. (2016), Guidelines for Accessible Textual UML Modeling Notations. In *International Conference on Computers Helping People with Special Needs (ICCHP2016)*, 67 - 74.

[6] Mazanec M., Macek O. (2012). On General-Purpose Textual Modeling Languages. In *Proceedings of the DATESO 2012 Workshop.* MATFYZPRESS, Prague, Czech Republic, 1-12.

[7] Roques A. (2009). PlantUML in a Nutshell. http://plantuml.com/ [abgerufen am 12.12.2016]

[8] Müller K. (2012). How to Make Unified Modeling Language Diagrams Accessible for Blind Students. In *International Conference on Computers Helping People With Special Needs 13th International Conference (ICCHP 2012)*, Springer-Verlag New York Inc., 186–190.

[9] Seifermann S., Groenda H. (2016). Survey on Textual Notations for the Unified Modeling Language. In: *MODELSWARD 2016*, SciTePress, 20–31.

[10] Seifermann S., Groenda H. (2015). Towards Collaboration on Accessible UML Models. In *Mensch und Computer 2015 - Workshop-Band*, De Gruyter Oldenbourg, 411-417.

[11] Groenda H., Seifermann S., Müller K. and Jaworek G. (2015). The Cooperate Assistive Teamwork Environment for Software Description Languages. In *Proceedings of the 13th European Conference of the Association for the Advancement of Assistive Technology in Europe (AAATE 2015)*, IOS Press, 111-118.

[12] Cooperate Projektteam. Cooperate-Project *Github*. https://github.com/Cooperate-Project [abgerufen am 12.12.2016]

[13] Seifermann S., Henß J (2017). Comparison of QVT-O and Henshin-TGG for Synchronization of Concrete Syntax Models. In *Proceedings of the 6th International Workshop on Bidirectional Transformations* (Bx 2017), accepted to appear.