**Universität Leipzig**

Institut für Informatik

Abteilung Betriebliche Informationssysteme (BIS)

**Hausarbeit zum Thema**

**Abbildung von Constraints für Benutzeroberflächen**

Betreuer: Hr. Johannes Schmidt

Bearbeiter: Hans-Georg Schladitz

Matr.-Nr.: 2137652

4. Semester

Eingereicht am: 29.06.2016

**Abbildung von Constraints für Benutzeroberflächen**

Hans-Georg Schladitz

**Abstract**

UML als Standard ist ein weite verbreiteter und in der Praxis häufig eingesetzter Ansatz zur Modellierung von Spezifikation, Konstruktion und Dokumentation von Software und Systemen. Das Klassendiagramm als bekanntester Vertreter von UML und gleichzeitig als Ausgangsmodelltyp der modellgetriebenen Softwareentwicklung, eignet sich auf Grund der hohen Abstrahierbarkeit auch zur Modellierung von Benutzeroberflächen. Während die Struktur und Beziehungen einer Software leicht über Modelle und Generatoren zum lauffähigen Programmen gebracht werden können, verhält es sich mit der Definition von Einschränkungen (Constraints) wegen der hohen Ausdrucksmächtigkeit und der Notwendigkeit der detaillierten Beschreibung, entgegengesetzt. Ziel dieser Arbeit ist es – bezogen auf UML-Klassendiagramme – einen Überblick über aktuelle Ansätze (Frameworks und Konzepte) und Herausforderungen bei der Verwendung von Constraints zu geben. Grundlage hierfür ist ein kurzer theoretischer Überblick über UML, Constraints und ihre Zusammenhänge.

Schlüsselwörter

*Constraint, OCL, UML, DSL, MDSD, MDE*

Inhaltsverzeichnis

[Abbildungsverzeichnis 4](#_Toc451002066)

[Abkürzungsverzeichnis 4](#_Toc451002067)

[1 Einleitung 5](#_Toc451002068)

[2 Theorie 5](#_Toc451002069)

[3 Analyse 7](#_Toc451002070)

[4 Zusammenfassung 7](#_Toc451002071)

[Literaturverzeichnis 8](#_Toc451002072)

# Abbildungsverzeichnis

[Abbildung 1: Kompositionsbeziehung mit Multiplizitäten 6](#_Toc450578138)

# Abkürzungsverzeichnis

ANTLR Another Tool for Language Recognition

AST Abstract Syntax Tree

DSL Domain-Specific Language

EBNF Erweiterte Backus Naur Form

EMF Eclipse Modeling Framework

EMOF Essential Meta Object Facility

EPL Eclipse Public License

GPL General-Purpose Language

IDE Integrated Development Environment

JVM Java Virtual Machine

MDA Model Driven Architectue

MDE Model Driven Engineering

MDWE Model Driven Web Engineering

MDSD Model Driven Software Development

MWE Modeling Workflow Engine

MOF Meta Object Facility

UI User Interface

UML Unified Modeling Language

UWE UML-Based Web Engineering

OCL Object Contraint Language

OMG Object Management Group

oAW openArchitectureWare

XMI XML Metadata Interchange

# Einleitung

Mit dem Ziel UML-Klassendiagramme als Ausgangsbasis zur Erstellung von Benutzeroberflächen wie z.B. Eingabeformulare zu verwenden, bedarf es einiger theoretischer Grundlagen. Das Klassendiagramm ist ein UML-Model. D.h. es ist ein Standard der OMG. Über Klassen und Beziehungen können Objekte aus der realen Welt (von Systemen) abstrahiert und abgebildet werden. So kann die Komplexität von Systemen, wie z.B. Software, reduziert werden, was die Analyse der Systeme oder die Entwicklung solcher erleichtert. Ein weiterer Verwendungszweck für das Klassendiagramm als Modell ist das Modell Driven Engineering (MDE). Hierbei geht es nach [Kraus et al 2007, 1], neben dem normalen Transformieren und Ausarbeiten von Modellen zur Erstellung von Software, das Ziel des Model Driven Software Development (MDSD) ist, ebenfalls um die direkte Ausführung von Modellen. UML-Modelle eigenen sich wegen der hohen Verbreitung und der verbreiteten Toolunterstützung, wodurch sich, das UML-based Web Engineering (UWE) etabliert hat. UWE basiert auf Model Driven Web Engineering (MDWE), welches auf MDE beruht und sich auf Web-Systeme und nutzt demzufolge webnahe Technologien bezieht. Der Vorteil von MDSD und MDE liegt in der Trennung der Aufgabenbereiche. Modellierer kümmern sich um ihren Problemraum - den Modellen - wohingegen Programmierer sich auf den Lösungsraum - den Code bzw. dem Generator - konzentrieren können. Ebenfalls wird eine Trennung von der Funktionalität des Systems und den Implementierungsdetails geschaffen. Auf die strikte Trennung von Technologie und Funktionalität beruht das Model Driven Architecture (MDA) (vgl. [OMG 2014, 9]). MDA ist ein Mittel zur Implementierung von MDSD und MDE.

# Theorie

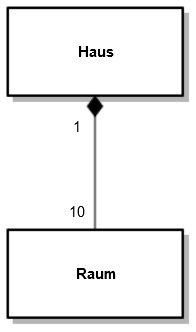
Im Mittelpunkt steht nach Aufgabenstellung nur der Modelltyp der Klassendiagramme von UML. Eine Möglichkeit MDSD umzusetzen ist das Eclipse Modeling Framework (EMF). Es ist ein Java-Tool zum Generierung von Code, dass sich auf Klassendiagramme beschränkt. Dem EMF liegt das Ecore-Metamodell zu Grunde. Dadurch ist es möglich Objektmodelle mit unterschiedlicher Repräsentation zu importieren, indem sie in die kanonische Ecore-Form konvertiert werden. Die kanonische Ecore-Form lässt sich wiederrum in das Standard-Format „XML Metadata Interchange“ (XMI) von OMG transformieren. Dadurch schlägt es die Brücke zum MetaObject Facility (MOF), dem Metamodell von UML. Ecore und MOF sind beides Metamodelle die sich selbst definieren (vgl. [OMG 2015b, 9] und [Kuhn 2008]). Dadurch und wegen ihrer Ausdrucksmächtigkeit eignen sich beider für die Erstellung eigener bzw. neuer Modellierungssprachen. Besonderen Fokus hierbei haben domänenspezifische Sprachen (DSL). Eine DSL ist eine Sprache zur einfachen Darstellung Sachverhalten aus einer bestimmten Domäne für bestimmte Personen (Domänenexperten), die sich in diesem Gebiet auskennen. Dieses Wissen fehlt im normalen Fall dem Programmierer einer Software. Im Sinne des MDSD können mittels Generatoren aus Modellen einer DSL, Generatoren erstellt werden, welche aus den Informationen der Modelle Code generieren. Der Vorteil hierbei liegt wiederum in einer Trennung, der Trennung zwischen der fachlichen und technischen Sicht. Der Programmierer braucht nur die Generatoren für den Modelltyp zu erstellen. Dadurch reduziert sich die Programmierzeit und die Programmierfehler. Der Domänenexperte kann ohne Programmierkenntnisse sein Expertenwissen in Modellen übertragen. Per Kopfdruck wird aus den Modellen entsprechender Quellcode einer bestimmten Programmiersprache (abhängig von der Erstellung des Generators) generiert. Das UML Klassendiagramm ist genau betrachtet bereits eine DSL zur Beschreibung von Anwendungssystemen. Über Klassen, die Konzepte der realen Welt abbilden und Beziehungen zwischen diesen, wird auf konzeptioneller Ebene die Struktur von domänenabhängigen Systemen definiert. Hierbei können über die Arten der Beziehungen konkrete Aussagen über die Konzepte getroffen werden. So ist es beispielsweise möglich über die Vererbungsbeziehung (Generalization) zwischen Klassen komplette Klassenhierarchien zu beschreiben. Klassen selbst haben neben ihren Namen auch Eigenschaften (Attributes) und ein Verhalten (Behavior), dass über Methoden zum Ausdruck gebracht wird. Neben der Vererbungsbeziehung gibt es noch die Enthaltenseinbeziehung (Aggregation, Composition), Abhängigkeitenbeziehung (Dependency) und die normale Assoziation (Association).

Abbildung : Kompositionsbeziehung mit Multiplizitäten

Grundform aller Beziehungen ist die normale binäre Assoziation (vgl. [OMG 2015, 200]). Sie ist definiert durch einen Anfang und ein Ende. An den Endpunkten sind jeweils Klassen referenziert. Dadurch ist es möglich, wie in Abb. 1 gezeigt, mittels Intervallen (Multiplizitäten), Wertigkeiten der Beziehung zwischen diesen Klassen beschreiben zu können. Dies entspricht einer Einschränkung und damit einer Art von Zusicherung (Constraint), da eine Ober- bzw. Untergrenze die Anzahl der Möglichkeiten begrenzt.

Es werden sechs Arten von Contraints unterschieden:

* Invariants,
* Pre-/Postconditions,
* Initial & derived Values,
* Definitions,
* Body Definitions und
* Guards.

Invariants sind Zusicherungen, die zu jeder Zeit für eine Instanz oder Assoziation gelten. Im Fall der Multiplizität heißt dies am Beispiel das die Abb. 1 illustriert, dass ein Haus in diesem Fall genau 10 Zimmer enthalten muss. Die Vor- bzw. Nachbedingung müssen zur Ausführung einer bestimmten Funktion davor bzw. danach gelten. Initial & derived Values beschreiben Bedingungen für Ausgangs- und abgeleitete Werte. So kann z.B. festgelegt werden, dass innerhalb eines Formular-Klassendiagramm, das Attribute „Value“ vom Objekt mit dem Namen „name“ des Typs „Textarea“ mit dem Startwert „John Doe“ initialisiert wird. Über Definitionen können Attribute und Operationen definiert werden, die nicht im Klassendiagramm enthalten sind. Body Definitions beschreiben die Art der Operation. Mit z.B. „isQuery = true“ wird eine Operation zur Abfrageoperation. Guards sind Zusicherung, die bei einem Zustandsübergang gelten müssen. In dieser Arbeit wird, aufgrund des kleingehaltenen Rahmens, der Focus auf Invariants gelegt. Invariants werden von dem UML-Klassendiagramm bereits ausreichend unterstützt. D.h. sie können bereits über die UML-Sprachelemente ausgedrückt werden.

# Analyse

Eine Sprache zur Beschreibung von Constraings für UML-Klassendiagramme ist OCL.

OCL erweitert UML um die Möglichkeit zusätzliche Randbedingungen beschreiben zu können. OCL-Ausdrücke sind widerspruchsfrei und können darum von Programmen verarbeitet werden. Bei der Codegenerierung von einem UML-Klassendiagramm zu Programmcode, dient OCL der Überwachung. Mittels OCL wird demnach überprüft, ob die definierten Constraints beim Generieren von Code erfüllt sind. Weitere Beispiele die mittels OCL in Form von Constraints definiert werden können sind Wertebreiche oder einzuhaltende Restriktionen zwischen Objekten. Beispiel hierfür ist das Alter einer Person, der nicht negativ sein darf, oder die Bedingung, dass eine Person jünger sein muss als die Eltern. Problematisch bei OCL ist die Komplexität. Aufgrund der Ausdrucksmächtigkeit der Sprache können Constraints sehr detailliert beschrieben werden, was bei der zudem bestehenden schweren Syntax oft viel Zeit erfordert und hinreichendes Know-how zu OCL selbst, als auch zur Domäne, die es abzubilden gilt. Jedoch, führt der Einsatz von OCL zu einem höheren Generierungsgrad.

Als erstes Framework zu Modellierung von UML-Klassendiagrammen und zur Definition von Constraints wird Papyrus untersucht. Papyrus ist eine Modellierungsumgebung die auf der Entwicklungsumgebung (IDE) Eclipse basiert. Papyrus konzentriert sich auf die Umsetzung von Standards. Aktuell wird der UML 2.5.0 und OCL 2.3.1 Standard unterstützt, sodass es für die Untersuchung geeignet ist. Ziel bei der Untersuchung ist es ein Klassendiagramm zu erzeugen, welches ein Eingabeformular beschreibt. Dazu soll betrachtet werden, inwiefern Constraints mittels OCL zu diesem Formular beschrieben werden können. So soll durch Zusicherungen verhindert werden, dass Passwort und Name identisch sein können, oder dass es bei einem Angebot von Pizzen, kein Pizza-Typ doppelt angeboten wird.

Grundsätzlich gibt es X Möglichkeiten um aus UML-Klassendiagrammen zusammen mit definierten Constraints Code zu generieren. Diese sind:

* OCL,
* Profiles und
* DSLs.

OCL ungeeignet, da sehr ausdrucksstark und darum sehr komplex, schwer zu erlernen.

Profiles TODO

DSLs -> xText

MWE (Modeling Workflow Engine)

Welche grundlegenden Arten von Constraints gibt es? Geben Sie einen Überblick.

Welche Constraints können bereits mit den Sprachelementen der UML ausgedrückt werden?

Welche Möglichkeiten gibt es, Constraints zu den Entitäten / Modellelementen anzugeben? Geben Sie einen Überblick.

Wie können diese Constraints in der Generierung von Software angewendet bzw. umgesetzt werden?

ToDo:

* MPS
* XText
* Eclipse Papyrus
* EMF-Rest

<http://modeling-languages.com/emf-rest-now-with-data-validation-security-and-full-restful-support/>

* <https://github.com/netceteragroup/valdr#getting-started>
* <http://netceteragroup.github.io/valdr/> (define model-based constraints in JSON)

Eine Plattform für die modellgetriebene Softwareentwicklung ist die openArchitectureWare (oAW). Sie ist unter der Open-Source-Lizenz Eclipse Public License (EPL) frei verfügbar. Die oAW bietet die Möglichkeit für eine Vielzahl von Modelltypen Codegeneratoren zu erzeugen um beliebigen Quellcode zu erzeugen. Modelle die verarbeitet werden können sind unter Anderem EMF- und UML-Modelle. Aktuell wird oAW unter dem neuen Namen Modeling Workflow Engine (MWE) im Eclipse Modelling Projects weiterentwickelt. MWE besteht im Kern aus einer Sprachfamilie, die aus drei Teilen zusammengesetzt wird:

* Xtend
* Check und
* Xpand.

Xtend ist eine funktionale (Programmier-)Sprache die zur Erweiterung (Extention) bestehender Metamodelltypen dient. Sie ist ähnlich zu Java und bietet jedoch eine kompaktere Syntax und erweiterte Konzepte bzw. Funktionalitäten. Dadurch ist er Umgang mit ihr schnell zu erlernen, vor allem, wenn bereits Vorkenntnisse in Java bestehen. Check ist äquivalent zu OCL. Jedoch kann Check, im Gegensatz zu OCL nicht nur auf MOF-kompatible Modelle angewendet werden. Beispielsweise kann Check auf Erweiterungen von Xtend zugreifen. Xpand ist eine Templatesprache die speziell für die Codegenerierung entwickelt wurde und bietet dafür wichtige Features. Die Workflow-Engine ist ein Komponentenframework, dass zur Erstellung komplexer Generatoren dient. Dafür wird ein Generator in einzelne Cartridges aufgeteilt. Ein Cartridge ist ein beliebiger Teil eines Generators. Diese Aufteilung in eine Art Modulen ermöglicht eine Wiederverwendung bereits erstellter Generatorteile. Eine Workflowbeschreibung entspricht hier einem Bauplan des Generators und definiert die Schnittstellen der verwendeten Cartridges. Dadurch kann ein Generator als Ganzes innerhalb einer Jar-Datei an Dritte weitergegeben werden und mit der Workflow-Schnittstelle ausgeführt werden. Das Zusammensetzen der Komponenten und das anschließende Ausführen geschieht über Workflows. Workflows werden über eine einzelne Java Virtual Machine (JVM) ausgeführt. Es werden ebenso Transformationen und Artefakt-Erzeugungen über Workflows definiert (vgl. [Lorenzo 2013, 28ff]).

XText benutzt die MWE(2) DSL um die Erstellung von Artefakten zu konfigurieren.

Wird ein Workflow ausgeführt, erzeugt XText alle notwendigen Artefakte für ein UI-Editor der betrachteten DSL. Außerdem wird eine ANTLR-Spezifikation für die DSL abgeleitet. Dieses Tool unterstützt die Erzeugung von Parsern, Lexern und TreeParsern für Gramatiken. Bei XText sorgt ANTLR dafür, dass Abstrakte Syntaxbäume (ASTs) beim Parsen erzeugt werden. XText beruht außerdem auf dem Eclipse Modeling Framework (EMF). Dem EMF zu Grunde, liegt das Meta(meta)modell Ecore. Ecore entspricht einer abstrakten Sprache zur Definition von Metamodellen ohne Technologieabhängigkeit, sowie einem Framework. Diese Metamodelle können wiederum Sprachen anderer Modelle sein. Es dient, angelehnt an der Meta Object Facility (MOF) von der OMG, der Überbrückung zwischen unterschiedlichen Metamodellen, durch Schaffung einer allgemeinen Grundlage. Sind zwei verschiedene Metamodelle MOF- bzw. Ecore-konform, dann können Modelle die auf ihnen basieren gemeinsam, durch z.B. Modelltransformation, verarbeitet werden. Eine Untermenge von MOF 2.0 ist EMOF (Essential MOF). Diese ist weitgehend kompatibel zu Ecore. Folglich sind alle Java-Klassen die von EMF erzeugt werden Unterklassen von EObject, einem Element von Ecore, das als EMF-Äquivalent von java.lang.Object gesehen werden kann. Analog hierzu korrespondiert EClass zu java.lang.Class. Bei der Erstellung einer DSL mit XText wird gewöhnlich mit der Struktur (Grammatik) begonnen. Diese wird intern automatisch in einem AST umgewandelt. Die Grammatik besteht aus Features bzw. Regeln. Die erste Regel entspricht dem Wurzelknoten vom AST. Die Beschreibungsausdrücke einer Regel werden hierbei angelehnt an der erweiterten Backus-Naur-Form (EBNF) rekursiv aufgelöst. Aus dem resultierenden AST können anschließend EMF-Klassen generiert werden. Instanzen der Klasse werden durch eine statische Factory (Fabrikmethode) erzeugt. Aus diesem Grund existieren innerhalb der Klassen keine Konstruktoren. Die Attribute (Features) werden durch Getter- und Setter-Methoden initialisiert (vgl. [Lorenzo 2013, 34ff]).

TODO: XBase

# Zusammenfassung

# Literaturverzeichnis

[Damus 2015] Damus, C. W., Fun with OCL in Papyrus Mars 2015, URL: http://www.damus.ca/blog/2015/6/15/fun-with-ocl-in-papyrus-mars.

[Kraus et al 2007] Kraus, A, Knapp, A, Koch, N., Model-Driven Generation of Web Applications in UWE, 2007.

[Kuhn 2008] Kuhn, S., Diplomarbeit, Entwicklung eines domänenspezifischen UML Diagramms zur

Benutzeroberflächenmodellierung, 2008, URL:

<https://wiki.eclipse.org/images/d/d0/DA_StefanKuhn.pdf>.

[Lorenzo 2013] Bettini, Lorenzo, Implementing Domain-Specific Languages with Xtext and Xtend, 2013.

[OMG 2015] OMG, UML Specification, 2015, URL:

<http://www.omg.org/spec/UML/2.5/PDF>.

[OMG 2015b] OMG, MOF Specification, 2015, URL:

<http://www.omg.org/spec/MOF/2.5/PDF>.

[OMG 2014] OMG, Model Driven Architecture (MDA), MDA Guide rev.2.0, 2014, URL: <http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf>.

[OMG 2014b] OMG, OCL Specification, 2014, URL:

<http://www.omg.org/spec/OCL/2.4/PDF>.