**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 -> TODO

# Abbildungsverzeichnis

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

Abkürzungsverzeichnis

DSL Domain-Specific Language

EMF Eclipse Modeling Framework

GPL General-Purpose Language

IDE Integrated Development Environment

MDA Model Driven Architectue

MDE Model Driven Engineering

MDWE Model Driven Web Engineering

MDSD Model Driven Software Development

MOF MetaObject Facility

UML Unified Modeling Language

UWE UML-Based Web Engineering

OCL Object Contraint Language

OMG Object Management Group

XMI XML Metadata Interchange

# Einleitung

Mit dem Ziel 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 Standard der OMG. Über Klassen und Beziehungen können Objekte aus der realen Welt (von Systemen) abstrahiert und abgebildet werden. So kann die Komplexität komplexer Systeme 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 (MDSD) zur Erstellung von Software, 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]).

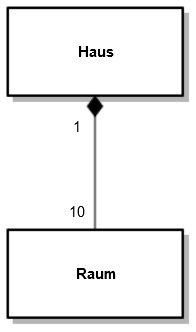
Es ist ein Mittel zur Implementierung von MDSD und MDE. Im Mittelpunkt steht nach Aufgabenstellung nur der Modelltyp der Klassendiagramme von UML. Eine Möglichkeit MDSD umzusetzen ist das Eclipse Modeling Framework (EMF). Es ist ein Java-Tool zum Generierung von Code, dass sich auf Klassendiagramme beschränkt. 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 (ReferecenceType) 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 1: 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.

# Hauptteil

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.

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

# 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>.

[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>.