Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Karsten Wilken

Charaktererstellung für reglementierte Rollenspiele
mit einer domänenspezifischen Sprache

Karsten Wilken

Charaktererstellung für reglementierte Rollenspiele mit einer domänenspezifischen Sprache

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Martin Hübner
Zweitgutachter : Lutz Behnke, MSc.

Abgegeben am 09. Dezember 2016

Karsten Wilken

Thema der Bachelorarbeit

Charaktererstellung für reglementierte Rollenspiele mit einer domänenspezifischen Sprache

Stichworte

Domain Specific Language, XText Framework, Rollenspielsysteme, Charaktererstellung

Kurzzusammenfassung

Diese Bachelorarbeit befasst sich mit den Darstellungsmöglichkeiten mittels einer domänenspezifischen Sprache die Charaktererstellung von einem schriftlichen in Prosa verfassten Rollenspiel-Regelwerk zu einem maschinenverständlichen Konstrukt zu verarbeiten. Unter Verwendung des XText-Frameworks werden grundlegende Konzepte erstellt und überprüft, ob sie geeignet sind zur Repräsentation der Regelwerke. Der Schwerpunkt liegt dabei auf der Überprüfung der Machbarkeit und eventueller Feststellung weiter benötigter Konzepte. Die Umsetzbarkeit wird prototypisch überprüft. Einführend werden die grundlegenden Themengebiete erläutert. Abschließend werden die Ergebnisse zusammengefasst.

Karsten Wilken

Title of the paper

Character creation for regulated role-playing systems with a domain specific language

Keywords

Domain Specific Language, XText Framework, Role-playing system, character creation

Abstract

This bachelor thesis deals with the presentation possibilities by means of a domain-specific language to transform the character creation from prose-written roll-play rules to a machine-understandable construct to process. Using the XText framework, basic concepts are created and checked to determine whether they are suitable for representing the rules. The focus is on the verification of the feasibility and possible determination of further required concepts. The feasibility is checked prototypically. The basic topics are explained in to require the needed knowledge to understand them. Finally, the results are summarized.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung

Seit über 30Jahren ist die Charaktererstellung essenzieller Bestandteil in Rollenspielen (Pen & Paper Rollenspiele). Sie repräsentieren den Charakter in der Spielwelt, in der sie mit anderen Spielern Geschichten und Abenteuer erleben können. Mit der Entwicklung der Computer und des Internets haben sich auch die technischen Hilfsmittel für Rollenspiele weiterentwickelt. Online-Rollenspiele stellen inzwischen eine eigene Kategorie im Bereich der Computerspiele dar. Beliebte Rollenspiele dienen oft als Vorlage für diese Spiele (z.B. "Dungeons and Dragons" für "Dungeons and Dragons Online" oder "Das schwarze Auge" für "Drakensang").
Sowohl "Pen & Paper" Spiele als auch Computerspiele werden durch in Prosa verfasste Regelwerke beschrieben, die die Umwelt und die Interaktionen beschreiben, die für die Spieler möglich sind. Die Regelwerke umfassen die gesamte Spielwelt und werden mit wachsendem Inhalt und Komplexität größer und aufwändiger.
Die Charaktererstellung ist oftmals der erste Kontakt eines neuen Spielers mit den Regeln der Spielwelt. Die Zusammenhänge, Bedingungen und Notwendigkeiten beim Erstellen eines Charakters werden in den Regelwerken mittels eigener Kapitel oder ganzen Heften aufgeführt. Es ist oftmals eine intensive Vorarbeit nötig um einen Charakter nach den Regeln zu erstellen. Ob jedoch alle Regeln und ihre Abhängigkeiten beachtet und angewendet wurden, kann nicht direkt überprüft werden.

In Computerspielen sind die Regeln der Spielwelt bekannt und überprüfbar. Es kann sichergestellt sein, dass die Charaktererstellung nach den vorgegebenen Regeln und ihren Abhängigkeiten richtig erfolgt.
Dank des Internets steigt die Verbreitung von Charaktergeneratoren für Rollenspieler, um die technischen Vorteile und erleichterten Zugangsmöglichkeiten nutzen zu können. Zur Erstellung dieser Generatoren ist nicht nur das Verständnis bzgl. der Regeln nötig sondern auch Programmierkenntnis. Die Nutzung der Charakter-Generatoren beschränkt sich oftmals auf ein spezielles Rollenspiel, für das es entworfen wurde.

Ein weiterer Aspekt ist die generelle Konvertierung der Regeln für Pen & Paper Rollenspiele in maschinenverständliche Formate (z.B. bei der Erstellung eines Computerspiel basierend auf einem Pen & Paper Rollenspiel). Die Barriere der Programmierkenntnisse verhindert, dass die Regelwerkersteller selbst ihre Regeln zur Verfügung stellen können.

1.2 Zielsetzung

Eine „Domain Specific Language“ (DSL) ist eine Beschreibungssprache. Sie stellt eine Abstraktionsebene zur technischen Umsetzung des Sachverhaltes dar, die gleichzeitig dafür zu sorgen hat den Inhalt klarer und einfacher verständlich darzustellen. Im Gegensatz zu universellen Programmiersprachen benötigt man weniger Einstiegswissen und keine konkreten Programmierkenntnisse.

Sie bietet den Experten ihrer Domäne (hier Regelwerk Erstellung/Bearbeitung) somit die Möglichkeit Regeln und Bedingungen anzugeben ohne tiefere Programmierkenntnisse besitzen zu müssen. Die Regeln bestehen hierbei aus schriftlich formulierten Regeln und nicht aus Regeln im Sinne des Fachbereiches künstliche Intelligenz.
Die DSL dient als Schnittstelle zur Darstellung der Regeln in einer Programmiersprache. Durch die Verwendung der DSL kann man den Arbeitsschritt der Eingabe der Regeln nicht nur anwenderfreundlicher gestalten, sondern erlangt auch eine weitere Abstraktionsebene. Durch die klare Trennung kann das Datenmodell für den Computer in einer beliebigen Sprache entwickelt werden und gegeben falls auch ausgetauscht werden. Um diesen Wechsel zu vollziehen, ist nur eine Anpassung in der Konvertierung (Parser) der DSL erforderlich. All dies kann im Hintergrund geschehen ohne dass der Anwender der DSL es bemerkt.

Zentrale Anforderungen an eine DSL sind dabei: gute Lesbarkeit, einfache Erlernbarkeit und Hilfestellung (Dokumentation, Validierung, Vervollständigung).

Sowohl bei der Erstellung der Charaktere für Pen & Paper Rollenspiele als auch für die Umsetzung eines schriftlichen Regelwerks der Charaktererstellung zu einem maschinenverständlichen Regelwerk wird der Einsatz dieser DSL den Einstieg, die Pflege und die allgemeinere Machbarkeit vereinfachen. Für den Anwender werden keine Programmierkenntnisse vorausgesetzt.

Anhand drei repräsentativer Regelwerke soll gezeigt werden, wie unterschiedliche Regelwerke zur Charaktererstellung mit dieser DSL umgesetzt werden können.
Dieses Projekt beschränkt sich konkret auf die Umsetzung der Regeln der Charaktererstellung um den Aufwand und Umfang zu begrenzen. Komplette Regelwerke können einen Umfang von mehreren Büchern beinhalten, was den Rahmen der Bachelor-Arbeit sprengen würde.

Im Rahmen des Projektes soll eine DSL (Name der Sprache: „Create“) entworfen werden, mit der die Erstellung eines Charakters basierend auf einem schriftlichen Regelwerk ermöglicht werden kann. Es soll überprüft werden, ob die Überführung der Regelwerke zur Charaktererstellung mit einer DSL machbar sind.

Hierbei werden die drei repräsentativen Regelwerke nur in Modelle überführt. Der Fokus der Arbeit liegt auf der Umsetzung der Regelwerke zu maschinenverständlichen Verarbeitung. Dieser Schritt wird mit der Umsetzung der Modelle aufgezeigt. Die Erstellung einer Instanz fällt in den handwerklichen Aspekt und wird für die meisten Modelle ähnlich aber jedoch nicht identisch sein. Wichtige Aspekte für die Erstellung einer Instanz werden prototypisch dargestellt um die Konzepte zu veranschaulichen. Die Umsetzung aller Modelle in konkrete Instanzen zur Veranschaulichung erfordert Aufwand und Zeit in einem Bereich, der nicht im Fokus und Ziel dieser Projektarbeit liegt.

Die automatische Generierung der Anzeige und Darstellung des erzeugten Charakters ist nicht Bestandteil des Projekts (3D-Modell, automatische Erzeugung von html Seiten oder z.B. Aussehen anhand der umgesetzten Eigenschaftswerte des Charakters). Die angebotene Ausgabe über verschiedene Medien (z.B. js oder html) sollen lediglich mögliche Verwendungsmöglichkeiten aufführen.
Nachfolgend eine Übersicht über die grobe Aufteilung und die unterschiedlichen Bereiche:

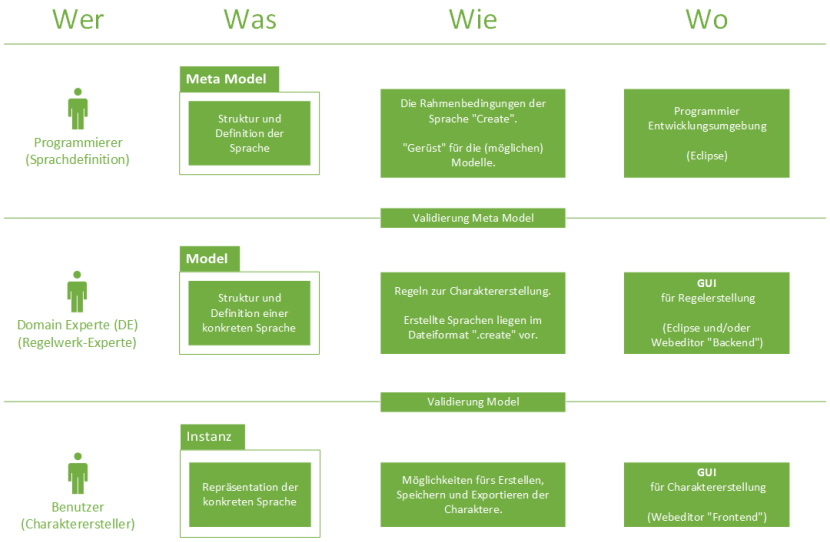

Abbildung 1: 1.1 Veranschaulichung

Die Grafik soll Bereiche abgrenzen und die Zugehörigkeiten veranschaulichen. Wer ist beteiligt und zu welchem Bereich gehören die angesprochenen Elemente. Die Grafik bzw. ihre einzelnen Bereiche werden im Zuge der Arbeit detaillierter beschrieben.

1.3 Aufbau der Bachelorarbeit

Im ersten Kapitel (Einleitung) wird der Rahmen der Arbeit erläutert. Es werden die Ziele beschrieben und die Grenzen des Projekts aufgezeigt. Das zweite Kapitel (Anforderungen) bietet den ersten Einblick in die Welt der "Domain Specific Languages" (DSL) und zeigt eine Einordnung auf, was eine DSL ausmacht und welche Anforderungen an sie gestellt werden.
Mit dem dritten Kapitel (Stand der Kunst) wird das grundlegende Verständnis vermittelt um die Mechanismen und Arbeitsweisen bzgl. der DSL's und die Verwendung im Rahmen der Projektarbeit zu verstehen und einordnen zu können.
Die nötigen Schritte um von einem Regelwerk auf eine Beschreibung in meiner DSL Sprache „Create" zu gelangen, werden im Kapitel 4 (Der Weg zur Sprache „Create") erörtert.
Im fünften Kapitel (Detailbeschreibung) wird die Realisierung der Projektarbeit dargestellt und die verwendeten Komponenten beschrieben.
Abgeschlossen wird die Arbeit mit dem sechsten Kapitel (Zusammenfassung) über die erlangten Erkenntnisse und einem Ausblick wie die Ergebnisse verwendet werden können und welche Möglichkeiten die Umsetzung mittels einer DSL bieten.

1.4 Abgrenzung des Projektumfangs

Der Fokus der Arbeit beschränkt sich auf die Überprüfung der Machbarkeit der Übersetzung eines Regelwerks zur Charaktererstellung in eine maschinenverständliche Form. Die verwendeten und im Verlauf erklärten Mechanismen werden nur prototypisch umgesetzt um die Verwendung zu veranschaulichen. Die eingesetzte Umwicklungsumgebung bietet sehr umfangreiche und vielseitige Entwicklungsmöglichkeiten, deren komplette Umsetzung für die angestrebte Sprache „Create" im Rahmen des Projektes zu umfangreich und weitreichend wäre. Daraus folgt auch, dass die Codegenerierung für die konkreten Regelwerke (z.B. „Das schwarze Auge") nicht umgesetzt wird. Anhand von beispielhaften Codefragmenten wird jedoch das Schema aufgezeigt und dargelegt, dass dieser Schritt möglich und machbar ist.

2 Anforderungen

2.1 Einführende Erläuterung

Eine domänenspezifische Sprache (kurz DSL – „Domain Specific Language“)[1] beschreibt eine formale Sprache. Im Gegensatz zu den uns bekannten konkreten bzw. gesprochenen Sprachen dienen formale Sprachen dazu abstrakte und mathematische Zusammenhänge zu beschreiben. Sie werden unter anderem in der Logik (z.B. Beschreibung des logischen Schlussfolgerns), in der theoretischen Informatik (z.B. formale Darstellung einer Programmiersprache) und sogar in der Linguistik (z.B. Beschreibung des Satzaufbaus) verwendet.[2]

Eine DSL zeichnet sich in erster Linie dadurch aus, dass sie auf ein spezielles Problem („domain specific“) ausgelegt ist und nur Lösungen für dieses Problem anbietet. Dadurch unterscheidet sie sich im Kontext der Programmierung von universellen Programmiersprachen („General Purpose Language“). Diese zielen darauf ab Lösungsmöglichkeiten für viele Problemstellungen anzubieten. Die Programmiersprache Java bietet unter anderem z.B. die Möglichkeiten grafische Oberflächen zu programmieren oder komplette Webanwendungen zu erstellen.
Da sich eine DSL nur auf ein konkretes Problem konzentriert, bietet sie einige Vorteile gegenüber einer universellen Programmiersprache. Diese setzen ein umfangreiches Wissen voraus über die Funktionsweise und die konkrete Struktur (Syntax) der Sprache. Damit das Ziel erreicht werden kann viele Problemstellungen zu bewältigen, ist das Umfang an Wörtern und Kombinationen in den universellen Programmiersprachen komplexer, unüberschaubarer und schwerer zu verstehen.

[1] Quelle Wikipedia: Domänenspezifische Sprachen [5]
[2] Quelle Wikipedia: Formale Sprachen [4]

Ein Grund zur Verwendung von DSLs ist der im Gegensatz überschaubare Rahmen an Komplexität. Dieser ermöglicht es auch schwierigere Aspekte der Entwicklung und Darstellung zu vereinfachen um z.B. die dahinterliegende Programmierlogik vor dem Benutzer zu verbergen. Diese Kapitel geht unter anderem darauf ein welche Anforderungen an eine DSL gestellt werden und wie sie sich dadurch von universellen Programmiersprachen abgrenzen.

2.2 DSL

Eine domänenspezifische Sprache ist eine Computer-Programmierungssprache mit begrenzten Ausdrucksmöglichkeiten und ausgerichtet auf ein konkretes Problem.[3]

Wie in vielen Bereichen der Informatik gibt es mehrere genauere Definitionen zu domänenspezifischen Sprachen und welche konkreten Sprachen als DSL bezeichnet werden können bzw. dürfen. Nach Fowler werden vier zentrale Merkmale zur Definition einer DSL angeführt:[4]

Computer-Programmiersprache:

Eine DSL wird von Menschen verwendet um dem Computer Anweisungen zu geben. Die Struktur soll für den Menschen einfach verständlich sein, aber weiterhin für den Computer ausführbar.
Beispiel: „Roboter, bewege dich einen Schritt vor“. Einfache Anweisung des Menschen wird von Computer verstanden und umgesetzt.

Sprachnatur:

Eine DSL ist eine Programmiersprache und die Möglichkeiten sich auszudrücken dürfen sich nicht nur auf die Wörtern begrenzen sondern auch von Kombinationsmöglichkeiten miteinander.
Beispiel: „Roboter, wende dich nach links und danach gehe 2 Schritte vor.“ Aneinanderreihung von Anweisungen kann ausgeführt werden.

Fokus auf Domäne:

Eine begrenzte Sprache ist nur dann sinnvoll, wenn sie einen klaren Fokus auf eine spezielle Aufgabenstellung(Domäne) hat.
Beispiel: „Stelle grundlegende Bewegungsmöglichkeiten eines Roboters dar.“

Begrenzte Ausdrucksfähigkeit:

Eine universelle Programmiersprache besitzt diverse Möglichkeiten und macht sie somit schwer lern- und verstehbar. Eine DSL stellt nur so viel Unterstützung zur Verfügung, um das Problem zu beschreiben und zu lösen.

---

[3] Quelle Buch: Domain-Specific Languages, M.Fowler [1]
[4] Quelle Buch: Domain-Specific Languages, M.Fowler [1, pp. 27-28]

Es ist nicht möglich ein komplettes Softwaresystem damit zu realisieren. Es wird lediglich für einen speziellen Teil des Systems verwendet.
Beispiel: „Roboter, lerne Salsa und fliege zum Mond“. Der Roboter könnte die Aufgaben wahrscheinlich erfüllen, allerdings nicht mit einer DSL, die sich mit dem Teilsystem „Roboter-Bewegung“ alleine befasst. Ein komplettes Softwaresystem ist dazu nötig. Die einzelnen Aufgaben könnten dann durch ihre spezielle Problemstellung wiederum von einer DSL beschrieben werden. (z.B. Salsa-DSL)

2.3 Rollenspiele

Der Bereich der Rollenspiele umfasst inzwischen eine riesige Bandbreite an Variationen und Spielsystemen. Für dieses Projekt steht der Bereich der Pen-&-Paper-Rollenspiele[5] im Fokus. Hierbei handelt es sich um Spielsysteme, bei denen die Spieler in fiktive Rollen schlüpfen und mit Hilfe von Stift und Papier (Pen & Paper) gemeinsam eine durch Erzählung ein Abenteuer erleben. Genau wie in fast allen anderen Aspekten des sozialen Miteinanders basiert auch die erdachte und erlebte Fantasiewelt auf einem Regelwerk, an das sich die Beteiligten halten sollten und ihnen Orientierung bietet.

Grundsätzlich unterscheidet man Rollenspiele in zwei verschiedene Kategorien von Rollenspielarten[6]. Freie und spontane Rollenspiele unterliegen keinen strikten bzw. festen Regeln. Sie werden unter den Teilnehmern kommuniziert bzw. abgesprochen. Weithin bekannte Beispiele dafür sind: „Räuber und Gendarm“ oder „Cowboy und Indianer“.
Die andere Kategorie wird als reglementierte Rollenspiele bezeichnet und umfasst auch den Bereich der Pen-&Paper-Rollenspiele. Weitere Beispiele, die in diese Kategorie gehören: Live-Rollenspiele, bei denen die Abenteuer im realen Leben mit z.B. Kostümen nachgestellt und erlebt werden und Computer-Rollenspiele, bei denen am Computer eine virtuelle Umgebung zur Verfügung gestellt wird, in der die Akteure agieren können.

Die Grenzen von freien Rollenspielen liegen meist in der Vorstellungskraft der Beteiligten. Die Möglichkeiten lassen sich kaum in endlicher Form darstellen. Sie bieten daher keine geeignete Grundlage um die vereinbarten Regeln in maschinenverständliche Rahmen zu setzen. Daher schließt das Projekt diesen Bereich der Rollenspiele komplett aus.

Reglementierte Rollenspiele setzen Regelwerke voraus um die Rahmenbedingungen der Fantasiewelt vorzugeben. Es wäre somit möglich für alle diese Arten Regelwerke aufzusetzen und zu versuchen diese mittels einer DSL in maschinenverständliche Form umzuwandeln.
Ein solches Szenario macht jedoch nicht für alle Rollenspiel-Arten Sinn. Computer-Rollenspiele sind z.B. bereits in einer Computer verständlichen Form vorhanden.

[5] Quelle Wikipedia: Pen-&-Paper Rollenspiele [6]
[6] Quelle Wikipedia: Rollenspiele [7]

Die Komplexität der Fantasiewelten kann deutlich variieren und der Umfang hängt oftmals auch von den Zielen der Rollenspiele ab. Die Varianten reichen von detaillierten Attributen und Rollen mit Abhängigkeiten und Bedingungen bis hin zu simplen Beschreibungen in Textform.
Regelwerke für Rollenspiele umfassen oftmals mehrere Bände um die Zusammenhänge ganzheitlich zu erfassen.
Es ist nicht möglich im Rahmen der Projektarbeit ein komplettes Regelwerk mit all seinen Beziehungen und Abhängigkeiten abzubilden.
Daher wird nur ein kleinerer Teil der Rollenspiele betrachtet und versucht umzusetzen.
Die Charaktererstellung ist ein Teil des Regelwerkes und bietet einen überschaubaren Rahmen an Regeln um zu überprüfen, ob es möglich ist, diese Regel Konstrukte in Computer verständliche Form umzusetzen.

2.4 Verwendung

Die Charaktererstellung ist der Einstieg in jede Begegnung mit einem Rollenspiel. Ein Szenario für die Verwendung einer DSL für die Charaktererstellung ist die Umsetzung bereits vorhandener Rollenspiele zu einer Computerspiel Variante oder die Bereitstellung von elektronischen Hilfsmitteln zur Charaktererstellung bzw. zum regelkonformen Erstellen eines Charakters.
Die DSL dient hierbei nicht nur zur Überprüfung der Korrektheit der Regeln, sondern ist als zusätzliche Abstraktion der Regeln auch ein Mittel zur Vereinfachung der Präsentation und Abkapselung von Komplexität vor dem Rollenspiel-Einsteiger.

3 Stand der Kunst

3.1 Einführende Erläuterung

Dieses Kapitel erläutert die grundlegenden Details zu den wichtigen Einzelbereichen des Projektes. Es vermittelt den notwendigen Wissensstand um die Umsetzung des Projektes verstehen zu können und es wird auf einige der gängigen Mittel und Methoden in dem jeweiligen Gebiet verwiesen.

3.2 DSL

Domänenspezifische Sprachen stellen eine Abstraktionsebene dar, um Sachverhalte vereinfacht beschreiben zu können. Sie werden für eine konkrete Problemstellung(Domäne) eingesetzt. Ein Benutzer (Domänenexperte) kann mit Hilfe der DSL Sachverhalte aus seinem Wissensgebiet darstellen ohne konkrete Kenntnisse über die Umsetzung auf maschinenverständlicher Ebene haben zu müssen.

3.2.1 Warum eine DSL verwendet wird

Universelle Programmiersprachen besitzen die Möglichkeiten die Sachverhalte darzustellen, die eine DSL bietet. Es bestehen also bereits vorhandene Möglichkeiten einen Lösungsansatz für die Problemstellung zu finden. Warum wird also eine eigene DSL für so eine Problemstellung erstellt?

Zugang ohne Programmierkenntnisse & Erlernbarkeit

Das Erlernen einer Programmiersprache ist langwierig und beinhaltet einen langen Lernprozess. Um die verwendeten Konzepte und Vorgehensweisen dieser Sprachen zu erlernen, benötigt es Zeit und Übung um alle Aspekte beherrschen zu können. Hinzu kommt, dass der Domänenexperte am Besten in seinem Fachgebiet eingesetzt ist.

DSLs bieten die Möglichkeit sein Fachwissen in einer Form auszudrücken, die für einen Computer verständlich ist. Ein zentrales Merkmal einer DSL ist die einfache und simplere Struktur um Sachverhalte der jeweiligen Domäne auszudrücken.

Erhöhte Lesbarkeit & kein Programmiercode

Die Struktur und Syntax einer DSL können auf das jeweilige Spezialgebiet angepasst werden. Es kann auf die Terminologie des Sachgebietes eingegangen werden und somit die Beschreibung vereinfacht werden.
Moderne Programmiersprachen bieten bereits fast lesbare Verkettungen von Methoden und Variablen, so dass sie ansatzweise durchgehend lesbar sind und sich ihr Sinn daraus ableiten kann. DSLs bieten eine weitere Möglichkeit dieses Ziel weiter zu fokussieren. In ihren Sprachedefinitionen wird auf die Minimierung der Steuerzeichen geachtet, die die Sprache steuern.

3.2.2 Die Arten von DSLs

Es bestehen zwei unterschiedliche Ansätze um eine neue domänenspezifische Sprache zu entwerfen:

Interne DSL

Mit diesem Begriff werden interne oder eingebettete Sprachen beschrieben, die sich bereits vorhandene universelle Programmiersprachen (z.B. Java, C/C++, Ruby) oder Beschreibungssprachen (z.B. UML) zu Nutze machen und die dort vorhanden Elemente und Tools nutzen. Dadurch wird der Entwicklungsaufwand reduziert. Die Definition der Sprachelemente, die die DSL verwendet, begrenzt die Konstrukte der verwendeten Muttersprache. Es wird nur eine Teilmenge verwendet. Eine interne DSL stellt somit eine echte Teilmenge der Muttersprache dar.
Ein weiterer Effekt ist, dass die Nähe zur ursprünglichen Sprache den Lerneinstieg für die DSL vereinfachen kann, sofern man mit der Sprache bereits vertraut ist.

Externe DSL

Sie orientieren sich nicht an bereits vorhandenen Sprachen. Dadurch erhalten sie die Freiheit nicht an bestehende Konzepte gebunden zu sein. In einer externen DSL wird alles eigenständig definiert. Einerseits bringt dies den Vorteil, dass man nicht auf die Mechanismen einer Muttersprache angewiesen ist und andererseits muss der Entwickler der externen DSL alle Komponenten der Verarbeitung erstellen und überprüfen. Für moderne Programmiersprachen wird heutzutage eine weite Bandbreite an Werkzeugen zur Verfügung gestellt, die für die externe DSL neu erstellt werden müssten wie z.B. Editoren, Debugger oder Testumgebungen.

3.2.3 Welche Grenzen und Probleme es gibt

DSLs beschreiben nur eine konkrete Problemstellung. Es muss daher sichergestellt sein, dass der Aufwand der Erstellung und des späteren Erlernens der neuen Sprache ein gerechtfertigter Nutzen gegenüber steht (z.B. in Form von Verwendungshäufigkeit, Ablauferleichterung oder vereinfachter digitaler Erfassung von Ergebnissen/Fakten).

Sobald eine DSL verwendet und weiter entwickelt wird, erweitern sich häufig auch die Anwendungsfälle, die mit Hilfe der DSL abgedeckt werden sollen. Hierbei besteht die Gefahr, dass sich durch die Erweiterung eine Entwicklung zu einer universellen Programmiersprache abzeichnet und somit der zentrale Fokus auf eine Problemstellung verloren geht. Eine DSL soll nur ein Modell ihres Aufgabenbereiches darstellen und nur diese Sachverhalte aus der Domäne darstellen können. Das beinhaltet ebenfalls, dass sichergestellt ist, dass ungewünschte oder unmögliche Konstrukte nicht dargestellt werden können. Jede Erweiterung beinhaltet höchstwahrscheinlich Verbindungen und Abhängigkeiten zu den bereits beschriebenen Sachverhalten. Dies wiederum erhöht die Vielschichtigkeit und Sprachstruktur der DSL. Auch hier ist ein Bruch zu den Zielen einer DSL zu erwarten: Einfachheit & geringe Komplexität.[7]

Eine DSL stellt eine neue Computersprache dar. Sie enthält neue Konstrukte und neue Sprachelemente. Es gibt eigene Definitionen der Syntax und der Semantik für das beschriebene Problem. Es müssen Mechanismen entwickelt werden, die die neuen Elemente maschinenverständlich verstehen, umsetzen und überprüfen können. Wird die Sprache verändert, führt dies zu vielen Punkten, die in der dahinter versteckten Verarbeitung der Sprache angepasst und überprüft werden müssen. Sowohl bei internen als auch bei externen Sprachen sind die Überprüfungsmöglichkeiten auf die momentane Struktur der Sprache ausgelegt und die eigens erstellten Konzepte müssten nicht nur in der Sprachdefinition angepasst werden. Selbst kleine Fehler oder übersehen Elemente bzw. Beziehungen zwischen Elementen können weitreichende Auswirkungen haben und Fehler verursachen.

3.3 Rollenspiele und Charaktere

Um die Regelelemente der Charaktererstellung für den Computer verständlich umzusetzen, müssen die grundlegenden Gemeinsamkeiten und anzunehmenden Beschreibungsformen erläutert werden. Im Folgenden soll darauf eingegangen werden, welche Strukturen allgemein verwendet werden um Charaktere in Rollenspielen zu beschreiben. Dabei werden nicht nur feste Werte und Beschreibungen betrachtet, sondern auch Auswahlmöglichkeiten und mögliche dynamischen Ereignisse, die Einfluss beim Erstellen des Charakters haben können.

[7] Quelle Buch: Domain-Specific Languages, M.Fowler [1]

3.3.1 Werte, Beschreibungen und Eigenschaften

In der Fantasiewelt, in der die zukünftigen Charaktere existieren, benötigen sie nicht nur beschreibende Eigenschaften, die den Mitspielern und auch sich selber eine bildliche Idee des Helden (Charakters) vermitteln, sondern auch den Umgang mit der Spielwelt darstellen und klären. Die unten angeführte Kategorisierung ist eine weitverbreitete Form der Unterteilung. Die Konzepte sind in ähnlichen und abgewandelten Formen auch in den drei verwendeten Beispiel-Regelwerken vorhanden.

Beschreibende Eigenschaften

Sie können die sichtbaren Merkmale eines Charakters beschreiben (z.B. Haare und Augenfarbe). Entscheidend ist, dass ihnen eine textuelle Beschreibung ausreicht. Es muss kein Wert zugewiesen werden, der die Qualität oder Quantität der Eigenschaft beschreibt. Es können ebenfalls Charaktereigenschaften beschrieben werden, die keinen direkten Einfluss auf die Spielregeln haben, sondern eine soziale Hilfestellung für Mitspieler bieten sollen. Dies wäre z.B. der Fall in Rollenspielsystemen, bei denen die Regelwerke nur grobe Rahmenbedingungen bereitstellen sollen und die Interaktion der Charaktere untereinander auf sozialer Basis stattfinden soll. Ein Beispiel für solch eine Darstellung der Regeln wird im später betrachteten Regelwerk von „FreeFate“[8] erläutert. Beispielhaft seien hier Eigenschaften wie „immer mitten drin“ oder „engstirnig“ genannt.

Wertende und Fähigkeitseigenschaften

Zur Darstellung wie gut jemand etwas beherrscht oder welchen Bonus er bei einer Ausführung der Eigenschaft oder Fertigkeit vorgewiesen werden kann, gibt es in den meisten reglementierten Rollenspielen Zahlensysteme zur Bewertung von Eigenschaften. Diese haben meist einen möglichen Wertebereich (z.B. [0-20]). D.h. aufgrund unterschiedlichster Begabungen in dem beschriebenen Bereich kann der Eigenschaft ein Wert zugewiesen werden und unter anderem auch verändert werden (z.B. durch Training oder Ausüben).

Üblicherweise werden diese Werte im konkreten Spiel durch ein Würfelereignis überprüft. (z.B. muss die gewürfelte Augenanzahl über oder unter dem zugewiesenen Wert sein um einen Erfolg oder Misserfolg auszulösen)

Zahlenbelegte Werte sind ein wichtiger Bestandteil im Rollenspielsystem. Einerseits sind sie die Rahmenbedingungen für ein gemeinsames und abgeglichenes Verständnis der Spielwelt und der geltenden Regeln und andererseits eine notwendige Voraussetzung um die Mechanismen in ein für den Computer verständliches Format umzuwandeln und benutzen zu können.

Erwähnenswert ist auch, dass es Eigenschaften geben kann, die sich und ihren Wert aus anderen Eigenschaften zusammensetzen. (z.B. Angriffswert = Körperkraft + Gewandtheit)

[8] Quelle Webseite: FreeFate.de [8]

Klassen und Rollen

Einflüsse, die nicht nur auf einen einzelnen Charakter zutreffen oder größere Konzepte kombinieren, werden in Rollenspielsystemen häufig in Klassen und Rollen zusammengefasst. In den unterschiedlichen Regelsystemen kann es dazu kommen, dass die beiden Begriffe synonym oder ähnlich verwendet werden.
Sie beschreiben kompliziertere Aspekte, die oft nicht durch simple Eigenschaften beschrieben werden können (z.B. Rasse, Beruf oder Werte, die aus kombinierten Eigenschaften entstehen).
Ein Merkmal dieser Gruppierung ist, dass sie keine eigene oder einzelne Eigenschaft haben, die sie beschreiben. Vielmehr haben sie Einfluss auf unterschiedliche bzw. mehrere Eigenschaften. Als Beispiel sei hier der Beruf des Schmiedes angeführt. Er hat die Vorrausetzung, dass der Charakter einen hohen Wert in der Eigenschaft Körperkraft hat und z.B. nicht hitzeempfindlich ist. Die Auswirkung der Rolle besteht darin, dass von ihm erstellte Gegenstände hochwertigere Eigenschaften besitzen und er z.B. mit einem Hammer als Waffe höhere Angriffswerte hat.

3.3.2 Wertzuweisungen

Die meisten reglementierten Rollenspiele haben die gleichen Mechanismen um Werte zu zuweisen und zu überprüfen. Es kann nicht ausgeschlossen werden, dass noch weitere Mechanismen existieren, allerdings stellen die hier aufgelisteten Methoden die bekannten und meist verwendeten Mechanismen dar. Diese Aussage beruht auf Rücksprache mit zweier Quellen, die sich intensiv mit unterschiedlichsten Regelwerken auseinander gesetzt haben.

1. Emailgespräch mit Thomas Echelmeyer, Mitarbeiter des Unternehmens Ulisses Spiele[9], sinngemäß nach eigenen Angaben sammelt er „Pen&Paper-Rollenspiele und über 500 verschiedene Grundregelwerke besitzt“.
2. Lutz Behnke, Zweitbetreuer dieser Bachelorarbeit und passionierter Rollenspieler in diversen Formaten.

Zuweisung

Der Wert wird fest ein Zahlenwert zugewiesen.

Zufall

Der Zufall wird in Pen-&Paper-Rollenspielen meist in Form eines Würfels dargestellt. Um eine bestimmte Kombination von möglichen Zahlenwerten zu ermöglichen gibt es Würfel mit unterschiedlichen Augenzahlen (z.B. 6er, 8er oder 20iger) und gegebenenfalls werden mehrere Würfe miteinander kombiniert.
Um z.B. einen Wert im Bereich von 8 bis 13 darzustellen würde ein sechsseitiger Würfel verwendet werden und dem Wurfergebnis 7 hinzu addiert.

[9] Quelle Webseite: Ulisses-Spiele.de [9]

Wertkombination

In Fällen, in denen die Eigenschaft einen Wert beschreibt, der sich aus anderen Eigenschaften zusammengesetzt wird, kann eine arithmetische Kombination verwendet werden, die sich dadurch auch mit veränderten Einzeleigenschaften verändert.
Beispielhaft könnte eine Eigenschaft Angriffswert aus dem Mittelwert von Körperkraft, Mut und Geschicklichkeit berechnet werden.

3.3.3 Listen und Ereignisse

Listen sind eine Möglichkeit um einen Werte in einem bekannten und begrenzten möglichen Rahmen zu beschreiben. Bei Listen handelt es sich meist um Eigenschaften, denen zwar ein Zahlenwert zugewiesen werden kann, aber gleichzeitig eine beschreibende Bedeutung zur Verfügung gestellt werden soll. Ein Beruf könnte dadurch z.B. in eine Bewertung bekommen. (unbeholfen, fortgeschritten, meisterhaft)
Ebenso kann über eine Liste über Definierung von Wertebereichen eine Einstufung vornehmen. In einem Wertebereich von 1 bis 20: 1 bis 5 schlecht, 6-10 akzeptable, 11-15 gut und 16-20 meisterlich.

In den Bereich der Ereignisse fallen die bereits erwähnten Würfelereignisse. Diese haben nicht nur Auswirkungen wie z.B. bei der Vergabe der Werte sondern können auch gleichzeitig Auslöser sein. Dies tritt meist direkt im Spielverlauf auf.
Sie können auch mit Listen kombiniert werden. Ein Wurfergebnis kann anhand der unterschiedlichen Listeneinträge andere Ereignisse auslösen. Ein beliebtes Beispiel dafür ist eine „Patzertabelle“. Diese bestimmt das Ausmaß des Scheiterns des Heldens und die damit verbundenen negativen Auswirkungen wie z.B. ausrutschen oder im schlimmsten Fall schwere körperliche Schäden bis hin zum Todesfall.

3.4 Compiler

Bei der Verwendung einer DSL wird in der Regel ein Compiler eingesetzt als Schnittstelle um für den Anwender (Domänenexperten) verständlich ausgedrückten Sachverhalt in einen maschinenverständlichen Datensatz umzuwandeln. In den folgenden Abschnitten wird ein grober Überblick über das Thema Compiler vermittelt. Der Bereich des Compilerbaus ist ein eigenständiger Bereich der Informatik und bietet vielfältige Themengebiete, Vertiefungsmöglichkeiten und Optimierungsansätze.[10] [11]

[10] Quelle Buch: Compilers: Principles, A.Aho, M.Lam, R.Sethi und H.Ullman [10]
[11] Quelle Buch: Domain-Specific Languages, M.Fowler [1]

3.4.1 Was ist ein Compiler

Ein Compiler ist ein Computerprogramm, das darauf ausgelegt ist, einen Quellcode in eine angegebene Zielsprache zu übersetzen. In unserem Fall den Code der Regelwerke, die wir in der DSL beschreiben.
Die Übersetzung findet dabei in mehreren Schritten statt. Zu seinen Aufgaben gehören das Überprüfen der Richtigkeit des zu übersetzenden Codes und naheliegender Weise die Überführung der Anweisungen in Befehle, die die Zielsprache ausführen kann.
Die meisten Hochprogrammiersprachen setzen einen Compiler voraus, der den erstellten Code zu ausführbaren Maschinencode umwandelt.

3.4.2 Aufbau und Funktionsweise

Ein Compiler ist in verschiedene Bereiche eingeteilt, die jeweils ein abgeschlossenes Aufgabengebiet haben.
Im ersten Schritt erfolgt die Überprüfung des Quellcodes. Im Fokus stehen hierbei die richtige Reihenfolge der Zeichen und Steuerungselemente. Entscheidend ist nicht nur die Reihenfolge sondern auch die Absicherung, dass alle Konstrukte so beendet werden wie es vorgesehen ist. Die Struktur der möglichen Anweisungen folgt dabei einer vorher definierten Grammatik.[12] Äquivalent zu den uns bekannten Grammatiken, die die gesprochene Sprache definieren und dadurch eine Satzstruktur definiert und erkannt werden kann, kann der Compiler sich an der Grammatik der jeweiligen Sprache orientieren.
Um dieses zu gewährleisten muss der Compiler Strategien besitzen, um längere Zeichenketten zu analysieren und sich merken können, welche Ausdrücke zusammengehören. Unter anderem gehören dazu diverse „Look-Ahead“ Strategien, mit denen der Compiler die Folgezeichen analysieren kann und so zusammenhängende und bekannte Strukturen erkennt. Hierbei gibt es auch Fälle zu beachten, die der Compiler schwer oder gar nicht selber entscheiden oder feststellen kann. Bekannte Beispiele dafür sind z.B. die Linksrekursion und Schleifenbildung. Das bedeutet, dass aufgrund gleicher Elemente in der Sprache der Compiler nicht entscheiden kann welche Elemente zusammengehören bzw. welche Elemente als nächstes auszuwählen sind. Viele moderne Compiler verfügen über die Möglichkeiten Schleifen zu entdecken. In einigen Situationen ist jedoch der Entwickler gefordert um z.B. das Auftreten einer Linksrekursion zu unterbinden.
Der Quellcode wird in einzelne Elemente zerlegt (sogenannte Tokens). Hier wird z.B. unterschieden zwischen Schlüsselwörtern der Sprache, Bezeichnern und Zahlen.
Entsprechend dem Aufbau der Sprache wird, sofern keine Fehler auftreten, ein Syntaxbaum (AST – Abstract Syntax Tree) erstellt, der die Zusammenhänge und Beziehungen der einzelnen Elemente wiederspiegelt.

[12] Quelle Wikipedia: Formale Grammatik [11]

Der Syntaxbaum stellt eine Form des Zwischencodes zwischen den beiden Sprachen dar. Moderne Compiler führen auf diesen Zwischencode noch zusätzliche Schritte wie z.B. Codeoptimierung oder Anpassungen auf unterschiedliche Plattformen durch.
Im letzten Schritt wird aus dem Syntaxbaum die Codegenerierung für die Zielsprache ausgeführt.

3.5 Language Workbench

Der Begriff wurde von Martin Fowler geprägt und wurde bereits 2010 in seinem Buch zum Bau von domänenspezifischen Sprachen als zukünftig relevante Methode zum DSL-Bau eingestuft.[13]

Sie beschreiben die Entwicklung von DSLs innerhalb bereits bestehender Entwicklungsumgebungen unter Verwendung der dort vorhandenen Werkzeuge. (z.B. integriert in Entwicklungsumgebungen wie Eclipse[14] oder IntelliJ[15])
Vorausgesetzt werden Konzepte zur Beschreibung von Metamodellen (Modelle zum Beschreiben und Definieren von Sprachen), Bearbeitungsumgebungen für DSLs (Editoren) und Möglichkeiten der Codeerzeugung oder Interpretation.[16]

Es werden somit alle Bequemlichkeiten mit zur Verfügung gestellt, die man in der Programmierung universeller Sprachen kennt und zu schätzen weiß.

Im Laufe der letzten Jahre wurden ganze Frameworks zum Bau von DSLs in diesem Bereich entwickelt. Ein Framework ist eine Zusammenstellung von Softwaretechnik, die ein Rahmenkonstrukt bereit stellt um ein konkretes Softwareprogramm zu entwickeln. Ziel ist es diverse wiederkehrende und notwendige Schritte abzunehmen, so dass sich der Entwickler auf die konkrete Problemstellung konzentrieren kann. Der Entwickler muss sich lediglich an gewisse Konventionen und Konfigurationen halten, die durch das Framework vorgegeben sind. Dafür stehen im oftmals vorgefertigte Architekturstrukturen zur Verfügung und er kann auf vorbereitete Entwurfsmuster zurückgreifen.

[13] Quelle Buch: Domain-Specific Languages, M.Fowler [1]
[14] Quelle Wikipedia: Eclipse (Software) [14]
[15] Quelle Wikipedia: IntelliJ IDEA [13]
[16] Quelle Webseite: Martin Fowler - Language Workbench [12]

4 Der Weg zur Sprache „Create“

Dieses Kapitel beschreibt den Entwicklungsprozess der Sprache „Create“.
Der Name ist ein humorvolles Akronym für: „**C**reate **R**eally **E**mbarrassing **A**nd **T**alentless **E**ntities“
Im Abschnitt „3.3 Rollenspiele und Charaktere“ wurden die Merkmale eines Charakters generell dargestellt. In den folgenden Abschnitten wird auf den Entwicklungsprozess zur DSL eingegangen.

4.1 Das Erschaffen der Sprache (Metamodel)

Es muss grundlegend geklärt werden, ob die verwendeten Konzepte der Charakterstarstellung in einer allgemeingültigen Form repräsentiert werden können, die möglichst alle Aspekte umfasst. Es ist unmöglich die Machbarkeit aller möglichen Formen von Regelwerken der Charakterstellung zu überprüfen. Die Gültigkeit der hier aufgeführten Konzepte wird anhand dreier Beispielregelwerke überprüft. Bei der Auswahl der Regelwerke wurde darauf geachtet, dass sie einerseits zu den größeren Vertretern der Spielsysteme gehören und ihre Regelsysteme auf einer Basis beruhen, die in vielen weiteren Systemen in Abwandlungen angewendet werden.
„Das schwarze Auge“[17] repräsentiert das größte und bekannteste deutschsprachige Regelsystem. International ist „Dungeons & Dragons“[18] sehr weit verbreitet. Es gibt laut Verlagsinformationen[19] keine offiziellen Zahlen oder Statistiken, die den Verbreitungsgrad der Regelwerke beurteilen oder anhand von Verkaufszahlen belegen könnten. Solche Unterlagen werden nicht von den Verlagen veröffentlicht. Im Internet werden oftmals nur subjektive Listen von Privatpersonen veröffentlicht. Auch hier beruht meine Beurteilung der Regelwerke auf Rücksprache mit den in „3.3.2 Wertzuweisungen“ aufgelisteten Experten.

[17] Quelle Wikipedia: Das schwarze Auge [15]
[18] Quelle Wikipedia: Dungeons & Dragons [16]
[19] Quelle Webseite: Ulisses-Spiele.de [9]

Mit „FreeFate“[20] wird auch noch ein anderes Regelwerk betrachtet, dass andere Konzepte und Einteilungen verwendet.

Es gilt zu zeigen, dass die folgenden Elemente umsetzbar sind:

Eigenschaften (1 zu 1 Beziehung)

Sie müssen sowohl die Möglichkeit bieten eine textuelle Beschreibung anzugeben, als auch eine Wertzuweisung zu ermöglichen. Je nach Regelwerk kann beides zutreffen oder nur eine der beiden Beschreibungsformen.

Es muss möglich sein einen Wertebereich für die möglichen Zahlenwerte zu definieren.

Gegebenenfalls muss vorgegeben werden, wie dieser Wert bestimmt wird. Die möglichen Varianten sind bereits in „3.3.2 Wertzuweisungen“ beschrieben worden: Zuweisung, Zufall und Wertkombination.

Kombinierte Eigenschaften (n zu 1 Beziehung)

Im Gegensatz zu den Eigenschaften wird die Wertzuweisung bzw. Ermittlung über eine mathematische Kombination anderer Werte bestimmt.

Viele andere Aspekte entsprechen denen der normalen Eigenschaft. Es muss z.B. ein Wertebereich definierbar sein.

Kombination (1 zu n Beziehung)

Unter dem Begriff Kombination vereine ich die Begriffe Rollen und Klassen. Nicht nur in der Informatik sondern auch im Bereich der Rollenspiele sind diese Begriffe mehrfach belegt und haben unterschiedliche Bedeutungen und Interpretationen zur Folge.

Im Gegensatz zu Eigenschaften können sie mehrere Ziele (andere Eigenschaften) haben.

Ein weiterer Unterschied ist, dass sie nicht nur Auswirkungen haben, sondern auch Vorbedingungen haben können. Einerseits um diese Rolle oder Klasse wählen zu können und andererseits um Rahmenbedingungen zu beschreiben unter welchen Umständen sie gewählt werden können.

Würfel (Zufallsentscheidungen)

Die Modellierung von Zufallsentscheidungen muss möglich sein. Es müssen unterschiedliche Typen(Augenzahlen) von Würfeln definiert werden können.

[20] Quelle Webseite: FreeFate.de [8]

Listen

Es muss eine Aufzählung möglich sein. Die jeweiligen Zeilen müssen über einen Wert angesprochen bzw. identifiziert werden können. (z.B. Zahl aus Würfelergebnis). Sie benötigen Vorbedingungen und Möglichkeiten die Auswirkungen zu beschreiben.
Es soll gegebenenfalls auch eine textuelle Beschreibung angegeben werden.

Steigerungsmöglichkeiten

Die Charaktererstellung beinhaltet unter Umständen auch Formen, die bereits bei der Charaktererstellung einen Stufenanstieg oder Verbesserungsmöglichkeiten vorsehen.
Die möglichen Varianten gleichen den Wertzuweisungen. Zusätzlich kann eine Anzahl von Versuchsmöglichkeiten angegeben werden.

4.2 Einsatz für Domainexperten (Model)

Für den Anwender der DSL, dem Domainexperten bzw. Regelexperten, muss eine Bearbeitungsmöglichkeit zur Verfügung gestellt werden, in der er die Regeln aufschreiben kann. Es stehen vor allem die Ziele zum Einsatz einer DSL im Vordergrund. Nur wenn diese eingehalten werden, ist der Einsatz einer DSL sinnvoll und führt zu einer erhöhten Produktivität. Die Struktur muss einfach gehalten sein und für den Anwender verständlich präsentiert werden. Eine einfache Struktur und wenige Steuerungselemente sind nicht nur erwünscht, sondern entscheidend für den Erfolg der DSL.

Wünschenswert sind auch die bereits in universellen Programmiersprachen bekannten und in den Entwicklungseditoren verfügbaren Erleichterungen bei der Eingabe und der Darstellung der DSL. Zentrale Elemente sind hierbei:

Autovervollständigung & Vorschläge

Schlüsselwörtern können ergänzt werden und ggf. sogar das Einfügen gesamter Elemente sofern erkannt wird, dass es sich um eine zusammenhängende Beschreibung handelt.

Syntax-Hervorhebung & Validierung

Steuerungselemente der DSL, Zahlen und Texte können in Editoren besonders hervorgehoben oder eingefärbt werden.
Zusätzlich besteht die Möglichkeit dem Benutzer Informationen über Fehler bei der Verwendung anzuzeigen. Die dafür notwendige Überprüfung des gesamten Textes (Validierung) findet dafür idealerweise automatisch im Hintergrund statt.

Formatierung

Moderne Editoren bieten die Möglichkeit nach vorgegebenen Mustern eine Formatierung des Codes zu erstellen.
Korrekte Einrückungen des Codes helfen unter anderem Zusammenhänge und Grenzen besser darzustellen.

Als abschließenden Schritt in der Bearbeitung muss dem Domänenexperten die Möglichkeit zur Verfügung gestellt werden das in der DSL beschriebene Regelwerk für die Charaktererstellung in eine computernutzbare Form umzuwandeln. Beispielsweise Generierung von Javacode , der in geeigneter Form als Rahmengerüst für eine reale Instanz der Sprache verwendet werden kann. Die Realisierung der konkreten Instanz liegt nicht unbedingt im Aufgabengebiet des Domänenexperten. Er hat durch die Bearbeitung mit der DSL den Sachverhalt in eine Form überführt, die vom Computer verstanden werden kann und zusätzlich in anderen Umfeldern weiter eingesetzt werden kann.

Vorab sei erwähnt, dass die Codegenerierung und somit die Realisierung einer Instanz nicht teil des Projektumfangs ist. Das später aufgeführte Vorgehen zeigt neben der Beschreibung in Form der DSL beispielhaft, dass die Generierung von z.B. Java Klassen aus der beschriebenen DSL möglich ist. Die nötige Codegenerierung hängt auch mit der konkreten weiteren Verwendung zusammen und den daran gestellten Erwartungen. Eine pauschale Vorführung der kompletten Generierung wäre somit nicht vorteilhaft und müsste für den jeweiligen Anwendungsfall überarbeitet werden oder gar grundsätzlich neu implementiert werden.

4.3 “Create” in der Verwendung (Instanz)

Auch wenn die Erstellung der Instanz nicht im Rahmen des Projektes realisiert wird, möchte ich eine mögliche Verwendung skizzieren.

Denkbar ist die Erstellung einer Webpräsenz mit deren Hilfe sich interessierte Spieler Charaktere und die dazugehörigen Charakterbögen erstellen können. Diese sind bereits aufgrund der vorhandenen Beziehungen und beschriebenen Abhängigkeiten komplett regelkonform und erleichtern somit den Erstellungsprozess.

Eine weitere Anwendungsmöglichkeit wäre es in der Entwicklung eines Computerspiels den Prozess der Charaktererstellung modular über die DSL und den daraus resultierenden generierten Code zu repräsentieren. Die einfache und schnelle Erstellung neuer Regeln und deren Abänderung in der DSL ermöglicht zusätzlich eine einfache und dynamische Entwicklungsmöglichkeit, die den Entwicklungsprozess des Spiels vorantreiben kann.

5 Detailbeschreibung

Nachdem die Grundlagen zu DSLs und der Charaktererstellung erläutert wurden, ist es Zeit die konkrete Modellierung und Umsetzung der DSL zu betrachten. In diesem Kapitel wird das verwendete Framework vorgestellt und die konkrete Struktur der DSL dargestellt. Es werden die Elemente der Beispielsprachen aufgelistet und auf die Elemente in der Grammatik verwiesen, die diese Konstruktion abdecken können.

Als Hinweis sei erwähnt, dass von konkreten Erklärungsbeispielen des Frameworks anhand von Codefragmenten abgesehen wird. Bei der Einarbeitung in das Framework anhand des auf der Webseite empfohlenen Buches: „Implementing Domain-Specific Languages with Xtext and Xtend“[21] hat sich gezeigt, dass durch die stetige Weiterentwicklung des Frameworks bestehende Konstrukte und Verwendungsweisen schnell überholt sind. Oftmals ist die Verwendung verbessert und die Handhabe vereinfacht worden. In diesem Projekt wird XText in der Version 2.10.0 verwendet. Über Internetforen und diverse Beispiele der Onlinedokumentation ist sogar ersichtlich, dass die Weiterentwicklung innerhalb weniger Monate zu überholter Dokumentation und Verwendungsproblemen führen kann.[22] Die Community ist äußerst hilfsbereit und viele der Entwickler geben sich Mühe den verzweifelten Nutzern zu helfen. Hier offenbart sich auch ein weiterer Nachteil eines riesigen Frameworks. Um die erklärten Anwendungsfälle beurteilen und rekonstruieren zu können, müssen viele Details angegeben werden. Oftmals sogar aus vordergründig nebensächlichen individuellen Einstellungen des Frameworks. Beispielsweise trifft ein bereitgestellter Lösungsansatz nur in bestimmten Kombinationen einen noch verfügbaren Lösungsweg oder das beschriebene Problem ist bereits in den neueren Versionen gelöst oder anders implementiert worden. Sofern man nicht bereits ein Experte in dem Framework ist, kann man die Unterschiede in den angeführten Tutorial-Anweisungen, Forenlösungen und aktualisierten Vorgehensweisen kaum bis gar nicht unterscheiden.
Der verwendete Programmcode bezieht sich nur auf die eigen definierte DSL.

[21] Quelle Buch: Implementing Domain-Specific Languages with XTtext and Xtend, L.Bettini [2]
[22] Quelle Webseite: Eclipse XText Forum [24]

5.1 Das Framework – XText

5.1.1 Auswahl des Frameworks

Es gibt eine Vielzahl verschiedener Ansätze von Frameworks zur Erstellung von DSL. Zwei der größten derzeit vorhandenen und umfangreichsten Frameworks sind einerseits XText[23] und anderseits MPS (Meta Programming System)[24].
Die beiden Systeme nutzen unterschiedliche Entwicklungsumgebungen (IDE Integrated Development Environment). XText ist in Eclipse eingebettet und entwickelt worden. Die Entwickler von MPS bietet die Möglichkeit verschiedenen Entwicklungsumgebungen zu verwenden. Für Java-Entwicklungen wird IntelliJ verwendet. Für andere Sprachen wie z.B. Python, PHP oder Ruby werden ebenfalls Umgebungen zur Verfügung gestellt.

Die Entscheidung über die Auswahl des verwendeten Frameworks fand in einem sehr frühen Stadium der Projektarbeit statt, da die Einarbeitung in ein Framework einen erhöhten Lernaufwand mit sich bringt. Nicht nur die vielen Konzepte und Syntax müssen verstanden werden, sondern zusätzlich gibt es viele Abhängigkeiten und Arbeitsschritte, die das Framework anbietet, aber vermehrt im Hintergrund ausführt oder voraussetzt. Um die positiven Effekte der Entwicklungsumgebung ausnutzen zu können, muss man auch Zeit zur Ergründung dieser Bereiche mit einplanen.
Neben den unterschiedlichen Entwicklungsumgebungen verfolgen die beiden Frameworks unterschiedliche Ansätze. XText verfolgt die Strategie sich stark auf die „Java-Welt“ auszulegen und die in Eclipse dafür bereitgestellten Mechanismen auszunutzen. Durch die jahrelange Verwendung und hohe Verbreitung von Java sind die Entwicklungsumgebungen dafür erprobt und sehr weit ausgereift. Des Weiteren wird durch die starke Anlehnung an Java einem breiten Spektrum an Entwicklern der Einstieg in das Framework erleichtert. XText bietet den kompletten Umfang der in Java verfügbaren Möglichkeiten und hat durch die eigene Spracheerweiterung Xtend die Konzepte von Java erweitert und viele Erleichterungen wie schlankere Syntax Varianten ermöglicht (z.B. entfallen an vielen Stellen geschweifte Klammern).
MPS verfolgt einen komplett anderen Ansatz. Es bietet Komptabilität zu mehreren Zielplattformen wie bereits oben beschrieben (z.B. Java, C, Python, Ruby oder PHP). Ein zentraler Aspekt um diese Verfügbarkeit für mehrere Plattformen zu erreichen ist die unterschiedliche Struktur in der MPS den Code verwaltet. Anstatt den erzeugten textuellen Code in einem Zwischenschritt in maschinenverständliche Form zu überführen, der ausgeführt werden kann (siehe Kapitel „3.4 Compiler“), wird der geschriebene Code direkt in einem abstrakten Syntaxbaum (AST) verwaltet. Für den Benutzer werden Mechanismen zur Verfügung gestellt, die die Erstellung des Codes wie übliches textuelles Programmieren wirken lassen.

[23] Quelle Webseite: Eclipse XText [17]
[24] Quelle Webseite: JetBeans: MPS [18]

Zusätzlich bietet MPS die Möglichkeit der grafischen Programmierung. Hierbei werden der Code und die Beziehungen visualisiert und eine Bearbeitung ähnlich wie in Beschreibungs- und Modellierungssprachen (z.B. UML-Unified Modeling Language)[25] ermöglicht.

Als Eckpunkte zur Auswahl des Frameworks habe ich mich an den damals bekannten Anforderungen für die DSL orientiert. Im zentralen Fokus stand die Erlernbarkeit der DSL auch für Domänenexperten ohne Programmierkenntnisse. Die Anlehnung und Fokussierung von XText auf die textuelle Beschreibung von DSLs war ein entscheidender Faktor. Da die Schlichtheit der DSL im Vordergrund liegt, bestand die Gefahr, dass die vielen unterschiedlichen Möglichkeiten des MPS-Frameworks die resultierende DSL und den Benutzer überfordern könnten. XText hingegen schien darauf ausgelegt zu sein eine schlichte und simple Syntax zu produzieren und die komplizierteren Programmieraspekte nicht nur vor dem Domänenexperten zu verbergen, sondern auch dem Programmierer viele Arbeitsschritte abzunehmen und zu erleichtern.
Persönlich kam hinzu, dass ich den Umgang mit der Entwicklungsumgebung Eclipse gewohnt bin und die Eigenheiten, die alle größeren Entwicklungsumgebungen durch ihren Umfang beinhalten, einschätzen und umgehen konnte.
Somit fiel die Wahl des Frameworks auf XText.

Zurückblickend und mit der erlangten Erfahrung im XText-Framework stelle ich diese Entscheidung jedoch in Frage. Im Laufe des Projektes und mit den konkreteren Anforderungen, die zur Umsetzung der Beispielsprachen nötig sind, hat sich gezeigt, dass eine reine textuelle Beschreibung wie in XText auch an ihre Grenzen stoßen kann.
Im Rahmen der Erstellung der Bachelorarbeit habe ich die Einstiegsbeschreibungen auf der MPS Webseite erneut betrachtet und einzelne Konzepte entdeckt, die ich mir bei der Umsetzung in XText ebenfalls erwünscht hätte.
Als Beispiel sei hier das Konzept der Entscheidungstabellen („DecisionTables“) erwähnt, dass durch Martin Fowler als ein wichtiges Konzept in seinem Buch „Domain-Specific-Languages“ ausgeführt wird.[26] Es beschreibt eine Möglichkeit mehrere Bedingungen miteinander und ihren möglichen Kombinationen zu modellieren und die jeweiligen Konsequenzen daraus abzubilden. Folgendes Bild veranschaulicht die Vorgehensweise einer solchen Tabelle:

[25] Quelle Wikipedia: Unified Modeling Language [19]
[26] Quelle Buch: Domain-Specific Languages, M.Fowler [1, pp. 114-118]

Printer troubleshooter

		Rules							
Conditions	Printer prints	N	N	N	N	Y	Y	Y	Y
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is recognized by computer	N	Y	N	Y	N	Y	N	Y
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

Abbildung 2: 5.1 Beispiel eines Decision Tables[27]

In MPS ist dieser Designentwurf enthalten und kann sehr benutzerfreundlich verwendet werden. In XText habe ich über Umwege eine ähnliche Repräsentation erstellt. Das Konzept ist aber nicht vorgesehen und mit der Funktionsweise und der Ableitung der Arbeitsschritte in dieser Form nicht möglich.

Im Laufe des Projektes hat sich ebenfalls angedeutet, dass die möglichst einfach gehaltene DSL nicht alle nötigen Konzepte beschreiben kann. Einerseits da die Auswahlmöglichkeiten in der Charaktererstellung zu mehrdeutigen Ableitungsregeln für den Compiler führen konnten und dieser somit nicht beurteilen konnte, welche nächsten Schritte gewählt werden sollen. Das Framework hat diese Doppeldeutigkeit allerdings feststellen können und über eine Fehlerausgabe auf die Mehrdeutigkeit hingewiesen. Damit diese beseitigt werden kann, mussten aber weitere Steuerungssymbole eingeführt werden. Die DSL bleibt weiterhin einfach und überschaubar, allerdings wurde dadurch ein Argument für das XText Framework gemindert.

Warum das XText Framework jedoch eine große Erleichterung bei der Erstellung von DSLs sein kann und die Verwendung auch in diesem Projekt zu großen Vorteilen geführt hat, wird in den folgenden Abschnitten näher erläutert.

[27] Quelle Wikipedia: Decision Table [20]

5.1.2 Framework-Rahmen und Entwicklungsumgebung

Wie eingangs erwähnt ist das Framework XText in die Entwicklungsumgebung von Eclipse eingebunden. Zur Verwaltung und Erstellung der Projekte stehen Hilfsprogramme zur Verfügung wie sie Eclipse auch den universellen Programmiersprachen zur Verfügung stellt. Die Erstellung eines X-Text Projektes kann z.B. durch einen sogenannten Wizard durchgeführt werden, der in Form eines Leitfadens durch die möglichen Konfigurationen führt und alle gewünschten bzw. ausgewählten Projektstrukturen anlegt.
Moderne Programmierprojekte stützen sich meistens auf eine gewisse Form eines „Build-Management-Systems“.[28] (z.B. Ant, Maven oder Gradle)
Diese Systeme steuern den Aufbau und die Generierung benötigter Elemente des Projekts. Dies können automatisch generierte Klassen sein (z.B. generierter Code, der den Sachverhalt der DSL repräsentiert) aber auch komplette Ablaufsteuerungen (z.B. die Erstellung aller benötigten Klassen, das Verpacken der Daten in ein bestimmtes Format um einen Webserver oder Webservice bereitzustellen und nach fehlerfreier Zusammenstellung das Ziel gleichzeitig zu starten)
Diese Steuerung wird in XText durch das Framework „Gradle“ bereitgestellt.[29]
Es bietet eine Vielzahl an Möglichkeiten zur Individualisierung der Ablaufsteuerung im Projekt. Von einer detaillierten Beschreibung des Gradle-Frameworks wird an dieser Stelle jedoch abgesehen, da es ein eigenständiges Framework ist und somit einen großen Umfang an Funktionalitäten mit sich bringt. Im Rahmen des Projektes wurden kaum Änderungen an den Konfigurationsdateien von Gradle benötigt.
Das Gradle-Framework gibt auch eine Ordnerstruktur für den Code vor. Hier wird z.B. eine Unterteilung in vom Benutzer erzeugten Code und generierten Code unternommen. Generierter Code sollte niemals händisch ergänzt oder verändert werden. In der Regel wird der generierte Code bei Veränderungen an den selbst erstellten Klassen komplett erneut generiert und somit gehen alle eigenen Veränderungen verloren. Sind Anpassungen beim generierten Code nötig, wird dies über meist über Konfigurationsdateien oder entsprechende Elemente im selbst erstellten Code durch das Build-System verwaltet.

XText verwendet bei Organisation der Projektstrukturen einzelne Unterprojekte, die modular auf spezielle Aufgabengebiete ausgelegt sind. Die Zusammensetzung wurde bei der Erstellung des Projektes gewählt. Für das Create-Projekt werden vier Projektstrukturen angelegt:

Parent-Projekt

Die Hauptaufgabe besteht in der Organisation der anderen Projekte. Physikalisch sind die anderen Projekte Unterprojekte. Das Projekt steuert den Aufbau und die Verwendung der anderen Projekte.

---

[28] Quelle Wikipedia: Erstellungsprozess [21]
[29] Quelle Webseite: Gradle.org [22]

DSL Projekt

Hier wird Form der DSL beschrieben und es bestehen diverse Möglichkeiten, die Darstellung und Arbeitsweise der späteren DSL zu individualisieren. Es bestehen Möglichkeiten die Codegenerierung, Formatierungen und z.B. eigene Formen der Validierung bereitzustellen.

IDE Projekt

Es stellt eine Schnittstelle zur verwendeten Entwicklungsumgebung dar. Es gibt unterschiedliche Auswahlmöglichkeiten für den Editor der DSL. Dies kann in einer eigenen Eclipse Instanz erfolgen und hier würden die in Eclipse vorhandenen Programmiertools zur Verfügung stehen oder es wird ein WebEditor ausgewählt.
Für den gewählten Editor wird ein eigenständiges Projekt bereitgestellt, da unterschiedliche Hilfen bereitgestellt werden müssen.

Web Projekt

Um dem Benutzer eine einfache Zugangsmöglichkeit zur Bearbeitung zu gewährleisten, wurde für dieses Projekt ein WebEditor ausgewählt.
In diesem Projekt sind die nötigen Mechanismen generiert worden um ein Gerüst für einen lokalen Webserver bereitzustellen, auf dem der Editor ausgeführt werden kann. Die Verwendung eines WebEditors ermöglicht einen leichten Einstieg ohne die umständliche und für Anfänger eventuell überladene Entwicklungsumgebung von Eclipse.
Um dem Benutzer des Editors jedoch ähnliche Bearbeitungshilfen zur Verfügung zu stellen, bietet das XText-Framework drei Bibliotheken, mit denen im WebEditor aus Eclipse bekannte und mächtige Entwicklungshilfen zur Verfügung gestellt werden.
Eine detaillierte Beschreibung der drei Möglichkeiten (Orion, Ace oder CodeMirror) kann in der Online-Dokumentation von XText nachgelesen werden.[30]
Sie unterscheiden sich in einzelnen Funktionalitäten und Möglichkeiten, die sie zur Verfügung stellen. Für dieses Projekt wurde Orion eingebunden, da es mehr Funktionalitäten unterstützt als die beiden anderen Möglichkeiten.

5.1.3 Funktionsweise von XText

Die Sprachdefinition unserer Sprache „Create“ wird in einem Texteditor erstellt. Das Framework kann die Datei anhand ihrer Dateiendung zuordnen (.xtext).

Äquivalent zur Gültigkeit von Java-Klassen und deren gültigen Namensbereichen wird auch ein Namensbereich definiert, der die Gültigkeit auf diesen Namensbereich beschränkt. Zur Veranschaulichung in Nicht-Programmier-Terminologie hilft hier vielleicht das Bild von Längeneinheiten.

[30] Quelle Webseite: Eclipse Xtext [17]

Eine Einheit kann in unterschiedlichen Regionen anders definiert sein: In den USA und Europa können Längeneinheiten in Kilometer bzw. Meilen beschrieben werden. In dem einen Bereich stellt eine Längeneinheit somit 1000m (metrisches System) dar, während in dem anderen System die Längeneinheit ~1609m (imperiale System) darstellt. In unterschiedlichen Systemen können gleichbenannte Gruppierungen wie z.B. Kunde unterschiedliche Details beinhalten. Ohne Abgrenzung der Gültigkeit der Definition kann es zu Konflikten kommen.

In dem „.xtext“ Dokument werden die Regeln zur Charaktererstellung definiert. Auf die unterschiedlichen Kombinationsmöglichkeiten wird im folgenden Abschnitt eingegangen, der sich mit der Grammatik beschäftigt.

Sobald man alle Regeln definiert hat oder überprüfen möchte, ob die angegebenen Konstrukte richtig übersetzt und generiert werden können, wird über einen Menüpunkt ausgewählt, dass das Framework die sogenannten Artefakte erzeugen soll. Damit ist die Generierung der ausführbaren Klassen und Abhängigkeiten gemeint. Der interne Compiler führt zusätzlich weitere Überprüfungen durch. Es wird z.B. überprüft, ob durch die Reihenfolge der Regelaufrufe gegebenenfalls eine nicht eindeutige Auswahl der Regeln möglich ist. Ebenso werden Schleifen Konstrukte erkannt. Das Framework übernimmt selbstständig ohne zusätzlichen Eingriff die Validierung der angegebenen Regeln. Es können für diesen Bereich weitere Überprüfungen erstellt werden, um besondere individuelle Überprüfungen durchzuführen oder die Vorhandenen zu erweitern.

Das Framework erstellt bei der ersten Generierung Default-Konfigurationen und Klassen, die zur Steuerung des Frameworks nötig sind. Diese sind Einstiegspunkte zur Ergänzung des Ablaufs oder Veränderung des Prozesses. Werden keine besonderen Anpassungen vorgenommen, wird der vorgesehene allgemeine Ablauf eingeleitet.

Zu jeder in der DSL angegebenen Regel werden sowohl ein Interface (Schnittstelle) definiert als auch eine Implementierungsklasse in Form einer konkreten Java-Klasse mit allen angegebenen Variablen und Zugriffsmöglichkeiten, die in dem Interface definiert sind. Das Framework erstellt zusätzlich automatisch weitere Hilfsmethoden wie z.B. eine „toString“ Methode, mit deren Hilfe die Variablen und Werte der Klasse ausgegeben werden können.
Es werden ebenfalls „JavaDoc“ Elemente für die Klassen, Variablen und Methoden generiert. „JavaDoc“ ist ein Format mit dem Zusatzinformationen über die Funktionsweise und alle relevanten Informationen an z.B. eine Java-Klasse oder Methode angefügt werden kann. Diese Informationen stehen dem Entwickler dann bei Aufrufen der Klassen oder Methoden in der Entwicklungsumgebung zur Verfügung. Sie helfen beim Verständnis der Klassen und sind eine Stütze zur langfristigen Dokumentierung des Programmcodes.

Xtext verwendet als Organisationsstruktur der Klassenhierarchie und Modellierung das in Eclipse eingebundene Eclipse Modeling Framework (EMF)[31] Es beinhaltet eine eigene DSL bzw. eigenes Metamodel, mit dem Klassendiagramme und den darin enthaltenen Klassen und weiteren Konstrukten darstellen kann. Auch hier wird von einer näheren Beschreibung des Frameworks abgesehen. Die Erläuterung dient zur Verdeutlichung wie die erzeugten Regeln in XText verwaltet und erzeugt werden.

Nach der Erzeugung der Java-Klassen und den durchgeführten Überprüfungen durch das Framework kann die erstellte DSL nun verwendet werden. Dies kann je nach Projekteinstellung über eine eigene Instanz von Eclipse stattfinden oder wie in diesem Projekt über einen Webeditor.
Gesteuert wird der Aufruf bei einem Eclipse-Editor über eine interne Runtime-Configuration. Für den Webeditor hingegen sind weitere Vorbereitungsschritte nötig.
Hier stellt das Build-System Gradle die nötigen Operationen zur Verfügung. Der vorgegebene Standartablauf, der mit einem Gradle Befehl initiiert wird, erstellt aus dem generierten Code und den vorgegebenen Einstellungen (gesetzt im Web Projekt des Frameworks) ein WebServlet[32]. Ein WebServlet beschreibt eine konkrete Java-Klasse (bzw. idie Instanz der Klasse), die es einem Webserver ermöglicht Anfragen eines Klienten (z.B. Webbrowsers) entgegen zu nehmen und zu beantworten. Beispielhaft sei hier die Anfrage des WebEditors genannt, ob das vom Experten erstellte Dokument alle Regeln einhält oder Fehler beinhaltet.
Das Framework verwendet als Standartwebserver „Jetty“.[33]

Der Webeditor ist auf dem lokal gestarteten Server unter der in der Konfiguration angegebenen Adresse zu erreichen: „localhost:8080/create/index.html“. Für das Create Projekt habe ich einen eigenen Pfad angegeben um von der Standartimplementierung abzuweichen.

5.2 Die Grammatik

XText bietet unterschiedliche Möglichkeiten an die bereits vorhandenen Grammatiken zu erweitern. Standartmäßig ist als Vorgabe eine DSL eingestellt, die darauf ausgelegt ist simple textuelle Beschreibungen darzulegen: „Terminals“. Für dieses Projekt ist die Erweiterung „XBase“ verwendet worden. Sie zielt darauf ab, dass in den erstellten DSLs direkt Java-Klassen und Typen referenziert werden können. Entgegen der ursprünglichen Annahme wird dies in der jetzigen DSL allerdings nicht verwendet.

[31] Quelle Wikipedia: Eclipse Modeling Framework [23]
[32] Quelle Wikipedia: Servlet [25]
[33] Quelle Webseite: Eclipse Jetty [26]

Generell anzumerken ist, innerhalb der DSL nur Definitionen (z.B. Attribute oder Würfel) referenziert werden können, die zeitlich vor der aktuellen Regel erstellt wurden.
Die Reihenfolge der Variablen (mit ihren Schlüsselwörtern) ist fest vorgegeben und können nur in dieser Reihenfolge angegeben werden. Eine Vertauschung oder Verdrehung der Reihenfolge resultiert in einen Fehler.

Als Einstiegspunkt der DSL wird eine Regel definiert, die als Rahmen für alle weiteren abgeleiteten Regeln agiert. Diese wird wie in den Einstiegsbeispielen Domainmodel bezeichnet.

```
9  Domainmodel:
10     'game' name = QualifiedName '('
11         elements+=Entity*
12     ')';
```

Abbildung 3: 5.2 Regel Domainmodel

Schlüsselwörter wie „game“ werden als einfache Zeichenkette definiert. „name“ ist ein vordefiniertes Wort und der zugewiesene Wert stellt den konkreten Namen der später erzeugbaren Klasse dar. „QualifiedName“ ist eine aus der vorgegebenen DSL abgeleitete Regel zur Darstellung von Strings. Innerhalb der Klammern wird dann eine Liste von Entities definiert. „elements“ beschreibt dabei die Listen-Variable, der die einzelnen aufgelisteten „Entity“ zugewiesen werden. Der „*“ ist ein Element der EBNF Schreibweise (EBNF – Erweiterte Backus-Naur-Form)[34]und gibt die Kardinalität an (d.h. wie häufig das Element „Entity“ vorkommen kann. Der „*“ bedeutet kein oder mehrfach).

```
15 Entity:
16     Character |
17     Combination |
18     Attribute |
19     CombinedAttribute |
20     Dice |
21     Raise |
22     Category |
23     DecisionTable |
24     ValueAssign
25 ;
```

Abbildung 4: 5.3 Regel Entity

Die Regel stellt einen Container für die möglichen Elemente des Spiels dar. Die jeweiligen Regeln werden an dieser Stelle definiert. Später können diese erstellten Regeln referenziert werden.

[34] Quelle Wikipedia: Erweiterte Backus-Naur-Form [27]

```
27 Character:
28     'character' name = QualifiedName '('
29         properties += Property*
30     ')'
31 ;
```

Abbildung 5: 5.4 Regel Character

Der Charakter besitzt die gleiche Struktur wie die Regel Domainmodel. Es wird ein Name zugewiesen und die Variable „properties“ beinhaltet alle definierten Regeln „Property“.

```
34 Property:
35     'attribute:' attrValue = [Attribute] |
36     'combinedAttribute:' combAttrValue = [CombinedAttribute] |
37     'combination:' combValue = [Combination]
38 ;
```

Abbildung 6: 5.5 Regel Property

Die Regel Property bietet Möglichkeiten unterschiedliche Formen der Eigenschaften zu definieren. Jede Form der Eigenschaft (normale Eigenschaft, kombinierte Eigenschaft und Kombination) können anhand ihrer Schlüsselwörter identifiziert werden. Zu beachten ist hierbei Alternativentscheidungen möglich sind. Das Zeichen „|“ steht für ein logisches „ODER“. D.h. eine Property kann nur eine der drei Formen annehmen und darstellen.
Jede der drei Typen hat ihre eigene Variable mit unterschiedlicher Bezeichnung. Dies ist für das Framework wichtig, da später bei Hilfsmitteln wie der Autovervollständigung im abstrakten Syntaxbaum ansonsten Probleme auftreten. In den eckigen Klammern werden die jeweiligen Regeln referenziert. Es wird also keine weitere Regel aufgerufen, sondern auf eine vorher definierte Regel und ihren repräsentativ Namen verwiesen.
Das Schlüsselwort der unterschiedlichen Eigenschaften ähnelt dem Schlüsselwort bei ihrer Definition und unterscheidet sich nur durch den Doppelpunkt um eine Zuweisung darzustellen.
Da eine Property nur innerhalb eines Charakters verwendet wird, führt dies nicht zu Konflikten oder Doppeldeutigkeiten.

```
48 Attribute:
49     'attribute' name = QualifiedName  '('
50     ('value:' (intValue=INT | stringValue=STRING))?
51     ('initialValue:' initialValue = [ValueAssign])?
52     ('type:' valueType= ('Integer' | 'String'))?
53     ('range[' minValue = INT ',' maxValue = INT ']')?
54     ('category:'category = [Category])?
55     ('description:' description = STRING )?
56     ')'
57 ;
```

Abbildung 7: 5.6 Regel Attribute

Damit die unterschiedlichen Varianten von Eigenschaften definiert werden können, sind die jeweiligen Variablen mit dem Konstrukt (Expression)? versehen. Diese Schreibweise stellt dar, dass der Wert entweder gar nicht oder einmal vorkommen kann. Dadurch wird ermöglicht, dass nicht jede Variable in dem Attribute beschrieben oder belegt werden muss. Zusätzlich zu den in den Anforderungen bereits notwendigen Feldern gibt es die Möglichkeit die Eigenschaft eine vorher definierte Kategorie zuzuordnen uns z.B. eine detaillierte Beschreibung anzugeben.
Mit dem Schlüsselwort „initialValue“ wird die Form der Wertzuweisung dargestellt. Die Regel „ValueAssign“ wird später beispielhaft für die möglichen Mechanismen die Generierung der Zuweisungsmöglichkeiten aufzeigen.

```
59 AttributeRef:
60     attribute=[Attribute] '.' part=AttributeIntPart
61 ;
66 enum AttributeIntPart:
67     attrValue | attrMinValue | attrMaxValue
68 ;
```

Abbildung 8: 5.7 Beispiel für Referenzierung innerhalb der DSL

Um auf einzelne Felder der Attribute-Regel innerhalb der DSL zugreifen zu können, muss eine Referenzierungsregel definiert werden. Sie besteht aus zwei Elementen. „AttributeRef“ repräsentiert die Referenz auf ein vorher definiertes Attribut. „AttributeIntPart“ stellt eine Enumeration dar und ermöglicht die Unterscheidungsmöglichkeit unterschiedliche Werte anzugeben. Hierbei muss darauf verwiesen werden, dass diese Werte nur String-Literale sind. Es muss bei der späteren Codegenerierung also aufgrund des Enumerationstyps eine Unterscheidung stattfinden. Der Umweg über eine Referenzregel ist notwendig, da es zwar vorgesehen ist Javatypen referenzieren zu können, allerdings zu dem Zeitpunkt der Erstellung der DSL kein entsprechendes Javaelement definiert ist, auf das sich bezogen werden könnte. Hier befindet man sich in der rein textuellen Beschreibung des Sachverhaltes.
Äquivalent zu dem hier aufgezeigten Beispiel enthält die Grammatik Referenzen für kombinierte Eigenschaften und Kombinationen.

```
89 CombinedAttribute:
90     'combinedAttribute' name = QualifiedName '('
91     'range[' minValue = INT ',' maxValue = INT ']'
92     'combinedValue:' combValue = MathExpression
93     ('category:'category=[Category])?
94     ('description:' description = STRING )?
95     ')'
96 ;
```

Abbildung 9: 5.8 Regel CombinedAttribute

Die Regel ähnelt dem normalen Attribute. Die einzige Unterscheidung ist, dass die Variable für den Wert nicht optional ist, da eine kombinierte Eigenschaft ohne diesen Wert keinen Sinn ergibt. Die mögliche Berechnungskombination wird in der Regel „MathExpression“ dargestellt.

```
113 MathExpression:
114     Addition
115 ;
116
117 /* Add & Sub */
118 Addition returns MathExpression:
119     Multiplication ({Addition.left=current} op=('+' | '-') right=Multiplication)*
120 ;
121
122 /* Mult & Div */
123 Multiplication returns MathExpression:
124     Primary ({Multiplication.left=current} op=('*' | '/') right=Primary)*
125 ;
126
127 Primary returns MathExpression:
128     IntRef |
129     AttributeRef |
130     DiceRef |
131     '(' Addition ')'
132 ;
133
134 IntRef:
135     intVal = INT
136 ;
```

Abbildung 10: 5.9 Regel MathExpression

Es ist eine weit verbreitete Herausforderung in XText die arithmetischen Kombinationsmöglichkeiten darzustellen. Das Regelkonstrukt ist von einem Beispiel zur Umsetzung mathematischer Ausdrücke übernommen worden. Es können in XText manuell Rückgabewerte angegeben werden. „returns MathExpression“. Sofern dieses Konstrukt nicht angegeben ist, wird automatisch der Typ der Regel erstellt.
Der wichtige Aspekt ist die Verhinderung einer möglichen Linksrekursion, sofern die Ableitungsform nicht korrekt abgebildet ist. Mit der Zuweisung „{Regel}.left=current“ wird dem abstrakten Syntaxbaum der jeweils vorher ausgewertete linke Teil der Regel (z.B. die Multiplikation) zugewiesen und dadurch sichergestellt, dass die Rekursion im gegebenen Fall nur in eine Rechtsrekursion enden kann. Die Reihenfolge in der die Multiplikation und Addition angeordnet sind, spiegelt auch die Priorität der Auswertungsreihenfolge dar.
Das auch im XText-Buch beschriebene Beispiel ist in der Regel „Primary“ darauf ausgelegt, dass eine Zahl, ein Attribute oder ein Würfel referenziert werden kann.

```
176 Dice:
177     'dice' name = ID '('
178     maxValue=INT
179     ')'
180 ;
183 DiceRef:
184     dice=[Dice] '.' part=DicePart
185 ;
190 DicePart:
191     throw = 'throw'
192 ;
```

Abbildung 11: 5.10 Regel Dice

Eine besondere Eigenschaft hat die Definition des Würfels. Er hat nur einen Maximalwert, der die maximale Würfelhöhe definiert. Um das Ergebnis eines Würfelwurfes zu modellieren, kann daher in der Referenz ein „throw“ dargestellt werden. In der Codegenerierung würde die Referenz dann zu einer zufallsgenerierten Zahl umgewandelt, die im Wertebereich zwischen 1 und dem Maximalwert des Würfels liegt.

```
145 Combination:
146     'combination' name = ID '('
147         (conditions+=CombinationCondition)*
148         raise=[Raise]
149         initValue = ValueAssign
150     ')'
151 ;
```

Abbildung 12: 5.11 Regel Combination

Zur Erinnerung: Eine Kombination stellt Rollen und Klassen dar. Hier gibt es eine mögliche Reihe von Vorbedingungen, die erfüllt sein müssen.
Mit der Variable „raise“ kann man eine vorher definierte Steigerungsform angeben. Wie bei den Eigenschaften kann über die Regel ValueAssign eine Methode definiert werden, wie die Werte erstellt werden.

```
165 CombinationCondition:
166     'combinationCondition' name = ID '('
167     condition=XExpression
168     ')'
169 ;
```

Abbildung 13: 5.12 Regel CombinationConditon

Diese Regel bedient sich der ausdrucksstarken XExpression der XBase DSL. Die Konditionen sind eine der wenigen Regeln der DSL, die ich nicht weiter eingeschränkt habe. Sie kann jedoch äquivalent zu den vorherigen Beispielen der MathExpression konkretisiert werden.

```
198 Raise:
199     AttributeRaise | CombinationRaise
200 ;
203 AttributeRaise:
204     'attributeRaise' name=ValidID '('
205         target=[Attribute]
206         (tries=INT)?
207         valueChange=AttributeRaiseType
208     ')'
209 ;
212 AttributeRaiseType:
213     DiceBasedRaise | ConstantBasedRaise | ExpressionRaise
214 ;
217 DiceBasedRaise:
218     'diceRaise' '('
219         diceToUse=[Dice]
220         diceResult=DiceRef
221     ')'
222 ;
225 ConstantBasedRaise:
226     'constantRaise' '('
227         constResult=INT
228     ')'
229 ;
231 ExpressionRaise:
232     'expressionRaise' '('
233         raiseExpression=MathExpression
234     ')'
235 ;
238 CombinationRaise:
239     'combinationRaise' name = ValidID '('
240     (actions+=CombinationRaiseType)+
241     ')'
242 ;
244 CombinationRaiseType:
245     AddCombination | DeleteCombination
246 ;
248 AddCombination:
249     'addComb(' add=Combination ')'
250 ;
251
252 DeleteCombination:
253     'delComb(' delete=Combination ')'
254 ;
```

Abbildung 14: 5.13 Regeln für Raise

Es sind nicht alle Formen der möglichen Steigerungen modelliert. Zum Darstellen der Möglichkeiten habe ich prototypisch zwei Formen der Steigerungen erstellt.
Die Steigerung eines Attributes und das Entfernen oder Hinzufügen einer Rolle Kombination. Attribute können ähnlich der Wertzuweisung eine konstante Werterhöhung erfahren, eine Zufallssteigerung durch ein Würfelereignis erlangen oder in Form einer mathematischen Kombination dynamische Werte zugewiesen werden.
Vorgesehen, aber noch nicht konstruiert ist der Zusammenhang der Vorbedingungen von Kombinationen und den Abhängigkeiten bei Steigerungsversuchen.

Bei Attribute-Steigerungen besteht zusätzlich noch die Möglichkeit Zielattribute anzugeben und eine Anzahl an möglichen Versuchen aufzulisten.

```
265 DecisionTable:
266     'decisionTable' name=ID
267     ('('    numberConditionRows=INT ',' numberConsequenceRows=INT ')')?
268     '['
269     rows+=Row*
270     ']'
271 ;
273 Row:
274     'i=' index=INT ':'
275     conditions+=Condition ('-' conditions+=Condition)*
276     consequences+=Consequence ('|' consequences+=Consequence)*
277 ;
288 Condition:
289     condName = ValidID '('
290     expression= ConditionTargets op=RelOpTo INT
291     ')'
292 ;
294 RelOpTo:
295     '=' | '<' | '>' | '!=' | '<''=' | '>''='
296 ;
302 ConditionTargets:
303     AttributeRef | CombinedAttributeRef | CombinationRef
304 ;
307 Consequence:
308     target=ConsequenceRef
309     effect=[Raise]
310 ;
313 enum ConsequenceRef:
314     AttributeRef | CombinationRef
315 ;
```

Abbildung 15: 5.14 Regeln für DecisionTables

Die Regel der DecisionTable ist nur prototypisch umgesetzt. Der Aufbau der Bedingungen ist in vereinfachter Form zur reinen Darstellung umgesetzt. Auch hier müsste für eine spätere Codegenerierung die Bedingungen noch ergänzt werden.
Die Besonderheit an diesen Regeln ist, dass versucht wird eine Tabellenform nachzustellen, in einem Framework, bei dem Referenzen und Abhängigkeiten innerhalb der DSL schwer darzustellen sind. Wie bereits in „Abbildung 2: 5.1 Beispiel eines Decision Tables“ aufgezeigt wurde, besteht bei einer Tabelle nicht nur eine horizontale Abhängigkeit bzw. Verbindung, sondern auch in vertikaler Richtung. Diese Abhängigkeiten können in XText nicht modelliert werden. Ein direkter Zusammenhang ist in der Regel und den darin enthaltenen ableitenden Regeln enthalten.
Mein Ansatz ist es eine beliebige Anzahl von Bedingungen zu ermöglichen. Darauf folgend wird eine beliebige Anzahl an Konsequenzen aufgeführt. XText kann die beiden Elemente anhand ihrer führenden Steuerelemente „-„ oder „|“ unterscheiden.

```
ValueAssign:
    'valueAssign' name = QualifiedName
    '<' generator=ValueGenerator ','
    'count=' genCount=INT ','
    '['targets=TargetGenerator']' '>'
 ;
ValueGenerator:
    '(' generatorExpression = MathExpression ')'
 ;
TargetGenerator:
    'targetAttributes:' attributes+=[Attribute]  (',' attributes+=[Attribute])* ';' |
    'targetCombinations:' combinations+=[Combination] (',' combinations+=[Combination])* ';'
 ;
```

Abbildung 16: 5.15 Regeln für Wertzuweisung

Der Wertegenerator ist ein prototypisches Beispiel dafür, wie die Struktur und Möglichkeiten der DSL um weitere Funktionalitäten erweitert werden können.
Derzeit bietet der Generator die drei bereits bekannten Möglichkeiten der Wertzuweisung, die in der Regel „MathExpression“ angegeben sind. Über die Variable „count“ bietet er die Möglichkeit einer beliebigen Anzahl an Werten gemäß der angegebenen ValueGenerator-Methode zu erstellen und auf die in den Targets angegebenen Eigenschaften oder Kombinationen zu verteilen.

Die hier aufgeführten Regeln veranschaulichen die grundsätzliche Machbarkeit der Darstellung der nötigen Elemente für die Erzeugung von Charakteren. Die nötigen Schritte um aus diesen Formen der Beschreibung die erforderlichen Java-Klassen zu generieren, werden nicht in dieser Projektarbeit näher aufgeführt. Die Machbarkeit der Codegenerierung der in der DSL beschrieben Regeln zu nutzbaren Java-Klassen ist jedoch eindeutig in der XText Dokumentation belegt und erläutert.[35]
Das Prinzip verfolgt eine simple Strategie. Für jede vorhandene Regel wird eine „compile“ Methode erstellt. Variablen und Methoden werden mittels von dem Framework zur Verfügung gestellter IF-ELSE Konstrukte überprüft und generiert.
Die Umsetzung der Compile-Methoden ist im DSL-Projekt prototypisch anhand einiger Beispiele dargestellt:

```
def compile(Attribute e) {
    '''
    package «e.eContainer.fullyQualifiedName»

    public class «e.name» {
        private «IF e.valueType.equalsIgnoreCase("Integer")»Integer value = null;
                «ELSEIF e.valueType.equalsIgnoreCase("String")»String value = "";
                «ENDIF»

            private Integer intValue = «IF Integer.valueOf(e.intValue) != null»«e.intValue» «ELSE» «null» «ENDIF»;
            private String stringValue = «IF e.stringValue != null» «e.stringValue» «ELSE» «null» «ENDIF»;

        «IF e.initialValue != null»
            «compile(e.initialValue)»
        «ENDIF»
        private Integer minValue = «IF Integer.valueOf(e.minValue)!=null»«e.minValue»«ELSE»«null»«ENDIF»
        private Integer maxValue = «IF Integer.valueOf(e.maxValue)!=null»«e.maxValue»«ELSE»«null»«ENDIF»

        private String category = «IF e.category != null»«e.category»«ELSE»«null»«ENDIF»
        private String description = «IF e.description != null»«e.description»«ELSE»«null»«ENDIF»
    }
    '''
}
```

Abbildung 17: 5.16 Beispiel einer compile-Methode in der Codegenerierung

[35] Quelle Webseite: Eclipse Xtext Dokumentation - Codegenerierung [28]

Das Bild kann nicht größer übernommen werden, da XText die Formatierung über Mehrfachzeilen-Strings darstellt. Die resultierende Einrückung ist z.B. über die grau hinterlegten Blöcke erkennbar. Würde der IF-ELSE Block in eine tiefere Zeile verschoben werden, würde auch der generierte Code auf mehrere Zeilen verteilt sein. Das Bild zeigt die Erstellung der Variablen und die initiale Wertzuweisung abhängig davon, ob der Wert gesetzt wurde oder nicht.

```
62          private Integer getDiceEvent() {
63              return «IF e.diceEvent != null» «e.diceEvent.compile» «ELSE» «null» «ENDIF» ;
64          }
```

Abbildung 18: 5.17 Beispiel der Umsetzung einer Methode in der Codegenerierung

Die Abbildung zeigt den erstellten Methodenrumpf. Nach demselben Prinzip können alle nötigen Codegenerierungen aus der DSL erzeugt werden.
Wichtig ist zu beachten, dass die Übersetzungsmethoden sich nicht an eine veränderte DSL anpassen, sondern mit jedem Entwicklungsschritt händisch angepasst werden müssen.

5.3 Die Beispiel-Modelle

Die folgenden Ausschnitte sind im Webeditor erstellt worden und demonstrieren die Umsetzbarkeit der Charaktererstellung mittels einer DSL.

Ein notwendiger Hinweis bei der Umsetzung der Regelwerke ist, dass es in einigen Regelwerken bereits bei der Charaktererstellung Aspekte gibt, die die Ausrüstung des späteren Charakters beschreiben. Ausrüstungen sind ein weiterer großer Teil der Umsetzung eines Regelwerkes und umfassen viele unterschiedliche Details, die ich nicht im Projektrahmen modellieren kann. Mit der Gestaltung der Ausrüstungsgegenstände kann ein komplett eigenständiges Projekt gefüllt werden. Deshalb werden die Bereiche der Regelwerke, die eine Darstellung von Ausrüstungsgegenständen voraussetzen nicht bei der Charaktererstellung berücksichtigt. Ebenso wird die Steigerung der Eigenschaften und Kombinationen nicht demonstriert. Sie sind in erster Linie ein Mechanismus des Spielablaufes und weniger ein Instrument bei der Charaktererstellung. Grundlegend sind die nötigen Strukturen in der DSL beschrieben und für das konkrete Spiel/Regelwerk im Gesamtkonzept verfügbar, aber die Demonstration der konkreten DSL beschränkt sich der Übersichtlichkeit halber rein auf die Charaktererstellung.

Die kompletten Quellen der Regelwerke werden auf den beiliegenden CDs oder über die in den jeweiligen Untersektionen verknüpften Hinweise des Literaturverzeichnisses eingesehen werden. Es werden jeweils die Konzepte erläutert und beispielhaft anhand einer konkreten Darstellung in der DSL demonstriert. Jedes Regelwerk besitzt identische Typen von Eigenschaften, die meist eine Gruppierung von Eigenschaften repräsentieren (z.B. körperliche oder geistige Talente). Eine komplette Auflistung aller einzelnen Regeln und deren Ausformulierung erbringen keine zusätzlichen Wissenserkenntnisse und führen sogar zu einer weitaus unübersichtlicheren Auflistung.

5.3.1 Das schwarze Auge

Die verwendeten Quellen sind kein konkretes Regelwerk. Diese sind einerseits kostenpflichtig und anderseits nicht öffentlich im Internet verfügbar. Anhand eines Charakterbogens des Regelwerkes DSA 3.0[36] werden die vorhandenen Eigenschaften und Fähigkeiten beschrieben. Die Vorgehensweise bei der Charaktererstellung erfolgt anhand einer Zusammenstellung von Kurzregeln.[37]

Textuell beschriebene Eigenschaften:

```
game DSA3 (
    category 'beschreibend'
    attribute geschlecht (type: String category: beschreibend)
    attribute charaktername (type: String category: beschreibend)
    attribute rasse (type: String)
)
```

Abbildung 19: 5.18 Textuelle Eigenschaften

Die Abbildung veranschaulicht die Darstellung einer Eigenschaft durch eine textuelle Beschreibung. Optional kann der Eigenschaft auch eine Kategorie zugewiesen werden.

[36] Quelle Webseite: Helden.de [29]
[37] Quelle Webseite: Taladas.de [30]

Eigenschaften mit Werten und dazugehöriger Wertzuweisung:

```
    category 'positiveEigenschaften'
    category 'negativeEigenschaften'
    dice w6(6)
    valueAssign positiveWerte
                    <   (w6.throw + 7),
                        count=8,
                        [targetAttributes: mut, koerperkraft, gewandtheit;]
                    >
    valueAssign negativeWerte
                    <   (w6.throw + 2),
                        count=7,
                        [targetAttributes: aberglaube;]
                    >
    attribute mut
                    (   initialValue: positiveWerte
                        range[0,20]
                        category: positiveEigenschaften
                    )
    attribute koerperkraft
                    (   initialValue: positiveWerte
                        range[0,20]
                        category: positiveEigenschaften
                    )
    attribute gewandtheit
                    (   initialValue: positiveWerte
                        range[0,20]
                        category: positiveEigenschaften
                    )
    attribute aberglaube
                    (   initialValue: negativeWerte
                        range[0,6]
                        category: negativeEigenschaften
                    )
```

Abbildung 20: 5.19 Eigenschaften mit Werten und Wertzuweisung

Beispielhaft sind hier die drei positiven und eine negative Eigenschaft aufgelistet. Der Wertgenerator ist darüber definiert. Im Regelwerk werden insgesamt 8Würfe (count) vorgesehen für die sieben positiven Werte. Die Zuweisung und Auswahlmöglichkeiten müssen durch die konkrete Instanz der Charaktererstellung beschrieben werden. In den Targets sind in dem Bespiel nur die drei vorhanden Eigenschaften aufgelistet um die Machbarkeit darzustellen. Die vollständige Liste müsste alle sieben Eigenschaften enthalten.

Bei den negativen Eigenschaften sind sieben Würfe vorgesehen. Alle sieben Werte müssen einer der sieben negativen Eigenschaften zugewiesen werden.
Ein identisches Vorgehen kann bei den Fertigkeitsattributen wie z.B. Kampf- oder Handwerksfertigkeiten verwendet werden.

Kombinierte Eigenschaften:

```
41      category 'Kampf'
42      combinedAttribute basisAttacke
43                      (   range[0,20]
44                          combinedValue: ( mut.attrValue +
45                                           koerperkraft.attrValue +
46                                           gewandtheit.attrValue )
47                                           / 5
48                          category: Kampf
49                      )
50  )
```

Abbildung 21: 5.20 Kombinierte Eigenschaften

Am Beispiel des Basis-Attacke Wertes wird dargestellt wie eine kombinierte Eigenschaft repräsentiert werden kann.

Listenauswahl:

```
51      attributeRaise mutPlusEinsRaise
52                      (
53                          mut // Ziel
54                          2 // Versuche
55                          constantRaise (1) // Art der Steigerung
56                      )
57      attributeRaise mutPlusZweiRaise
58                      (
59                          mut // Ziel
60                          2 // Versuche
61                          constantRaise (2) // Art der Steigerung
62                      )
63      attributeRaise mutDiceRaise
64                      (
65                          mut // Ziel
66                          2 // Versuche
67                          diceRaise
68                                  (   w6 // verwendeter Würfel
69                                      w6.throw
70                                  )
71                      )
72
73      decisionTable standartFertigkeitenListe
74                  [
75                      i= 1 : AttributeRef mutPlusEinsRaise
76                      i= 2 : AttributeRef mutPlusZweiRaise
77                      i= 3 : AttributeRef mutDiceRaise
78                  ]
```

Abbildung 22: 5.21 Auswahl aus einer Liste

Die Darstellungsform der Steigerungen ist noch nicht vollständig umgesetzt worden, da die Steigerungen im Laufe des Projektes als Spielmechanik eingestuft wurden und daher die Regeln nicht weiter aktualisiert worden. Die Abbildung zeigt jedoch, dass selbst in diesem Zustand eine Darstellung möglich ist.
Die Darstellung einer Liste kann durch eine decisionTable ohne Bedingungen erzeugt werden. Die Struktur der DecisionTable ist jedoch auch noch prototypisch und es werden auch noch weitere Veränderungen wahrscheinlich sein, sobald man sich mit den Details der Codegenerierung aus einer solchen Liste beschäftigt.
Das zugrundeliegende Prinzip kann aber bereits erläutert werden:
Die einzelnen Zeilen sind über die Variablenzuweisung „i“ möglich und können somit ähnlich einem Indexzugriff angesteuert oder referenziert werden. „AttributeRef“ kann dann beispielsweise ein Attribute-Ziel angeben. Derzeit ist als Platzhalter ein Enumerationswert hinterlegt.

```
80      decisionTable tableMitBedingungen
81                  [
82                      i= 1: mutGroesser11(mut.attrValue > 11) AttributeRef mutPlusEinsRaise
83                      i= 2: mutKleiner11(mut.attrValue < 11) AttributeRef mutDiceRaise
84                  ]
```

Abbildung 23: 5.22 DecisionTable mit Bedingung

Das Bild zeigt, dass die DSL selbst in ihrer jetzigen prototypischen Form eine DecisionTable mit Bedingungen und Konsequenzen darstellen kann. Natürlich wird die Darstellung um so unübersichtlicher je mehr Bedingungen und Konsequenzen diese hat. Sie ist somit kaum vergleichbar mit dem eigentlichen Entwurfsmuster.

Wie bereits einführend erwähnt, sind weder die Steigerungen noch die Zuweisung von Ausrüstungsgegenständen beschrieben.

5.3.2 Dungeons & Dragons

Die Regeln für dieses Rollenspiel werden anhand des Handbuches beschrieben. Es liegt PDF-Format der angefügten CD bei. Herausgegeben wird das Regelbuch von dem Unternehmen „Wizards oft he Coast, Inc.“, die das Spiel publizieren.[38]

Dungeons & Dragons erweist eine deutlich ähnliche Struktur und Aufbau der Beschreibungen auf wie „das schwarze Auge“. Der Aufbau der beschreibenden und wertbelegten Eigenschaften hat denselben Aufbau und entspricht den oben aufgezeigten Formaten. Das Regelwerk unterscheidet sich hier lediglich von den Bezeichnungen und der Anzahl an beschriebenen Eigenschaften.

[38] Quelle Webseite: Wizards of the Coast Inc. [31]

Die Wertzuweisungen können wahlweise durch direkte Auswahl, Listeneinträge und Berechnungen stattfinden. Auch diese Aspekte wurden bereits im Abschnitt „5.3.1 Das schwarze Auge“ demonstriert. Deshalb werden an dieser Stelle nicht alle bereits aufgezeigten Regeln erneut aufgelistet.
Die bisher in der DSL beschriebenen Auswahlmöglichkeiten genügen zum Abdecken der Generierungsmöglichkeiten noch nicht komplett. Im Kapitel 3 auf Seite 17 des Handbuches werden z.B. die Möglichkeiten zur Generierung der Wertzuweisungen beschrieben. Es wird auf drei unterschiedliche Methoden verwiesen. In der momentanen Form der DSL kann nur eine einzige Form der Wertzuweisung stattfinden. Dieser Generator kann zukünftig allerdings um weitere Methoden erweitert werden, die die folgenden drei Zuweisungsmethoden darstellen können:

Methode 1

Eine Liste von vorgegebenen Werten kann den einzelnen Eigenschaften beliebig zugeordnet werden. Aufgelistet sind hier beispielsweise[16, 14, 13, 12, 11, 10]

Methode 2

Individualisierung vorgegebener Startwerte [8, 10, 10, 10, 10, 10] mit 22Steigerungspunkten zur Verbesserung der Werte. Anhand einer Kostentabelle können die Werte dann weiter modifiziert werden.

Methode 3

Zufallsbestimmung. Man würfelt vier Mal mit einem sechsseitigen Würfel und addiert die drei besten Werte.

Alle drei Methoden sind derzeit nicht enthalten, basieren aber auf bereits in der DSL beschriebenen Konzepten und Methoden und könnten somit ebenfalls als weitere Variante für den Generator hinzugefügt werden.

5.3.3 FreeFate

Das Regelwerk ist frei verfügbar über das Internet auf der Homepage von FreeFate erhältlich. [39]

FreeFate ist ein Spielsystem, dass mehr auf beschreibende und interkommunikative Darstellung der Eigenschaften ausgelegt ist.
Grundsätzlich gelten allerdings auch hier die Einstufungen in textuelle und wertbasierte Eigenschaften. FreeFate unterscheidet dabei vier unterschiedliche Gruppen.

[39] Quelle Webseite: FreeFate.de [8]

Konzept

Es beschreibt, was den Charakter zentral ausmacht. Lapidar könnte man dies auch als eine Form „Label“ oder Klischee bezeichnen. Beispielsweise wird angeführt: Der knallharte Detektiv oder Die geniale Wissenschaftlerin.
Diese Eigenschaften werden bereits oben dargestellt durch textuelle Eigenschaften repräsentiert.

Aspekte

Sie beschreiben feingranularer die Eigenschaften des Spielcharakters. Es wird explizit darauf hingewiesen, dass auch diese Eigenschaften keine Werte besitzen.
Angeführte Beispiele: „Immer mitten drin“ oder „Der Tod meines Partners verfolgt mich im Traum“
Sofern die Eigenschaften auch negativ ausgelegt werden können und zum Nachteil angeführt werden können, erhält der Charakter Fatepunkte als Ausgleich. (Dieses fällt mit in den Bereich Steigerungen und wird daher zu den Spielmechanismen gezählt)
Insgesamt können 8 Aspekte bei der Charaktererschaffung zugeteilt werden. Es ist allerdings auch möglich nicht alle Aspekte zu Beginn zu verteilen um im Laufe des Spiels eine weitere Wendung für den Charakter herbeizuführen. (Dieser Teil kann durch leere bzw. nicht zugewiesene Eigenschaften/Aspekte realisiert werden. Allerdings ist dies erst im Bereich der Codegenerierung und der damit zusammenhängenden Überprüfung möglich.

Fertigkeiten

Sie beschreiben Talente und erlernbare Fähigkeiten. Das Spiel sieht 28 unterschiedliche Fertigkeiten vor, von denen 10 ausgewählt werden. Diese können direkt durch wertbasierte Eigenschaften in der DSL beschrieben werden.
Zusätzlich zu einem repräsentativen Zahlenwert wird der Bewertung der Fähigkeit auch eine textuelle Beschreibung zugewiesen.
Dieses kann in Form einer decisionTable ohne Bedingungen dargestellt werden. In der momentanen Implementation der Tabelle fehlt lediglich ein zusätzliches beschreibendes Textfeld ähnlich der Variable „description“ in den Attributen.
Die Verteilung der Fertigkeiten auf eine gleichmäßige Breite wird durch eine sogenannte Fertigkeiten-Pyramide repräsentiert. Konkret bedeutet dies, dass nur folgende Fertigkeitskombinationen erlaubt sind:

1x Großartig (+4)
2x Gut (+3)
3x Ordentlich (+2) und
4x Durchschnittlich (+1).
(in der Summe 20Punkte)

Die Vergabe der Fertigkeitspunkte geschieht durch direkte Auswahl bzw. Zuweisung des Zahlenwertes. Dies ist in der DSL und dem Zuweisungsgenerator modelliert.

Stunts

Sie stellen Steigerungsmöglichkeiten der Fertigkeiten dar. Es können nur Fertigkeiten als Stunt ausgewählt werden, wenn die Fertigkeit besser als „mäßig“ ist. Insgesamt können 4 Stunts gewählt werden, aber genauso wie bei den Aspekten können sie im Laufe des Spiels ergänzt werden.
An dieser Stelle wird auch eine Grenze offenbart bzgl. der Beschreibbarkeit für einen Computer. Stunts können kontextbezogen in der Spielentwicklung eingebunden und verwendet werden. Diesen Raum an Möglichem zu beschreiben oder strukturell darzustellen, ist in Form einer DSL nicht möglich und gegebenenfalls auch grundsätzlich nicht darstellbar. Computer können in erster Line vorhandenes und konkretes beschreiben bzw. definieren.

Ein Abschnitt der Charaktererstellung befasst sich mit der Zuweisung von Ausrüstungsgegenständen. Wie bereits erwähnt ist dieser Aspekt der DSL zur Charaktererstellung überhaupt nicht implementiert.

Abschließend beschreibt die Charaktererstellung in FreeFate die Beschreibung zweier Eigenschaften, die die körperliche und geistige Belastung des Charakters beschreiben sollen. In der jetzigen Form kann dies noch nicht vollständig beschrieben werden, da der Wert neben einem festen Zahlenwert von 5 zusätzlich noch anhand eines Tabelleneintrages in Abhängigkeit eines Fertigkeitswertes (z.B. Ausdauer oder Willenskraft) beeinflusst wird.
Auch diese Form des Generators ist noch nicht in der DSL enthalten, kann aber um diese Generatorform erweitert werden. Die zugrundeliegenden Regeln sind dafür bereits vorhanden (Listen und DecisionTables).

6 Zusammenfassung

6.1 Zusammenfassung

Das Ziel der Arbeit war es eine domänenspezifische Sprache zu entwickeln, die die Charaktererstellung für Rollenspielregelwerke beschreiben kann und in eine computerverständliche Form umzuwandeln. Umgesetzt wurde dieses Ziel unter Verwendung des XText-Frameworks.

Der Hauptfokus der Projektarbeit liegt auf der Überprüfung der Machbarkeit einer solchen Beschreibung der Charaktererstellung. Es ist nicht darauf abgezielt worden eine komplette oder vollständige DSL zu entwickeln. Im Rahmen der Entwicklung und des Abgleichens mit Beispielregelwerken konnten grundlegende Gemeinsamkeiten der Regelwerke aufgezeigt werden und zusätzlich weitere Möglichkeiten und Notwendigkeiten der Erweiterung der verwendeten DSL-Konzepte ausfindig gemacht werden.

In fast allen noch nicht abgedeckten Beschreibungsfällen konnte anhand der bereits vorhandenen Regeln in der DSL aufgezeigt werden, dass die benötigten Konstrukte und Methoden zur Erstellung der nötigen Beschreibung existieren und umgesetzt werden können. Somit ist die Machbarkeit belegt.

Einzelne Sonderfälle, die einen unendlichen Lösungsraum haben, können höchstwahrscheinlich nicht in endlicher Form für einen Computer zugänglich gemacht werden und bleiben (derzeit) noch der menschlichen Fantasie vorbehalten.

Wert-Generatoren, Listen Konstrukte und Steigerungen sind teilweise nur prototypisch umgesetzt worden oder nicht umgesetzt worden um den Projektrahmen zu begrenzen.

Das Projekt hat nur die Modelle umgesetzt. Zur maschinellen Nutzung der DSL sind noch weitere Schritte nötig, z.B. die Codegenerierung. Diese wurde anhand von compile-Methoden ebenfalls prototypisch vorgeführt.

6.2 Bewertung

Die Verwendung eines großen Frameworks hat den Arbeitsaufwand im Rahmen des Projektes deutlich verschoben. Es hat die ursprünglich als großen Aufwand eingeschätzten Bereiche des Compilerbaus komplett in den Hintergrund verdrängt. Die Bereitstellung eines Compilers für die DSL ist unumgänglich. Der Aufbau und die Konstruktion des Compilers beinhalten viele Stolperstricke und Möglichkeiten der Fehleranfälligkeit besonders bei einem hohen Ausmaß an Iterationsschritten in der Entwicklung.

Ein Kriterium, das mich von einem Test-Driven-Development der DSL abgehalten hat, waren die häufigen Iterationen, die die Struktur der DSL an sich verändert haben und somit auch eine häufige und fortlaufende Anpassung der Tests vorausgesetzt hätten. An dieser Stelle muss man sich bewusst sein, dass z.B. beim Testen und verbessern eine kleine Syntaxformel oder von Schlüsselwörtern bereits ebenfalls eine Überprüfung der Tests einher geht und diverse Stellen im Entwicklungsprozess bei jeder Veränderung überprüft werden müssen.

Der Erleichterung durch die Bereitstellung des nötigen Compilers und weiterer Hilfsklassen stehen jedoch die unendlichen Tiefen eines riesigen Frameworks gegenüber. Framework-Dokumentationen und Tutorials beschreiben im Regelfall nur den Soll- oder Idealzustand und nur selten bis gar nicht mögliche Fehler.
Der Zeitaufwand für die Fehlersuche ist jedoch wahrscheinlich wesentlich geringer als das Rad neu zu erfinden um einen umfangreichen und funktionierenden Compiler zu entwerfen.

Die größte Überraschung im Projektrahmen war festzustellen, dass der vom Framework bereitgestellte Webeditor primär nur zur Darstellung und Bearbeitung einer DSL angedacht ist. Es gibt keine angedachten Methoden zur Codegenerierung oder weiterführenden Nutzung des Ergebnisses. Individuelle Implementierungen sind selbstverständlich immer möglich, allerdings zielt ein Framework auch auf eine weite Bandbreite von Anwendungsfällen ab, die sie unterstützen können. Der DSL-Editor in Eclipse ist für die Generierung von Code ausgelegt und hier gilt diese Verwendung als zentraler Anwendungsfall.

Auch wenn es mir nicht möglich war alle Fälle der Regelwerke im Rahmen des Projektes abzudecken, freut es mich umso mehr, dass grundlegenden Konzepte die Basis für eine Weiterentwicklung und Vervollständigung bieten und in den Beispielregelwerken bereits absehbar ist, dass die fehlenden Elemente abbildbar sind.

6.3 Ausblick

Die Konstruktion der Grundzüge der DSL hat gezeigt, dass viele unterschiedliche Sachverhalte der Rollenspiele abgebildet werden können. Es ist durchaus denkbar, dass die Regelwerke vollständig mit DSLs modelliert werden können und somit eine leichte Überführung der Regelwerke möglich ist. Ein Ansatz zur Realisierung wäre eine modulare Beschreibung der Teilaspekte in ähnlicher Form wie der Charaktererstellung. Ergänzend sei an dieser Stelle erwähnt, dass das XText-Framework die Möglichkeit bietet mehrere DSLs miteinander zu verbinden bzw. zu referenzieren. Ein möglicher Ansatzpunkt für eine ganzheitliche Darstellung eines Rollenwerkes.
In den Beispielregelwerken wurde ersichtlich, dass die Kombinationsmöglichkeiten der Wert-Generatoren sehr viele verschiedene Varianten ermöglicht und eine DSL in vielen Fällen für neue Kombinationsmöglichkeiten angepasst bzw. erweitert werden muss. Eine universelle Lösung der Wertzuweisung ist wahrscheinlich aufgrund der unendlichen Kombinationen nicht möglich.

Es bleibt der faszinierende Aspekt der enormen Ausdrucksstärke und Vielfalt, die mit einer DSL umgesetzt werden können.

Literaturverzeichnis

[1] M. Fowler, Domain-Specific Languages, 2010.

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

[3] S. P. Fannon, Gamer's Bible.

[4] Wikipedia, „Wikipedia: Formale Sprache," [Online]. Available: https://de.wikipedia.org/wiki/Formale_Sprache. [Zugriff am 5 12 2016].

[5] Wikipedia, „Wikipedia: Domänenspezifische Sprachen," [Online]. Available: https://de.wikipedia.org/wiki/Dom%C3%A4nenspezifische_Sprache. [Zugriff am 5 12 2016].

[6] Wikipedia, „Wikipedia: Pen-&-Paper Rollenspiel," [Online]. Available: https://de.wikipedia.org/wiki/Pen-%26-Paper-Rollenspiel. [Zugriff am 5 12 2016].

[7] Wikipedia, „Wikipedia: Rollenspiel," [Online]. Available: https://de.wikipedia.org/wiki/Rollenspiel_(Spiel). [Zugriff am 5 12 2016].

[8] K. Schad, „FreeFate.de," [Online]. Available: http://freefate.de/. [Zugriff am 5 12 2016].

[9] Ulisses-Spiele, „Ulisses-Spiele.de," [Online]. Available: http://www.ulisses-spiele.de/. [Zugriff am 5 12 2016].

[10] A. Aho, M. Lam, R. Sethi und J. Ullman, Compilers: Principles, 1986.

[11] Wikipedia, „Wikipedia: Formale Grammatik," [Online]. Available: https://de.wikipedia.org/wiki/Formale_Grammatik. [Zugriff am 5 12 2016].

[12] M. Fowler, „Martin Fowler: Language Workbench," [Online]. Available: http://martinfowler.com/bliki/LanguageWorkbench.html. [Zugriff am 5 12 2016].

[13] Wikipedia, „Wikipedia: IntelliJ IDEA," [Online]. Available: https://en.wikipedia.org/wiki/IntelliJ_IDEA. [Zugriff am 5 12 2016].

[14] Wikipedia, „Wikipedia: Eclipse,“ [Online]. Available: https://en.wikipedia.org/wiki/Eclipse_(software).

[15] Wikipedia, „Wikipedia: Das Schwarze Auge,“ [Online]. Available: https://de.wikipedia.org/wiki/Das_Schwarze_Auge. [Zugriff am 5 12 2016].

[16] Wikipedia, „Wikipedia: Dungeons & Dragons,“ [Online]. Available: https://de.wikipedia.org/wiki/Dungeons_%26_Dragons. [Zugriff am 5 12 2016].

[17] Eclipse-Foundation, „Eclipse: XText,“ [Online]. Available: https://eclipse.org/Xtext/. [Zugriff am 5 12 2016].

[18] JetBrains, „JetBrains: MPS,“ [Online]. Available: https://www.jetbrains.com/mps/. [Zugriff am 5 12 2016].

[19] Wikipedia, „Wikipedia: Unified Modeling Language,“ [Online]. Available: https://de.wikipedia.org/wiki/Unified_Modeling_Language. [Zugriff am 5 12 2016].

[20] Wikipedia, „Wikipedia: Decision Table,“ [Online]. Available: https://en.wikipedia.org/wiki/Decision_table. [Zugriff am 5 12 2016].

[21] Wikipedia, „Wikipedia: Erstellungsprozess,“ [Online]. Available: https://de.wikipedia.org/wiki/Erstellungsprozess.

[22] Gradle, „Gradle.org,“ [Online]. Available: https://gradle.org/. [Zugriff am 5 12 2016].

[23] Wikipedia, „Wikipedia: Eclipse Modeling Framework,“ [Online]. Available: https://de.wikipedia.org/wiki/Eclipse_Modeling_Framework. [Zugriff am 5 12 2016].

[24] Eclipse-Foundation, „Eclipse: XText Forum,“ [Online]. Available: https://www.eclipse.org/forums/index.php/f/27/. [Zugriff am 5 12 2016].

[25] Wikipedia, „Wikipedia: Servlet,“ [Online]. Available: https://de.wikipedia.org/wiki/Servlet. [Zugriff am 5 12 2016].

[26] Eclipse-Foundation, „Eclipse: Jetty,“ [Online]. Available: https://eclipse.org/jetty/. [Zugriff am 5 12 2016].

[27] Wikipedia, „Wikipedia: EBNF,“ [Online]. Available: https://de.wikipedia.org/wiki/Erweiterte_Backus-Naur-Form. [Zugriff am 5 12 2016].

[28] Eclipse-Foundation, „Eclipse: XText Dokumentation - Codegenerierung,“ [Online]. Available: https://eclipse.org/Xtext/documentation/103_domainmodelnextsteps.html. [Zugriff am 5 12 2016].

[29] Helden.de, „DSA3 Charakterbogen im Excelformat,“ [Online]. Available: http://www.helden.de/downs/fantasy/item/975-dsa-3-charakterbogen-im-exelformat. [Zugriff am 5 12 2016].

[30] Taladas.de, „DSA-Kurzregelwerk,“ [Online]. Available: http://www.taladas.de/Taladas/html/RPGRegelnDSA15.html. [Zugriff am 5 12 2016].

[31] Wizards, „Webseite: Wizards of the Coast,“ [Online]. Available: https://dnd.wizards.com/. [Zugriff am 5 12 2016].

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 09. Dezember 2016 _____