Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit Anwendungen 1

Dennis Dedaj

Game Engineering

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Dennis Dedaj

Game Engineering

Seminarausarbeitung im Rahmen der Veranstaltung Anwendungen 1
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft

Abgegeben am 25. Juli 2008

Inhaltsverzeichnis

Tabellenverzeichnis **4**

Abbildungsverzeichnis **5**

1 Einführung **6**
1.1 Motivation und Zielsetzung 6
1.2 Struktur der Arbeit 7

2 Grundlagen **8**
2.1 Spielgenres 8
2.1.1 Action Games 8
2.2 MDSD - Model Driven Software Development 10
2.3 Modellnotationen 11

3 Methodik **12**
3.1 Auswahl eines Vorgehensmodells 12
3.2 Erste Identifizierung von Darstellungsmöglichkeiten für Action Games 12
3.2.1 Tokens 13
3.2.2 Interaktionsmatrix 14
3.2.3 Token-Zustände 14
3.2.4 Game Flow Diagramme 14
3.3 Auswählen der Autorenwerkzeug-Technologie 15
3.4 Auswählen der Transitionstechnologie 15

4 Technologien **16**
4.1 Autorenwerkzeuge 16
4.2 Technologie für das Erzeugen von Quellcode 17

5 Zusammenfassung **19**

Literaturverzeichnis **21**

Tabellenverzeichnis

2.1 Spielgenres 9

4.1 Übersicht der Evaluation von grafischen Werkzeugen 18

Abbildungsverzeichnis

1.1 Schnittstelle zwischen Entwurf und Programmierung 6

2.1 Ein einfaches Domänenmodell in der UML-Notation 11

3.1 Storyboard eines einfachen Action Games 13
3.2 Die Tokenhierarchie als UML-Klassendiagramm 13
3.3 Die Darstellung der möglichen Interaktion zwischen verschiedenen Tokens . 14
3.4 Die Zustände und deren Übergänge eines Tokens 15

1 Einführung

In diesem Kapitel wird die Motivation für diese Ausarbeitung verdeutlicht, das Ziel definiert und abschließend eine kurze Übersicht über die Struktur der Arbeit gegeben.

1.1 Motivation und Zielsetzung

Die Entwicklung von Computerspielen ist interdisziplinär. Das sogenannte Gamedesign (Erstellung eines Spielentwurfs) wird zum Großteil von kreativen Künstlern entwickelt. Bei der Erstellung eines Gamedesigns wirken 2D- und 3D-Grafiker, Gamedesigner, Musiker, Produzenten und weitere Rollen mit. Nach der Erstellung eines solchen Gamedesigns sind Programmierer gefragt, die geforderten Konzepte umzusetzen.
Die Transition von einem kreativen Gamedesign zu einem softwaretechnischem Produkt erfordert eine Schnittstelle zwischen den Gamedesignern und den Programmierern. In Abbildung 1.1 wird dies verdeutlicht. Diese Schnittstelle könnte durch eine unterstützende Anwendung verbessert werden.

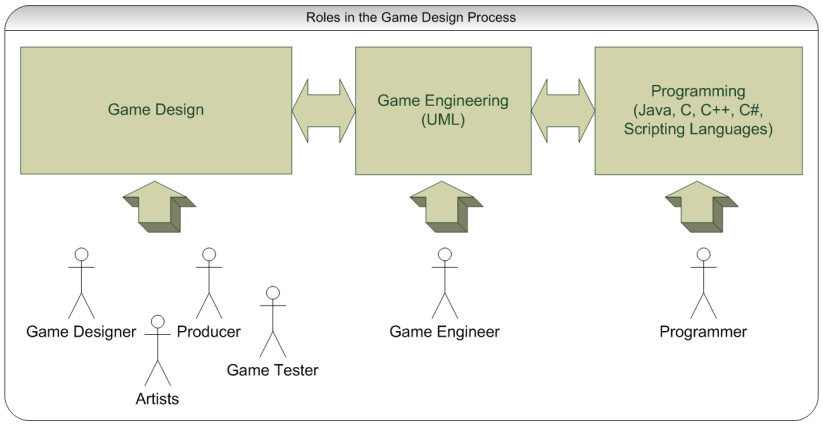

Abbildung 1.1: Schnittstelle zwischen Entwurf und Programmierung

Ziel dieser Ausarbeitung ist es einen Ansatz zu liefern auf dem eine, wie im vorigen Abschnitt erwähnte Schnittstelle, aufgebaut werden kann. Perspektivisch soll die Entwicklung einer unterstützenden Anwendung für Gamedesigner bzw. Programmierer betrachtet werden. Es soll eine Grundlage für verschiedene Darstellungsmöglichkeiten von Computerspielen erarbeitet werden, um notwendige Bausteine eines Autorenwerkzeuges zu identifizieren. Diese Arbeit soll weiterhin die technologischen Grundlagen für die Entwicklung eines Autorenwerkzeuges vorstellen.
In der Masterarbeit könnte die Entwicklung eines generischen Autorenwerkzeuges für Gamedesigner in Betracht gezogen werden.

1.2 Struktur der Arbeit

Nach dieser Einführung wird in Kapitel 2 die Anwendungsdomäne Computerspiele mit ihren verschiedenen Spielgenres vorgestellt. Anschließend wird eine kurze Einführung in modellgetriebene Softwareentwicklung und ein Überblick über geeignete Modellnotationen gegeben. Das Kapitel 3 „Methodik“ stellt die wichtigsten Schritte zum Erreichen des Ziels dar. Es wird auf das Vorgehensmodell eingegangen, mögliche Darstellungsformen der Spielinhalte vorgestellt und in den letzten beiden Kapiteln wird die Auswahl der technologischen Basis für das Autorenwerkzeug und die Transitionstechnologie erläutert. In Kapitel 4 wird auf konkrete Technologien zur Erstellung eines Autorenwerkzeuges eingegangen. Die Zusammenfassung in Kapitel 5 fässt die Ergebnisse zusammen, beschreibt erkannte Risiken und nächste Schritte.

2 Grundlagen

Hier werden die Grundlagen für diese Arbeit erläutert. Vorerst muss die Domäne erläutert werden, auf welche sich die Arbeit bezieht: Computerspiele. Im Folgenden wird ein Überblick über verschiedene Spielgenres gegeben, um später geeignete Darstellungsmöglichkeiten zu erarbeiten. Es ist erforderlich, dass zwischen den Genres unterschieden wird. Jedes Genre erfordert unterschiedliche Herangehensweisen und enthält andere Tokens[1]. Weiterhin wird sich diese Arbeit auf die Subdomäne „Action Games“ beschränken.
Daraufhin wird modellgetriebene Softwareentwicklung eingeführt, um zu verdeutlichen wie der technische Weg von einem abstrakten Modell zu konkretem Quellcode bestritten werden könnte.
Bevor das nächste Kapitel „Methodik“ begonnen wird, werden verschiedene Modellnotationen vorgestellt.

2.1 Spielgenres

Adams u. Rollings [2006] identifizieren acht wichtige Genres. Für jedes dieser Genres werden eigene Spielelemente und Spielkonzepte beschrieben. Diese sind in verkürzter Form in Tabelle 2.1 zusammengefasst. Im Weiteren wird, um eine Vertiefung zu ermöglichen, konkreter auf den Genre „Action Games“ eingegangen. Das folgende Kapitel 2.1.1 erläutert diesen Genre ausführlicher.

2.1.1 Action Games

Zur Bildung des gemeinsamen Verständnisses über diesen speziellen Genre werden zuerst die Tokens genauer erklärt und daraufhin ein kurzer Überblick über den Spielablauf gegeben. Action Games enthalten nach Tabelle 2.1 folgende wichtigen Tokens:

- Avatar - Die Repräsentation des Spielers.

[1] Die Objekte eines Spiels werden Token genannt, um die Verwechslung mit C/C#-Objekten zu vermeiden. (Morris u. Rollings [2004])

Genre	Spielelemente	Spielkonzepte
Adventure Games	Objekte und Charaktere	Rätsel lösen und Geschichten erleben
Strategy Games	Einheiten und Fabriken	Kämpfen, Diplomatie, Spionage, Religion oder Kultur
Puzzle Games	Diverse, meist abstrahierte Objekte	Rätsel und Denkaufgaben lösen
Artificial Life	Alle, die auch in der Realität vorkommen können	Bedürfnisse decken, Fähigkeiten trainieren
Construction and Management Games	Gebäude, Fabriken, Straßen, Gewässer ... Berater, Bürger, Arbeiter ...	Simulation von Markt, Ökonomie, Produktion, Kultur ...
Sports Games	Spielfelder, Spielgegenstand, Spieler, Gegner und Teams	Spielregeln Physiksimulation
Vehicle Simulation	Autos, Flugzeuge, Züge, Schiffe, Rennbahnen oder andere Umgebung	Realistische Physiksimulation
Action Games	Gegner, Sammelobjekte, Teleporter und Level	Leben / Energie, Zeitlimit und Punkte

Tabelle 2.1: Die verschiedenen Spielgenres mit Ihren wichtigsten Elementen und Konzepten nach Adams u. Rollings [2006]

- Gegner
- Sammelobjekte - Tokens, die von dem Spieler gesammelt werden können.
- Teleporter - Transportieren den Spieler zu einem anderen Ort.
- Level - Virtuelle Welten, die von dem Spieler durchspielt werden.
- Leben / Energie - Darstellung der Anzahl der Leben bzw. der Lebensenergie des Spielers.
- Zeitlimit - Innerhalb dessen der Spieler ein bestimmtes Ziel (z.B. Ende eines Levels) erreichen muss.
- Punkte - Quantifizierbare Darstellung des Spielerfolgs.

Bei dieser Art von Spielen ist es das Ziel des Spielers, seinen Avatar so durch die Levels zu steuern, dass er dabei möglichst wenig Leben bzw. Energie verliert und gleichzeitig einen möglichst hohen Punktstand erreicht. Die Gegner werden in den Levels platziert, um dem Spieler die Verfolgung seiner Ziele zu einer Herausforderung zu machen. Durch verschiedene Erweiterungen wie das Platzieren von Teleportern und Sammelobjekten kann die Komplexität solcher Spiele beliebig erweitert werden.

2.2 MDSD - Model Driven Software Development

Dieses Kapitel gibt einen Überblick über Model Driven Software Development (MDSD) und legt damit das Fundament für die technologischen Ausführungen dieser Arbeit. Die Arbeit folgt der Definition von MDSD nach Efftinge u. a. [2007]:

> Modellgetriebene Softwareentwicklung (Model Driven Software Development, MDSD) ist ein Oberbegriff für Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen.

Die drei zugrunde liegenden Prinzipien werden im Folgenden erläutert.

Formale Modelle sind in diesem Kontext als „formal“ zu bewerten, wenn das Modell einen Aspekt -nicht aber alles- eines Softwaresystems vollständig beschreibt und automatisiert in ausführbaren Code überführt werden kann (nach Efftinge u. a. [2007]). Formale Modelle tauchen in unterschiedlichen Formen auf.
Es gibt textuelle formale Modelle wie z.B. XText. XText stellt eine vereinfachte extended Backus-Naur-Form (EBNF) zur Darstellung von abstrakter und konkreter Syntax zur Verfügung, aus welcher dann ein Parser, ein Metamodell und ein Eclipse-basierter Editor zur Verfügung gestellt wird (siehe openArchitectureWare Group [2008]).
Weiterhin ist die Verwendung von grafischen Modellen stark verbreitet. Als die bekannteste Notation von grafischen Modellen ist UML 2.0 zu erwähnen. Bei UML-Modellen ist jedoch erneut zu erwähnen, dass auch diese nur dann als formal zu bezeichnen sind, wenn sie das Modell auch vollständig beschreiben. Das ist durch die Verwendung der Object Constraint Language (siehe OMG [2005]) erreichbar, da mit dieser Regeln zwischen verschiedenen Objekten definiert werden können.

Erstellen von lauffähiger Software ist das Leitziel bei MDSD. Das Endprodukt sollte eine lauffähige Software sein. Hierbei sollten zwei Arten der automatischen Erzeugung von Quellcode unterschieden werden.
Ein Generator ist Software, die -ähnlich einem Compiler- aus einem Modell Quelltext erzeugt, welcher daraufhin ausgeführt werden kann. Generatoren gehören somit dem Build-Prozess an.
Ein Interpreter hingegen liest das Modell erst zur Laufzeit ein und führt -abhängig von bestimmten Regeln oder Ínhalten des Modells- verschiedene Aktionen aus.

Automatisierung ist das wesentliche Merkmal der modellgetriebenen Softwareentwicklung. Um die Automatisierung erreichen zu können muss das Modell vollständig spezifiziert werden, da es nicht nur zur manuellen Implementierung verwendet wird.

2.3 Modellnotationen

Die Auswahl von geeigneten Modellnotationen ist für das Erstellen von Tools zur automatischen Generierung von Quellcode wichtig, da die verschiedenen Notationen wiederum anderes Vorgehen bei dem Erstellen des Quellcodes erfordern. Wie bereits in dem vorigen Kapitel erwähnt, kann zwischen textuellen und grafischen Repräsentationen differenziert werden.

Durch den interdisziplinären Charakter des Gamedesigns ist es notwendig eine allgemein verständliche Notation zu wählen. Eine textuelle Notationssprache eignet sich in der Spiele Domäne nur bedingt, da die Anwender eventuell keine Programmierkenntnisse besitzen. Als kurzes Beispiel von textuellen und grafischen Notationen wird ein kleines Beispiel eingeführt. Das Domänenmodell kann wie folgt allgemein beschrieben werden:

```
Entity{
  String name
  Adresse adresse
}
Entity Adresse{
  String strasse
  String plz
  String ort
}
```

In einer textuellen Notation, hier eine XText-Grammatik, würde folgendes Modell resultieren:

```
Model      : (entities+=Entity)+;
Entity     : "Entity"' name=ID "}" (attributes+=Attribute)* "}";
Attribute  : type=ID name=ID;
```

Durch die Verwendung einer grafischen Beschreibungssprache -hier UML- kann eine vereinfachte Darstellung erreicht werden: Im weiteren Verlauf der Arbeit wird die UML-Notation

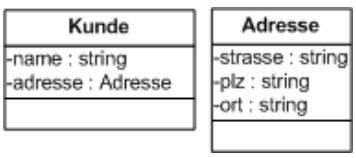

Abbildung 2.1: Ein einfaches Domänenmodell in der UML-Notation

vorerst bevorzugt, da diese besonders einfach zu erstellen ist und für das grundsätzliche Verständnis jedem Leser klar sein sollte.

3 Methodik

In diesem Kapitel wird auf das geplante Vorgehen zum Erreichen der in der Einführung in Kapitel 1.1 genannten Ziele ausgeführt. Im folgenden Kapitel 3.1 wird zuerst auf die Auswahl des Softwareentwicklungsprozess eingegangen. Daraufhin werden erste Schritte zur Auswahl geeigneter Darstellungsmöglichkeiten erarbeitet. Im Anschluss werden nötige Arbeitsschritte in Bezug auf das Autorenwerkzeug und die Transitionstechnologie erläutert.

3.1 Auswahl eines Vorgehensmodells

Bei der Spieleentwicklung und der modellgetriebene Softwareentwicklung werden iterative Vorgehensmodelle bevorzugt (siehe Morris u. Rollings [2004] und Efftinge u. a. [2007]). Das hängt primär davon ab, dass sich beide während der Entwicklung stark ändern.
Spiele müssen während der Entwicklung immer weiter- und neuentwickelt werden. Speziell der tatsächliche Spaßfaktor für den Spieler lässt sich nicht durch eine noch so durchdachte Architektur im Vorwege sichern. Aber auch in anderen Bereichen wie die Benutz- und Bedienbarkeit sind stetige Verbesserungen zu leisten (siehe Adams u. Rollings [2006]).
MDSD erfordert iterative Vorgehensweisen, da die Komplexität von Metaarchitekturen nur schwer zu überblicken ist und durch iterative Entwicklung die Qualität der Architekturen verbessert werden kann. Efftinge u. a. [2007] betonen dabei besonders, dass der erste Entwurf meist nicht in jeder Hinsicht vollkommen ist.

3.2 Erste Identifizierung von Darstellungsmöglichkeiten für Action Games

In Kapitel 2.1.1 wurden bereits die essenziellen Spielelemente von Action Games identifiziert. In diesem Kapitel geht es nun darum, geeignete Darstellungsmöglichkeiten für die Elemente und weiterführend auch für deren Interaktionen und Zustandsänderungen zu erarbeiten. Die Basis zu den hier entwickelten Darstellungsmöglichkeiten bieten die in Dedaj [2008] zusammengefassten Darstellungsmöglichkeiten. Im Zuge der Masterarbeit gilt es, diese zu Verfeinern und weitere Beschreibungen (unter anderem textuelle Notationen) zu

entwickeln. Als Basis für sämtliche folgenden Beispieldarstellungen wird ein kleines Action Game verwendet, welches in Abbildung 3.1 in Form eines Storyboards dargestellt ist.

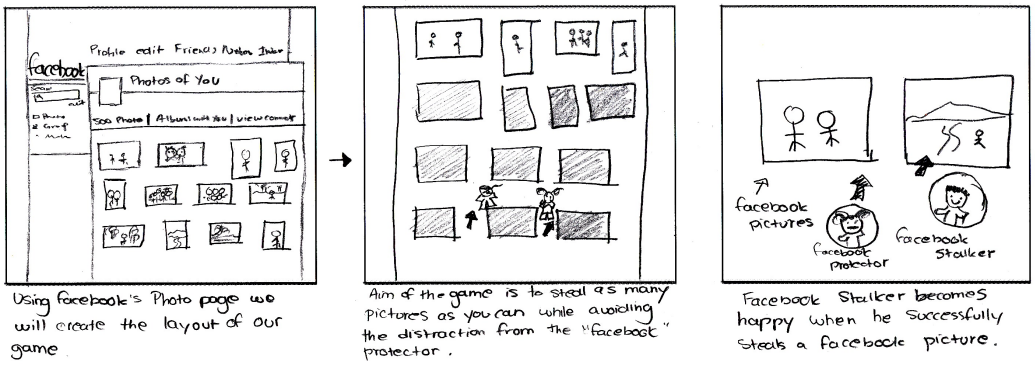

Abbildung 3.1: Storyboard eines einfachen Action Games

3.2.1 Tokens

Die Darstellung von Tokens, den Spielobjekten, kann nach Morris u. Rollings [2004] in der Form von UML-Klassendiagrammen erfolgen. Die konkreten Token Stalker, Picture, FacebookProtector, Wall und Score sind Unterklassen des abstrakten Tokens. Das abstrakte Token bietet Schnittstellen, um Grundfunktionalitäten wie 'zeichnen' und 'skripten' zu unterstützen. In Abbildung 3.2 ist die Hierarchie verdeutlicht.

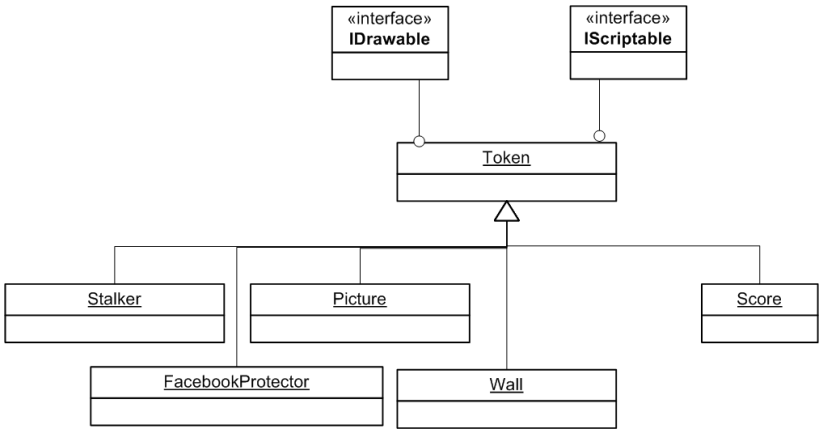

Abbildung 3.2: Die Tokenhierarchie als UML-Klassendiagramm

3.2.2 Interaktionsmatrix

Eine weitere und wichtige Darstellung ist die Interaktionsmatrix (Morris u. Rollings [2004]). Eine beispielhafte Interaktionsmatrix ist in Abbildung 3.3 erkenntlich. Mit dieser wird dar-

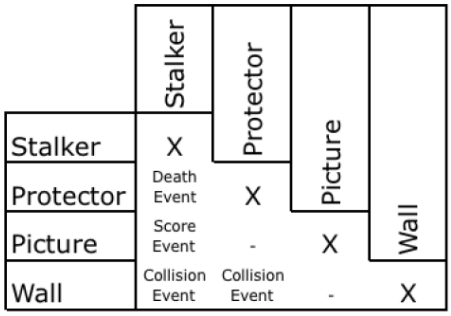

Abbildung 3.3: Die Darstellung der möglichen Interaktion zwischen verschiedenen Tokens

gestellt, ob und wie die verschiedenen Tokens miteinander agieren können. Sie bietet eine besonders einfach verständliche Darstellung aus welcher leicht Quellcode abgeleitet werden kann. Für die Masterarbeit entsteht in Bezug auf diese Darstellung die Frage der Machbarkeit für die Umsetzung der tabellarischen Form in dem Autorenwerkzeug.

3.2.3 Token-Zustände

In dem vorigen Unterpunkt wurde bereits auf die Darstellung der Interaktion zwischen verschiedenen Tokens eingegangen. Mit der folgenden Darstellung soll der Fokus auf die einzelnen Tokens und deren Zustandsänderung gesetzt werden. Sobald ein Token -oft aber nicht zwingend durch Interaktion mit einem weiteren Token- seinen Zustand ändert, muss es auch für die Zustände und insbesondere den Zustandsübergängen eine geeignete Darstellung geben. Eine mögliche Darstellung ist in Abbildung 3.4 ersichtlich.

Diese Darstellungsform bietet einen weiteren Vorteil. UML-Diagramme bieten mit ihren Zustandsdiagrammen eine äquivalente Darstellungsmöglichkeit, welche bereits durch bestehende Tools unterstützt wird.

3.2.4 Game Flow Diagramme

Um ein größeres Bild des gesamten Spiels darzustellen, erarbeiten Taylor u. a. [2006] eine geeignete Darstellung, um einen Überblick des Spielablaufs von Computerspielen zu erzeugen.

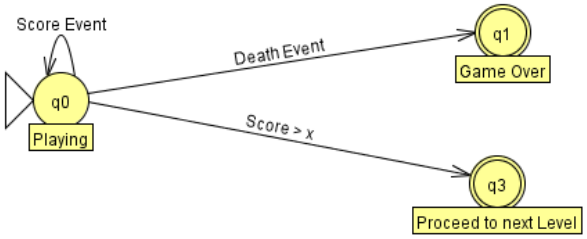

Abbildung 3.4: Die Zustände und deren Übergänge eines Tokens

3.3 Auswählen der Autorenwerkzeug-Technologie

Es soll ein Werkzeug für Anwender aus dem Bereich Spieleentwicklung erstellt werden mit dem es möglich ist auf besonders einfach und schnelle Art und Weise Modelle von Computerspielen zu erzeugen. In Kapitel 2.3 wurde UML bereits als mögliche Darstellungsform identifiziert. Im Zuge der Masterarbeit wäre es unerlässlich weitere Darstellungsmöglichkeiten zu evaluieren.
Das Autorenwerkzeug kann entweder aus einer eigenen Implementierung entstehen oder auf Basis eines bestehenden Werkzeuges entwickelt werden. Außerdem ist bei beiden Ansätzen die Nutzung von bereits bestehenden Frameworks zu berücksichtigen. Ein Anfang zu den genannten Punkten wird in Kapitel 4.1 getan.

3.4 Auswählen der Transitionstechnologie

Das grafische Bearbeiten von Spielinhalten ist ein Schwerpunkt der Entwicklung eines Autorenwerkzeuges. Die anspruchsvollere und komplexere Aufgabe ist jedoch die Transition der erstellten Modelle in lauffähigen Quellcode. Auch hier sollten verschiedene Technologien, bestehende Werkzeuge und Frameworks tiefgreifend analysiert, miteinander verglichen und evaluiert werden. Einen Überblick über mögliche Technologien gibt Kapitel 4.2.

4 Technologien

Für die Erstellung eines Autorenwerkzeuges ist ein grafisches Werkzeug von Vorteil, da die Autoren nicht Programmieren möchten bzw. nicht das erforderliche Fachwissen mitbringen. Eine Technologieempfehlung wird in dem folgenden Kapitel gemacht. Daraufhin wird die Technologie, die zur Transition der vom Autoren erstellten Modelle in Quellcode dient, vorgestellt.

4.1 Autorenwerkzeuge

Autorenwerkzeuge sollten möglichst einfach gehalten werden und sich auf das Wesentlichste beschränken. Aus diesem Grund wird in dieser Arbeit eine grafische Umgebung für die Autoren empfohlen.

Vorerst werden hier Kriterien für grafische Autorenwerkzeuge erarbeitet, um die zu evaluierenden Werkzeuge vergleichbar zu machen:
Der zu *erwartende Aufwand* sollte in der Bewertung der Werkzeuge betrachtet werden. Ein zu hoher Aufwand könnte zu extremen Verzügen bei der Masterarbeit führen. Ein weiteres wichtiges Kriterium ist die *Anpassbarkeit* des jeweiligen Tools, da derzeit zwar nur die Subdomäne „Action Games“ betrachtet wird, später die Erweiterung um andere Domänen jedoch nicht ausgeschlossen werden darf. Um die Zusammenarbeit zwischen dem Autorenwerkzeug und der Quellcodegenerierung zu ermöglichen, ist ein gewisser Grad an *Interoperabilität* zu gewährleisten. Die Bedienoberfläche des Autorenwerkzeugs sollte besonders einfache *Bedienbarkeit* vorweisen. Die hier genannten Kriterien werden nach der Auflistung und Auswertung der Werkzeuge in der darauf folgenden Tabelle 4.1 als Bewertungskriterien verwendet.

In der folgenden Auflistung werden die verschiedenen Werkzeuge vorgestellt:

- Microsoft Visio
  Das grafische Werkzeug von Microsoft wird in der Softwareentwicklung primär genutzt, um einfache UML-Diagramme zu zeichnen. Die Anpassungsfähigkeit von Visio sind durch das Erstellen von eigenen Elementen gegeben. Die Bedienbarkeit von Visio ist hervorzuheben, es ist schnell und einfach möglich Metamodelle zu erstellen. In

Hinblick auf die Interoperabilität zeigt Visio jedoch leichte defizite, das hängt primär damit zusammen, dass es erstmal nur Schnittstellen zu Programmiersprachen aus dem Hause Microsoft mitbringt. Es bleibt offen, die Interoperabilität mit Blick auf andere Programmiersprachen zu analysieren.
Visio bietet ein hohes Potenzial durch seine Anpassungsfähigkeit und der guten Bedienbarkeit zeigt jedoch Defizite in Blick auf die Interoperabilität.

- Eclipse Graphical Modeling Framework (GMF) (Eclipse.org [2008b])
  GMF ist ebenfalls ein grafisches Modellierungswerkzeug, welches in die Eclipse Entwicklungsumgebung eingebettet ist. Es unterstützt die automatische Generierung von grafischen Editoren und ist durch dieses Feature besonders anpassbar, dies könnte aber gleichzeitig zu einer Metametamodellierung führen, was erhöhten Aufwand bedeuten würde. Ein weiterer Vorteil ist die Unabhängigkeit von den Programmiersprachen. Dieses Tool kann in Verbindung mit Java, aber auch den Sprachen aus der C-Familie arbeiten. Bei dem ersten Arbeiten mit diesem Werkzeug fiel eine weniger intuitive, aber dennoch annehmbare Bedienbarkeit dieses Werkzeuges auf.

- MetaEdit+ (Metacase [2008])
  Dieses kommerzielle Werkzeug konnte bisher noch nicht evaluiert werden. In der Vorbereitung der Masterarbeit sollte versucht werden, eine Lizenz für dieses Werkzeug zu bekommen. Es ist besonders interessant, da es zusätzlich zu den grafischen Notationen auch tabellarische Notationen unterstützt. Eine tabellarische Notation wäre speziell für die Interaktionsmatrizen aus Kapitel 3.2.2 hilfreich.

- Eigene Implementierung
  Eine eigene Implementierung auf Basis von Technologien wie z.B. Java und Swing sollte in Betracht gezogen werden, wenn sich herausstellen sollte, dass keins der vorgestellten Werkzeuge den Anforderungen gerecht werden kann. Dabei sollte jedoch der erhöhte Aufwand in Relation zu der hohen Flexibilität bewertet werden. Die Flexibilität kann zur Erhöhung der Interoperabilität, der Bedienbarkeit und der Anpassbarkeit führen.

Die grundlegende Evaluation sollte in der Masterarbeit fortgeführt werden und um zusätzliche Aspekte erweitert werden. In Tabelle 4.1 wird die derzeitige Evaluation übersichtlich dargestellt.

4.2 Technologie für das Erzeugen von Quellcode

Nachdem wir uns verschiedene Technologien für das Erstellen bzw. Verwenden von grafischen Werkzeugen angeschaut haben, wird im Folgenden ein Blick auf verschiedene Technologien zur Transition der Modelle in Quellcode gelegt. Für die Transition ist es erforderlich,

Werkzeug	Aufwand	Anpassbarkeit	Interoperabilität	Bedienbarkeit
MS Visio	?	+	-	++
GMF	+	++	++	+
MetaEdit+	?	?	?	?
Eigene Implementierung	- -	++	+	++

Tabelle 4.1: Übersicht der Evaluation von grafischen Werkzeugen

dass ein Modell formal analysiert und daraus eindeutige Anweisungen generiert werden können. Um dieses Ziel zu erreichen ist es notwendig, dass das Modell von dem Transitionstool eingelesen werden kann. Das wird durch eine interoperable interne Notation möglich. Für diese Repräsentation des Modells sollten die im vorigen Kapitel vorgestellten Werkzeuge das jeweilige Modell in eine allgemeine Notation wandeln können. Aus diesen allgemeinen Notationen wird in dem darauffolgenden Schritt der Quellcode generiert.

Für das Generieren des Quellcodes wurden bisher keine Technologien konkret evaluiert. Bei der Recherche zeigten sich jedoch bereits die folgenden potenziellen Technologien auf:

- Eclipse Modeling Framework - EMF (Eclipse.org [2008a])
  Das Eclipse Modeling Framework ist durch das Ecore Metametamodell beschrieben. Es bietet bereits diverse Möglichkeiten ein Modell in Quellcode verschiedener Sprachen zu transformieren. Es ist jedoch zu beachten, dass es nur dann wirklich gut benutzbar ist, wenn das Autorenwerkzeug auch in Java programmiert ist, da es nur eine Java-API bietet.

- openArchitectureWare (openArchitectureWare Group [2008])
  Das openArchitectureWare bietet ein eigenes System, mit dem es möglich ist auf beliebigen generischen Objektstrukturen typsicher zu arbeiten (Efftinge u. a. [2007]). Weiterhin ist openArchitectureWare als Eclipse-Plugin realisiert, wodurch es besonders attraktiv erscheint, wenn es in Verbindung mit anderen Eclipsetechnologien verwendet werden soll.

- Eigene Implementierung
  Im Falle der Notwendigkeit einen eigenen Codegenerator zu schreiben, sind verschiedene Technologien zu verwenden. So würde ein Compilerbauwerkzeug wie beispielsweise Another Tool for Language Recognition (AntLR), yacc oder JavaCC den erforderlichen Parser generieren. Die Notationssprache des Modells müsste ebenfalls entwickelt werden, was auf Basis von Technologien wie eXtensible Markup Language (XML) geschehen könnte. In dieser ersten Analyse wurde die eigene Implementierung vorerst zurückgestellt, da der erhebliche Mehraufwand bisher nicht gerechtfertigt scheint.

5 Zusammenfassung

Ziel dieser Ausarbeitung war es einen Überblick über die nötigen Entwicklungsschritte und Technologien zu geben ein Autorenwerkzeug zu Erstellen. Dazu wurde in Kapitel 2 auf die Spielgenres eingegangen, um dem Leser die Anwendungsdomäne näher zu bringen. Im Anschluss wurde zur Abgrenzung und mit Blick auf die Vertiefung von verschiedenen Darstellungsmöglichkeiten der spezielle Spielgenre „Action Games“ näher erläutert.
In Kapitel 3 wurden die erforderlichen Schritte zur Entwicklung eines Autorenwerkzeuges beschrieben. Von der Auswahl eines geeigneten Vorgehensmodells über Vorschläge verschiedener Darstellungsmöglichkeiten bis hin zur Auswahl einer Basistechnologie für das Autorenwerkzeug und der Transition wurde das Vorgehen erläutert.
In dem letzten Hauptkapitel 4 - „Technologien“ wurde auf konkrete Technologien eingegangen. In dem ersten Unterkapitel wurden verschiedene Werkzeuge, welche als Basis für das Autorenwerkzeug dienen könnten vorgestellt. Daraufhin wurde ein kurzer Überblick über Technologien, die zur Transition von dem Modell zum Quellcode geeignet sind, gegeben.

Im folgenden werden **Risiken** genannt und eventuelle Gegenmaßnahmen vorgeschlagen.

- Das Entwickeln von Referenzarchitekturen für Spiele und insbesondere das Übertragen dieser auf ähnliche Domänen könnte durch hohen Zeitaufwand eingeschränkt werden.
- Das Anwendungsgebiet ist sehr komplex. Aus diesem Grund wurde der Themenbereich der Arbeit auf Action Games beschränkt. Es könnte sich herrausstellen, dass das Erstellen von generischen Architekturen sich selbst für diese Subdomäne als zu aufwendig zeigt und somit eine automatisierte Generierung der Software nicht mehr möglich sein könnte. Sollte dieser Fall auftreten, so wäre in Betracht zu ziehen, ob weitere Einschränkungen in der Subdomäne machbar sind. Die Masterarbeit könnte sich auf Teilaspekte einer Domäne beschränken.
- Die Entwicklung eines eigen zu implementierenden Autorenwerkzeuges sollte aufgrund von hohem Aufwand vermieden werden. Sollte diese dennoch unvermeidbar werden, müssen Einschränkungen bezüglich der in Kapitel 4.1 genannten Anforderungen gemacht werden.

- Probleme mit der Schnittstelle zwischen dem Autorenwerkzeug und der Quellcodegenerierung sollten im Vorwege durch eine flexible und durchdachte Architektur minimiert werden.

Die als **nächstes durchzuführenden Schritte**, um dieses Thema für die Masterarbeit vorzubereiten, sind:

- Es sollten Spielearchitekturen identifiziert werden und Gemeinsamkeiten erkannt werden.
- Designpatterns für Spiele sollten identifiziert und erarbeitet werden, um später gute Architekturen mit dem Werkzeug erzeugen zu können.
- Es sollte verstärkte Kommunikation mit Spiele-Fachexperten geführt werden, um die Anforderungen an ein Autorenwerkzeug konkret definieren zu können.
- Eine prototypische Implementierung würde ebenfalls helfen die technischen und fachlichen Anforderungen konkreter und exakter zu definieren. Weiterhin würde diese helfen, das bisherige Vorgehen zu evaluieren und eventuell zu verbessern.

Literaturverzeichnis

Adams u. Rollings 2006

ADAMS, Ernest ; ROLLINGS, Andrew: *Fundamentals of Game Design (Game Design and Development Series)*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 2006. – ISBN 0131687476

Dedaj 2008

DEDAJ, Dennis: *Game Engineering*. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2008/dedaj/folien.pdf (Stand: 20.04.2008), 4 2008

Eclipse.org 2008a

ECLIPSE.ORG: *Eclipse Modeling Framework*. http://www.eclipse.org/emf/ (Stand: 18.07.2008), 7 2008

Eclipse.org 2008b

ECLIPSE.ORG: *Graphical Modeling Framework*. http://www.eclipse.org/gmf/ (Stand: 18.07.2008), 7 2008

Efftinge u. a. 2007

EFFTINGE, Sven ; HAASE, Arno ; STAH, Thomas ; VÖLTER, Markus: *Modellgetriebene Softwareentwicklung*. 2. Auflage. Heidelberg, BRD : dpunkt.verlag GmbH, 2007. – ISBN 9783898644488

openArchitectureWare Group 2008

GROUP openArchitectureWare: *openArchitectureWare Generator*. http://www.openarchitectureware.org (Stand: 21.07.2008), 7 2008

Metacase 2008

METACASE, Consulting: *MetaEdit+*. http://www.metacase.com (Stand: 18.07.2008), 7 2008

Morris u. Rollings 2004

MORRIS, Dave (Hrsg.) ; ROLLINGS, Andrew (Hrsg.): *Game Architecture And Design: A New Edition*. Indiana : Pearson Education, 2004. – ISBN 0735713634

OMG

OMG: *Object Management Group*. http://www.omg.org (Stand: 14.07.2008),

OMG 2005

OMG: *Object Constraint Language 2.0*. http://www.omg.org/docs/ptc/05-06-06.pdf (Stand: 15.07.2008), 6 2005

Taylor u. a. 2006

TAYLOR, M. J. ; GRESTY, D. ; BASKETT, M.: Computer Game-Flow Design. In: *Computer Entertainment* 4 (2006), Nr. 1