Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Mediensysteme

Entwurf eines Spieler-Modells für eine erweiterbare Spielplattform zur Ausbildung in der Bauphysik

Bachelorarbeit

Janina Held
geb. am: 31.12.1988 in Dinslaken

Matrikelnummer 70435

1. Gutachter: Prof. Dr. Oliver Kornadt
2. Gutachter: Dr. Matthias Hagen

Datum der Abgabe: 14. April 2011

Vorwort

Im Projekt *Intelligentes Lernen* beschäftigen sich die Professuren *Content Management und Web-Technologien*, *Systeme der Virtuellen Realität* und *Bauphysik* der Bauhaus-Universität Weimar mit der Entwicklung innovativer Informationstechnologien für eLearning-Umgebungen. In den Teilbereichen Retrieval, Extraktion und Visualisierung großer Dokumentkollektionen, sowie simulations- und planbasierter Wissensvermittlung werden Algorithmen und Werkzeuge erforscht, um eLearning-Systeme leistungsfähiger zu machen und um somit den Lernerfolg zu optimieren [Bau].
Ziel des Projekts, auf dem Gebiet des simulationsbasierten Wissenstransfers, ist die Entwicklung eines Multiplayer Online Games (MOG) zur Ausbildungsunterstützung in der Bauphysik.
Im Rahmen der vorliegenden Bachelorarbeit wird für diese digitale Lernsoftware ein Spieler-Modell zur Verwaltung der spielerspezifischen Daten entworfen und in das bestehende Framework integriert. Der Schwerpunkt der Arbeit liegt in der Organisation der erlernten Fähigkeiten des Spielers und in der an den Wissensstand angepassten Auswahl geeigneter Spielaufgaben. Für die Anwendung im eLearning-Bereich ist die Erweiterbarkeit des Modells um neue Lernkomplexe eine wesentliche Anforderung.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AWT	Abstract Window Toolkit
GBL	Game-Based Learning
GEF	Graphical Editor Framework
Id	Identifier
IDE	Integrated Development Environment
MOG	Multiplayer Online Game
OSGi	Open Service Gateway Initiative
RCP	Rich Client Platform
SWT	Standard Widget Toolkit
UML	Unified Modeling Language
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Einleitung

Der aktive Prozess des Lernens beschränkt sich längst nicht mehr nur noch auf die Kindheit. Rasante Entwicklungen, neue Technologien und der Wandel, hin zu einer Informations- und Wissensgesellschaft, fordern immer mehr die Fähigkeit, sich Wissen eigenständig anzueignen, um sein persönliches Qualifikationsprofil zu erweitern [MS03]. Der Begriff des *lebenslangen Lernens* rückt dabei zunehmend in den Mittelpunkt heutiger Diskussionen über Bildungsstrategien. Die größte Herausforderung besteht jedoch darin, einen flexiblen und möglichst zeit- und ortsunabhängigen Zugang zu spezialisiertem Fachwissen zu schaffen, um einen individuellen Lernprozess zu ermöglichen.
An diesem Punkt setzen eLearning-Systeme an. Sie verbinden moderne Kommunikationstechniken mit aktuellen Lernstrategien und stellen eine Wiederverwendbarkeit der Lerninhalte sicher. Die elektronische Unterstützung begleitet dabei den Lernprozess und fördert die Kommunikation mit anderen Lernenden. In dieser *Community of Practice* [Lav08], können Lernende vom Wissen anderer, die sich mit gleichen Problemstellungen beschäftigen, profitieren.

> *"When we think of games, we think of fun. When we think of learning we think of work. Games show us this is wrong."*
> (James Paul Gee, 2005) [Gee05]

James Paul Gee veranschaulicht damit einen der großen Nachteile von eLearning-Systemen. Den Lernenden fehlt oftmals die nötige Motivation, sich mit den Lerninhalten auseinanderzusetzen [MS03]. Daher soll durch den Einsatz spielerischer Elemente in eLearning-Umgebungen der Faktor des Lernens in den Hintergrund gerückt werden. Dem Spieler soll in einer realitätsnahen Umgebung die Möglichkeit geboten werden, erworbenes Wissen zu erproben. Die Theorie des *situierten Lernens* [Lav08] besagt, dass ein vollständiger Transfer des Wissens von lehrenden Personen zum Lernenden nicht möglich ist. Die Situation des Lernens muss in einem authentischen Kontext eingebettet sein. Wesentliche Bestandteile, um einen hohen Lernerfolg zu erzielen, sind kommunikative Auseinandersetzung, sozialer und kultureller Umgang mit den Inhalten und das Erproben von Lösungen. Virtuelle Spielumgebungen bieten diesen Raum der kritischen Auseinandersetzung.
James Paul Gee [Gee07] beschreibt wichtige Lernprinzipien, die in Computerspielen umgesetzt sind und deutet auf das Potenzial von spielbasiertem Lernen hin. Durch die Verschmelzung von Lern- und Spielziel profitiert der Lernerfolg von den in Spielen eingesetzten Mechanismen zur Aufrechterhaltung der Motivation.

Im Rahmen dieser Bachelorarbeit wird ein Spieler-Modell für ein bauphysikalisches Lernspiel zum Einsatz im eLearning-Bereich entwickelt und in das bestehende Framework

integriert. Ziel des Modells ist es, über ein Skill[1]-System (siehe 3.2.1) den Lernverlauf des Spielers zu verwalten. Dazu müssen erlernbare Fähigkeiten der realen Welt modelliert und für die virtuelle Spielwelt abgebildet werden. Diese können vom Spielteilnehmer, durch die Erledigung gezielte Aufgaben, erlernt werden. Unter dem Gesichtspunkt der motivierenden Belohnung muss eine Bewertung des Lernfortschritts stattfinden, durch die eine an den Wissensstand angepasste Auswahl von spielbaren Missionen erstellt werden kann. Diese Zusammenstellung gibt dem Spieler die Möglichkeit, gezielt Fähigkeiten zu trainieren und somit angestrebte Lernziele zu erreichen. Durch den sukzessiven Ausbau des Wissens ergeben sich stets neue Aufgaben für den Spieler.

Die vorliegende Arbeit ist in fünf Kapitel eingeteilt. Im Folgenden werden zunächst grundlegende Begriffe aus dem Bereich des spielbasierten Lernens geklärt. Anschließend werden die entscheidenden Aspekte für die Aufrechterhaltung der Motivation, sowie die Bedeutung von Belohnungen für den Spieler, untersucht. Im dritten Abschnitt wird auf Grundlage der vorangegangenen Erkenntnisse der Entwurf eines Spieler-Modells vorgestellt. Die Implementierungskonzepte und die Umsetzung der Integration in das Framework wird im vierten Kapitel beschrieben. Abschließend werden die gewonnenen Erkenntnisse zusammengefasst und die Möglichkeiten der Erweiterung des Modells vorgestellt.

[1] Fähigkeit, Fertigkeit

2 Spielbasiertes Lernen

Im Hinblick auf die Entwicklung eines Spieler-Modells zum Einsatz in einer digitalen Lernspiel-Umgebung, werden in diesem Kapitel zunächst die grundlegenden Aspekte spielbasierten Lernens analysiert. Nach einer Klärung des Begriffs, folgt ein Überblick über wichtige Gestaltungskriterien, sowie deren Auswirkung auf den Lernprozess. Die für den Lernerfolg wichtigste Komponente ist die Aufrechterhaltung der Spielermotivation. Daher bildet dieser Abschnitt den Schwerpunkt des Kapitels. Abschließend wird untersucht, welchen Einfluss Belohnungen auf den Spieler haben und wie sich diese auf die Motivation auswirken.

2.1 Begriffsklärung

Spiel

In der Literatur sind eine Vielzahl von verschiedenen Spieldefinitionen beschrieben. Allgemein lässt sich jedoch sagen, dass der Begriff *Spiel* eine Handlung beschreibt, die nicht zweckgebunden ist, sondern aus eigenem Vergnügen verrichtet wird. Zu den Kriterien, die als kennzeichnende Merkmale von Spielen aufgeführt werden, gehört zunächst das Vorhandensein einer Spielidee oder Geschichte, die für Motivation sorgt und den Rahmen der Handlung vorgibt. Der Spielablauf findet in einer Handlungssituation statt, die eine aktive Beteiligung des Spielers erfordert und die durch festgelegte Regeln gesteuert wird [MS03]. Weiterhin nennt [Pre07] sechs strukturelle Schlüsselelemente eines Spiels. Neben den Regeln und den festgelegten Zielen, zählen hierzu auch die Interaktionen, die Problemlösung von Konflikten, sowie die daraus resultierenden Ergebnisse und Rückmeldungen.

Spielbasiertes Lernen (engl. *Game-Based Learning* (GBL))

Spielbasiertes Lernen, oftmals synonym in der Literatur auch als *digitales Lernspiel* bezeichnet, beschreibt die Verbindung von Spielen und Lernen hauptsächlich in digitalen Spielumgebungen. „Als Lernspiel können nun Aktivitäten bezeichnet werden, deren Inhalte, Strukturen und Abläufe in pädagogischer Absicht und auf der Grundlage didaktischer Prinzipien gestaltet sind, die zugleich aber zentrale Merkmale von Spielen enthalten.“, so [MS03]. In einer spielbasierten Lernumgebung profitiert der Lernerfolg von den erweiterten Möglichkeiten, die durch die Spielumgebung geboten werden. Gestaltungskriterien,

die bei der Umsetzung einer spielbasierten Lernumgebung beachtet werden müssen, werden im folgenden Kapitel erläutert.

2.2 Lernprinzipien digitaler Spielumgebungen

Klassische Bildungsmaßnahmen sind vorwiegend darauf ausgelegt, Wissen bereitzustellen und zu vermitteln. Die Möglichkeit, dieses Wissen anzuwenden erfolgt meist in nur viel zu geringem Umfang. Der Erfolg wird ausschließlich an der Bewertung der Resultate gemessen. Dabei nimmt der Lehrende die aktive und der Lernende die passive Rolle ein. Vermitteltes Wissen ,, [...] ist oft nur ,,träges Wissen“, d.h. Wissen, das zwar theoretisch beherrscht wird, jedoch in einer konkreten Anwendungssituation nicht aktiviert werden kann“ (Meier und Seufert, 2003) [MS03].
Abbildung 2.1 verdeutlicht den typischen Lernverlauf in einer herkömmlichen Lehreinrichtung. Die Kurve weist einen linearen Verlauf mit sprunghaften Anstiegen auf. Diese Sprünge deuten auf das Stattfinden einer Leistungskontrolle hin, denn die Lernenden beschäftigen sich in dieser Vorbereitungszeit verstärkt und intensiv mit den Lerninhalten. Ein konstanter Lernverlauf findet aus eigener Motivation selten statt. Somit erfolgt in regelmäßigen Abständen innerhalb kurzer Zeit ein starker Wissenszuwachs. In den dazwischenliegenden Phasen, in denen der Lernende wieder eine passive Rolle einnimmt, lässt sich nur ein wesentlich geringerer Lernerfolg verzeichnen [BS09].

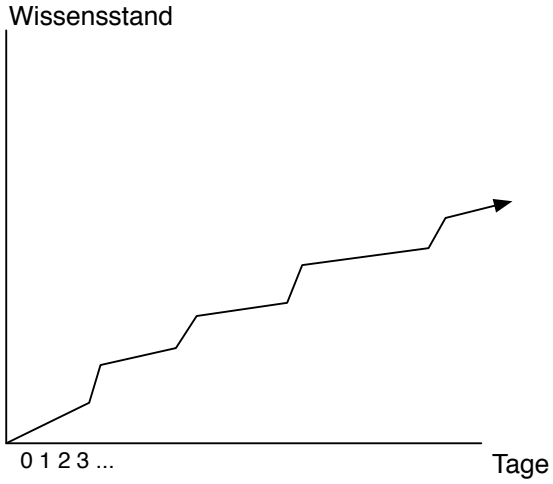

Abbildung 2.1: Typische Lernkurve einer herkömmlichen Bildungseinrichtung [BS09]

In der Abbildung 2.2 ist zum Vergleich der Levelanstieg in Abhängigkeit zu den gespielten Tagen, des bekannten MOG *World of Warcraft* dargestellt. Diese Abbildung verdeutlicht

eine typische Kurve für ein MOG. Zu Beginn erfolgt innerhalb kurzer Spielzeit ein starker Levelanstieg, wodurch die Motivation beim Spieler aufgebaut wird [BS09]. Im weiteren Spielverlauf flacht die Kurve zunehmend ab und die Aufrechterhaltung der Motivation erfolgt über andere Faktoren. Diese werden im folgenden Kapitel genauer beschrieben.

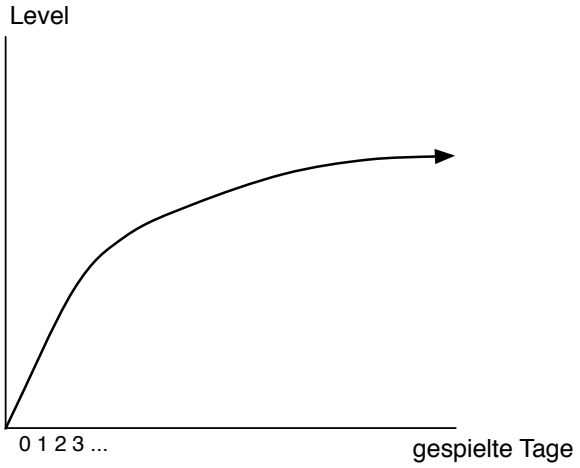

Abbildung 2.2: Levelkurve des bekannten MOG „*World of Warcraft*“ [BS09]

Die Übereinstimmung von Lernziel und Spielziel ist demzufolge ein wichtiges Kriterium bei der Entwicklung von spielbasierten Lernumgebungen. Der Lernerfolg profitiert, wenn der Spieler keine passiven Phasen im Lernverlauf durchlebt. Dieses Potenzial bieten Lernspiele, indem der aktive Lernende im Vordergrund einer problemorientierten Lernumgebung steht.
James Paul Gee beschreibt verschiedene Lernprinzipien, die in digitalen Spielen umgesetzt sind und deren Anwendung von ihm auch in klassischen Lehrsituationen als notwendig erachtet werden [Gee07]. Im Folgenden sind die vier wichtigsten Prinzipien zusammengefasst.

Semiotic Domain

Gee beschreibt unter dem Begriff *Semiotic Domain* eine Sinnwelt, in der Handlungen, Texte, Bilder, Symbole, Aufgaben und Ziele eine komplexe Einheit bilden. Der Lernende muss in einem authentischen Kontext Brücken zwischen neuem und bereits bekanntem Wissen schlagen können. Unverankertes Wissen geht sofort verloren. Virtuelle Spielwelten bieten dieses komplexe System.

Lernen durch Identitäten

Digitale Spiele fördern das Denken in verschiedenen Identitäten und Sichtweisen. Nach Gee nimmt der Spieler drei verschiedene Identitäten an. Dazu gehört die reale Identität, die virtuelle Identität und die projizierte Identität. Entscheidend beim Lernen in virtuellen Welten ist die Rückprojektion von der virtuellen in die reale Welt. Diese Brücke zum Wechseln zwischen den Identitäten bildet die projizierte Identität. Kann in einer spielbasierten Lernumgebung die virtuelle Welt mit der realen Welt in Zusammenhang gebracht werden, so ist die Grundlage zum Lernen geschaffen.

Prinzip des Trainings

In einer virtuellen Spielumgebung hat der Spieler eine ausreichende Menge an Übungsmöglichkeiten. Dabei sollte sich der Spielfortschritt an das Können des Spielers anpassen. Im Gegensatz zu klassischen Lernkonzepten, in denen Wissen themenweise behandelt und unabhängig vom Kenntnisstand fortschreitend vermittelt wird, bestimmt in virtuellen Spielen der Lernende selbst den Fortschritt. Hat die vom Spieler entwickelte Strategie das Ziel erreicht, kann der Spieler in eine höhere Stufe, beispielsweise in ein neues Level oder durch die Freischaltung neuer Möglichkeiten, aufsteigen.

Lernen durch Risikofreiheit

Spielwelten bieten einen angstfreien Raum zum Lernen. Spieler können größere Risiken eingehen, da die Konsequenzen harmloser als im realen Leben sind. Fehler werden nicht vermieden, sondern dienen dem Erfahrungslernen.

Digitale Lernspiele fördern somit verschiedene Lernprozesse (Vergleich [MS03]).
In einer Simulationsumgebung kann der Spieler experimentieren und ist dabei nicht auf eine systematische Wissensvermittlung angewiesen. Somit fördern digitale Lernspiele das *aktive Lernen*.
Lernspiele bieten eine problemorientierte Lernumgebung, in welcher der Spieler individuelle Erfahrungen sammelt und neu gewonnene Erkenntnisse mit seinem persönlichen Vorwissen verknüpft. Somit wird Lernen als *konstruktiver Prozess* gefördert.
Ein *selbstgesteuerter Lernprozess* wird erreicht, wenn der Spieler Lernwege und Lernzeiten selbst bestimmen kann. Spielbasierte Lernumgebungen bieten große Freiheiten, indem der Lernende die Verweilzeit im Spiel, sowie Spielhandlungen selbst festlegen kann.
Lernen als *sozialer Prozess* erfolgt vorwiegend in MOGs, in denen durch Chat-Systeme die Kommunikation zu anderen Spielern unterstützt wird. Oftmals sind die Spielaufgaben so gestaltet, dass sie nur gelöst werden können, wenn sich mehrere Spieler in einer Gruppe diesem Problem widmen. Dabei kann der Spieler vom Wissen anderer profitieren.
Die Möglichkeit der Identifikation mit dem virtuellen Spielcharakter integriert den Lernenden ganzheitlich in das Spielgeschehen und fördert Lernen als *emotionalen Prozess*.

In einer authentischen Lernsituation, in der die auftretende Problemstellung von hoher Relevanz ist, erfolgt Lernen als *situierter Prozess*. Der Lernende kann aus verschiedenen Sichtweisen und durch das Erproben unterschiedlicher Lösungsstrategien eine Konfliktsituation lösen.

Dieses Potenzial kann jedoch nur unter der Berücksichtigung folgender Kriterien genutzt werden [MS03]:

- Anpassung an die Zielgruppe
- Identifikation des Lernbedarfs
- Lernmotivation der Zielgruppe
- Gestaltung einer motivierenden Lernspielumgebung

Die Identifikation der Zielgruppe, des Lernbedarfs und der Lernmotivation wird als besonders bedeutsam erachtet. Für die Entwicklung eines *Spieler-Modells* ist die Gestaltung einer motivierenden Lernspielumgebung jedoch von größerem Interesse. Im folgenden Abschnitt werden Gestaltungskriterien untersucht, die einen Aufbau der Spielermotivation bewirken.

2.3 Spielermotivation

Im Zusammenhang mit einer Lernspielumgebung, beschreibt der Begriff *Motivation* die intrinsischen und extrinsischen Beweggründe eines Spielers für die engagierte Auseinandersetzung mit den Inhalten [Kon10]. Intrinsische Motivation ist charakterisiert durch einen inneren Antrieb. Extrinsische Motivation ist geleitet von der Aussicht auf eine von außen kommenden Belohnung.
Wie im Abschnitt 2.2 festgestellt wurde, bieten digitale Spielwelten eine authentische Lernumgebung, die den Anwender in das Spielgeschehen und somit in den Lernkontext hineinversetzt. Sie stellen einen Raum zur Verfügung, um aktiv Strategien zum Lösen von Problemen zu entwickeln und ermutigen zu systematischem Denken. Dabei sind die Konsequenzen der Handlungen wesentlich geringer als im realen Leben. Der Erfolg einer spielbasierten Lernumgebung wird jedoch maßgeblich von der Motivation des Spielers bestimmt. Ein langfristiger Spielspaß kann nur erreicht werden, wenn der Spieler über eine ausreichende Menge an intrinsischer und extrinsischer Motivation verfügt.

Spielzyklus

Abbildung 2.3 zeigt ein Modell, welches den Ablauf des Spielzyklus beschreibt [GAD02]. Dieser Zyklus besteht aus dem Spielerverhalten, der Rückmeldung des Programms und der Beurteilung des Spielers. Beeinflusst wird der Spielzyklus von den festgelegten Lerninhalte und den Spieleigenschaften. Zusammen mit diesen Komponenten liefert der Spielzyklus das Lernergebnis.

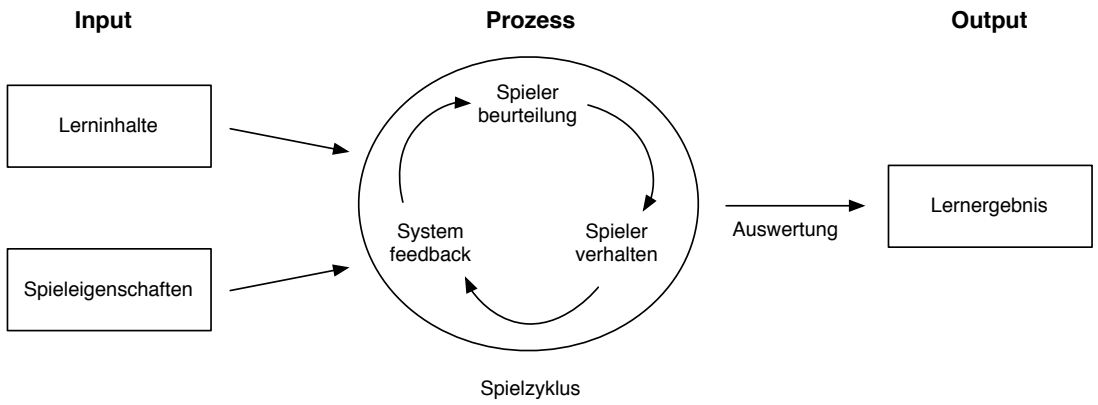

Abbildung 2.3: Input-Prozess-Output-Spielzyklus (nach Garris und Driskell in [GAD02])

Der Spielzyklus beschreibt einen ständigen Kreislauf, in dem sich der Spieler befindet. Führt der Spieler einen Spielzug aus, liefert die virtuelle Welt eine direkte Reaktion. Anhand dieser Rückmeldung kann der Spieler sein vorheriges Verhalten beurteilen und sich gegebenenfalls für eine neue Handlung entscheiden. Die kritische Komponente ist dabei die Rückmeldung des Systems. Der Spieler muss in der Lage sein, das Feedback bewerten zu können, um es für sich als erfolgreich oder gescheitert einzustufen. Konnte der Spieler ein positives Feedback vom System erfahren, wird er positiv bestärkt und das Interesse am Weiterspielen steigt. War die getätigte Handlung nicht erfolgreich, sieht sich der Spieler zunächst herausgefordert und sein Ehrgeiz wächst. Der Spieler wählt eine alternative Handlung um das Problem zu lösen. Dies erfolgt jedoch nur so lange, bis die Frustrationsgrenze des Spielers erreicht ist. Daher sollte der Spieler innerhalb des Zyklus die Kompetenzen erreichen, die zum Lösen des Problems nötig sind. Dadurch kann ein Abbrechen aufgrund fehlender Motivation vermieden werden. Wird dieses Ziel erreicht und der Spieler vergisst sich selbst und taucht vollkommen in die Spielwelt ein, dann ist die intrinsische Motivation so hoch, dass daraus ein Flow-Effekt entstehen kann.

Flow-Effekt

Mihály Csíkszentmihályi beschreibt in [Csi03] den Flow-Effekt als ein Erlebnis der vollkommenden Vertiefung in eine Tätigkeit mit einem hohen Maß an Freude und Erfüllung. Dieser Zustand wird erreicht, wenn die Herausforderungen mit den Fähigkeiten übereinstimmen. Wie Abbildung 2.4 verdeutlicht, entsteht das Flow-Erlebnis an der Grenze zwischen Über- und Unterforderung. Wenn die Herausforderung höher als die Fähigkeit ist, dann entsteht Überforderung und Frustration beim Spieler. Ist die Herausforderung geringer als die Fähigkeit des Spielers, entsteht durch die Unterforderung Langeweile. Flow entsteht nicht nur direkt an der Grenze, sondern jeder Spieler hat eine individuelle Flow-Zone.

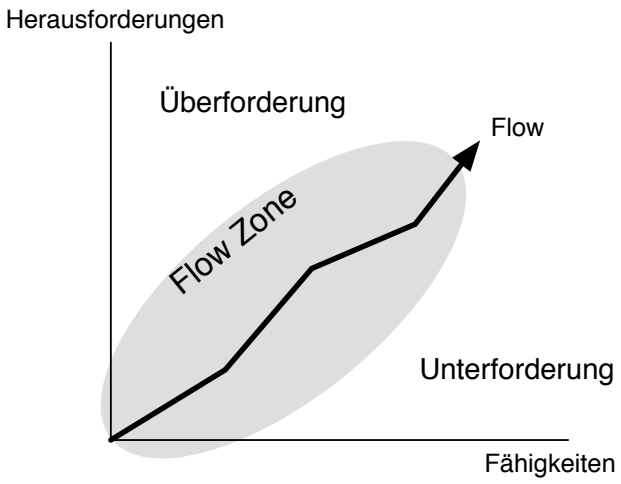

Abbildung 2.4: Der Flow-Effekt

Csíkszentmihályi beschreibt die folgenden acht notwendigen Komponenten für das Zustandekommen der Flow-Erfahrung [Csi03]:

- Verhältnis zwischen Anforderung und Fähigkeit
- klare Ziele
- direktes Feedback
- müheloser Handlungsablauf
- Verschmelzung von Handlung und Bewusstsein
- hohe Konzentration auf begrenzte Aktivität
- Gefühl von Kontrolle
- Veränderung des Zeiterlebens

Der Flow-Effekt in einem digitalen Lernspiel kann erreicht werden, wenn eine ausreichende Menge an intrinsischer Motivation vorhanden ist und dem Spieler Herausforderungen angeboten werden, die abgestimmt sind auf dessen individuellen Fähigkeiten. Ebenso muss der Spieler das Gefühl von Kontrolle über das Spiel haben [Che].

Durch die Integration weiterer motivierender Elemente kann die Ausdauer und das Vergnügen des Spielers zusätzlich unterstützt werden. Dazu zählt beispielsweise die grafische Gestaltung der Benutzeroberfläche: „Digitale Lernspiele sollten durch eine hohe Benutzerfreundlichkeit, einfache Navigation, einem einheitlichen Prinzipien folgendem Bildschirmdesign, eine klare Informationspräsentation und durch eine gefällige Ästhetik überzeugen“ (Meier und Seufert, 2003) [MS03].

Zur Steigerung der extrinsischen Motivation muss der Spieler von außen für erfolgreiche Handlungen belohnt werden. Als aussichtsreich gelten Systeme, die Spielpunkte für richtige Tätigkeiten vergeben, da sie eine direkte Verbindung zwischen Handlung und Belohnung darstellen und dem Spieler eine ständige Aussicht auf Belohnung bieten [BS09]. Im folgenden Abschnitt wird untersucht, welche Arten von Belohnungen in einer Spielumgebung motivierend eingesetzt werden können.

2.4 Bedeutung von Belohnungen

Belohnungen spielen eine wichtige Rolle im Lernprozess. Erfährt der Lernende auf seine Handlung eine positive Konsequenz, wird er diese Handlung erneut so ausführen. Folgt jedoch eine negative Auswirkung, überdenkt er seine Handlung und führt sie in geänderter Weise aus. Belohnungen wirken als positiver Anreiz und fördern gewünschtes Verhalten. Der Lernende fühlt sich gestärkt und die extrinsische Motivation steigt.
Dieser Ansatz wird in der Spieleentwicklung ebenfalls genutzt. Belohnungen geben dem Spieler das Gefühl von Erfolg. Dabei kommen verschiedene Arten von Belohnungen zum Einsatz [Bat]:

- Ein besonders faires und universell einsetzbares System zur Belohnung entsteht durch den *Einsatz von virtuellem Geld*. Spieler finden diese Art der Belohnung besonders motivierend, da sie die Möglichkeit bietet, Gegenstände zu kaufen, die der individuellen Empfindung nach am sinnvollsten erscheinen.
- Eine andere Art der Belohnung bietet die *Einstufung in ein Rang-System*. In der Regel erhält der Spieler dabei für erfolgreiche Tätigkeiten Punkte, die beim Erreichen einer bestimmten Höhe dazu führen, dass der Spieler in ein anderes Level aufsteigt. Dieses Bewertungssystem ist vergleichbar mit einer klassischen Lernsituation, in welcher der Lernende für seine Tätigkeiten benotet wird. Dabei kann die Bewertung der Handlung ganzheitlich erfolgen. Das bedeutet, es erfolgt eine Bewertung des Gesamtergebnisses ohne die Berücksichtigung der Teilbereiche. Bei einer analytische Bewertung wird jede Handlung eines Spielers analysiert und die Teilergebnisse liefern das Gesamtresultat [Mer01].
- Die *Erhöhung des Spielers in einer Statistik*, die eine entschiedene Relevanz für das Spiel haben muss, wird als weitere Möglichkeit zur motivierenden Belohnung angesehen.
- In einem Spielgeschehen, dessen Handlung auf einer Geschichte aufbaut, können durch das Einspielen von *Zwischenszenen*, aus denen ersichtlich wird, dass der Geschichtsverlauf einen bedeutsamen Vorsprung macht, belohnende Anreize entstehen.
- Als *emotionale Belohnung* wird das Gefühl des Spielers empfunden, wenn er etwas *für* einen anderen Spieler gemacht hat.
- Neue Gegenstände und die Freischaltung neuer Gebiete gehört zu den am meisten verbreiteten Belohnungen in Spielumgebungen. Diese *Erweiterungen* bieten neuen Anreiz zum Ausprobieren und zum Erkunden.

- Eine weitere Art der Belohnung entsteht durch *Vervollständigungen*, wie das Erreichen von 100% einer Aufgabe oder das Finden eines fehlenden Puzzleteils.
- Eine Belohnung durch die *Verleihung des Titels des Erfolgs*, umgesetzt durch die Auszeichnung des Spielers mit Zertifikaten, wird vorwiegend in MOGs eingesetzt. Ausgezeichnet werden meistens erfolgreich gelöste Aufgaben oder besondere Fähigkeiten.
- Das *Erreichen des Spielziels* stellt schließlich die größte Belohnung dar.

Um die gewünschte Steigerung der Motivation zu erzielen, müssen die Belohnungen eine bedeutsame Relevanz für das Spiel haben.

Die Auszahlung dieser Belohnungen kann auf unterschiedliche Arten erfolgen. Eine Belohnung nach einer festgelegten Anzahl von Aktionen führt zu einer hohen Aktivität des Spielers. Da das System für den Spieler leicht zu verstehen und schnell abzuschätzen ist, bewirkt es oftmals eine Spielunterbrechung nach dem Erreichen der Belohnung, da der Spieler weiß, dass für die nächste Belohnung viele Handlungen nötig sind. Dem wirkt eine zufällige Belohnung der Tätigkeiten entgegen. Das Interesse des Spielers bleibt erhalten, da er immer mit Belohnungen rechnen kann.
Eine weitere Möglichkeit besteht darin, den Spieler nach einer festgelegten Zeit zu belohnen. Dies kann beispielsweise über eine tägliche Auszahlung von Punkten erfolgen. Diese Art der Belohnung führt zu einer konstanten Aufmerksamkeit, da der Spieler es anstrebt, von der täglichen Belohnung zu profitieren. Eine Belohnung nach einer variablen Zeit steigert ebenfalls das Interesse, da der Spieler jederzeit mit einer Belohnung rechnen kann, führt aber oftmals auch dazu, dass der Spieler schnell merkt, dass seine Handlungen keine Auswirkungen auf Belohnungen haben [Bat].

> *"But the job of an achievement is to help the player have fun, [...]."*
> [McC]

Der richtige Einsatz der Belohnungen stellt die zentrale Herausforderung in der Entwicklung einer motivierenden Spielumgebung dar. Schwer zu erreichende Belohnungen sind vertretbar, wenn sie der Vervollständigung eines Kernelements des Spiel dienen [McC].

Dieses Kapitel bildet die theoretische Grundlage für die Entwicklung des Spieler-Modells, dessen Konzept im Folgenden vorgestellt wird.

3 Konzept des Spieler-Modells

Ein *Spieler-Modell* bildet die Grundlage für einen erfolgreichen Lernverlauf in einer Spielumgebung. Das Modell sollte dem Spieler einen individuellen Lernprozess ermöglichen, damit ein optimaler Lernerfolg erreicht werden kann. Die zentrale Aufgabe des Modells besteht in der Bewertung der Spielerfähigkeiten und in einer an den Lernfortschritt angepassten Festlegung der Spielaufgaben.

Im Projekt [MM] wurde auf Grundlage fundamentaler Kenntnisse aus dem Bereich des GBL eine Modellarchitektur für den Einsatz in einer Lernspielumgebung entwickelt. Diese vereint herkömmliche Techniken der digitalen Spieleentwicklung mit Strategien zur Wissensvermittlung. Das System besteht aus vier Kernelementen. Dazu gehören neben einer Game-Engine zur visuellen Darstellung der virtuellen Welt, ein *Spieler-Modell*, ein Agent zur Festlegung der Regeln und die Lerninhalten. Das *Spieler-Modell* erfasst die für den Spieler relevanten Daten. Diese beinhalten die gesammelten Punkte, das erreichte Level, erhaltene Belohnungen und gewählte Aufgaben. Im Kontext eines lernbasierten Einsatzes werden ebenso die erlernten Fähigkeiten, das aktuelle Wissen und die Lernziele über dieses Modell verwaltet. Die Festlegung der Spielregeln und die Anpassung des Schwierigkeitsgrades der Aufgaben erfolgt über einen gesonderten Agenten, der den aktuellen Wissensstand des Spielers bewertet.

In diesem Kapitel wird das Konzept eines vergleichbaren *Spieler-Modells* erarbeitet. Im Gegensatz zu der in [MM] beschriebenen Architektur, erfolgt sowohl die Verwaltung der spielerspezifischen Daten, als auch die Auswahl der Aufgaben und die Bewertung der Spielerhandlungen im *Spieler-Modell*.
Das Modell wird für ein bauphysikalisches Lernspiel, welches im eLearning-Bereich eingesetzt werden soll, konzipiert. Aus diesem Grund müssen die erlernbaren Inhalte durch den Fachanwender verwaltbar und erweiterbar sein. Es muss eine Schnittstelle geschaffen werden, über die auf die Spieldaten zugegriffen werden kann. Diese Daten bestehen einerseits aus erlernbaren Fähigkeiten und andererseits aus Missionen, in denen der Spieler durch das Lösen einer gezielten Aufgabe neue Fähigkeiten erlernen kann.
Zur Entwicklung des Modells werden neben den Anforderungen aus dem Bereich des GBL, aus einer Anwendungsfalluntersuchung benutzerspezifische Anforderungen an das System abgeleitet.

3.1 Anwendungsfälle

Für das zu entwickelnde System ergeben sich aus zwei verschiedenen Sichten Anwendungsfälle. Auf der einen Seite stellen die Spieler, die aktiv am Spielgeschehen teilnehmen, eine Benutzergruppe dar. Dem gegenüber stehen die Administratoren, die verwaltend in das System eingreifen und Lerninhalte zu Verfügung stellen. Für beide Benutzergruppen existieren verschiedene Anwendungsfälle.
Abbildung 3.1 verdeutlicht das Anwendungsfalldiagramm nach dem UML (Unified Modeling Language) 2.0 Standard [MH06]. Es dient der Darstellung der zu erwartenden Anwendungsfälle aus den verschiedenen Benutzersichten. Aus diesem Anwendungsfalldiagramm können die Spezifikationen der Anforderungen für das zu entwickelndes Softwaresystem abgeleitet werden.

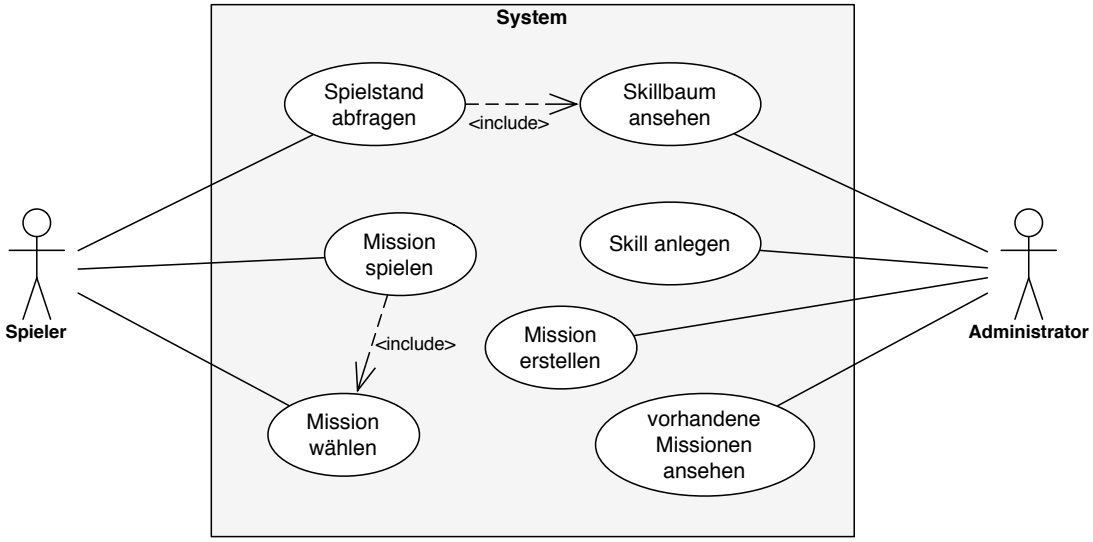

Abbildung 3.1: UML-Anwendungsfalldiagramm

Im Folgenden werden die verschiedenen Anwendungsfälle analysiert und die daraus resultierenden Anforderungen zusammengefasst.

3.1.1 Spieler

Der Spieler nimmt als Benutzer des Systems eine zentrale Rolle ein. Es existieren zwei möglichen Anwendungsfälle, in denen der Spieler mit dem System in Interaktion tritt. Dabei könnte sich der Spieler für seinen persönlichen Spielstand interessieren und sein Fähigkeitslevel abfragen, oder durch das Spielen einer Mission seinen aktuellen Spielstand erweitern.

Mission spielen

In diesem Anwendungsfall möchte der Spieler einen bedeutsamen Fortschritt im Spielverlauf machen. Dazu wählt er aus einer Liste von möglichen Missionen, die ihm vom System vorgeschlagen werden, eine Mission aus und startet diese. Anschließend versucht der Spieler, die in der virtuellen Welt auftretenden Probleme zu lösen. Sein Bestreben besteht darin, die Ziele der Mission zu erreichen. Dazu bewertet er die auf seine Handlungen folgenden Reaktionen des Systems (siehe Kapitel 2.3, Spielzyklus) und entscheidet, wann er die Mission beendet. Wurden die in der Mission festgelegten Aufgaben erfolgreich gelöst, werden dem Spieler Punkte für seine Tätigkeiten ausgezahlt.

Anforderungen *„Mission spielen“*

- Missionssystem
- Definition von Missionsvoraussetzungen
- Ermittlung spielbarer Missionen
- Festlegung von Inhalten und Zielen einer Mission
- Mechanismus zur Auswertung der Spielertätigkeiten
- Punkte-System zur Erhöhung der Spielerfähigkeiten
- Benutzeroberfläche zur Darstellung von spielbaren Missionen

Spielstand abfragen

Um einen Überblick über erreichte Erfolge zu bekommen, fragt der Spieler seinen aktuellen Spielstand ab. Dabei interessieren ihn die bereits erlernten Fähigkeiten, die gesammelten Punkte und die erreichten Ziele. Ebenso möchte er Informationen darüber erhalten, welche Skills als nächstes erlernt werden können und wieviele Skillpunkte zur Erlangung einer neuen Fähigkeit fehlen.

Anforderungen *„Spielstand abfragen“*

- Skillsystem zur Verwaltung der Fähigkeiten
- Möglichkeit zum Erlernen der Skills durch Vergabe von Skillpunkten für erfolgreiche Handlungen in den Missionen
- Speicherung der spielerspezifischen Daten
- Benutzeroberfläche zur Darstellung der Skillabhängigkeiten
- visuelle Hervorhebung der erlernten Skills
- Darstellung sonstiger erreichter Erfolgen

3.1.2 Administrator

In einem Lernspiel, welches in einer eLearning-Umgebung eingesetzt werden soll, müssen die Lerninhalte verwaltbar und erweiterbar sein. Diese Aufgabe übernimmt der Administrator, der einen Zugang zu den bereits vorhandenen Lerninhalten hat. Diese sollen gegebenenfalls anpasst werden oder durch neue Lernkomplexe erweitert werden. Zu den möglichen Anwendungsfällen des Administrators gehört daher das Anlegen neuer Lerneinheiten in Form von Skills, sowie das Erstellen von Missionen, über die neue Fähigkeiten erlernt werden können. Ebenso bildet die Informationsabfrage der im System vorhandenen Skills und Missionen je einen Anwendungsfall, die nachfolgend erläutert werden.

Skill anlegen

Dieser Anwendungsfall beschreibt eine mögliche Erweiterung des Systems durch den Administrator. Sollen neue Lerngebiete in das System aufgenommen werden oder gibt es Änderungen der bestehenden Inhalte, so muss es einen Zugriff auf die Spezifikationen der Skills geben. Es könnten die festgelegte Eigenschaften geändert werden oder neue Inhalte hinzugefügt werden. Um einen neuen Skill anzulegen, der im Spiel eine erlernbare Fähigkeit darstellt, müssen die Voraussetzungen und die Abhängigkeiten zu anderen Skills festgelegt werden. Neben dem Anlegen von Skills sollen unter dem Gesichtspunkt der motivierenden Bestärkung auch besondere Belohnungen für den Spieler festlegbar sein.

Anforderungen *„Skill anlegen“*

- Erweiterbarkeit des Systems durch neue Lerneinheiten
- Anlegen von Skills
- Skillabhängigkeiten
- Skillpunkte
- Zuordnung der Skills zu Kategorien
- Definition von Belohnungen

Skillbaum ansehen

In diesem Anwendungsfall fragt der Administrator die bereits im System integrierten Skills ab. In einer Ansicht verschafft er sich einen Überblick über alle erlernbaren Fähigkeiten, sowie deren Abhängigkeiten untereinander. Dabei müssen die Voraussetzungen zum Erlernen eines Skills und die Zuordnungen zu Themenkomplexen ersichtlich werden.

Anforderungen *„Skillbaum ansehen“*

- Benutzeroberfläche zur Darstellung des Skillbaums mit definierten Abhängigkeiten und Voraussetzungen

Vorhandene Missionen ansehen

Missionen bestehen aus festgelegten Aufgaben, die aus mehreren Tätigkeiten bestehen können. Sie werden dem Spieler im Spielverlauf angeboten und dienen dem Erlernen von Fähigkeiten. Sie besitzen unterschiedliche Schwierigkeitsgrade und sollten daher erst bei einem ausreichenden Kenntnisstand des Spielers gespielt werden. In diesem Anwendungsfall informiert sich der Administrator über alle im System vorhandenen Missionen. Von besonderem Interesse sind dabei die Missionsvoraussetzungen und die Ziele, da sie darüber entscheiden, ob alle Skills im System erlernt werden können. Eine weitere wichtige Information ist die Anzahl der Punkte, die für die Tätigkeiten vergeben werden.
Dieser Anwendungsfall ist vergleichbar mit dem Anwendungsfall „Spielstand abfragen“ des Spielers.

Anforderungen *„Vorhandene Missionen ansehen“*

- festgelegte Aufgaben in Form von Missionen
- Missionsvoraussetzungen
- Missionsziele
- Festlegung der erlernbaren Fähigkeiten in den Missionen
- Benutzeroberfläche zur Darstellung der vorhandenen Missionsspezifikationen

Mission erstellen

Dieser Anwendungsfall stellt ebenso eine Erweiterung des Systems dar. Werden neue Skills in das System aufgenommen, dann könnte der Administrator Missionen anlegen, um dem Spieler das Erlernen dieser Fähigkeiten zu ermöglichen. Dazu müssen zunächst skillbasierte Voraussetzungen definiert werden, damit sichergestellt ist, dass dem Spieler diese Mission nur angeboten werden, wenn er über das nötige Wissen verfügt. Des Weiteren legt der Administrator die Aufgabe der Mission fest und bestimmt, wann diese erfüllt ist. Außerdem bestimmt er die Höhe der Vergütung.

Anforderungen *„Mission erstellen“*

- Erweiterung des Systems um neue Missionen
- skillbasierte Voraussetzungen
- Missionsinhalte
- Auszahlung von Vergütungen bei erfolgreichem Missionsabschluss

3.2 Modellentwicklung

Aus der Analyse der benutzerspezifischen Anforderungen resultiert, dass bei der Entwicklung des *Spieler-Modells* zwei Komponenten eine wichtige Rolle spielen. Zum einen wird ein Skill-System benötigt, um Lernkomplexe zu definieren und die erlernte Fähigkeiten des Spielers zu verwalten. Zum anderen müssen dem Spieler in der virtuellen Welt realitätsnahe Problemsituationen zum Erlernen dieser Inhalte geboten werden. Dies erfolgt über Missionen, in denen der Spieler mit verschiedenen Problemen konfrontiert wird. Zum Lösen dieser Konflikte werden gezielte Handlungen und Tätigkeiten des Spielers benötigt,

durch die er neue Fähigkeiten erlernen kann. In einem System müssen diese Missionen, die einen unterschiedlichen Schwierigkeitsgrad besitzen und daher gewisse Vorkenntnisse zum Spielen benötigen, organisiert werden. Ein hoher Spielspaß und ein optimaler Lernerfolg kann nur erzielt werden, wenn der Spieler nicht über- bzw unterfordert wird. Das Schwierigkeitsniveau muss dem Kenntnisstand des Spielers angepasst sein, um einen Flow-Effekt zu erreichen (siehe Kapitel 2.3). Daher liegt eine besondere Bedeutung des Modells in der Auswahl der geeigneten Missionen für den Spieler.

Das entworfene Modell besteht aus einem Skill-System und einem Missionsmanager. Das Skill-System verwaltet die Fähigkeiten und bewertet die spielerspezifischen Daten. Der Missionsmanager ist für alle Angelegenheiten zuständig, die sich im Zusammenhang mit der Erstellung und der Verwaltung der Missionen ergeben.
Nachfolgend werden die Funktionalitäten und Zuständigkeiten dieser beiden Komponenten beschrieben. Anschließend erfolgt eine Schilderung des Modellablaufs.

3.2.1 Skill-System

Skill-Systeme dienen der Organisation der Fähigkeiten von Mitgliedern einer Interessensgruppe, die gleiche oder ähnliche Ziele verfolgen. Sie werden auch als Erfahrungspunkte-Systeme bezeichnet und können zur Lernfortschrittskontrolle eingesetzt werden.
In der Wirtschaft nutzt man Skillmanagement-Systeme, um in einem Unternehmen die Qualifikationen der Mitarbeiter zu verwalten und um diese dann gezielt einzusetzen. Auf Grundlage dieser Skillmanagement-Mechanismen werden in der digitalen Spieleentwicklung oftmals sehr komplexe Skill-Systeme zur Verwaltung der Spielerfertigkeiten entworfen. Dazu werden erlernbare Fähigkeiten bzw. Fertigkeiten in Form von Skills definiert. Diese besitzen meist unterschiedliche Abhängigkeiten untereinander und können durch verschiedene Spielmechanismen erlernt werden. Skill-Systeme finden vorwiegend im Bereich der MOGs ihren Einsatz.
Ein Beispiel für ein sehr populäres MOG ist *Eve-Online* [CCP]. Das in diesem Rollenspiel integrierte Skill-System enthält eine Vielzahl von erlernbaren Skills, die in Kategorien und Unterkategorien eingeteilt werden. Die Besonderheit liegt darin, dass die Fähigkeiten allein durch den Aufwand von Zeit erlernt werden können. Das bedeutet, die angeeigneten Skills beschreiben nicht erlernte Fähigkeiten des realen Spielers, sondern zeichnen nur den virtuellen Charakter aus. Die Zeit, die zum Erlernen eines Skills benötigt wird, berechnet sich aus einer Formel, in der unter anderem die vorhandenen Charaktereigenschaften eine Rolle spielen. Zu jedem Skill sind maximal 5 Levelstufen zu erlernen, für die unterschiedlich viele Punkten benötigt werden. Das Erlernen eines Skills kann wenige Minuten bis mehrere Tage dauern. Die Zeit läuft jedoch auch außerhalb der Spielzeit weiter.

Für das *Spieler-Modell* wird ein Skill-System entwickelt, welches zwei grundlegende Aufgaben erfüllt. Auf der einen Seite dient es zur Verwaltung der erlernbaren Fähigkeiten und zur Kontrolle des Lernfortschritts. Andererseits soll, unter Berücksichtigung der motivierenden Bestärkung, durch das Feedback des Skill-Systems und durch den Einsatz von belohnenden Elementen eine anhaltende Motivationssteigerung erreicht werden.

Da das *Spieler-Modell* für eine spielbasierte Lernumgebung konzipiert ist, in der eine möglichst realitätsnahe Wissensvermittlung stattfinden soll, stellen die erlernbaren Inhalte vorwiegend reale Fähigkeiten dar, die der Spieler in der virtuellen Welt erlernen kann. Es ist jedoch dem Anwender überlassen, in welchem Kontext das Modell eingesetzt werden soll.
Die Lerninhalte werden durch erlernbare Fähigkeiten (Skills) in der virtuellen Welt definiert. Diese Skills besitzen eine hierarchische Abhängigkeitsstruktur, die beliebig festgelegt werden kann und durch welche die grundlegende Reihenfolge beim Erlernen der Fähigkeiten festgelegt wird. Im Gegensatz zu *Eve Online* können diese Skills nicht über die Zeit erlernt werden, sondern nur über erfolgreich gelöste Aufgaben (Missionen), durch welche der reale Spieler seinen Fähigkeiten beweist.
Im Skill-Modell werden die Abhängigkeiten zwischen den Skills über benötigte Skillpunkte (Erfahrungspunkte) definiert. Diese Erfahrungspunkte kann der Spieler durch erfolgreich absolvierte Missionen sammeln. Dazu werden die Tätigkeiten des Spielers bewertet und bei Erfüllung einer festgelegten Bedingung, dem Spieler ausgezahlt. Dabei ist darauf zu achten, dass nicht nur das Spielziel, sondern auch das Lernziel in die Bewertung einfließt [Spr10]. Diese Forderung ist entscheidend, um sicher zu gehen, dass der Spieler wirklich die Lerninhalte erworben hat und nicht nur Wege gesucht hat, das Spielziel zu erreichen.
Eine wichtige Bedeutung hat in diesem Zusammenhang das Bewertungssystem. Eigentlich müsste es eine Zweiteilung geben, sodass einerseits die einzelnen Tätigkeiten in einer Mission bewertet werden und andererseits die eigentlichen Fähigkeiten des Spieler. Über diese Zweiteilung könnte eine differenzierte Beurteilung der Fähigkeiten stattfinden. Im Rahmen der Bachelorarbeit wurden die Missionen jedoch nur prototypisch umgesetzt, da die Entwicklung des *Spieler-Modells* im Vordergrund steht und die Spielinhalte (Missionsinhalte) in dem Framework, in dem das Modell integriert werden soll, noch nicht implementiert sind. Aus diesem Grund bestehen die Missionen nur aus Elementen, die für das Modell eine entscheidende Rolle spielen. Die einzelne Tätigkeiten in einer Mission sind noch nicht definiert. Daher basiert das Skill-System nur auf Skillpunkten, die nach einer Entscheidung über den Erfolg einer Mission, ausgezahlt werden. Das Sammeln der Skillpunkte führt schließlich zum Erlernen der Skills. Dabei ist keine maximal zu erreichende Punkthöhe festgelegt. Der Spieler kann sich wie im realen Leben immer weiter spezialisieren. Lediglich über die Abhängigkeiten wird festgelegt, welche Punkthöhe erreicht werden muss, um den entsprechenden Skill zu erlernen. Da das Lösen einer Aufgabe immer Kenntnisse aus verschiedenen Bereichen voraussetzt und Lernen nie in einem thematisch abgeschlossenen Raum stattfindet, ermöglicht die Umsetzung des Modells, das Erlernen mehrerer Skills in einer Mission. Diese Skills müssen nicht aus dem selben Themengebiet stammen. Die Einteilung der Lerninhalte in einzelne Themengebiete und Kategorien wird ebenfalls durch das Modell ermöglicht. Dabei können Skills gruppiert und somit Lerneinheiten festlegen werden.

Zusammenfassend ergibt sich, dass die Zuständigkeit des Skill-Systems in der Bereitstellung und Verwaltung der Lerninhalte liegt. Ebenfalls müssen von dieser Systemkomponente die Funktionalitäten zur Verwaltung der spielerbezogenen Skilldaten bereitgestellt

werden. Die extrinsische Motivation wird über zusätzliche Belohnungen, die in der Spieleentwicklung als *Badges* bezeichnet werden und durch das Erfahrungspunkte-System gesteigert. In den folgenden Abschnitten werden die einzelnen Elemente des Skill-Systems beschrieben.

Skills

Der Administrator kann die im Spielverlauf erlernbaren Fähigkeiten über Skills definieren. Dabei können zunächst Kategorien angelegt werden, zu der beliebig viele Skills erstellt werden können. Diese Skills bestehen aus einer Beschreibung der Fähigkeit, einer Zuordnung zu einer Kategorie und den Voraussetzungen zum Erlernen. Über die Definition von Voraussetzungen, die aus Skills und einer benötigten Skillpunkthöhe bestehen, kann eine hierarchische Abhängigkeitsstruktur (Skillbaum) realisiert werden. Diese Möglichkeit ist in einer lernbasierten Umgebung von großem Interesse, da in vielen Lernsystemen einzelne Lernkomplexe aufeinander aufbauen und zum Erlernen neuer Inhalte ein bestimmtes Vorwissen benötigt wird. Besitzt ein Skill keine Voraussetzungen, kann dieser jederzeit vom Spieler erlernt werden. Üblicherweise wird dieser Fall bei Grundlagenskills angewendet. Diese variable Festlegung der Abhängigkeiten ermöglicht es, das Modell auch in einem Kontext einzusetzen, in dem die erlernbaren Fähigkeiten nicht aufeinander aufbauen sollen.
Über ein Erfahrungspunkte-System können in den Missionen Punkte gesammelt werden. Diese stellen einerseits den Lernverlauf dar und belohnen andererseits den Spieler für richtiges Handeln. Ein solches Punkte-System hat wie im Kapitel2.4) beschrieben, eine motivierende Wirkung.

Skillbaum

Ein Skillbaum repräsentiert die Abhängigkeiten zwischen den Skills. Dieser kann in einer Benutzeroberfläche, die vom Skillmanager bereit gestellt wird, angezeigt werden. Dabei unterscheiden sich zwei Anwendungsfälle. Auf der Serverseite wird eine Benutzeroberfläche bereitgestellt, die alle im System integrierten Skills und deren Abhängigkeiten anzeigt. Diese Visualisierung ist für den Administrator wichtig, um einen Überblick über alle Lernkomplexe und erlernbaren Fähigkeiten zu erhalten. Auf der Clientseite kann der Spieler sich seinen persönlichen Skillbaum anzeigen lassen. Diese Veranschaulichung stellt eine Erweiterung der Benutzeroberfläche des Administrators dar. Dabei werden neben der Visualisierung des allgemeinen Skillbaums auch die spielerspezifischen Skillausprägungen angezeigt.
Die Kontrolle des eigenen Lernverlaufs hat eine entscheidende Bedeutung für den Lernerfolg [TPR+92]. Eine wesentliche Steigerung kann erreicht werden, wenn der Lernende seinen eigenen Lernprozess überwachen und bewerten kann. Ebenso müssen erreichbare Ziele veranschaulicht werden, damit der Spieler Strategien zum Erreichen dieser Ziele entwickeln kann. Die Visualisierung des Skillbaums, mit den Abhängigkeiten und den benötigten Voraussetzungen zum Erlernen eines neuen Skills, bieten dem Spieler die Möglichkeit, in einem

individuellen Lernverlauf neue Kompetenzen zu erlangen. In der Ansicht des Skillbaums werden dem Spieler ebenfalls seine erreichten Skillpunkte angezeigt und erlernte Skills hervorgehoben.

Badges

Als *Badges* werden in einer virtuellen Spielwelt zusätzliche Belohnungen für den Spieler bezeichnet. Wie im Kapitel2.4) beschrieben, führen diese zu einer zusätzlichen Steigerung der Motivation. *Badges* belohnen durch die Verleihung des Titels des Erfolgs. Das bedeutet, der Spieler erhält sogenannte Zertifikate für festgelegte Ziele, welche dem Spieler persönliche Fähigkeiten bescheinigen. Diese spielen vor allem in MOGs eine wichtige Rolle. MOGs sind so konzipiert, dass verschiedene Problemsituationen nur gelöst werden können, wenn Spieler gemeinschaftlich Lösungsstrategien entwickeln und somit vom Können anderer profitieren. *Badges* helfen dabei, geeignete Spieler zu finden, die bereits das Spezialwissen auf dem benötigten Gebiet besitzen.
Das Skill-Modell ermöglicht neben der Definition von Skills auch das Anlegen von *Badges*. Diese werden vom Administrator angelegt und im Skill-System gespeichert. Ein *Badge* besteht aus einer Beschreibung und den benötigten Voraussetzungen. Diese sind ebenfalls über Skills und deren Skillpunkthöhe definiert.

3.2.2 Missionsverwaltung

Die zweite grundlegende Komponente des *Spieler-Modells* ist ein Missionsverwaltungssystem (Missionsmanager). Dieser bietet eine Schnittstelle zum Anlegen von Missionen und stellt verschiedene Funktionalitäten zur Verwaltung der Missionen bereit. Dazu zählt zum einen die Speicherung der Missionsdaten und zum anderen die Auswahl der geeigneten Missionen für den Spieler. Dazu muss eine Kommunikation zwischen dem Missionsmanager und dem Skill-System stattfinden, um den aktuellen Fähigkeitsstand des Spielers abzufragen. In den folgenden beiden Abschnitten wird zunächst der Aufbau einer Mission beschrieben und anschließend das Verfahren zur Auswahl der geeigneten Missionen vorgestellt.

Missionen

In einer virtuellen Spielwelt werden Missionen erstellt, um dem Spieler durch das Lösen festgelegter Aufgaben eine gezielte Möglichkeit zum Trainieren und Erlernen bestimmter Fähigkeiten zu bieten. Sie sind darauf ausgelegt, in einer kontextbezogenen Problemsituation den Spieler zu systematischem Denken und zum Erproben von Lösungswegen zu motivieren. Dabei profitiert der Spieler von dem Prinzip der Risikofreiheit, da die Handlungen in einer virtuellen Spielwelt wesentlich geringere Folgen haben (siehe Kapitel 2.2). Die festgelegte Aufgabe in einer Mission, kann aus einer oder mehreren Tätigkeiten für den Spieler bestehen, durch die er Erfahrungen und Wissen sammelt.

Wie bereits erwähnt, wurden die Missionen nur prototypisch umgesetzt. Zu den notwendigen Elementen einer Mission für das Skill-Modell, zählen die Missionsvoraussetzungen und die erlernbaren Fähigkeiten. Die Voraussetzungen einer Mission beschreiben benötigtes Vorwissen, um die definierte Aufgabe lösen zu können. Diese Voraussetzungen werden über Skills und die benötigte Skillpunkthöhe festgelegt. Es ist möglich, Missionen anzulegen, die keine Voraussetzungen besitzen. Diese können dann jederzeit vom Spieler gespielt werden. Die dabei erlernten Fähigkeiten werden ebenfalls über Skills und eine festgelegte Skillpunkthöhe beschrieben. Dabei können Skillpunkte für mehrere Skills aus verschiedenen Kategorien vergeben werden.
Beim Anlegen der Missionsvoraussetzungen und den Missionsauszahlungen sollte sich grundsätzlich an das Skill-System gehalten werden. Denn das Skill-System definiert die Struktur der Lerninhalte. Um das Interesse des Spielers zu wecken, indem er in Skills *„reinschnuppern“* kann, für die er laut Skill-System noch nicht die vorausgesetzten Skillpunkte erfüllt, ist es sinnvoll, Zusatzmissionen anzubieten. Aus diesem Grund kann eine Mission sich über die Restriktionen des Skill-Systems hinwegsetzen. Das Modell bietet die Möglichkeit, solche zusätzlichen Missionen zu definieren. Diese Zusatzmissionen können als weitere Belohnungsart eingesetzt werden.

Missionsauswahl

Zu der zentralen Aufgabe des Missionsmanagers gehört die Auswahl der spielbaren Missionen für den Spieler. Dabei müssen die vorhandenen Skillpunkte des Spielers mindestens die in den Missionsvoraussetzungen geforderten Punkte erfüllen. Aus allen im System vorhandenen Missionen werden, soweit möglich, dem Spieler immer mehrere spielbare Missionen angeboten, damit der Spieler seine eigene Spielstrategie umsetzen kann. Er muss ebenfalls Informationen darüber erhalten, welche Skillpunkte er mit der Mission erreichen kann.
Dazu wird dem Spieler in einer Benutzeroberfläche eine Liste mit spielbaren Missionen und erreichbaren Punkten vom Missionsmanager angezeigt. Aus dieser Liste kann der Spieler eine Mission wählen und diese starten. Der Erfolg der Mission wird daran gemessen, ob die festgelegte Bedingungen beim Spielen erfüllt wurden. Abschließend werden dem Spieler die Skillpunkte zu seinem aktuellen Spielstand aufsummiert.

3.2.3 Exemplarischer Programmablauf

Um den Programmablauf zu demonstrieren wurde ein UML-Aktivitätsdiagramm erstellt (siehe Abbildung 3.2). Mit Hilfe eines Aktivitätsdiagramms kann der Ablauf der Aktivitäten eines zu modellierenden Systems dargestellt und somit das Verhalten des Systems beschrieben werden [MH06].
Ein schwarzer Punkt verdeutlicht den Startknoten in einem Aktivitätsdiagramm. Das Gegenstück dazu ist der Endknoten, dargestellt durch einen umrandeten schwarzen Punkt. Die Aktivitäten werden durch Rechtecke mit abgerundeten Ecken abgebildet. Der Übergang zwischen den Aktivitäten wird durch die Pfeile verdeutlicht. Dabei repräsentieren die

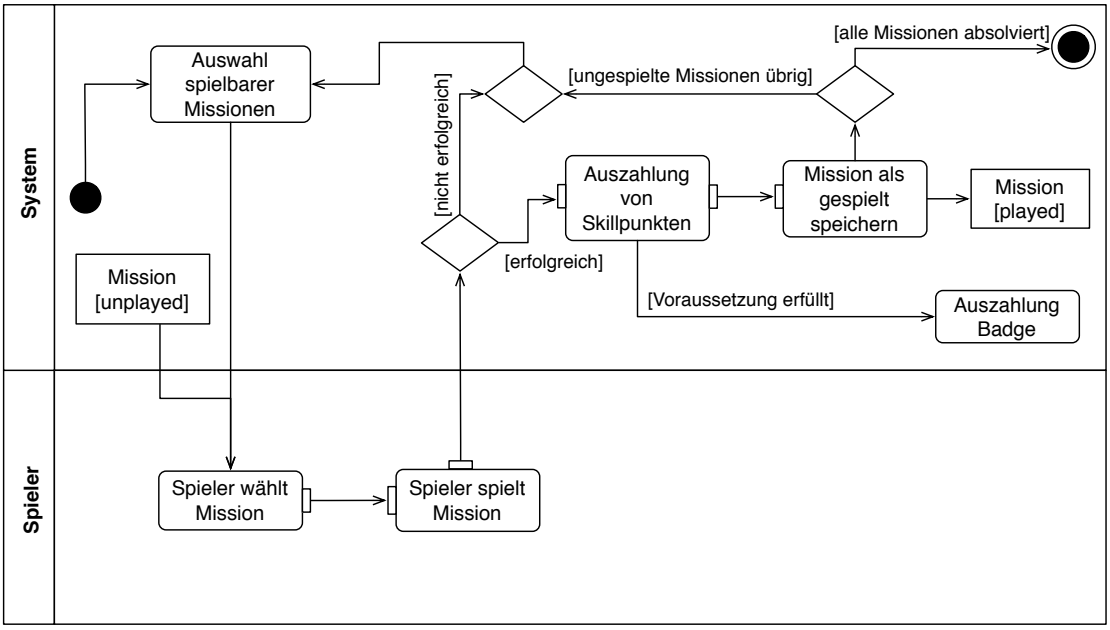

Abbildung 3.2: UML-Aktivitätsdiagramm zur Visualisierung des Programmablaufs

Rechtecke die Objektknoten. Die kleinen Rechtecke am Rande der Aktivitäten visualisieren die Eingabe- und Ausgabepins, das bedeutet, sie stellen die Eingabewerte einer Aktivität und die resultierenden Ausgabewerte dar. Rauten stehen für Verzweigungen bzw. Entscheidungen im Programmablauf.

Die in Abbildung 3.2 dargestellten Aktivitäten des *Spieler-Modells* sind in die Aktivitäten des Spielers und des Systems partitioniert. Das System startet mit der Auswahl der spielbaren Missionen. Durch diese Aktivität wird sichergestellt, dass der Spieler nur Missionen angeboten bekommt, für die er mindestens die benötigen Voraussetzungen erfüllt. Der Missionsmanager speichert zu jedem Spieler die noch nicht gespielten Missionen. Aus dieser Liste werden die Voraussetzungen der Missionen mit den Skillpunkten des Spielers verglichen. Alle Missionen, für die der Spieler die geforderten Voraussetzungen erfüllt, werden in einer Liste zusammengefasst. Aus dieser Liste wählt der Spieler eine Mission und spielt diese.
Da der Schwerpunkt der Bachelorarbeit in der Umsetzung der serverseitigen Verwaltungsprozesse der Skills und Missionen liegt, steht die Entwicklung der clientseitigen Tätigkeiten der Benutzer weniger im Mittelpunkt. Aus diesem Grund wurden die Tätigkeiten des Spielers nur auf die für das Modell benötigten Handlungen beschränkt und sind daher in den rudimentären Aktivitäten *Spieler wählt Mission* und *Spieler spielt Mission* zusammengefasst.
Nachdem der Spieler eine Mission gewählt hat, werden die Daten dieser Mission in einem aktiven Zustand vom System gespeichert und den weiteren Aktivitäten zur Verfügung ge-

stellt. Hat der Spieler die Mission beendet, wertet das System aus, ob die festgelegten Ziele der Mission durch die Tätigkeiten des Spielers erreicht wurden. Das Aktivitätsdiagramm besitzt an dieser Systemstelle durch die Raute eine Verzweigung. Wurden die Missionsziele nicht erreicht, bleibt die Mission als ungespielt im System gespeichert und wird bei einer wiederholten Auswahl von spielbaren Missionen erneut berücksichtigt. Konnte der Spieler die Mission erfolgreich beenden, werden ihm Punkte für seine Spieltätigkeiten ausgezahlt. Das System fragt von der aktuell gespielten Mission die festgelegten Skillpunkte ab. Dies können Punkte in beliebiger Höhe und für verschiedene Skills sein. Diese Auswertung übernimmt der Missionsmanager, der anschließend die Skillpunkte dem Skill-System mitteilt. Dadurch werden die Skillpunkte des Spielers erhöht und dessen Fähigkeiten erweitert. Bei jeder Skillauszahlung wird ebenfalls überprüft, ob die Voraussetzungen zur Auszahlung von Badges gegeben sind. Nach dieser Aktivität wird die Mission im System als gespielt gespeichert und daher bei einer erneuten Auswahl von spielbaren Missionen nicht wieder berücksichtigt. Nach der Beendigung dieses Prozesses werden, soweit weitere ungespielte Missionen für den Spieler im System vorhanden sind, erneut spielbare Missionen ermittelt. Wurden alle Missionen vom Spieler absolviert, erreicht das Modell seinen Endknoten.

4 Prototypische Implementierung

Aufbauend auf den Konzeptentwurf, wird in diesem Kapitel zunächst die prototypische Implementierung erläutert und anschließend die Integration in das bestehende Framework beschrieben.
Die Grundlage für die Implementierung bildet eine Anforderungsanalyse. Die Anforderungen ergeben sich aus dem Konzeptentwurf und den Vorgaben der bestehenden Simulationssoftware. Anschließend wird die Systemarchitektur vorgestellt und die Umsetzung der Funktionalitäten des *Skill-* und *Missionsmanagers* beschrieben. Unter dem Punkt 4.4 werden die verschiedenen Benutzeroberflächen, die vom Modell bereitgestellt werden, sowie das Feedback-System für den Spieler vorgestellt. Im letzten Abschnitt des Kapitels wird die Integration in das bestehende Framework, sowie die Möglichkeit der Erweiterung des Modells durch den Administrator, erläutert.

4.1 Anforderungsanalyse

Ein *modularisiertes Konzept* ist eine wichtige Anforderung an die Implementierung des Modells, um einen flexiblen Einsatz zu ermöglichen. Bei diesem Ansatz werden die einzelnen Systemelemente in verschiedenen Modulen umgesetzt. Ein Softwaremodul beschreibt ein abgeschlossenes System, welches die Funktionalitäten einer Systemeinheit zusammenfasst. Durch eine modulare Softwarestruktur wird das Prinzip der Kapselung realisiert, indem die Daten und Funktionalitäten vor dem Zugriff von außen verborgen werden. Die Kommunikation zu anderen Modulen erfolgt ausschließlich über eine eindeutig definierte Schnittstelle.
Für die Entwicklung des *Spieler-Modells* wurden zwei Module implementiert. Dazu zählt ein *Skillmanager* zur Verwaltung des Skill-Systems und ein *Missionsmanager*. Da in dem konzipierten Modell die einzelnen Skills nicht zwangsläufig mit den Missionen verflochten sind, besitzt der *Skillmanager* keine Abhängigkeiten zu anderen Modulen. Der *Missionsmanager* ist vom *Skillmanager* abhängig, da die Missionen auf den Skills aufbauen. Die Integration der Daten[1] erfolgt über zwei festgelegte Schnittstellen des *Skill-* und *Missionsmanagers*, durch welche die Daten von mehreren Modulen eingelesen werden können. Dies ist für eine Software sinnvoll, in denen die Module einzelne Lehrbereiche darstellen.

Aus diesem Sachverhalt ergibt sich, dass die *Konsistenzerhaltung beim Laden der Daten* berücksichtigt werden muss. Werden Daten eines Moduls geladen, kann es passieren, dass

[1] Mit Daten sind die vom Administrator festgelegte Spielinhalte gemeint.

diese von einem anderen Modul referenziert wurden, bevor diese Daten geladen werden. Dieser Fall kann beispielsweise eintreten, wenn ein Modul einen Skill anlegt, dessen vorausgesetzte Skills von einem anderen Modul definiert werden. Um dem Anwender trotzdem eine flexible Gestaltung der Abhängigkeiten zwischen den Skills zu ermöglichen, muss ein Mechanismus entwickelt werden, der die Einhaltung der Konsistenz der Daten überprüft und sicherstellt.

Die *Speicherung der spielerbezogenen Daten* stellt ebenfalls eine wichtige Anforderung an die Implementierung dar. Dabei muss auf die Verhinderung von Datenverlusten und auf Datensparsamkeit geachtet werden. Um die spielerbezogenen Daten zu speichern, müssen neben den allgemeinen Skills und Missionen, für die Spieler zusätzlich die konkreten Zustände der Missionen bzw. Ausprägungen der Skills gespeichert werden. Die allgemein existierenden Skills und Missionen stellen somit nur eine Schablone für die entsprechenden Zustände dar. Sie beschreiben eine Art Erweiterung der Skill- und Missionsobjekte. Diese sollen im Sinne der Datensparsamkeit im System nur einmal existieren und nicht für jeden Spieler mehrfach angelegt werden. Damit Änderungen sich auf das gesamte System auswirken dürfen von den Objekten auch gar keine Kopien angelegt werden. Auch an anderen Stellen im System, wie z.B. bei den Vorraussetzungen der Skills und Missionen dienen konkrete Ausprägungen der Skills dazu einen klar festgelegten Betrag eines Skills zu definieren.

Die Implementierung eines *Bewertungssystems* ist eine weitere wesentliche Anforderung, um die Handlungen des Spielers und den Spielfortschritt zu beurteilen. Dafür muss ein flexibles Verfahren implementiert werden, welches Entscheidungen ermöglicht, ob eine Mission erfolgreich beendet wurde. Es ist ebenfalls nötig, eine Benachrichtigungsstruktur zu entwerfen, um beim Starten und Beenden der Missionen auf den entsprechenden Spielzustand zu reagieren. Dazu zählt unter anderem die Auszahlung der Belohnungen für den Spieler bei erfolgreichem Missionsabschluss.

4.2 Systemarchitektur

Die im Konzept vorgestellten Systeme zur Verwaltung der Skills und Missionen wurden in den Modulen *Skillmanager* und *Missionsmanager* umgesetzt (siehe Abbildung 4.1). Der *Skillmanager* stellt das Basismodul zur Verwaltung des Skill-Systems dar. Dieser besitzt keine Abhängigkeiten zu anderen Modulen. Der *Missionsmanager* ist eine inkrementelle Erweiterung des *Skillmanagers* und ist daher von diesem Modul abhängig.
Beiden Module besitzen jeweils eine Managerklasse, die als Container für die jeweiligen Inhaltsdaten dienen. Sie speichern sowohl die angelegten Skills und Missionen, als auch die spielerbezogenen Ausprägungen zu diesen Schablonen. Die Managerklassen sind jeweils als *Singleton* implementiert. In der objekt-orientierten Programmierung beschreibt ein *Singleton* ein Entwurfmuster aus dem Teilbereich der Erzeugungsmuster [GHJV95]. Als Entwurfsmuster (Design Pattern) werden vorgefertigte Lösungsstrategien für wiederkehrende Entwurfsprobleme in der Softwareentwicklung bezeichnet. Sie stellen einen rezeptartigen Vorschlag zur Lösung dieser Entwicklungsprobleme dar.

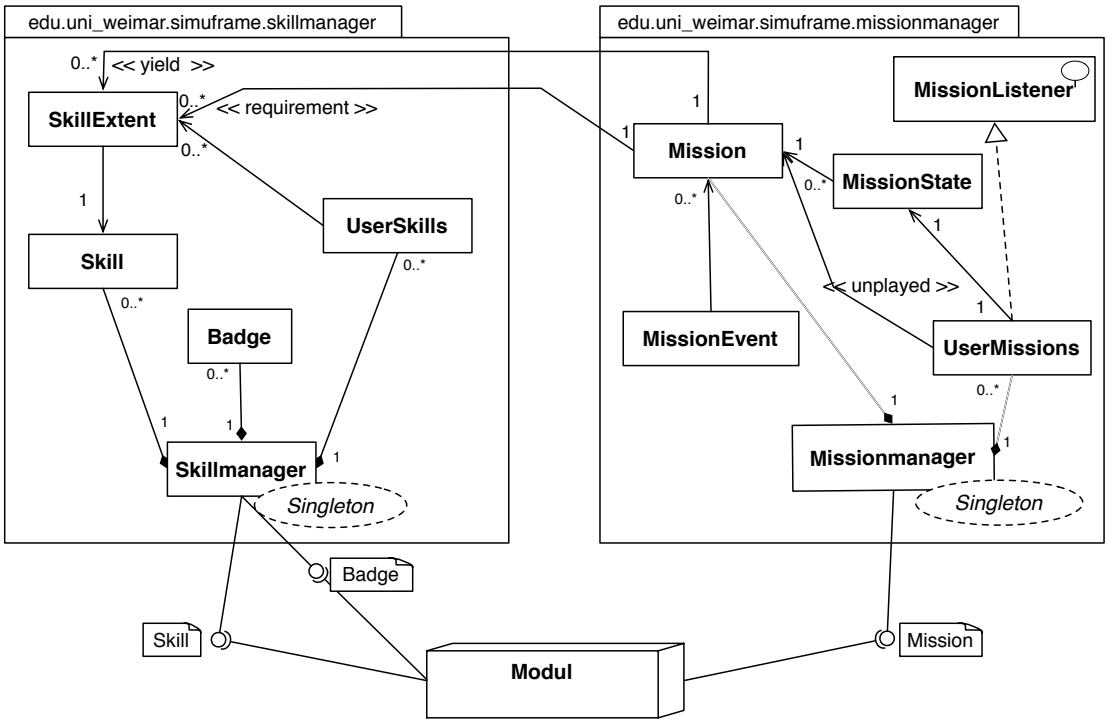

Abbildung 4.1: Darstellung der Systemarchitektur

Das *Singleton-Pattern* wird typischerweise bei der Implementierung von Managerklassen eingesetzt. Im Programmablauf darf vom Objekt *Manager* nur eine Instanz erzeugt werden, auf welche global zugegriffen werden muss. Dieses Prinzip findet auch beim Skill- und Missionsmanager Anwendung, welche Skills und Missionen global verwalten.
Zur Realisierung des *Singleton-Pattern* muss die direkte Instantiierung des Objektes über den Standardkonstruktor verhindert werden, indem der Standardkonstruktor als `private` gekennzeichnet wird. Dies hat zur Folge, dass nur die Klasse selber auf den Konstruktor zugreifen kann. Über die statische Methode `getInstance()` kann außerhalb der Klasse auf die einzige Instanz des Objekts, welche in der Klasse selbst als statisches Attribut gespeichert ist, zugegriffen werden. Dabei wird zunächst überprüft, ob bereits eine Instanz dieses Objekts existiert. Gegebenenfalls wird eine Instanz angelegt.
Durch dieses Vorgehen wird sichergestellt, das jeweils nur ein *Skillmanager* und ein *Missionsmanager* zur Verwaltung der Daten existiert. Im Folgenden werden die Implementierungen der beiden Module genauer beschrieben.

4.2.1 Skillmanager

Im *Skillmanager* sind die für das Skill-System benötigten Klassen und Funktionalitäten implementiert. Abbildung 4.2 zeigt das UML-Klassendiagramm des *Skillmanagers*.

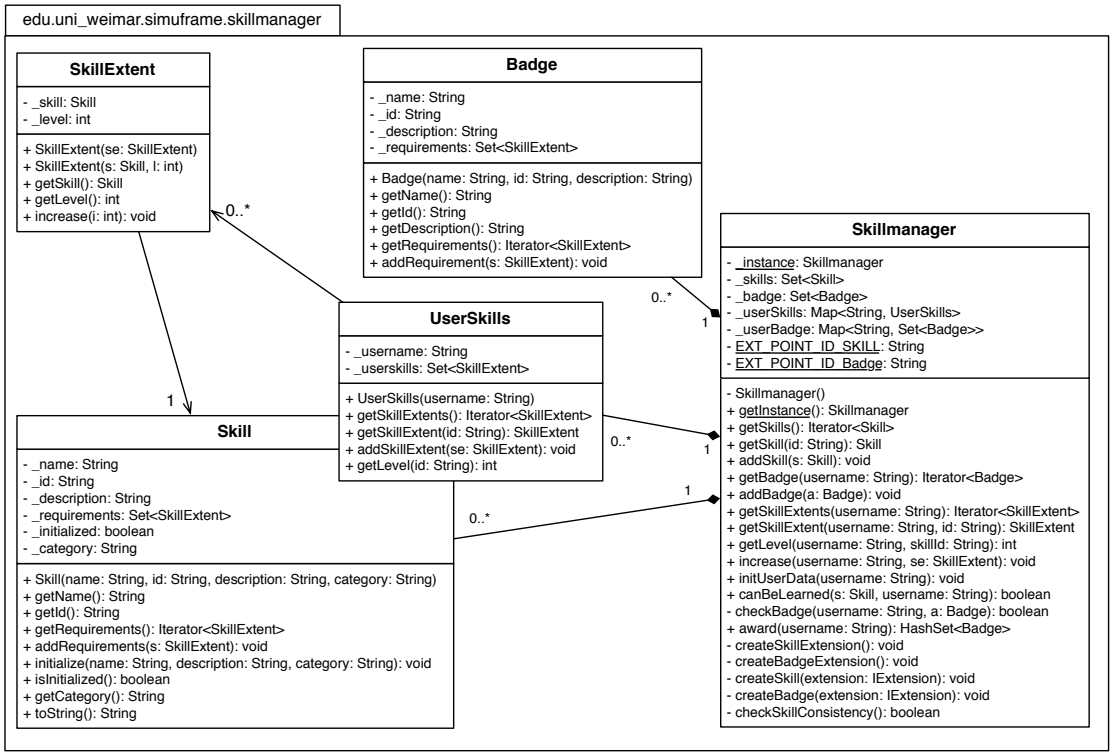

Abbildung 4.2: UML-Klassendiagramm des Skillmanagers

Neben dem bereits beschriebenen *Skillmanager* gehören die Skillobjekte zu den grundlegenden Bestandteilen. Diese Skills bestehen aus einem Namen, einem Identifier (Id), einer Beschreibung und einer Zugehörigkeit zu einer Kategorie. Über die Festlegung von Voraussetzungen werden die Skills in eine hierarchische Struktur (Skillbaum) gebracht [Joh09]. Wie in den Anforderungen erläutert, dürfen für die Festlegung der skillbasierten Voraussetzungen keine Kopien der Skillobjekte angelegt werden. Die Klasse `SkillExtent` stellt die Ausprägung zu einem Skill dar und speichert den Level, der die Skillpunkthöhe festlegt. Somit können in den einzelnen Skills vorausgesetzte `SkillExtents` gespeichert werden.
Das Einlesen der Daten erfolgt über zwei bereitgestellte *Extension Points*[2] des *Skillmanagers*, über welche die Skills und Badges in das System integriert werden können (siehe

[2] Erweiterungspunkte

Abbildung 4.1). Die genaue Beschreibung dieses Mechanismus erfolgt im Kapitel 4.3. Über diesen *Extension Point* können andere Module, die Inhaltsdaten bereitstellen, mit dem *Skillmanager* kommunizieren. An dieser Stelle kann das in den Anforderungen beschriebene Problem der Konsistenzerhaltung beim Einlesen der Daten auftreten. Der Administrator legt beim Anlegen eines Skills fest, welche Skills vorausgesetzt werden sollen. Dies erfolgt über die Id des Skills. Werden nun die Skills geladen, kann es passieren, dass zu einer Id, die einen vorausgesetzten Skill referenziert, noch keine Instanz im System existiert. Dieser Fall tritt ein, wenn der vorausgesetzte Skill erst zu einem späteren Zeitpunkt vom System geladen wird. An dieser Stelle könnte es passieren, dass beim Anlegen der `SkillExtents` ein Skill referenziert werden soll, der eben noch nicht existiert. Zur Lösung dieses Problems werden sogenannte *Platzhalter*-Skills angelegt. Das bedeutet, wenn ein Skill referenziert werden soll, der nicht im System vorhanden ist, wird zunächst ein neuer Skill definiert, der nur aus der bekannten Id besteht. Alle anderen Eigenschaften bekommen den Wert `null` zugewiesen. Um diese *Platzhalter*-Skills noch korrekt zu initialisieren, ohne dass die Skills doppelt angelegt werden oder unvollständige Skills im System vorhanden sind, wird vor jedem Anlegen eines neuen Skills die Methode `isInitialized()` aufgerufen. Diese überprüft über die Id, ob für den Skills bereits ein *Platzhalter*-Skill angelegt wurde. Wenn dieser Fall eintritt, wird der *Platzhalter*-Skill über die `initialize()`-Methode mit dem Namen, der Beschreibung und der Kategorie vervollständigt. Existiert noch kein Skill mit der Id im System, kann dieser Skill ohne weitere Beachtung über seinen Konstruktor instantiiert werden. Jeder Skill besitzt zusätzlich eine Variable zur Überprüfung auf korrekte Initialisierung. Nach dem Anlegen aller Daten dürfen im System keine Skills mehr existieren, deren Überprüfungsvariable nicht den Wert `true` liefert. Durch diesen Mechanismus ist es möglich, modulunabhängige Skillvoraussetzungen festzulegen. Der folgende Programmcode verdeutlicht diesen Mechanismus.

```
Skill s = getSkill(id);
s.initialize(name, description, category);

for (final IConfigurationElement configurationChild : configurationElement
    .getChildren()){

    Skill requiredSkill = getSkill(configurationChild.getAttribute("id"));
    if(requiredSkill == null)
        requiredSkill = new Skill(null, configurationChild.
            getAttribute("id"), null, null);
        addSkill(requiredSkill);

        int level = 0;
        try {
            level = Integer.parseInt(configurationChild
                .getAttribute("level"));
        }
        catch (Exception E){
```

```
            System.out.println(requiredSkill.getName() +
                "has invalid requirement level");
        }
        SkillExtent sl = new SkillExtent(requiredSkill, level);
        s.addRequirement(sl);
}
```

Zur Verwaltung der spielerspezifischen Daten wurde die Klasse `UserSkills` erstellt (siehe Abbildung 4.2). Diese dient der Verwaltung der vom Spieler erzielten Skillausprägungen. Für jeden Spieler wird dieses Objekt angelegt, in dem der zugehörigen Spielername und die `SkillExtents` des Spielers gespeichert werden. Die `UserSkills` werden im *Skillmanager* verwaltet. Sie betreiben mit den `SkillExtents` *lazy instantiation*, eine Vorgehensweise, die ebenfalls zu den Entwurfsmustern zählt [Bis]. *Lazy instantiation* beschreibt ein Verfahren der Instantiierung, bei der die Instanz eines Objektes erst dann angelegt wird, wenn das Objekt benötigt wird. Somit können Ressourcen gespart werden.

Die Erhöhung der Skillpunkte erfolgt über die `SkillExtents`. In den Missionen sind ebenfalls `SkillExtents` gespeichert, welche die Skills und die zu erreichende Punkthöhe angeben. Diese `SkillExtents` werden durch die `increase()`-Funktion zu den `SkillExtents` des Spielers hinzugerechnet. Dabei ist die maximal zu erreichende Punkthöhe für einen Skill nicht festgelegt.

Badges bestehen aus einem Namen, einer Id, einer Beschreibung und den Voraussetzungen. Die Voraussetzungen werden über `SkillExtents` definiert. Nach dem Aufruf der `increase`-Funktion und der Auszahlung der `SkillExtents` von den Missionen, wird überprüft, ob der Spieler die benötigten Voraussetzungen für die Verleihung eines *Badges* erfüllt.

4.2.2 Missionsmanager

Der *Missionsmanager* stellt das Modul zur Verwaltung der Missionen dar. In der Abbildung 4.3 sind die implementierten Klassen dieses Moduls dargestellt.

Die Missionen können ebenfalls über einen *Extension Point* vom Administrator angelegt werden (siehe Abbildung 4.1). Zur Demonstration der Funktionalitäten des Modells sind die Missionen als Prototyp implementiert und besitzen nur die Elemente und Funktionalitäten, die für das Skill-System benötigt werden. Eine Mission besteht aus einem Namen, einer Id, einer Beschreibung, den Voraussetzungen zum Spielen und den Auszahlungen bei einem erfolgreichen Missionsabschluss. Zusätzlich speichert eine Mission *Conditions* (Bedingungen), welche beim Spielen der Mission erfüllt werden müssen, damit die Mission erfolgreich beendet werden kann.

Vergleichbar mit der Verwaltung der Skillausprägungen im *Skillmanager*, werden im *Missionsmanager* die Zustände der Missionen ebenfalls spielerbezogen gespeichert. Dafür wurde

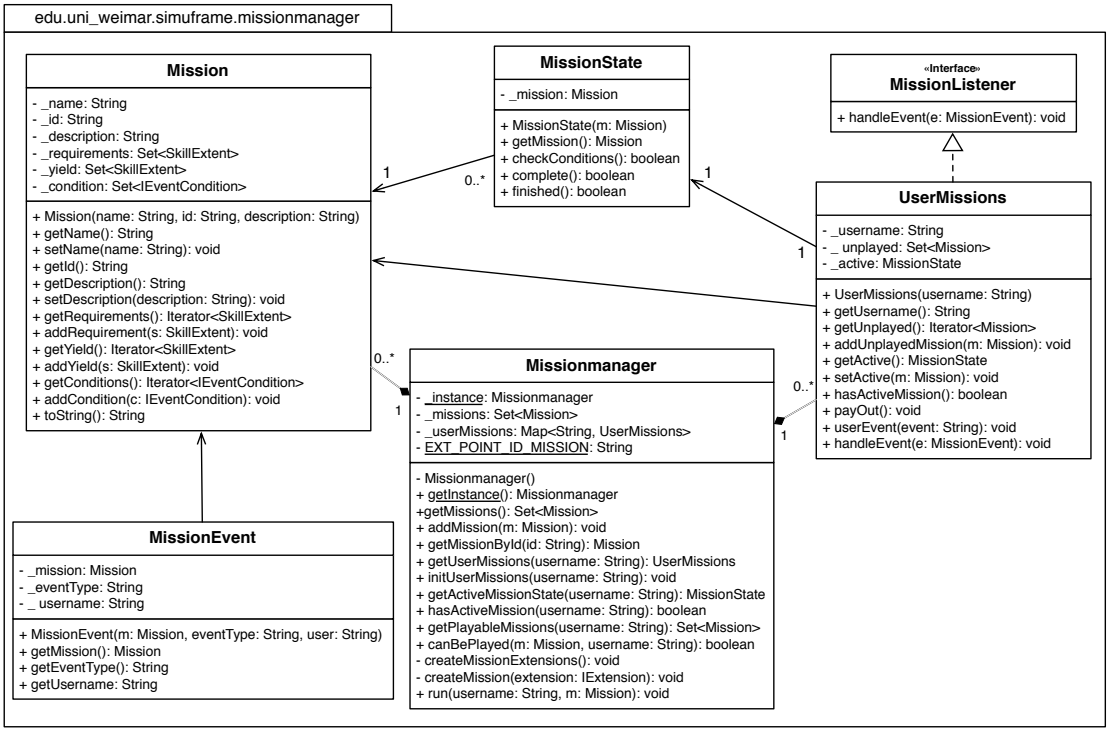

Abbildung 4.3: UML-Klassendiagramm des Missionsmanagers

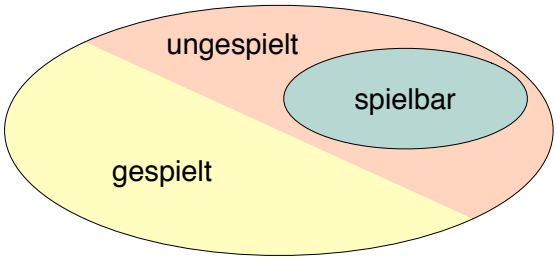

Abbildung 4.4: Diagramm zur Veranschaulichung der Teilmengen der im System gespeicherten Missionen

die Klasse `UserMissions` erstellt. (Vergleich: `UserSkills`) Diese speichert zu einem Spieler die ungespielten Missionen. Durch die Abbildung 4.4 wird deutlich, warum es sinnvoll ist, alle ungespielten Missionen zu speichern.

Die Gesamtmenge der im System vorhandenen Missionen kann in die Menge der gespielten und der noch nicht gespielten Missionen eines Spielers eingeteilt werden. Werden zu jedem Spieler die ungespielten Missionen gespeichert, dann muss zur Ermittlung der spiel-

baren Missionen keine Komplementbildung vorgenommen werden. Zudem wird nach jeder erfolgreich gespielten Mission die Menge der ungespielten Missionen kleiner und dadurch auch der Aufwand zur Ermittlung der spielbaren Missionen geringer.
Die Auswahl der spielbaren Missionen erfolgt auf Grundlage der vorhandenen Skillpunkte des Spielers. Dadurch wird sichergestellt, dass dem Spieler nur Missionen angeboten werden, für die er die benötigten Voraussetzungen erfüllt. Dazu werden die in den Missionen festgelegten `SkillExtents` mit den vorhandenen `SkillExtents` des Spielers verglichen. Um eine Mission spielen zu können, muss der Spieler mindestens die vorausgesetzte Skillpunkte für den entsprechenden Skill besitzen. Aus allen ungespielten Missionen wird eine Liste mit spielbaren Missionen erstellt, aus welcher der Spieler eine Mission wählen kann. Diese Liste wird nach jedem Missionsabschluss neu generiert. Startet der Spieler eine Mission, wird von dieser Mission ein `MissionState`-Objekt angelegt, welches den konkreten Zustand der Mission speichert. Nach dem Prinzip der Datensparsamkeit, existiert dieses Objekt nur für die Spieldauer der Mission.
Nach dem Beenden einer Mission wird überprüft, ob die definierten *Conditions* vom Spieler erfüllt wurden. Die *Conditions* implementieren das *Strategie*-Entwurfsmuster [GHJV95]. Dieses Pattern beschreibt einen Mechanismus, in dem das Verhalten eines Objekts zur Laufzeit durch den Einsatz unterschiedlicher Algorithmen beeinflusst werden kann. Die verschiedenen Algorithmen werden dabei in unterschiedliche Klassen gekapselt und implementieren ein gemeinsames Interface. Zur Demonstration des Sachverhalts wurde eine einfache *Condition* implementiert, die ein vom Framework bereitgestelltes Interface implementiert. Im Kapitel4.3 wird diese Bedingung genauer beschrieben. Nach dem Beenden einer Mission wird überprüft, ob die geforderten *Conditions* erfüllt wurden. Dazu wurde nach dem *Listener*-Pattern eine Benachrichtigungstruktur implementiert. Das *Listener*-Pattern wird angewendet, wenn Änderungen eines Objekts verschiedene abhängige Objekte interessiert. Dazu wird ein Interface implementiert, welches eine Schnittstelle zum an- und abmelden von Beobachtern definiert. Ändert sich der Status des Objekts, werden die Beobachter informiert [GHJV95].
Wurden alle *Conditions* erfüllt, wird ein Event abgesetzt. Das Event wird an den `UserMission` gesendet, der in diesem Fall das *MissionListener*-Interface implementiert. Die weitere Verarbeitung obliegt dem *UserMission*. Dieser ruft die `payOut()`-Funktion auf, durch die die Auszahlung der Skillpunkte stattfindet. Dabei werden von der aktuellen Mission die auszuzahlenden `SkillExtents` abgefragt. Der *Skillmanager* ruft anschließend die `increase()`-Funktion mit dem Spieler und den `SkillExtents` auf. Dadurch werden die `SkillExtents` des Spieler um die auszuzahlenden `SkillExtents` der Mission erhöht. Dieses Verhalten kann als Addition von `SkillExtent`-Objekten verstanden werden.

4.3 Integration in das bestehende Framework

In dem Projekt *Intelligentes Lernen* wird ein MOG zur Ausbildungsunterstützung in der Bauphysik entwickelt. Das in dieser Arbeit konzipierte *Spieler-Modell* wurde zur Integration in diese bestehende Software entworfen. Den zentralen Rechenkern des Simulations-

spiels stellt die *SimuFrame*-Software dar. Dieses modular erweiterbare Framework basiert auf der *Eclipse Rich Client Platform* (RCP) und ist in Java implementiert.
Seit der Eclipse Version 3.0 wurde die Eclipse RCP eingeführt, die es ermöglicht, generische Applikationen zu implementieren, die nicht auf der Eclipse *Integrated Development Environment* (IDE) aufsetzen. Ebenfalls wurde die Eclipse Runtime auf das *OSGI*-Konzept (Open Service Gateway Initiative) umgestellt. Diese Initiative beschreibt eine Standardisierung von Diensten in einem lokalen Netzwerk mit eingebetteten Geräten. Seit dieser Umstellung ist Eclipse selber nur noch der Ablaufkern, der die Rolle des *OSGi*-Servers erfüllt. Dieser lädt die einzelnen Module, die nun ebenfalls dem *OSGi*-Standard entsprechen [Dau05]. In der Entwicklung mit Eclipse wird der Begriff *Plugin* oftmals synonym für den Begriff *Modul* eingesetzt. Die *Plugins* beschreiben die *OSGi*-Komponenten (Bundles) und stellen die eigentlichen Funktionalitäten bereit. Sie werden über Erweiterungspunkte (*Extension Points*) an den Rest der Plattform angeschlossen. Diese *Extension Points* beschreiben eine Registrierungsstelle, für die sich *Plugins* anmelden. Die *Plugins* können auch eigene *Extension Points* definieren und somit untereinander gekoppelt werden. Der *Extension-Point*-Mechanismus ermöglicht dadurch das Einklinken eines Moduls in die Funktionalitäten eines anderen Moduls [Dau05]. In der Datei *plugin.xml* wird festgelegt, auf welche *Extension Points* das *Plugin* zugreift und welche Erweiterungspunkte von diesem *Plugin* bereitgestellt werden. Wird im Programmlauf eine Stelle erreicht, die durch einen *Extension Point* erweitert werden soll, wird im System nachgefragt, ob sich ein oder mehrere Module angemeldet haben. Wenn dies der Fall ist, werden die Funktionalitäten dieser *Plugins* nacheinander ausgeführt.
Die *SimuFrame*-Software nutzt den beschriebenen *Extension-Point*-Mechanismus zur Kopplung der einzelnen Module.

Damit das entwickelte *Spieler-Modell* flexibel in eine Lernspielumgebung integriert werden kann, muss eine Schnittstelle geschaffen werden, die eine Anbindung an eine bestehende Software ermöglicht. Im Folgenden wird beschrieben, wie diese Schnittstelle definiert ist und wie das Modell in das bestehende Framework *SimuFrame* integriert wurde.
Die Schnittstelle wird über *Extension Points* realisiert. Dazu stellen die Module *Skillmanager* und *Missionsmanager* verschiedene Erweiterungspunkte zur Datenintegration bereit. In der Abbildung 4.5 sind die angelegten *Extension Points* dargestellt. Das Modul *Skillmanager* stellt die *Extension Points Skill* und *Badge* bereit. Der *Missionsmanager* definiert den Erweiterungspunkt *Mission*. Zum Anlegen eines Erweiterungspunktes muss zunächst im entsprechenden Modul in der Datei *plugin.xml* ein *Extension Point* definiert werden. Dies erfolgt über eine Id, einen Namen und der Angabe einer Schema-Datei, in der die Struktur für die zugehörige XML-Datei festgelegt wird. In der XML-Datei werden die Daten zur Integration nach den in der Schema-Datei festgelegten Vorgaben angelegt. Im AnhangA sind die Definitionen der *Extension Points* der beiden Module abgebildet.
Die Datei *skill.exsd* (siehe Anhang A) beschreibt das Schema zum Anlegen eines Skills. Ein Skill ist definiert über einen Namen, eine Id, eine Beschreibung und den Voraussetzungen. Die Voraussetzungen bestehen aus den Elementen Id, zur Identifikation eines Skills, und einem Level. Wie bereits erläutert, wird aus diesen Vorgaben ein `SkillExtent` erstellt, der die Ausprägung zu einem Skill beschreibt. Über den zweiten *Extension Point* des *Skillma-*

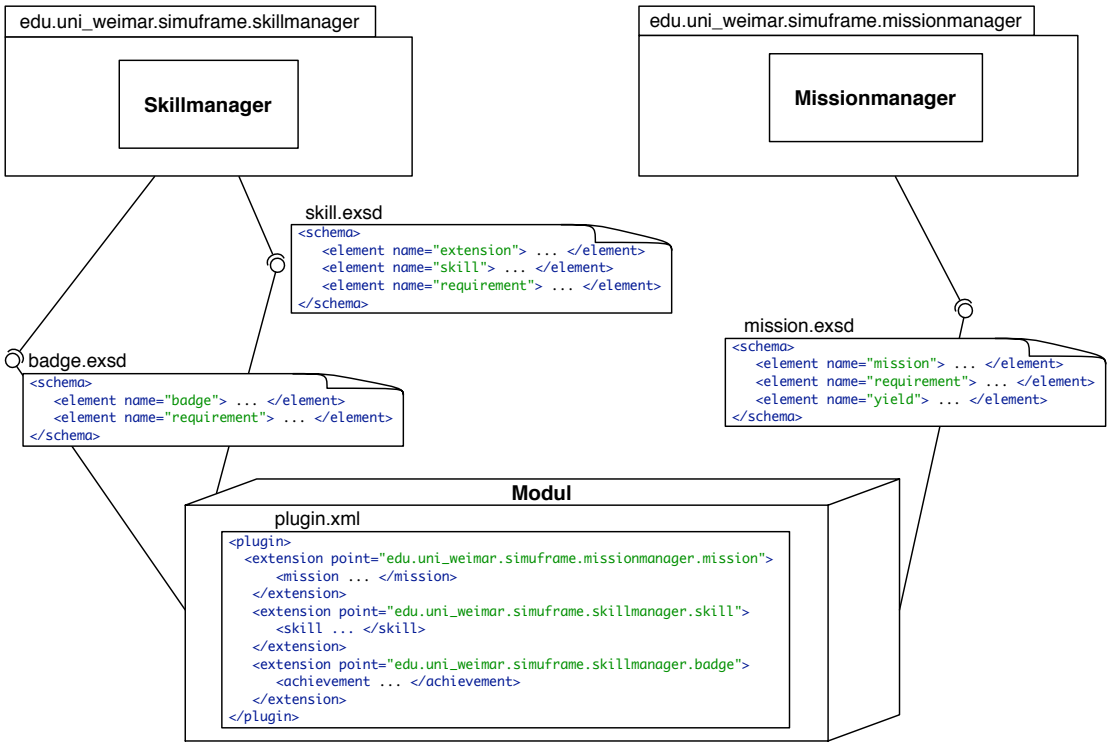

Abbildung 4.5: Datenbereitstellung über Extension Points

nagers können *Badges* angelegt werden. Diese bestehen ebenfalls aus einem Namen, einer Id und den Voraussetzungen zum Auszahlen des *Badges*. Die Voraussetzungen sind skillbasiert und daher wie die Voraussetzungen der Skills definiert. Die Schema-Datei *badge.exsd* ist ebenfalls im Anhang A dargestellt. Der *Missionsmanager* stellt den *Extension Point Mission* bereit. Eine Mission besteht aus einem Namen, einer Id und einer Beschreibung. Zusätzlich können skillbasierte Voraussetzungen, bestehend aus einer Id und einem Wert, und auf gleiche Weise auch skillbasierte Auszahlungen festgelegt werden. (siehe Anhang A, mission.exsd)
Für diese drei *Extension Points* können sich andere Module registrieren, um Spieldaten zu definieren. Das in der Abbildung 4.5 dargestellte allgemeine Modul steht stellvertretend für alle Module, die sich für die *Extension Points* anmelden. In der Datei *plugin.xml* eines Moduls können nach den Spezifikationen der Schema-Datei nun Skills, Badges und Missionen angelegt werden. Das Anlegen der Daten erfolgt durch den Administrator, der in den einzelnen Modulen die Inhalte definiert. In der *SimuFrame*-Software existieren verschiedene Module, die Teilbereiche der Bauphysiklehre darstellen. Für jedes Modul können nun zugehörige erlernbare Skills und Missionen festgelegt werden. Zur Demonstration des Modells wurden zu den Modulen *Acoustics* und *Heat* des *SimuFrames* beispielhafte Skills und Missionen definiert. Im Anhang A sind die *plugin.xml* Dateien abgebildet, sowie Ta-

bellen (siehe Anhang B), die das integrierte Datenmodell veranschaulichen. Diese Daten können jederzeit durch den Administrator verwaltet, erweitert oder verändert werden, ohne in den Programmcode eingreifen zu müssen. Dies ist eine wichtige Eigenschaft für ein System zum Einsatz im eLearning-Bereich.

Das Einlesen der Daten erfolgt zur Laufzeit des Programms. Der folgende Programmcode verdeutlicht am Beispiel der Skills, wie dieser Prozess abläuft.

```
private static final String EXT_POINT_ID_SKILL =
    "edu.uni_weimar.simuframe.skillmanager.skill";
private static final String EXT_POINT_ID_BADGE =
    "edu.uni_weimar.simuframe.skillmanager.badge";

private void createSkillExtensions() {
    final IExtensionRegistry registry = Platform.getExtensionRegistry();
    final String extensionPoint = EXT_POINT_ID_SKILL;
    final IExtensionPoint point = registry.getExtensionPoint(extensionPoint);
    Assert.isLegal(point != null,"extension" + extensionPoint +"not found");
    for (final IExtension extension : point.getExtensions()) {
        createSkill(extension);
    }
    checkSkillConsistency();
}
```

Zunächst wird ein Verzeichnis (`IExtensionRegistry`) aller in der Plattform definierten *Extension Points* und deren zugehörigen Erweiterungen erstellt. Dies erfolgt durch die Methode `getExtensionRegistry()`. Anschließend werden von dem *Extension Point* mit der entsprechenden Id alle Extensions abgefragt, die sich für diesen registriert haben. In der Funktion `createSkill(extension)` werden die Skillobjekte der jeweiligen Extension angelegt. Dazu werden von den konfigurierten Elementen einer Extension die in der XML-Datei angegebenen Attribute abgefragt und die Skills mit diesen Werten initialisiert. Die zugehörige Kategorie eines Skills wird über den Namen des Moduls festgelegt. Nach dem Anlegen aller Skills erfolgt in der Funktion `checkSkillConsistency()` eine Überprüfung auf die Konsistenz der Daten.

```
private void createSkill(final IExtension extension){
    for (final IConfigurationElement configurationElement : extension
        .getConfigurationElements()) {

        final String name = configurationElement.getAttribute("name");
        final String id = configurationElement.getAttribute("id");
        final String description = configurationElement
            .getAttribute("description");
        final String category = configurationElement
```

```
            .getDeclaringExtension().getNamespaceIdentifier();

        Skill s = getSkill(id);
        s.initialize(name, description, category);
        for (final IConfigurationElement configurationChild :
            configurationElement.getChildren()){
            Skill requiredSkill = getSkill(configurationChild
                .getAttribute("id"));
            if(requiredSkill == null)
                requiredSkill = new Skill(null, configurationChild
                    .getAttribute("id"), null, null);
            addSkill(requiredSkill);
            int level = 0;
            try {
                level = Integer.parseInt(configurationChild
                    .getAttribute("level"));
            }
            catch (Exception E){
                System.out.println(requiredSkill.getName() +
                    "has invalid requirement level");
            }
            SkillExtent sl = new SkillExtent(requiredSkill, level);
            s.addRequirement(sl);
        }
        addSkill(s);
    }
}
```

Die in den Missionen festgelegten *Conditions* stellen einen Anknüpfungspunkt zur Integration des Modells in das Framework dar. Diese Bedingungen beschreiben Zustände des Systems, die der Spieler erfüllen muss, damit die Mission als erfolgreich bewertet werden kann. Die Systemzustände werden vom Framework verwaltet und bereitgestellt. Im *SimuFrame* sind *Conditions* festgelegt, die einen Systemzustand beschreiben. Dafür wird ein Interface `IEventCondition` bereitgestellt, welches von beliebigen Klassen implementiert werden kann und wodurch unterschiedliche Bedingungen festgelegt werden können. Die festgelegten Bedingungen einer Mission werden durch Objekte vom Typ einer Klasse, die das Interface implementieren, bestimmt. Dies ermöglicht das Festlegen variabler Bedingungen, die beim Spielen einer Mission erfüllt werden müssen und die nach dem Beenden einer Mission abgefragt werden können.

4.4 Benutzeroberflächen

Über die Eclipse Workbench kann zu einer RCP-Anwendung eine grafische Benutzeroberfläche erstellt werden. Die Workbench wird von dem Plugin *org.eclipse.ui* bereitgestellt. Der zunächst leere Applikationsrahmen kann über *Extension Points* durch andere *Plugins* mit Inhalten befüllt werden.
Das *SimuFrame* verfügt bereits über eine grafische Benutzeroberfläche, die mit der SWT-Bibliothek (Standard Widget Toolkit) auf Grundlage der Eclipse Workbench erstellt wurde [Varb]. Die Benutzeroberfläche stellt verschiedene Ansichten für den Spieler und für den Entwickler bereit. Im aktuellen Entwicklungsstand werden die Benutzeroberflächen des Spieler noch nicht auf der Clientseite dargestellt, da derzeit an einem Übertragungsprotokoll zwischen Client und Server gearbeitet wird. Die grafische Spieloberfläche soll später mit Hilfe der Software *Unity3d* [Vara] auf der Clientseite realisiert werden.
Zur Visualisierung des Modellablaufs wurden vier verschiedene Views erstellt. Zwei dieser Views sind für den Spieler konzipiert und zwei weitere für den Administrator. Eine View bezeichnet einen *Workbench-Part* (ViewPart), der das *Workbench-Fenster* über Reiterkarteien strukturiert [Dau05]. Über den Extension Point *org.eclipse.ui.views* kann ein Plugin bzw. Modul eine View für die *Workbench* registrieren. Dabei muss eine Id, ein Name und die implementierende Klasse für die View angegeben werden.
Im nächsten Abschnitt werden die einzelnen Views für den Spieler und anschließend die Views für den Administrator vorgestellt.

4.4.1 Spieler

Der *Missionsmanager* stellt eine Benutzeroberfläche bereit, die dem Spieler eine Interaktion mit dem System ermöglicht. Die Abbildung 4.6 visualisiert diese Ansicht, in der zunächst eine Liste mit allen spielbaren Missionen dargestellt wird. Aus dieser Liste kann der Spieler eine Mission wählen.

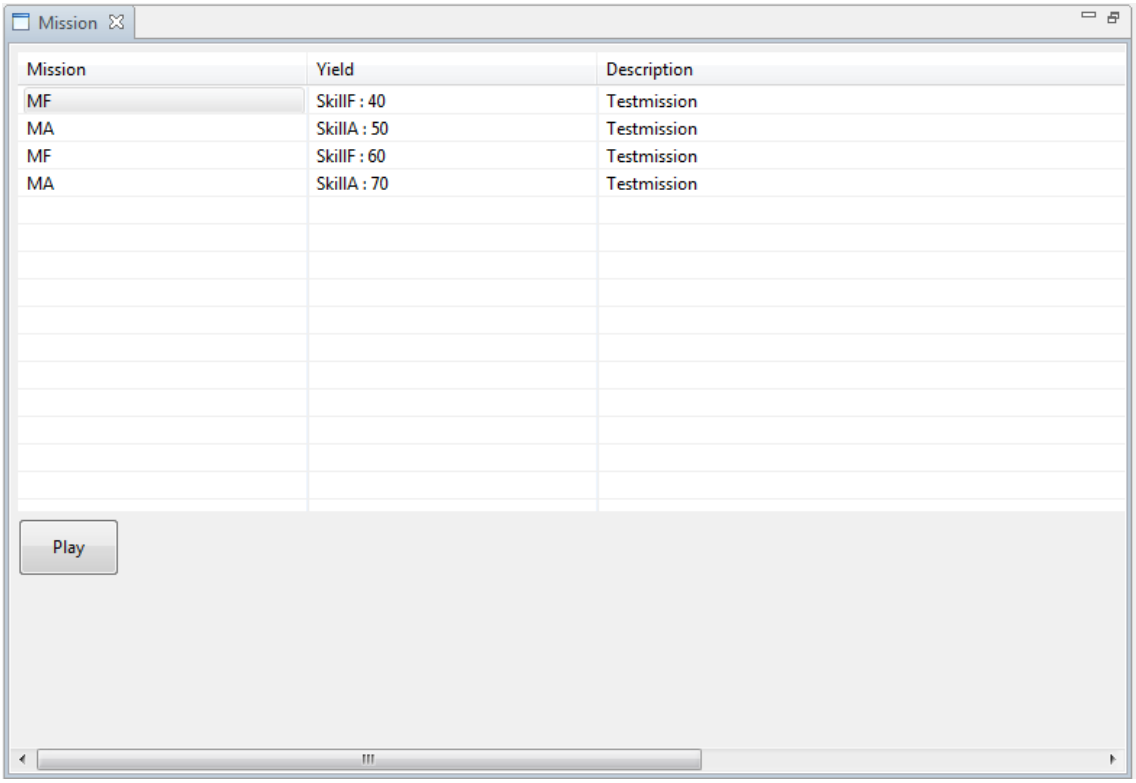

Abbildung 4.6: View zur Auswahl einer spielbaren Mission

Seine Auswahl kann er auf Grundlage der erreichbaren Skillpunkte treffen, da diese, neben der Beschreibung der Mission, in der Liste ablesbar sind. Über den *Play-Button* wird die ausgewählte Mission gestartet. Wie bereits erwähnt, wurden im Rahmen der Bachelorarbeit die eigentlichen Missionen nur prototypisch umgesetzt. Nach dem Starten der Mission müsste ein virtuelles Spielszenario geladen werden, in dem der Spieler interagieren kann. Dabei muss die Aufgabenstellung der Mission durch virtuelle Problemsituationen ersichtlich werden. Durch verschiedene Tätigkeiten und Handlungen des Spielers werden verschiedene Systemzustände erreicht. Erfüllen diese Zustände die in der Mission festgelegten Bedingungen, dann gilt die Missionsaufgabe als erfolgreich gelöst. Zur Demonstration des Modells wurde an dieser Stelle eine prototypische Mission angelegt. Die Abbildung 4.7 veranschaulicht den Missionsinhalt.

Abbildung 4.7: Missionsprototyp

Dargestellt wird der Name der Mission, ein Feld in dem ein Haken gesetzt werden kann und ein Button zum Beenden der Mission. Um den *Condition*-Mechanismus zu demonstrieren, wurde für den Prototypen eine *Condition* festgelegt, die von außen auf wahr oder falsch gesetzt werden kann. Setzt der Spieler den Haken, wird die *Condition* als erfüllt gesetzt. Wird die Mission über den *Finish-Button* beendet, überprüft das System zunächst, ob die *Conditions* erfüllt sind. Wenn dies der Fall ist, wird anschließend ein Event abgesetzt, welches den `UserMission` darüber informiert, dass sich der Status der Mission auf *erfolgreich gespielt* geändert hat. Daraufhin findet die Auszahlung der Skillpunkte statt. Diese *Conditions* können durch die Verwendung des `IEventCondition`-Interface durch jede beliebige andere *Condition* ersetzt werden.
Nach dem Beenden einer Mission wird für den Spieler basierend auf den vorhandenen Skills eine neue Liste mit spielbaren Missionen erstellt. Wurde die Mission nicht erfolgreich beendet, bleibt sie im System als ungespielt gespeichert und kann erneut gespielt werden.

Eine weitere wichtige Ansicht für den Spieler ist die Darstellung der Skillabhängigkeiten in einem Skillbaum. In diesem Skillbaum sollten die einzelnen Skills, den entsprechenden Kategorien zugeordnet, dargestellt werden. Die Abhängigkeiten zwischen den Skills sollten über Verbindungen, die auch die benötigten Skillpunkte veranschaulichen, ersichtlich werden. Ebenfalls sollte der Spieler seinen eigenen Spielstand ablesen können, damit er einen Überblick über bereits erlernte Fähigkeiten bekommt und erkennt, welche Skills noch trainiert werden müssen.
Da Eclipse zur Visualisierung von Graphen keine eigenen Plugins zur Verfügung stellt, ist man auf externe Bibliotheken angewiesen oder man muss eine eigene Visualisierung implementieren. Leicht in die Eclipse RCP-Anwendung zu integrieren, ist das *Zest-Visualisierungstoolkit für Graphen* [Ecl]. Es basiert auf *SWT* und ermöglicht die Implemen-

tierung von Views für die Eclipse Workbench. Für die Visualisierung des Skillbaums ist jedoch kein geeigneter Layoutalgorithmus vorhanden. Um die Anforderungen an die Visualisierung des Skillbaums umzusetzen, müsste ein eigener Layoutalgorithmus implementiert werden, da die vorhandenen Algorithmen für die Darstellung nicht in Frage kommen. Nachteilig ist ebenfalls, dass für dieses Toolkit nur ein geringer Dokumentationsumfang vorhanden ist.
Ein weiteres weit verbreitetes Visualisierungstoolkit zur interaktiven Darstellung von Daten ist *Prefuse* [Hee]. Dieses sehr umfangreiche Framework besitzt viele Funktionalitäten und eine sehr umfangreiche Dokumentation. Jedoch basiert *Prefuse* auf *AWT* (Abstract Window Toolkit). Zur Nutzung von *AWT Widgets* gibt es eine *SWT-AWT-Komponente*, die eine Integration in die Eclipse Workbench ermöglichen würde. Ein geeigneter Layoutalgorithmus zur Darstellung des Skillbaums müsste ebenfalls selber implementiert werden, was durch den enormen Funktionsumfang zu einer hohen Einarbeitungszeit führt.
Eine weitere Möglichkeit besteht in einer eigenen Implementierung der Visualisierung, durch die eine individuelle Darstellung erreicht werden kann. Diese hat ebenfalls einen hohen Aufwand zur Folge, da nicht auf bereitgestellte Funktionalitäten eines Visualisierungstoolkit zurückgegriffen werden kann.
In der Tabelle 4.1 sind die Vor- und Nachteile der verschiedenen Visualisierungslösungen zusammenfassend gegenübergestellt.

	Zest	Prefuse	eigene Implementierung
Aufwand	+	–	–
Funktionsumfang	0	+	gering
Qualität [1]	–	–	+
Dokumentation	–	+	nicht vorhanden

+ gut
0 neutral
– schlecht

Tabelle 4.1: Gegenüberstellung verschiedener Visualisierungslösungen

Da unter Verwendung eines der vorgestellten Visualisierungstoolkits auf jeden Fall der Layoutalgorithmus hätte selber implementiert werden müssen, wurde für die Darstellung des Skillbaums eine eigene Visualisierung implementiert. Diese basiert auf *SWT-Graphics* und wurde ohne die Verwendung eines der untersuchten Toolkits erstellt. Die Abbildung 4.8

[1] Qualität der in Frage kommenden Layoutalgorithmen für die hier betrachtete Anwendung.

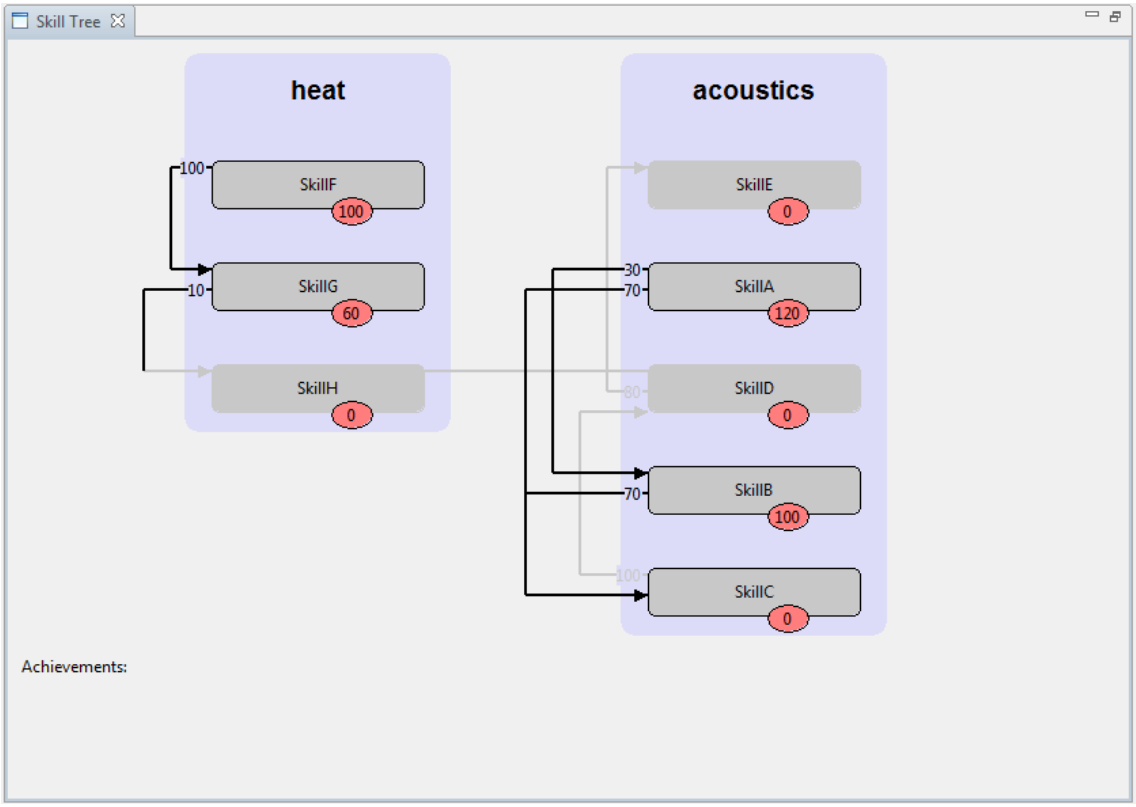

Abbildung 4.8: Darstellung des Skillbaums in der Spieler-Ansicht

veranschaulicht die entstandene Visualisierung des Skillbaums in der Spieler-Ansicht. Diese View wird vom Skillmanager bereitgestellt.
Der Spieler erhält eine komplette Übersicht über alle Skills, die im Spielverlauf erlernt werden können. Diese sind untereinander in den entsprechenden Kategorien angeordnet. Die Verbindungen verdeutlichen die Abhängigkeiten zwischen den Skills. Diese sind auch kategorieübergreifend möglich. Die benötigten Skillpunkte, die der Spieler besitzen muss, um den nächsten Skill zu erlernen, werden ebenfalls auf den Verbindungen angezeigt. Die roten Ellipsen an den Skills zeigen dem Spieler die aktuelle Punkthöhe an. Skills, für die bereits Punkte erreicht wurden und Verbindungen zu anderen Skills, für die die Voraussetzungen erfüllt sind, werden dem Spieler hervorgehoben angezeigt. Der restliche Skillbaum wird ausgegraut dargestellt. Die erreichten Badges werden in einer Liste unten links angezeigt. Nach jeder erfolgreich gespielten Mission wird der Skillbaum aktualisiert, um dem Spieler immer den aktuellen Spielstand anzuzeigen. In der Abbildung 4.8 ist ein möglicher Spielzustand dargestellt, bei dem der Spieler noch keine Badges erreicht hat, aber bereits Punkte für die Skills A, B, F und G gesammelt hat.

4.4.2 Administrator

Der Administrator übernimmt die Verwaltung der Spiel-und Lerninhalte. Diese können wie im Kapitel 4.3 beschrieben, in das Modell integriert werden. Um, gegenüber den XML-Dateien, einen besseren Überblick über die im System vorhandenen Daten zu bekommen, werden zwei Views zur grafischen Darstellung benötigt.
Für den Administrator wird vom *Skillmanager* ebenfalls eine Visualisierung des Skillbaums angeboten. Diese ist vergleichbar mit der Darstellung für den Spieler. Jedoch werden die spielerbezogenen Ausprägungen der Skills nicht dargestellt, (siehe Abbildung 4.9).

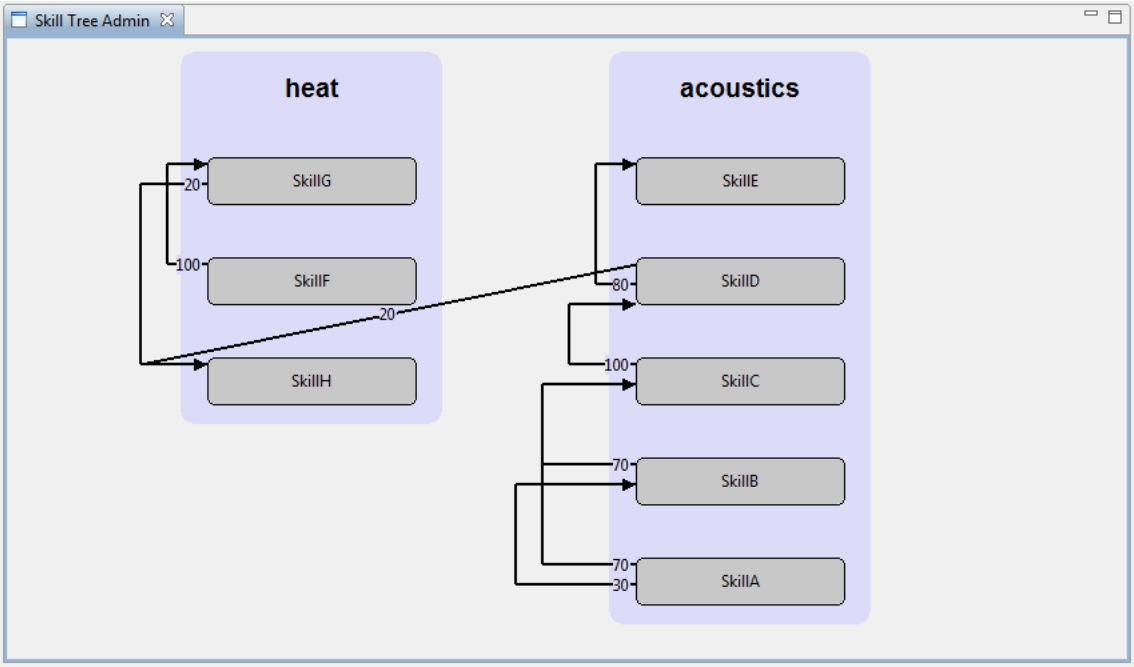

Abbildung 4.9: Darstellung des Skillbaums in der Administrations-Ansicht

Diese Ansicht ist besonders wichtig, da die Abhängigkeiten zwischen den Skills in der XML-Datei bei großen Datenmengen schnell unübersichtlich werden können. Durch die grafische Visualisierung bekommt der Administrator eine übersichtliche Zusammenfassung und er kann leicht überprüfen, ob das Datenmodell die gewünschten Eigenschaften erfüllt. Kommen neue Skills oder weitere Kategorien hinzu, wird der Skillbaum dynamisch angepasst.

Um alle Skills erlernen zu können, müssen ausreichend viele Missionen im System vorhanden sein. In der View zur Darstellung der integrierten Missionen (Abbildung 4.10) kann der Administrator überprüfen, ob durch die vorhandenen Missionen alle Skills erreicht werden können. Ebenfalls sollten die Voraussetzungen der Missionen und die in einer Mission zu erreichenden Skillpunkte nach den Skillabhängigkeiten festgelegt werden.

Es besteht jedoch die Möglichkeit, Zusatzmissionen zu definieren, in denen Skills erlernt werden können, für die der Spieler noch nicht die Voraussetzungen erfüllt. Die Bedeutung solcher Missionen wurde im Kaptel 3.2.2 beschrieben.
Die in Abbildung 4.10 dargestellte View ist in zwei Bereiche eingeteilt. In der oberen Tabelle werden alle im System vorhandenen Missionen zusammengefasst. Dabei ist der Name und die Beschreibung der Mission aufgelistet. Durch die Auswahl einer Mission, kann der Administrator im unteren Teil der View die zusätzlichen Informationen über die Mission abfragen. Neben der Id sind die vorausgesetzten Skills und die Skills, die durch erfolgreiches Ausführen der Mission erlangt werden können, verzeichnet.

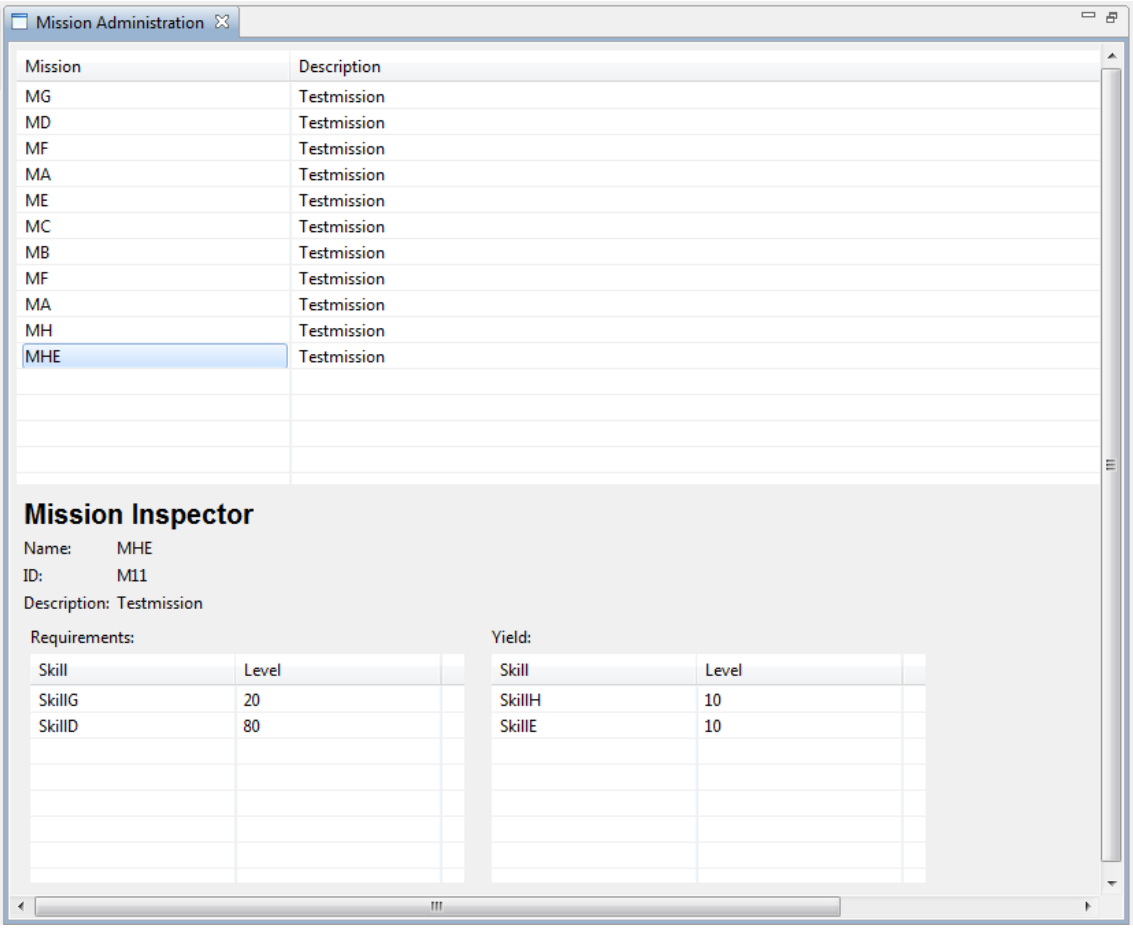

Abbildung 4.10: Darstellung der integrierten Missionsdaten

5 Zusammenfassung und Ausblick

In dieser Bachelorarbeit wurde ein *Spieler-Modell* für eine Spielplattform zum Einsatz im eLearning-Bereich entwickelt. Im Kontext einer Spielumgebung wird die Aufmerksamkeitsbereitschaft des Spielers durch Motivationsmechanismen gesteuert. Der Lernerfolg ist von dieser Aufnahmebereitschaft abhängig. Daher ist die Motivation Grundlage von Lernspielen. Der Rücktransfer der Lerninhalte in die Realität erfolgt über die *projizierte* Identität.

Um eine positive Lernerfahrung zu erreichen, braucht der Spieler eine freie Kontrolle über den Spielverlauf und die Möglichkeit, sich selbst Ziele abzustecken. So entsteht ein spielerischer Umgang mit dem Lernsystem der Vergnügen und Spaß bereitet. Die aufeinander aufbauenden erlernbaren Fähigkeiten (Skillbaum, Abschnitt 3.2.1) geben dem Spieler eine Orientierung, welche Wege ihn zu einem gesetzten Ziel führen können. Diese Strukturierung bewirkt eine aufbauende Reihenfolge einzelner Lernziele, die eine Steigerung des Schwierigkeitsgrades vorgibt. Es verbleiben aber genug freie Entscheidungsräume für einen individuellen Spielablauf.

Bei den Motivationsformen unterscheidet man in *intrinsische* und *extrinsische* Motivation. Beide tragen zur Steuerung der Lernbereitschaft bei. Sie sollten daher intensiv durch das Spielmodell unterstützt werden.

Intrinsische Motivation ist die innere Antriebskraft die der Spieler entwickelt und ihn vollkommen im Spiel versinken lässt. Sie wird durch den Flow-Effekt (Abschnitt 2.3) erreicht. Dazu müssen Fähigkeiten und Herausforderungen im Gleichgewicht stehen. Der Spielablauf erfolgt üblicherweise in Zyklen (Abschnitt 2.3) die dem Spieler ermöglichen sein Spielverhalten anhand des System-Feedbacks zu bewerten. Innerhalb dieses Vorgangs darf weder eine Unter- noch eine Überforderung stattfinden, damit sich der Flow-Effekt entwickeln kann und keine Frustration eintritt.
Dies wird über ein Kontrollsystem erreicht, in welchem dem Spieler anhand seiner vorhandenen Fähigkeiten angemessene Herausforderungen angeboten werden. Der Spieler entscheidet sich anhand seiner Zielsetzung für eine der Aufgabenstellungen. Seine Aktionen bewirken Veränderungen in der Spielwelt, über die die Qualität der Handlungen beurteilt wird.

Extrinsische Motivation entsteht durch das explizit in Aussicht stellen von Belohnung und Erfolg (Abschnitt 2.4). Primär wird der Spieler mit Erfahrungspunkten ausgezeichnet, die ihm einen Überblick über seine bisher erreichten Kompetenzen verschaffen. Besonderes Interesse für künftig erlernbare Wissensgebiete wird geweckt, durch die Möglichkeit, einzelne Missionen, die speziell zur Vorschau gedacht sind, bereits im Vorfeld zu absolvieren. An Schlüsselstellen können Badges vergeben werden die abgeschlossene Lerneinheiten

belohnen und Erfolge bescheinigen. Sein Bestreben liegt dabei in dem Erreichen aller verfügbaren Fähigkeiten und Auszeichnungen.

Das entwickelte System ist lose an das bestehende Framework gekoppelt, was eine leichte Anpassbarkeit und damit Übertragbarkeit an andere Systeme gewährleistet. Durch das standardisierte Laden der Daten ist es flexibel mit beliebigen Spielinhalten befüllbar und erweiterbar. Die Funktionalität ist je in einer Schnittstelle der beiden Manager zusammengefasst und kann über diese genutzt werden.

Ausblick

Das entstandene Modell ist in der Lage, den Kenntnisstand des Lernenden zu erfassen und durch angemessene Aufgaben zu erweitern. Für eine praktische Einsetzbarkeit sind jedoch noch weitere Entwicklungsschritte nötig. Neben der clientseitigen Benutzeroberfläche ist auch die Fertigstellung des Kommunikationsprotokolls Voraussetzung, um schließlich spielbare Missionen umzusetzen.

Für die Verwaltung der Spielinhalte ist es hilfreich, zur Laufzeit veränderliche Missionen und Skills zu ermöglichen. Dies könnte durch einen Missions- bzw. Skill-Editor erfolgen. Dabei sollte der Missions-Editor das Skill-Modell beachten, so dass die Voraussetzungen und Auszahlungen der Missionen nach Vorgabe des Skillbaums aufeinander abgestimmt sind. Trotzdem sollte das Anlegen von Zusatzmissionen ermöglicht werden. Da die *plugin.xml*-Datei zur Laufzeit nicht veränderbar ist, müsste eine (De-) Serialisierung[1] der aktuellen Spieldaten implementiert werden.

[1] Speichern und Laden

Literaturverzeichnis

[Bat] Bateman, C. *Only A Game - Designing Rewards in Games.* `http://onlyagame.typepad.com/only_a_game/2005/08/designing_rewar.html`. zuletzt abgerufen am 23.03.2011.

[Bau] Bauhaus Universität Weimar - Fakultät Medien. *Intelligentes Lernen - Innovative Informationstechnologien für das moderne Wissensmanagement.* `http://www.uni-weimar.de/cms/medien/projekte/innoprofile.html`. zuletzt abgerufen am 17.03.2011.

[Bis] Bishop, Philip and Warren, Nigel. *Java Tip 67: Lazy instantiation.* `http://www.javaworld.com/javaworld/javatips/jw-javatip67.html`. zuletzt abgerufen am 02.04.2011.

[BS09] J.D. Bayliss and D.I. Schwartz. Instructional design as game design. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 10–17. ACM, 2009.

[CCP] CCP hf. *Skill training.* `http://wiki.eveonline.com/en/wiki/Skill_training`. zuletzt abgerufen am 09.04.2011.

[Che] Jenova MFA Chen. Flow in Games.

[Csi03] Csikszentmihalyi, Mihaly. *Flow : das Geheimnis des Glücks.* Klett-Cotta, Stuttgart, 2003.

[Dau05] B. Daum. *Java-Entwicklung mit Eclipse 3.1.* dpunkt-Verl., 2005.

[Ecl] Eclipse Foundation, Inc. *Zest.* `http://www.eclipse.org/gef/zest/`. zuletzt abgerufen am 08.04.2011.

[GAD02] R. Garris, R. Ahlers, and J.E. Driskell. Games, motivation, and learning: A research and practice model. *Simulation & gaming*, 33(4):441, 2002.

[Gee05] J.P. Gee. Learning by design: Good video games as learning machines. *E-Learning and Digital Media*, pages 5–16, 2005.

[Gee07] Gee, James P. *What Video Games Have to Teach Us about Learning and Literacy.* Palgrave Macmillan, New York, 2007.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206. Addison-wesley Reading, MA, 1995.

[Hee] Heer, Jeffrey and others. *the prefuse visualization toolkit.* `http://prefuse.org/`. zuletzt abgerufen am 08.04.2011.

[Joh09] John Kirk III, Whitson. *Design Patterns of Successful Role-Playing Games*, pages 66–71. 2009.

[Kon10] G.D. Konetes. The Function of Intrinsic and Extrinsic Motivation in Educational Virtual Games and Simulations (Rising Scholar Paper). *Journal of Emerging Technologies in Web Intelligence*, 2(1):23–26, 2010.

[Lav08] Lave, Jean and Wenger, Etienne. *Situated Learning : Legitimate Peripheral Participation*. Cambridge Univ. Press, Cambridge, 2008.

[McC] McClanahan, Greg. *Achievement Design 101*. `http://www.gamasutra.com/blogs/GregMcClanahan/20091202/3709/Achievement_Design_101.php`. zuletzt abgerufen am 23.03.2011.

[Mer01] C.A. Mertler. Designing scoring rubrics for your classroom. *Practical Assessment, Research & Evaluation*, 7(25):1–10, 2001.

[MH06] R. Miles and K. Hamilton. *Learning UML 2.0*. O'Reilly Media, Inc., 2006.

[MM] P. Mohammed and P. Mohan. Sugar Coated Learning: Incorporating Intelligence into Principled Learning Games.

[MS03] C. Meier and S. Seufert. Game-based Learning: Erfahrungen mit und Perspektiven für digitale Lernspiele in der betrieblichen Bildung. *Handbuch E-Learning, Kap*, 4:1–17, 2003.

[Pre07] Prensky, Marc. *Digital Game-Based Learning*, pages 119–124. Paragon House, St. Paul, Minn., 2007.

[Spr10] Franziska Spring. personal communication, 2010.

[TPR+92] A.N. Thomas, J. Pellegrino, P. Rowley, M. Scardamalia, E. Soloway, and J. Webb. Designing collaborative, knowledge-building environments for tomorrow's schools. In *Conference on Human Factors in Computing Systems: Proceedings of the SIGCHI conference on Human factors in computing systems*. Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, 1992.

[Vara] Various Authors. `http://unity3d.com/`. zuletzt abgerufen am 10.04.2011.

[Varb] Various Authors. *SWT: The Standard Widget Toolkit*. `http://www.eclipse.org/swt/`. zuletzt abgerufen am 08.04.2011.

Abbildungsverzeichnis

Tabellenverzeichnis

A XML-Deklarationen

plugin.xml (edu.uni_weimar.simuframe.skillmanager)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension-point id="skill" name="skill" schema="schema/skill.exsd"/>
   <extension-point id="badge" name="badge" schema="schema/badge.exsd"/>
</plugin>
```

plugin.xml (edu.uni_weimar.simuframe.missionmanager)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension-point id="mission" name="mission" schema="schema/mission.exsd"/>
</plugin>
```

skill.exsd

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Schema file written by PDE -->
<schema targetNamespace="edu.uni_weimar.simuframe.skillmanager"
xmlns="http://www.w3.org/2001/XMLSchema">

   <element name="extension">
      <complexType>
         <sequence>
            <element ref="skill" minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
         <attribute name="point" type="string" use="required">
         </attribute>
         <attribute name="id" type="string">
```

```
        </attribute>
        <attribute name="name" type="string">
        </attribute>
      </complexType>
   </element>

   <element name="skill">
      <complexType>
        <sequence>
           <element ref="requirement" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="name" type="string" use="required">
        </attribute>
        <attribute name="id" type="string" use="required">
        </attribute>
        <attribute name="description" type="string">
        </attribute>
      </complexType>
   </element>

   <element name="requirement">
      <complexType>
        <attribute name="id" type="string" use="required">
        </attribute>
        <attribute name="level" type="string" use="required">
        </attribute>
      </complexType>
   </element>
</schema>
```

badge.exsd

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Schema file written by PDE -->
<schema targetNamespace="edu.uni_weimar.simuframe.skillmanager"
xmlns="http://www.w3.org/2001/XMLSchema">

   <element name="extension">
      <complexType>
        <sequence>
           <element ref="badge" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
```

```
         <attribute name="point" type="string" use="required">
         </attribute>
         <attribute name="id" type="string">
         </attribute>
         <attribute name="name" type="string">
         </attribute>
      </complexType>
   </element>

   <element name="badge">
      <complexType>
         <sequence>
            <element ref="requirement" minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
         <attribute name="name" type="string" use="required">
         </attribute>
         <attribute name="description" type="string" use="required">
         </attribute>
         <attribute name="id" type="string" use="required">
         </attribute>
      </complexType>
   </element>

   <element name="requirement">
      <complexType>
         <attribute name="id" type="string" use="required">
         </attribute>
         <attribute name="level" type="string" use="required">
         </attribute>
      </complexType>
   </element>
</schema>
```

mission.exsd

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Schema file written by PDE -->
<schema targetNamespace="edu.uni_weimar.simuframe.missionmanager"
xmlns="http://www.w3.org/2001/XMLSchema">

   <element name="extension">
      <complexType>
```

```
      <sequence>
         <element ref="mission" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="point" type="string" use="required">
      </attribute>
      <attribute name="id" type="string">
      </attribute>
      <attribute name="name" type="string">
      </attribute>
   </complexType>
</element>

<element name="mission">
   <complexType>
      <sequence>
         <element ref="requirement" minOccurs="0" maxOccurs="unbounded"/>
         <element ref="yield" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="name" type="string" use="required">
      </attribute>
      <attribute name="id" type="string" use="required">
      </attribute>
      <attribute name="description" type="string" use="required">
      </attribute>
   </complexType>
</element>

<element name="requirement">
   <complexType>
      <attribute name="id" type="string" use="required">
      </attribute>
      <attribute name="level" type="string" use="required">
      </attribute>
   </complexType>
</element>

<element name="yield">
   <complexType>
      <attribute name="id" type="string" use="required">
      </attribute>
      <attribute name="level" type="string" use="required">
      </attribute>
   </complexType>
</element>
```

```
</schema>
```

plugin.xml (edu.uni_weimar.simuframe.acoustics)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension point="edu.uni_weimar.simuframe.missionmanager.mission">
      <mission name="MA" id="M1" description="Testmission">
         <yield id="A" level="70"></yield>
      </mission>
      <mission name="MA" id="M2" description="Testmission">
         <yield id="A" level="50"></yield>
      </mission>
      <mission name="MB" id="M3" description="Testmission">
         <requirement id="A" level="30"></requirement>
         <yield id="B" level="100"></yield>
      </mission>
      <mission name="MC" id="M4" description="Testmission">
         <requirement id="A" level="70"></requirement>
         <requirement id="B" level="30"></requirement>
         <yield id="C" level="100"></yield>
      </mission>
      <mission name="MD" id="M5" description="Testmission">
         <requirement id="C" level="100"></requirement>
         <yield id="D" level="80"></yield>
      </mission>
      <mission name="ME" id="M6" description="Testmission">
         <requirement id="D" level="80"></requirement>
         <yield id="E" level="20"></yield>
      </mission>
   </extension>

   <extension point="edu.uni_weimar.simuframe.skillmanager.skill">
      <skill name="SkillA" id="A" description="Testskill">
      </skill>
      <skill name="SkillB" id="B" description="Testskill">
         <requirement id="A" level="30"></requirement>
      </skill>
      <skill name="SkillC" id="C" description="Testskill">
         <requirement id="A" level="70"></requirement>
         <requirement id="B" level="30"></requirement>
```

```
      </skill>
      <skill name="SkillD" id="D" description="Testskill">
         <requirement id="C" level="100"></requirement>
      </skill>
      <skill name="SkillE" id="E" description="Testskill">
         <requirement id="D" level="80"></requirement>
      </skill>
   </extension>
   <extension point="edu.uni_weimar.simuframe.skillmanager.badge">
      <badge description="Testbadge"
         id="edu.uni_weimar.simuframe.acoustics.badge"
         name="acoustic expertise">
         <requirement id="A" level="30"></requirement>
         <requirement id="B" level="30"></requirement>
         <requirement id="C" level="30"></requirement>
         <requirement id="D" level="30"></requirement>
         <requirement id="E" level="30"></requirement>
      </badge>
   </extension>
</plugin>
```

plugin.xml (edu.uni_weimar.simuframe.heat)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension point="edu.uni_weimar.simuframe.missionmanager.mission">
      <mission name="MF" id="M7" description="Testmission">
         <yield id="F" level="40"></yield>
      </mission>
      <mission name="MF" id="M8" description="Testmission">
         <yield id="F" level="60"></yield>
      </mission>
      <mission name="MG" id="M9" description="Testmission">
         <requirement id="F" level="100"></requirement>
         <yield id="G" level="60"></yield>
      </mission>
      <mission name="MH" id="M10" description="Testmission">
         <requirement id="D" level="10"></requirement>
         <requirement id="G" level="20"></requirement>
         <yield id="H" level="70"></yield>
      </mission>
```

```
      <mission name="MHE" id="M11" description="Testmission">
         <requirement id="D" level="80"></requirement>
         <requirement id="G" level="20"></requirement>
         <yield id="H" level="10"></yield>
         <yield id="E" level="10"></yield>
      </mission>
   </extension>
   <extension point="edu.uni_weimar.simuframe.skillmanager.skill">
      <skill name="SkillF" id="F" description="Testskill">
      </skill>
      <skill name="SkillG" id="G" description="Testskill">
         <requirement id="F" level="100"></requirement>
      </skill>
      <skill name="SkillH" id="H" description="Testskill">
         <requirement id="G" level="20"></requirement>
         <requirement id="D" level="10"></requirement>
      </skill>
   </extension>
   <extension point="edu.uni_weimar.simuframe.skillmanager.badge">
      <badge  description="Testbadge"
         id="edu.uni_weimar.simuframe.acoustics.badge"
         name="heat expertise">
         <requirement id="F" level="60"></requirement>
         <requirement id="G" level="60"></requirement>
         <requirement id="H" level="60"></requirement>
      </badge>
   </extension>
</plugin>
```

B Datenmodell

ID	Name	Requirement		Modul
		ID	Level	
A	Skill A	/	/	Acoustic
B	Skill B	A	30	Acoustic
C	Skill C	A B	70 30	Acoustic
D	Skill D	C	100	Acoustic
E	Skill E	D	80	Acoustic
F	Skill F	/	/	Heat
G	Skill G	F	100	Heat
H	Skill H	D G	10 20	Heat

Tabelle B.1: Skills

ID	Name	Requirement		Yield		Modul
		ID	Level	ID	Level	
M1	MA	/	/	A	70	Acoustic
M2	MA	/	/	A	50	Acoustic
M3	MB	A	30	B	100	Acoustic
M4	MC	A B	70 30	C	100	Acoustic
M5	MD	C	100	D	80	Acoustic
M6	ME	D	80	E	20	Acoustic
M7	MF	/	/	F	40	Heat
M8	MF	/	/	F	60	Heat
M9	MG	F	100	G	60	Heat
M10	MH	D G	10 20	H	70	Heat
M11	MH,E	D G	80 20	H E	10 10	Heat

Tabelle B.2: Missionen

ID	Name	Requirement		Modul
		ID	Level	
1	Acoustics Expertise	A	30	Acoustic
		B	30	
		C	30	
		D	30	
		E	30	
2	Heat Expertise	F	60	Heat
		G	60	
		H	60	

Tabelle B.3: Badges

Selbstständigkeitserklärung

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Weimar, den 17. Januar 2012

Janina Held