Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminarausarbeitung

Hauke Wittern

Modellgetriebene Entwicklung von Pervasive Games — Entwicklung einer domänenspezifischen Sprache

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung **2**

2 Grundlagen **3**
2.1 Pervasive Games . . . 3
2.2 Modellgetriebene Softwareentwicklung . . . 3
2.2.1 Definition . . . 4
2.2.2 Domänenspezifische Sprachen . . . 4
2.2.3 Gründe für die modellgetriebene Softwareentwicklung . . . 4

3 Modellgetriebene Softwareentwicklung von Pervasive Games **7**
3.1 Potential der modellgetriebenen Softwareentwicklung von Pervasive Games . 7
3.2 Verschiedene Abstraktionsebenen bei Pervasive Games . . . 8
3.3 Findung der konkreten Syntax einer DSL für Pervasive Games . . . 9
3.3.1 Anforderungen an die Syntax einer DSL . . . 9
3.3.2 Kategorisierung von DSLs . . . 10
3.3.3 Auswahl einer konkreten Syntax für Pervasive Games . . . 11
3.4 Entwicklungsablauf . . . 12
3.4.1 Entwicklung der Domänenarchitektur . . . 12
3.5 Werkzeugauswahl . . . 14
3.6 Risiken . . . 15

4 Zusammenfassung und Ausblick **16**

Abbildungsverzeichnis **17**

Glossar **18**

Quellenverzeichnis **19**

1 Einleitung

Pervasive Games sind eine relativ neue Art von Spielen, bei denen Elemente aus der von Computerspielen bekannten virtuellen Welt mit der realen Welt kombiniert werden. Die Spiele finden in der realen Welt statt, Computertechnologien haben aber einen großen Einfluss auf den Spielablauf. Insbesondere Technologien des Mobile-Computing spielen eine große Rolle bei der Umsetzung von Pervasive Games.

Die Vermischung der beiden Welten führt zu Problemen, die es bei klassischen Computerspielen nicht gibt. So ist zum Beispiel die aktuelle Position an der sich die Spieler gerade in der realen Welt befinden von Bedeutung für ein Pervasive Game. Die Position der Spieler muss deshalb mit geeigneten Technologien ermittelt werden. Außerdem werden bei Pervasive Games mobile Geräte verwendet, wie zum Beispiel Handys. Mit diesen werden virtuelle Inhalte in die reale Welt integriert. Es gibt viele verschiedene mobile Geräte, die auf unterschiedlichen Technologien basieren, unterschiedlich leistungsfähig sind und über einen unterschiedlichen Funktionsumfang verfügen. Es ist daher nicht einfach Spiele zu entwickeln, die auf möglichst vielen Geräten ausgeführt werden können. Ein Pervasive Game muss außerdem modelliert werden. Pervasive Games können an verschiedenen Orten gespielt werden. Es ist deshalb wichtig, das Model eines Spiels einfach an den Ausführungsort anpassen zu können. Die Entwicklung von Pervasive Games ist aus diesen Gründen komplex und es muss versucht werden, die Komplexität so weit wie möglich zu reduzieren.

In dieser Ausarbeitung wird die modellgetriebener Softwareentwicklung als Ansatz zur Reduzierung der Komplexität bei der Entwicklung von Pervasive Games vorgestellt. Mit domänenspezifischen Sprachen soll die Modellierung von Pervasive Games vereinfacht werden. Außerdem soll die Plattformunabhängigkeit erhöht werden. Die Umsetzung erfolgt in den nächsten beiden Semestern im Rahmen des Projekts im Masterstudiengang Informatik an der Hochschule für Angewandte Wissenschaften Hamburg.

Diese Ausarbeitung gibt einen Überblick über die wichtigsten Aspekte bei der modellgetriebenen Softwareentwicklung von Pervasive Games und untersucht, wie eine domänenspezifische Sprache für Pervasive Games aussehen sollte. Außerdem gibt diese Ausarbeitung einen Überblick über das geplante Vorgehen im Projekt und stellt einige Werkzeuge vor, die bei der Entwicklung eingesetzt werden könnten. Anschließend werden einige Risiken angesprochen, die im Projekt auftreten können.

2 Grundlagen

Dieses Kapitel stellt die wichtigsten Grundlagen vor, die in dieser Ausarbeitung von Bedeutung sind. Zunächst wird definiert, was Pervasive Games sind. Danach wird die modellgetriebene Softwareentwicklung vorgestellt.

2.1 Pervasive Games

In der Literatur gibt es unterschiedliche Definitionen, was ein Pervasive Game ist. [Hinske u. a. (2007)] hat den Begriff und die unterschiedlichen Definitionen näher untersucht und folgendermaßen zusammengefasst:

> Pervasive Games are a ludic form of mixed reality entertainment with goals, rules, competition, and attacks, based on the utilization of Mobile Computing and/or Pervasive Computing technologies. [Hinske u. a. (2007), S.12]

Pervasive Games finden demnach in einer gemischten Realität statt. Sie haben sowohl Eigenschaften von Spielen in der realen Welt (Real World Games) als auch Eigenschaften von Spielen in virtuellen Welten (Virtual Reality Games). Real World Games sind beispielsweise Brettspiele. Computerspiele sind Virtual Reality Games.

Pervasive Games bauen auf dem Pervasive Computing auf. Beim Pervasive Computing geht es darum, Computer allgegenwärtig in unserer Umwelt zu integrieren. Pervasive Games verwenden außerdem häufig Technologien (z. B. WLAN oder GPS) und Geräte (z.B. Handys) aus dem Bereich des Mobile Computing. [Hinske u. a. (2007)]

2.2 Modellgetriebene Softwareentwicklung

Dieses Unterkapitel erklärt, was man unter modellgetriebener Softwareentwicklung versteht, und stellt die wichtigsten Gründe für den Einsatz modellgetriebener Softwareentwicklung vor.

2.2.1 Definition

Bei modellgetriebener Softwareentwicklung (Model Driven Software Development, MDSD) nehmen Modelle überwiegend die Rolle von manuell geschriebenem Programmcode ein [Stahl u. a. (2007)]. Modelle werden hier zu Artefakten erster Klasse [Gruhn u. a. (2006)].

Modelle werden häufig zur Dokumentation oder als Bauplan eines Systems genutzt, nach dem das System manuell implementiert wird. [Stahl u. a. (2007)] nennt diese Art der Nutzung von Modellen modellbasiert. Bei modellgetriebener Softwareentwicklung beschreiben die Modelle nicht nur die Architektur eines Systems, die Modelle sind zugleich die Architektur. Änderungen am System werden bei MDSD an den Modellen vorgenommen, nicht am Programmcode. Hierfür ist es notwendig, aus den Modellen automatisiert lauffähige Software zu erzeugen. Dies wird entweder durch die Generierung von Code aus den Modellen vor der Ausführung oder durch die Interpretierung der Modelle zur Laufzeit erreicht. [Stahl u. a. (2007)]

2.2.2 Domänenspezifische Sprachen

Eine domänenspezifische Sprache (Domain Specific Language, DSL) ist eine Sprache, mit der Aspekte innerhalb einer Domäne formuliert werden können. Bei MDSD werden DSLs verwendet, um die Modelle zu formulieren.

Eine DSL besteht aus einem Metamodell und einer konkreten Syntax. Das Metamodell beschreibt die relevanten Konzepte der Domäne. Die Elemente des Metamodells und die Beziehungen zwischen den Elementen bezeichnet man auch als abstrakte Syntax. Die konkrete Syntax beschreibt, wie die Konzepte des Metamodells konkret dargestellt werden [Stahl u. a. (2007)]. In der Domäne von Pervasive Games könnte ein Metamodell beispielsweise ein Modellelement für Gegenstände der realen Welt und virtuellen Welt beinhalten. Eine grafische konkrete Syntax könnte solche Gegenstände in einem Diagramm mit speziellen Symbolen darstellen.

Abbildung 2.1 zeigt beispielhaft, wie ein Metamodell für Schnitzeljagden bei Pervasive-Games aussehen könnte. Dieses Metamodell wird in Abbildung 2.2 zur Modellierung einer Schnitzeljagd verwendet.

2.2.3 Gründe für die modellgetriebene Softwareentwicklung

Die Anwendung von modellgetriebener Softwareentwicklung verspricht eine Reihe von Vorteilen. Dieser Abschnitt stellt die wichtigsten Vorteile der MDSD vor.

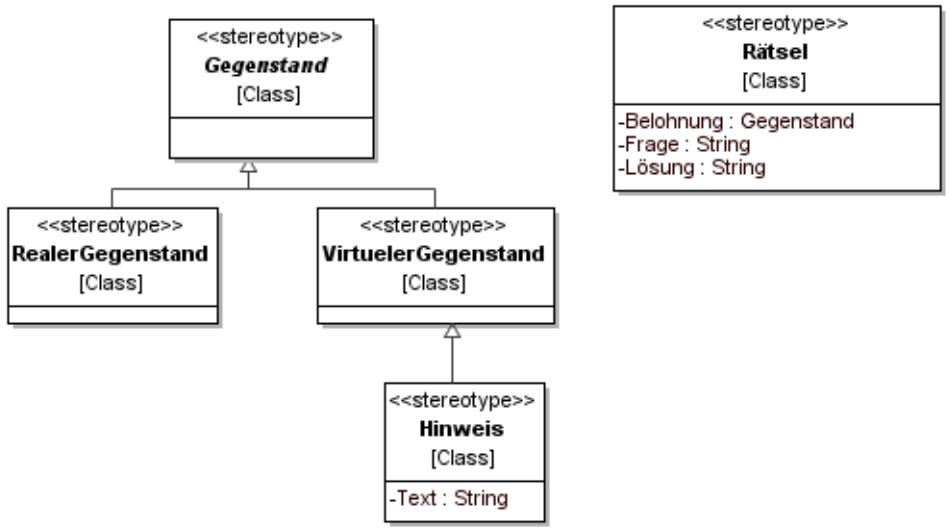

Abbildung 2.1: Beispiel-Metamodell (UML-Profil) für eine Schnitzeljagd

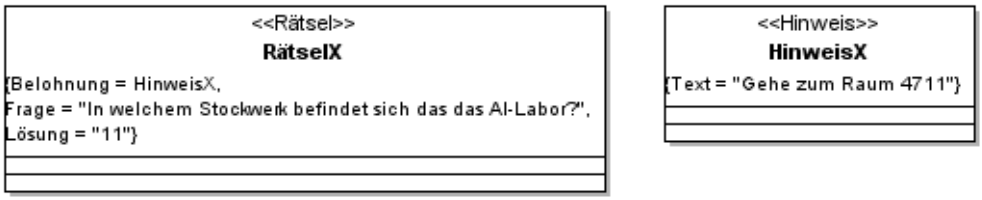

Abbildung 2.2: Beispiel-UML-Modell einer Schnitzeljagd

Hoher Abstraktionsgrad

Der wichtigste Grund für die MDSD sei laut [Stahl u. a. (2007)] die Möglichkeit, das System auf einer höheren Abstraktionsebene als der zugrundeliegenden Programmiersprache zu entwickeln. Dies ermöglicht es, das System näher an der fachlichen Domäne zu entwickeln und mit treffenderen Elementen zu beschreiben (z.B. Rätsel statt Klasse). Die Komplexität der Software-Artefakte soll dadurch reduziert werden. Details der technischen Umsetzung sind nicht im Modell enthalten und müssen dem Modell-Entwickler nicht bekannt sein [Stahl u. a. (2007)].

Die Komplexität des Softwareentwicklungsprozesses insgesamt reduziert der Einsatz von MDSD aber nicht automatisch. [Hailpern und Tarr (2006)] kritisiert, der Einsatz von MDSD führe lediglich zu einer Verlagerung der Komplexität an andere Stellen im Entwicklungsprozess.

Einheitliche Architektur

Wie in Abschnitt 2.2.1 erwähnt, werden bei MDSD Änderungen des Systems an den Modellen selbst vorgenommen. Dies hat den Vorteil, dass Inkonsistenzen zwischen dem Modell und der tatsächlichen Architektur vermieden werden [Stahl u. a. (2007)].

Bei MDSD werden die durch ein Modell beschriebenen Elemente automatisch übersetzt. Dadurch ist gewährleistet, dass die übersetzten Elemente auf eine einheitliche Art implementiert werden und die Architektur des Systems nicht verfälscht wird [Stahl u. a. (2007)].

Interoperabilität

Durch den hohen Abstraktionsgrad wird die Interoperabilität und Plattformunabhängigkeit potenziell erhöht. Es ist denkbar, ein und dasselbe Modell auf verschiedene Weise technisch zu realisieren. In der Praxis ist die Plattformunabhängigkeit nur eingeschränkt gegeben. Gründe hierfür seien laut [Stahl u. a. (2007)] Details der Plattform, die sich auf gewisse Weise doch in den Modellen niederschlügen. Hinzu käme, dass das System in der Regel auch aus nicht generiertem Code bestehe, der von Hand geschrieben und plattformspezifisch ist.

Produktivitätssteigerung

Die automatische Generierung von lauffähiger Software erspart das manuelle Schreiben von Quelltext. Eine wirkliche Produktivitätssteigerung ist bei MDSD aber nur mittelfristig realistisch, wenn Änderungen am System vorgenommen wurden [Stahl u. a. (2007)]. Die Zeitersparnis ergibt sich hier aus dem wiederholten automatisierten Generieren des geänderten Systems.

Fazit

Aufgrund der zuvor genannten Vorteile kann die Entwicklung von Software mit einem modellgetriebenen Vorgehen erleichtert werden. Dennoch muss bei jedem Projekt kritisch bewertet werden, ob der Einsatz von MDSD sinnvoll ist. Bei sehr kleinen Projekten lohnt es sich eher nicht, MDSD einzusetzen.

3 Modellgetriebene Softwareentwicklung von Pervasive Games

Dieses Kapitel erläutert, wie die Entwicklung von Pervasive Games durch den Einsatz von modellgetriebener Softwareentwicklung vereinfacht werden kann. Zunächst wird das Potential von MDSD bei Pervasive Games aufgezeigt. Danach wird gezeigt, wie eine DSL für Pervasive Games aussehen sollte. Anschließend wird der Entwicklungsablauf bei modellgetriebener Entwicklung von Pervasive Games skizziert und wie die Entwicklung der DSL darin eingeordnet ist. Bevor zum Schluss die wichtigsten Risiken erwähnt werden, werden Werkzeuge vorgeschlagen, die im Projekt[1] verwendet werden können.

3.1 Potential der modellgetriebenen Softwareentwicklung von Pervasive Games

Die grundlegenden Herausforderungen, die allgemein bei der Entwicklung von Pervasive Games auftreten, wurden in [Broll u. a. (2006)] zusammengestellt:

- Positionsbestimmung der Spieler. Der aktuelle Aufenthaltsort der Spieler hat eine große Bedeutung bei Pervasive Games.
- Kommunikation der Spieler untereinander und mit dem System
- Verwendung unterschiedlicher Geräte
- Entwicklung der Game-Engine
- Modellierung des Spiels
- Orchestrierung und Überwachung eines Spiels

[1] Mit Projekt ist hier das in den nächsten beiden Semestern im Rahmen des Masterstudiengangs Informatik durchgeführte Projekt an der Hochschule für Angewandte Wissenschaften Hamburg gemeint

Während der Entwicklung eines Pervasive Games müssen für diese Herausforderungen Lösungen gefunden werden. Für die Positionsbestimmung und die Kommunikation der Spieler untereinander und mit dem System kann eine Plattform für Pervasive Games Konzepte und Techniken bereitstellen. Ebenso können Abstraktionsmechanismen der Plattform die Verwendung unterschiedlicher Geräte erleichtern.

Die Modellierung eines Spiels ist eine Tätigkeit, welche die Plattform alleine kaum erleichtern kann. Die Plattform kann Progammierschnittstellen bereitstellen, welche die Programmierung der Plattform soweit wie möglich erleichtern. Die Konzepte eines Spiels lassen sich damit aber nicht effizient ausdrücken. An diesem Punkt verspricht die modellgetriebene Softwareentwicklung, die Entwicklung von Pervasive Games zu erleichtern. Der in Kapitel 2.2 erwähnte hohe Abstraktionsgrad bei MDSD ermöglicht es, die Komplexität bei der Entwicklung von Pervasive Games gering zu halten. Eine domänenspezifische Sprache ermöglicht es ein Spiel technikneutral mit den Konzepten der Domäne des Spiels zu beschreiben. Das Modell kann dadurch leichter verstanden und vom Entwicklerteam diskutiert werden.

Die MDSD kann darüber hinaus die Plattformunabhängigkeit des Systems erhöhen, wenn die Modelle technikneutral gehalten werden (siehe Kapitel 2.2.3). Aus den Modellen kann für die unterschiedlichen Geräte speziell angepasster Code generiert werden. Es muss dabei jedoch beachtet werden, dass ein System in der Regel nicht vollständig generiert werden kann. Die Artefakte des Systems, die nicht generiert werden können, müssen deshalb von Hand geschrieben werden.

Außerdem kann die Produktivität bei der Entwicklung von Pervasive Games durch MDSD beträchtlich gesteigert werden. Wenn Anpassungen eines Spiels erfordelich sind, beispielsweise bei einem Wechsel des Spielorts, kann Zeitgespart werden. Bei modellgetriebener Softwareentwicklung können die Modelle einfach angepasst werden und daraus automatisiert Code generiert werden. Der Anteil manuell zu implementierender System-Artefakte wird minimiert. Bei einem modelbasierten Vorgehen müssen dagegen alle Änderungen manuell vorgenommen werden.

3.2 Verschiedene Abstraktionsebenen bei Pervasive Games

Bei der Entwicklung von Pervasive Games kann man zwei Abstraktionsebenen unterscheiden, die eine besondere Bedeutung haben. Die erste Abstraktionsebene bildet die Plattform eines Pervasive Games. Die Plattform löst die grundsätzlichen Probleme (siehe Abschnitt 3.1), die bei Pervasive Games auftreten. Konzepte konkreter Spiele werden in dieser Abstraktionsebene nicht behandelt.

Die zweite Abstraktionsebene bilden die verschiedenen konkreten Spiel-Domänen. Spiele die zur selben Domäne (wie z. B. Schnitzeljagd) gehören bilden eine Software-Systemfamilie. In diesen Domänen findet man jedoch in der Regel nicht alle oder gar keine Konzepte der Plattform wieder. Eine allgemeine Plattform für Pervasive Games ist die Obermenge unterschiedlicher Familien von Pervasive Games.

Um diesen beiden Abstraktionsebenen gerecht zu werden, sind verschiedene DSLs sinnvoll. Es sollte eine DSL auf der Abstraktionsebene der Plattform geben und jeweils eine pro Spiel-Domäne. Die beiden Abstraktionsebenen können mit einer Modell-zu-Modell-Transformation überbrückt werden. Das bedeutet, dass ein konkretes Spiel ausschließlich mit der zur Spiel-Domäne gehörenden DSL modelliert werden muss. Dieses Modell wird durch eine Modell-Transformation automatisiert in die Plattformsprache übersetzt. Ein Entwickler, der ein Spiel modelliert, muss also nur eine Sprache verwenden.

3.3 Findung der konkreten Syntax einer domänenspezifische Sprache für Pervasive Games

Dieser Abschnitt untersucht, wie die konkrete Syntax einer DSL für Pervasive Games aussehen sollte. Dazu werden zunächst Anforderungen an die DSL aufgestellt. Danach werden die grundlegenden Kategorien untersucht, zu der eine DSL gehören kann. Zum Schluss wird abgewägt, welche Kategorie eine DSL für Pervasive Games haben sollte.

3.3.1 Anforderungen an die Syntax einer DSL

Bei der Entwicklung einer domänenspezifischen Sprache gibt es nach [Stahl u. a. (2007)] vier Anforderungen an die Syntax, die beachtet werden müssen:

[A1] Die Syntax muss geeignet sein, die Konzepte und Struktur der Domäne einfach und effizient darzustellen.

[A2] Die Syntax muss geeignet sein, Modelle einfach zu erstellen, zu lesen und zu verändern.

[A3] Die Erstellung und Weiterentwicklung der Syntax muss einfach sein.

[A4] Die bei der Verwendung der DSL benötigten Werkzeuge (Parser, Editoren) müssen sich gut in das Projekt integrieren lassen.

Diese Anforderungen sind auch für DSLs für Pervasive Games von Bedeutung. Da im Projekt nur begrenzte Zeit zur Verfügung steht und es einen erheblichen Aufwand erfordert, Editoren und Parser zu entwickeln, kommt folgende Anforderung hinzu:

[A5] Für die DSL müssen fertige Editoren und Parser verfügbar sein.

Anhand dieser Anforderungen wird im Folgenden entschieden, wie die Syntax von domänenspezifischen Sprachen für Pervasive Games prinzipiell aussehen sollte. Dafür werden zunächst die Kategorien vorgestellt, zu denen eine DSLs gehören kann.

3.3.2 Kategorisierung von DSLs

Grafische oder textuelle DSL

Die Form einer konkreten Syntax kann entweder grafisch oder textuell sein. Ob eine DSL grafisch oder textuell ist, hat einen entscheidenden Einfluss darauf, wie Modelle erstellt und betrachtet werden.

Grafische Modelle sind geeignet, die relevanten Dinge einer Domäne abstrakt darzustellen. Sie helfen Strukturen und Zusammenhänge zu verstehen. Grafische Modelle sind deshalb als Diskussionsgrundlage zwischen den am Projekt beteiligten Personen geeignet. Textuelle Modelle sind dagegen weniger gut lesbar. Insbesondere ist es bei diesen schwieriger Zusammenhänge zu verstehen als bei grafischen Modellen. Um einen Überblick zu bekommen, können in grafischen Modellen leicht unwichtige Details ausgeblendet werden. Bei textuellen Modellen können nur wenige Details ausgeblendet werden (z. B. durch Code Folding), ohne dass das Modell eine andere Bedeutung bekommt. Als Diskussionsgrundlage sind grafische DSLs deshalb besser geeignet als textuelle DSLs. [Stahl u. a. (2007)]

Ein Vorteil von textuellen Modellen ist, dass die Modellierung in Teamarbeit einfach zu realisieren ist. Es gibt eine Reihe von Werkzeugen, die es erlauben Änderungen an einem Text zusammenzuführen (Merge). Bei grafischen Modellen ist dies nicht so einfach möglich, da die Modelle in der Regel als unteilbarer Objektgraph gespeichert werden. [Stahl u. a. (2007)]

Interne oder externe DSL

Eine interne DSL ist eine Sprache, die in eine andere — als Host-Sprache auftretende — Sprache eingebettet ist. Streng genommen ist eine interne DSL keine eigene Sprache, sondern lässt eine vorhandene Sprache lediglich wie eine neue Sprache aussehen [Stahl u. a. (2007)]. Eine externe DSL ist hingegen eine eigenständige Sprache.

Mit der UML können grafische interne DSLs in Form von UML-Profilen definiert werden. In einem UML-Profil werden Stereotypen definiert, welche die Sprachelemente der internen DSL sind. Ein Profil ist also ein Metamodell. Die Sprache wird verwendet, indem die Elemente eines Modells mit Stereotypen gekennzeichnet werden. Die Attribute der Stereotypen werden mit sogenannten Tagged-Values angegeben. Abbildung 2.1 ist ein Beispiel für ein UML-Profil und Abbildung 2.2 zeigt, wie dieses verwendet wird.

Ein Vorteil von internen DSLs ist, dass die Werkzeuge der Host-Sprache wiederverwendet werden können. So können beispielsweise beim Einsatz von UML-Profilen die vorhandenen UML-Werkzeuge verwendet werden. Für externe DSLs müssen dagegen eigene Editoren und Parser erstellt werden, da sie nicht auf einer Host-Sprache basieren [Stahl u. a. (2007)]. Allerdings gibt es Werkzeuge, welche die Entwicklung von Editoren und Parsern vereinfachen. Zum Beispiel können mit dem Eclipse Graphical Modeling Framework (GMF, [URL:GMF]) grafische Editoren für externe DSLs erzeugt werden. Mit Xtext aus dem openArchitectureWare-Projekt (oAW, [URL:oAW]) können für textuelle DSLs Texteditoren und Parser generiert werden.

Der Vorteil von externen DSLs ist, dass man keine Einschränkungen bei der Definition der DSL hat. Die konkrete Syntax kann deshalb so gewählt werden, dass sie die Konzepte der Domäne angemessen darstellt. Bei internen DSLs ist die Syntax hingegen weitestgehend vorgegeben. Die Syntax interner DSLs kann die Konzepte einer Domäne deshalb unter Umständen nicht angemessen darstellen. [Stahl u. a. (2007)]

3.3.3 Auswahl einer konkreten Syntax für Pervasive Games

Bei der Entwicklung von Pervasive Games müssen die Strukturen und Zusammenhänge in den Modellen leicht verständlich sein. Die Entwickler müssen in einem Review leicht erkennen können, ob das Spiel so wie es modelliert wurde funktioniert. Reviews sind die einzige praktikable Art die Semantik eines Modells zu überprüfen [Spillner und Linz (2005)]. Es kann nicht ohneweiteres automatisiert überprüft werden, ob das Modell eines Spiels Fehlerzustände enthält, die das Spiel unspielbar machen. Die Fehlerwirkung des Fehlerzustands könnte man zwar beobachten, indem man das Spiel testweise spielt, der Fehlerzustand wird damit aber noch nicht gefunden. Außerdem bedeutet ein Testlauf eines Pervasive Games einen erheblichen Aufwand. Deshalb müssen die mit der Pervasive-Game-DSL verfassten Modelle für Reviews geeignet sein. Grafische DSLs sind besonders als Diskussionsgrundlage geeignet und sind damit auch für Reviews besser geeignet als textuelle DSLs.

Externe DSLs erfüllen die Anforderung, dass fertige Editoren und Parser verfügbar sein müssen nicht. Externe DSLs kommen daher für das Projekt im nächsten Semester zunächst nicht in Frage. Wenn es die Zeit zulässt, ist eine externe DSL jedoch einer internen vorzuziehen, da sie die Konzepte einer Spiel-Domäne am besten ausdrücken kann.

Eine grafische interne DSL erfüllt aus diesen Gründen die in Abschnitt 3.3.1 genannten Anforderungen am besten. Es wird deshalb an dieser Stelle empfohlen, die DSLs im Projekt als UML-Profil zu implementieren. Dies gilt sowohl für die DSL für die allgemeine Pervasive Game Plattform, als auch für die DSL einer speziellen Spiel-Domäne. Im zweiten Semester des Projekts kann zusätzlich, wenn es die Zeit zulässt, eine grafische externe DSL für die Spiel-Domäne entwickelt werden.

3.4 Entwicklungsablauf

Die Softwareentwicklung kann und sollte bei MDSD iterativ erfolgen, da die Konzepte einer Domäne nicht immer einfach zu verstehen sind. Die Analyse der Domäne und das Design einer DSL sollten deshalb in mehreren Iterationszyklen erfolgen. Außerdem müssen zur Domäne und Plattform passende Transformationen entwickelt werden, was einen wesentlichen Teil der modellgetriebenen Softwareentwicklung ausmacht. Auch bei der Entwicklung von Pervasive Games ist ein iteratives Vorgehen wichtig, da ein reibungsloser Spielablauf in der Regel erst nach zahlreichen Anpassungen möglich ist.

Der grobe Ablauf der Entwicklung (siehe Abb. 3.1) kann innerhalb eines Iterationsschritts zunächst in zwei parallel durchführbare Aktivitäten unterteilt werden: die Anwendungsanalyse und die Erstellung der Domänenarchitektur [Stahl u. a. (2007)]. Bei der Anwendungsanalyse werden die funktionalen Anforderungen eines konkreten Pervasive Games ermittelt. Parallel dazu kann die Domänenarchitektur entwickelt werden, welche für die modellgetriebene Entwicklung des Spiels benötigt wird. Die erstellte Domänenarchitektur besteht aus einem Metamodell für die Domäne Pervasive Games, einer Plattform als Laufzeitsystem und dazu passenden Transformationen. Die Transformationen dienen dazu, aus den Modellen eines Spiels ausführbaren Code zu erzeugen. Nach Abschluss dieser beiden Aktivitäten folgt die formale Modellierung. Während dieser Aktivität wird die Fachlichkeit des konkreten Spiels, unter Verwendung des zuvor entwickelten Metamodells, formal modelliert. Anschließend wird aus dem erstellten Modell Code generiert.

3.4.1 Entwicklung der Domänenarchitektur

Die Entwicklung der Domänenarchitektur ist die Aktivität während der u. a. die domänenspezifische Sprache und die zugehörigen Transformationen entwickelt werden. Die Entwicklung einer DSL wird ein Ziel des Projekts in den folgenden Semestern sein. Daher geht dieser Abschnitt näher auf diese Aktivität ein. Abbildung 3.2 gibt einen Überblick über die Aktivitäten bei der Entwicklung der Domänenarchitektur.

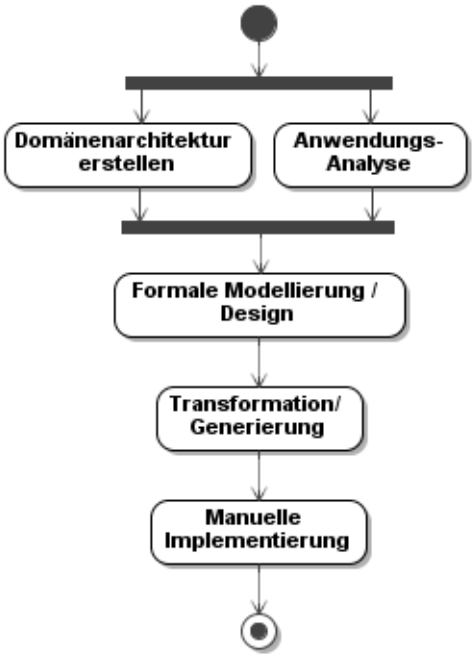

Abbildung 3.1: Überblick über die Aktivitäten bei modellgetriebener Softwareentwicklung (nach [Stahl u. a. (2007)])

Ein wichtiger Punkt zu Beginn der Entwicklung der Domänenarchitektur ist die Analyse der Domäne und der Entwurf einer domänenspezifischen Sprache. Die Domänenanalyse dient dazu, ein zu der Domäne passendes Metamodell zu finden. Für dieses Metamodell wird anschließend eine DSL entworfen.

Anschließend muss ein Editor für die DSL entwickelt werden. Wenn es Editoren gibt, die wiederverwendet werden können, kann diese Tätigkeit auch wegfallen. In Abschnitt 3.3.3 wurde die Verwendung von UML-Profilen für das Projekt in den nächsten Semestern vorgeschlagen. Da es zahlreiche UML-Editoren gibt, muss kein neuer DSL-Editor geschrieben werden und es wird Zeit gespart.

Wichtig ist die Erstellung eines Referenzmodells. Ein Referenzmodell wird mit der zuvor Entwickelten DSL beschrieben. Es soll an einem Beispiel zeigen, wie die DSL verwendet werden soll. Das Referenzmodell ist damit ein Teil der Dokumentation der Domänenarchitektur.

Ein Modell alleine ist noch keine ausführbare Software. Zusätzlich werden eine Plattform und Transformationen zwischen der DSL und der Plattform benötigt. Bei dem hier vorgestellten Vorgehen entwickelt man deshalb mit Hilfe von Prototypen eine Plattform, die als Laufzeitumgebung für die Modelle der Domäne dient. Eine Referenzimplementierung zeigt, wie das Referenzmodell bei Verwendung der Plattform umgesetzt wird. Damit ist die Refe-

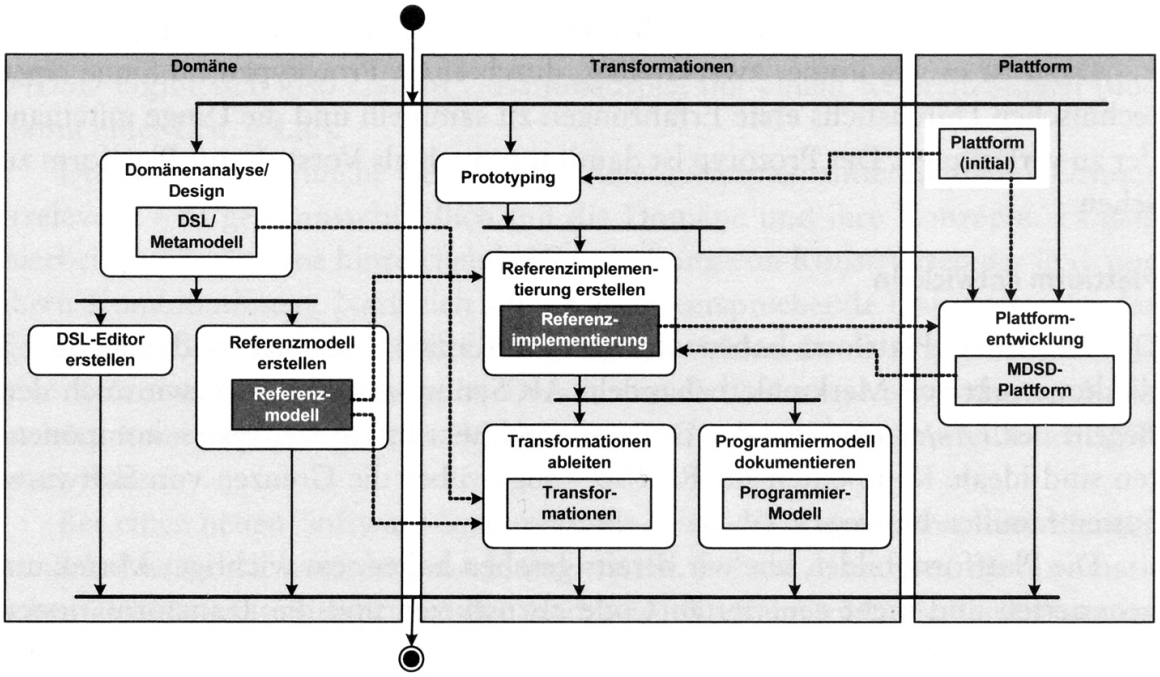

Abbildung 3.2: Erstellung einer Domänenarchitektur (aus [Stahl u. a. (2007)])

renzimplementierung wie auch das Referenzmodell ein Teil der Dokumentation der Domänenarchitektur. Die Referenzimplementierung ist besonders wichtig, da aus ihr schließlich die Transformationen abgeleitet werden.

Nach Abschluss dieser Tätigkeiten beginnt der nächste Iterationsschritt. Es ist wie gesagt wichtig, dass die Entwicklung iterativ erfolgt. Die Ergebnisse der einzelnen Tätigkeiten bei der Entwicklung der Domänenarchitektur liefern Erkenntnisse für die anderen Tätigkeiten. Diese Erkenntnisse müssen im nächsten Iterationsschritt beachtet werden.

3.5 Werkzeugauswahl

In [Wittern (2008)] konnte der Autor dieser Ausarbeitung bereits Erfahrungen mit drei verschiedenen MDSD-Werkzeugen sammeln. openArchitectureWare (oAW, [URL:oAW]) wurde in [Wittern (2008)] als derzeit am angemessensten für die modellgetriebene Softwareentwicklung erachtet. Die Verwendung von openArchitectureWare wird deshalb auch an dieser Stelle für das Projekt im nächsten Semester vorgeschlagen.

openArchitectureWare ist in die Eclipse-Entwicklungsumgebung integriert und bietet mit

XPand eine angemessene Template Sprache sowie mit XTend eine Sprache für Modelltransformationen. Mit Hilfe der Modelltransformationen und Templates kann openArchitectureWare aus Modellen Code generieren.

Die UML-Modelle müssen bei der Verwendung von openArchitectureWare mit einem externen UML-Werkzeug vorgenommen werden. Magic Draw ist ein UML-Werkzeug, dass besonders gut mit oAW zusammenarbeitet, da der Modellaustausch reibungslos funktioniert. Andere UML-Werkzeuge sind mit openArchitectureWare häufig nicht kompatibel.

Die endgültige Entscheidung welche Werkzeuge im Master-Projekt eingesetzt werden hängt davon ab, auf welche Plattform und im Projekt gesetzt wird und welche Entwicklungswerkzeuge dabei eingesetzt werden. Sollte Java als Plattform und Eclipse als Entwicklungsumgebung gewählt werden, ist der Einsatz von oAW sinnvoll.

3.6 Risiken

Dieser Abschnitt gibt einen kurzen Überblick über die Risiken, die bei der modellgetriebenen Entwicklung eines Pervasive Games im Rahmen des Master-Projekts in den folgenden Semestern auftreten können.

Das größte Risiko sind der Zeitfaktor und die Größe des ersten Iterationszyklus. Der erste Iterationszyklus dauert lange, da die Entwicklung der Domänenarchitektur am Anfang einen großen Aufwand erfordert. Es ist damit zu rechnen, dass die Entwicklung der Referenzimplementierung und der Transformationen den größten Teil der Zeit beanspruchen. Ein Problem bei der Entwicklung der Domänenarchitektur ist, dass die Referenzimplementierung und die Transformationen die Plattform benötigen. Das Projekt-Team muss deshalb bei jedem Iterationszyklus möglichst frühzeitig die Schnittstellen der Plattform vereinbaren.

Ein weiteres Risiko ist, dass die Syntax einer DSL für Pervasive Games nicht angemessen ist. Wenn die Syntax nicht angemessen ist, können Spiele nicht effizient mit der DSL modelliert werden. Unter Umständen zahlt sich der Einsatz von MDSD dann nicht aus.

Zudem besteht das Risiko, dass die benötigten Werkzeuge nicht zur Verfügung stehen oder nicht angemessen sind. Dieses Risiko ist im Master-Projekt jedoch gering, denn im vorherigen Abschnitt wurden einige Werkzeuge vorgestellt, mit denen der Autor dieser Ausarbeitung bereits Erfahrung hat. Sollten andere Werkzeuge eingesetzt werden, müssen diese zu Beginn des Projekt sorgfältig evaluiert werden.

4 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurde erläutert, wie die modellgetriebene Softwareentwicklung die Entwicklung von Pervasive Games erleichtern kann. Mit einer domänenspezischen Sprache ist es möglich, ein Spiel auf einer hohen Abstraktionsebene zu modellieren. Dadurch reduziert die MDSD die Komplexität bei der Entwicklung. Zusätzlich erhöht die MDSD die Plattformunabhängigkeit von Spielen. Aus technikneutralen Modellen kann gerätespezifischer Code generiert werden. Es muss dabei jedoch bedacht werden, dass nicht alle Details des Systems generiert werden können. Von Hand geschriebener Code ist nach wie vor nötig.

Eine grafische DSL wurde als angemessen für Pervasive Games befunden. Aus pragmatischen Gründen wurde nahe gelegt im Projekt die UML mit Profilen als DSL verwenden. Bei der Verwendung von UML müssen keine Werkzeuge zur Arbeit mit den Modellen entwickelt werden. Vorhandene UML-Werkzeuge können wiederverwendet werden.

Es wurden die Plattform für Pervasive Games und die Spiel-Domänen als zwei wichtige Abstraktionsebenen erkannt. Deshalb sollten für die Plattform und jede Spiel-Domäne spezielle DSLs entwickelt werden. Mit einer Modelltransformation werden die Spiel-spezifischen Sprachen in die Plattform-Sprache übersetzt.

Als MDSD-Werkzeug wurde für das Projekt das in Eclipse integrierte openArchitectureWare empfohlen. Magic Draw wurde als dazu passendes, UML-Werkzeug empfohlen.

Für den Entwicklungsablauf wurde ein iteratives Vorgehen für wichtig gehalten. Als größtes Risiko für das Projekt wurde die Größe des ersten Iterationszyklus identifiziert. Um dieses Risiko gering zu halten, müssen frühzeitig die Schnittstellen der Plattform festgelegt werden.

Als Ergebnis des Projekts sind fertige DSLs für die Pervasive Game Plattform und eine noch im Projekt festzulegende Spiel-Domäne sowie die nötigen Modelltransformationen vorgesehen.

Abbildungsverzeichnis

2.1 Beispiel-Metamodell (UML-Profil) für eine Schnitzeljagd 5
2.2 Beispiel-UML-Modell einer Schnitzeljagd 5

3.1 Überblick über die Aktivitäten bei modellgetriebener Softwareentwicklung (nach [Stahl u. a. (2007)]) 13
3.2 Erstellung einer Domänenarchitektur (aus [Stahl u. a. (2007)]) 14

Glossar

Domäne

Ein begrenztes fachliches oder technisches Interessen- oder Wissensgebiet. Eine Dömane kann in Subdomänen unterteilt werden.

Domänenarchitektur

Eine Domänenarchitektur besteht aus einem Metamodell für eine bestimmte Domäne, einer Plattform als Laufzeitsystem und dazu passenden Transformationen.

Fehlerwirkung

Die Wirkung eines Fehlerzustands in einer Software, die zur Laufzeit nach außen in Erscheinung tritt. Eine Fehlerwirkung wird als Abweichung eines Programms vom spezifizierten verhalten wahrgenommen (Sollwert $\neq$ Istwert).

Fehlerzustand

Ein Defekt in einer Software (z. B. eine falsch programmierte oder vergessene Anweisung), der die Ursache für eine Fehlerwirkung ist.

Review

Ein Review ist ein Prüfverfahren für Dokumente, das von Personen durchgeführt wird.

Software-Systemfamilie

Die Menge aller Produkte, die mit einer bestimmten Domänenarchitektur entwickelt werden können.

Quellenverzeichnis

Literatur[2]

[Broll u. a. 2006] BROLL, Wolfgang ; OHLENBURG, Jan ; LINDT, Irma ; HERBST, Iris ; BRAUN, Anne-Kathrin: Meeting technology challenges of pervasive augmented reality games. In: *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA : ACM, 2006, S. 28. – ISBN 1-59593-589-4

[Gruhn u. a. 2006] GRUHN, Volker ; PIEPER, Daniel ; RÖTTGERS, Carsten: *MDA – Effektives Software-Engineering mit UML und Eclipse*. Springer-Verlag, Berlin Heidelberg, 2006 (Xpert.press). – URL http://www.springerlink.com/content/w05681/. – ISBN 978-3-540-28746-9

[Hailpern und Tarr 2006] HAILPERN, Brent ; TARR, Peri: Model-driven development: the good, the bad, and the ugly. In: *IBM Syst. J.* 45 (2006), Nr. 3, S. 451–461. – URL http://www.research.ibm.com/journal/sj/453/hailpern.pdf. – ISSN 0018-8670

[Hinske u. a. 2007] HINSKE, Steve ; LAMPE, Matthias ; MAGERKURTH, Carsten ; RÖCKER, Carsten: *Classifying Pervasive Games: On Pervasive Computing and Mixed Reality*. Bd. 1. In: MAGERKURTH, Carsten (Hrsg.) ; RÖCKER, Carsten (Hrsg.): *Concepts and technologies for Pervasive Games - A Reader for Pervasive Gaming Research* Bd. 1. Aachen, Germany : Shaker Verlag, 2007. – URL http://www.vs.inf.ethz.ch/publ/

[Spillner und Linz 2005] SPILLNER, Andreas ; LINZ, Tiulo: *Basiswissen Softwaretest*. 3., überarbeitete und aktualisierte Auflage, korrigierter Nachdruck 2007. dpunkt.verlag GmbH, Heidelberg, 2005. – ISBN 3-89864-358-1

[Stahl u. a. 2007] STAHL, Thomas ; VÖLTER, Markus ; EFFTINGE, Sven ; HAASE, Arno: *Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management*. 2. Auflage. dpunkt.verlag, Stuttgart, 2007. – ISBN 978-3-89864-448-8

[2] Bei Quellen aus dem Internet wurden die URLs und der Inhalt am 15.12.2008 überprüft

[Wittern 2008] WITTERN, Hauke: *Entwicklung eines Assistenten für die modellgetriebene Softwareentwicklung*, Hochschule für Angewandte Wissenschaften (HAW) Hamburg, Bachelorarbeit, 2008

Webseiten[3]

[URL:GMF] FOUNDATION, Eclipse: *Eclipse Graphical Modeling Framework (GMF)*. – URL http://www.eclipse.org/modeling/gmf/

[URL:oAW] OPENARCHITECTUREWARE.ORG: *openArchitectureWare (oAW)*. – URL http://www.openarchitectureware.org

[3] Die Gültigkeit der URLs wurde am 15.12.2008 überprüft