Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit

Daniel Löffelholz

Modellbasiertes Testen mit UTP

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Daniel Löffelholz

Modellbasiertes Testen mit UTP

Seminararbeit eingereicht im Rahmen von Anwendungen 1
im Studiengang Informatik
Studienrichtung Master
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor : Prof. Bettina Buth

Abgegeben am 15. Dezember 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

Dieses Kapitel bietet einen kurzen Überblick über den Inhalt und den Aufbau dieser Arbeit.

1.1. Motivation

Ingenieure begegnen der Herausforderung der hohen Komplexität von Systemen aller Art mit dem Heranziehen von Modellen die den kompletten Entwicklungs- und Testprozess begleiten um Qualität des Systems und auch Qualität des Entwicklungsprozesses zu beurteilen. Diese Vorgehensweise hält durch die modellbasierten Softwareentwicklung (MDA) langsam auch in der Software-Industrie Einzug und kann bereits vorzeigbare Erfolge in Hinsicht Qualität und Entwicklungszeit vorweisen. Qualitativ hochwertige Software benötigt jedoch nicht allein eine effizienten und strukturierten Entwicklungs-, sondern auch einen entsprechenden Testprozess. Das modellbasierte Testen (MDT) als zu der MDA passender Testansatz birgt unterschiedliche Vorgehensweisen die in dieser Arbeit erläutert werden. Zudem wird mit UML Testing Profile (UTP) eine konkrete Technologie um einen MDT-Ansatz umzusetzen vorgestellt.

1.2. Überblick

Kapitel 2 stellt das modellbasiertes Testen als Testansatz für Informationssysteme vor. Das Kapitel gibt einen klassifizierenden Überblick über existierende Ansätze und Methoden des MDT. Es werden in diesem Kontext die Zusammenhänge zwischen Systemmodell, Testmodell, und konkreten Systemen erläutert. In Kapitel 3 wird das typischerweise eingesetzte UML Profil UTP vorgestellt um zu zeigen, wie eine konkrete Modellierungssprache für MDT aussehen kann. An einem beispielhaften Informationssystem wird grob gezeigt wie solche Modelle umgesetzt werden. Kapitel 4 bietet ein kurzes Fazit und schildert wie im Projekt im kommenden Semester die Erkenntnisse dieser Arbeit eingesetzt werden können.

2. Modellbasiertes Testen

In diesem Kapitel werden der Begriff des modellbasierten Testens (MDT) und dessen grundsätzliche Eigenschaften erläutert. Es werden die Vor- und Nachteile diskutiert und die Zusammenhänge zwischen Anforderungen, Modell, und System dargestellt.

2.1. Modellbasiertes Testen

Modellbasiertes Testen (MDT) ist ein Schlagwort, unter dem verschiedenste Techniken zur Nutzung von Modellartefakten im Testprozess verstanden werden. MDT soll helfen die Systematik und somit Effizienz des Testprozesses zu erhöhen um eine Qualitätssteigerung und Kostenreduktion zu erreichen. Obgleich erste modellbasierte Testansätze mit Hilfe von endlichen Automaten in den 60er Jahren zu finden sind, hat sich ein solcher Testansatz bisher noch nicht durchgesetzt. Dies ist wohl zum Großteil der Grundvoraussetzung des MDT geschuldet: Modellierungstechniken für Informationssystem werden erst seit vergleichsweise kurzer Zeit in der industriellen Softwareentwicklung eingesetzt. Dazu haben vor allem die Entwicklung einer standardisierten Modellierungssprache (Unified Modelling Language 2.0), welche viele heterogene Diagrammarten vereint, als auch die Propagierung eines modellbasierten Entwicklungsansatzes (Model Driven Architecture) beigetragen. Diese beiden Technologien enthalten jedoch keine expliziten Testmethoden. Für modellbasiertes Testen sind somit seperate Modellierungstechnologien und Testprozessschritte nötig. (vgl. Schieferdecker (2007))

2.2. Einordnung von modellbasiertem Testen

Modellbasiertes Testen kann sowohl als statisches als auch dynamisches Testverfahren eingesetzt werden.

Statisches Testen
Beim statischen Testen werden Informationen aus der Systemmodell abgeleitet und diese

überprüft. Grundsätzlich können drei Kategorien von Eigenschaften statisch überprüft werden: Stil, Semantik, und Syntax. (vgl. Unhelkar (2005))

Syntax
Syntaktische Korrektheit bedeutet in diesem Zusammenhang Korrektheit im Sinne der Modellierungsvorschrift. Dazu werden Informationen aus dem Metamodell der Modellierungssprache herangezogen und als Testkriterium verwendet. Die meisten Modellierungswerkzeuge unterstützen diese Art von Tests bereits automatisch. Durch eine syntaktische Korrektheit kann gewährleistet werden, dass das Modell stets gleich interpretiert wird.

Semantik
Beim Testen der Semantik kann man zwei Punkte unterscheiden: die Vollständigkeit im semantischen Sinn, sowie die semantische Konsistenz der unterschiedlichen Teilmodelle. Systeme werden oft mit vielen heterogenen Diagrammarten modelliert. Das Testen auf semantische Konsistenz muss hier wiedersprüchliche Aussagen der Teilmodelle aufdecken. Bei der Vollständigkeit des Modells im semantischen Sinn müssen alle Anforderungen an das System entsprechend im Modell umgesetzt sein.

Stil
Ob die Modellierung in einem “guten” Stil vorgenommen wurde lässt sich schwer formal beschreiben. Hier dienen Modellierungsstandards als Richtlinie. Eine Automatisierung der Tests dieser Kategorie ist jedoch kaum möglich.

Dynamisches Testen
Dynamische Testverfahren, also das Testen am laufenden System unterteilen sich in aktive und passive Verfahren. Das aktive Testen ist hier die Ausführung der zu testenden Software mit systematisch festgelegten Eingabedaten (Stimuli). Das passive Testen vergleich die Traces der Systemausführung mit der Spezifikation. Diese beiden Vorgehen ermöglichen unterschiedliche Testaspekte: während das aktive Testen eher das spezifizierte Systemverhalten mit Hilfe von Testfällen prüft, stützt sich das passive Testen auf das Einhalten von Systeminvarianten und Zustandsbedingungen. Die modellbasierten dynamischen Testverfahren sind jedoch in der Praxis oft nur schwer umzusetzen, da die Modelle oft nicht ausführbar sind. Dies liegt an unvollständiger Modellierung, die oft jedoch auf Grund der Komplexität von Informationssystemen nicht zu vermeiden ist. Werkzeuge wie Artisan[1] und Rhapsody[2] erlauben prinzipiell die Stimulation von Modellen. Ein weiteres Beispiel für die Ausführung von Modellen bietet *ActiveCharts* von Prof. Sarstedt (Sarstedt u. a. (2005)). (Schieferdecker (2007))

[1] Artisan ist ein integrierte Entwicklungswerkzeug-Suite welche System- und Softwaremodellierung, speziell für technische Systeme, unterstüzt
[2] Telelogic Rhapsody ist eine MDD-Umgebung für Echtzeit-Systeme

2.3. Zusammenhänge zwischen Test und System

In Abbildung 2.1 werden die typischen Zusammenhänge zwischen Anforderungen, Modell, und System dargestellt.

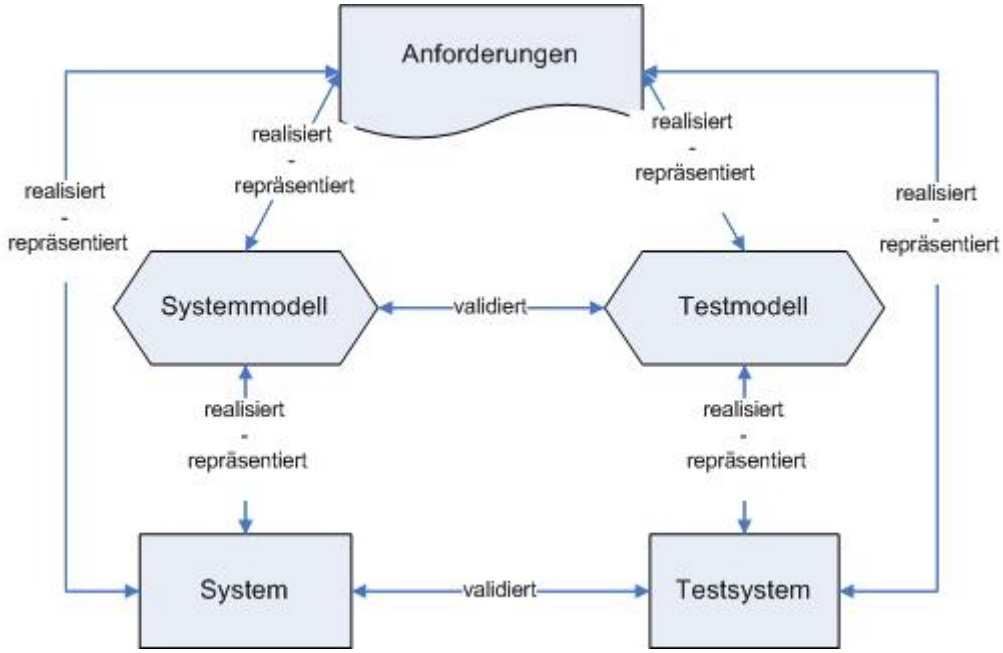

Abbildung 2.1.: Rollen der Modelle

Die Anforderungen sind die Basis für die Entwicklung des Test- und Systemmodells welche somit deren Realisierung darstellen. Beide können gegeneinander validiert werden da sie auf den selben Anforderungen basieren. Auf Basis dieser Modelle werden System und Testsystem erzeugt. Diese beiden System realisieren dann optimaler weise weiterhin die anfangs gestellten Anforderungen und können somit auch gegeneinander validiert werden. Durch diese Eigenschaften ist es möglich lückenlos die Anforderungen aus dem Anforderungsdokument bis zum konkreten System zu verfolgen. Sie ermöglichen zudem die frühzeitigere “top down” Integration der Testvorbereitungsphasen in das Entwicklungsmodell und Validierung des Systemmodells. Baker u. a. (2007)

2.4. Varianten

Es gibt mehrere Varianten des modellbasierten Testens. Sie unterscheiden sich vor allem in der Art wie Anforderungen, Testmodell, Systemmodell, System, und Testsystem zur

gegenseitigen Generierung genutzt werden. Hier gibt es grundsätzlich drei unterschiedliche Ansätze, die sich in dem Punkt unterscheiden, aus welchem oder welchen Modellen die konkreten Systeme generiert werden. (Schieferdecker (2007))

systemmodell-getrieben
Aus dem Systemmodell wird das Testsystem und das konkrete System bzw. Teile davon generiert. Dabei ist es möglich den Zustandsraum des Systems dynamisch zu explorieren, oder aus dem Systemmodell direkt Testcode zu erzeugen. Beispielsweise können Unit-Tests aus Zustandsautomaten generiert werden (vgl Kim u. a. (1999)). Vorteil ist hier, dass nur ein Modell erstellt werden muss welches bereits Artefakt eines modell-getriebenen Entwicklungsprozesses ist. Großer Nachteil ist die Abhängigkeit von System und Testsystem. Im Modell bereits enthaltene Fehler können nicht erkannt werden.

testmodell-getrieben
Analog wird hier das Testmodell herangezogen um das Testsystem und optional das System zu generieren. Für den Entwurf des Testmodells können standardisierte Testmodellierungstechniken herangezogen werden, welche im Laufe der Arbeit noch genauer erläutert werden. Falls man zudem noch das Testmodell zur Generierung des Systems nutzt, hat man erneut das Problem der Abhängigkeit der beiden Systeme.

test- und systemmodell-getrieben
Hierbei werden für die Ableitung von System und Testsystem Modelle herangezogen und der modellbasierte Ansatz wird somit am umfassendsten umgesetzt. Die drei Ausprägungen dieses Vorgehens gehen systemmodell-getrieben, testmodell-getrieben, oder separiert vor. Es findet also eine Modelltransformation entweder aus dem Systemmodell zum Testsystemmodell bzw. vom Testsystemmodell zum Systemmodell statt, oder es werden aus den Anforderungen unabhängige Modelle erstellt. Vorteile der Verfahren ist die Möglichkeit bereits vor der Implementierung der Systeme die Modelle auf Korrektheit und Konsistenz untereinander, sowie Überdeckungsmaße zu überprüfen (siehe 2.2). Die letzt genannte aufwändigste Ausprägung hat natürlich den Vorteil der Unabhängigkeit der beiden Modelle.

2.5. Erzeugung der Modelle

Den von der OMG empfohlenen Systemartefakten der MDA kann man auf jeder Abstraktionsstufe eine passendes Testmodell gegenüber stellen (OMG (2003)). Man realisiert somit einen test- und systemmodell-getriebenen Testansatz (siehe 2.4) parallel zur MDA. Analog zur Abbildung 2.2 stellt man den Systemodellstufen CIM (Computation Independent Model), PIM (Platform Independent Model), PSM (Platform Specific Model), die Testmodellstufen CIT (Computation Independent Tests), PIT (Platform Independent Tests), PST (Platform Specific

Tests) gegenüber. Wie in 2.4 genannt, wird somit schon früh nicht nur mit der Testsystemvorbereitung begonnen, sondern auch mit der Überprüfung der abstrakten Modelle. In 2.2 wurden die Testverfahren vorgestellt deren Anwendung hier möglich ist. Vor allem Inkonsistenzen zwischen den Modellen und unzureichende Anforderungsüberdeckung kann hier aufgedeckt werden. Aus den neu hinzugekommenen Systemartefakten können dann auf jeder Abstraktionsstufe Testmodelle abgeleitet werden. Beispielsweise ist es möglich, aus jeder UML-Diagrammart Tests zu generieren. (siehe Anhang A). Oft muss jedoch dazu das generierte Testmodell mit Testeigenschaften angereichert werden. Mit dem UML Testing Profile (UTP) existiert eine Erweiterung der UML welche dies ermöglicht. Im Kapitel 3 wird diese genauer vorgestellt. Am Ende der Modelltransformationen kann aus dem PSM und dem PST Code generiert werden. Die Vollständigkeit des Codes richtet sich jedoch immer nach der Vollständigkeit des Modells. (Dai (2005), Moll (2004))

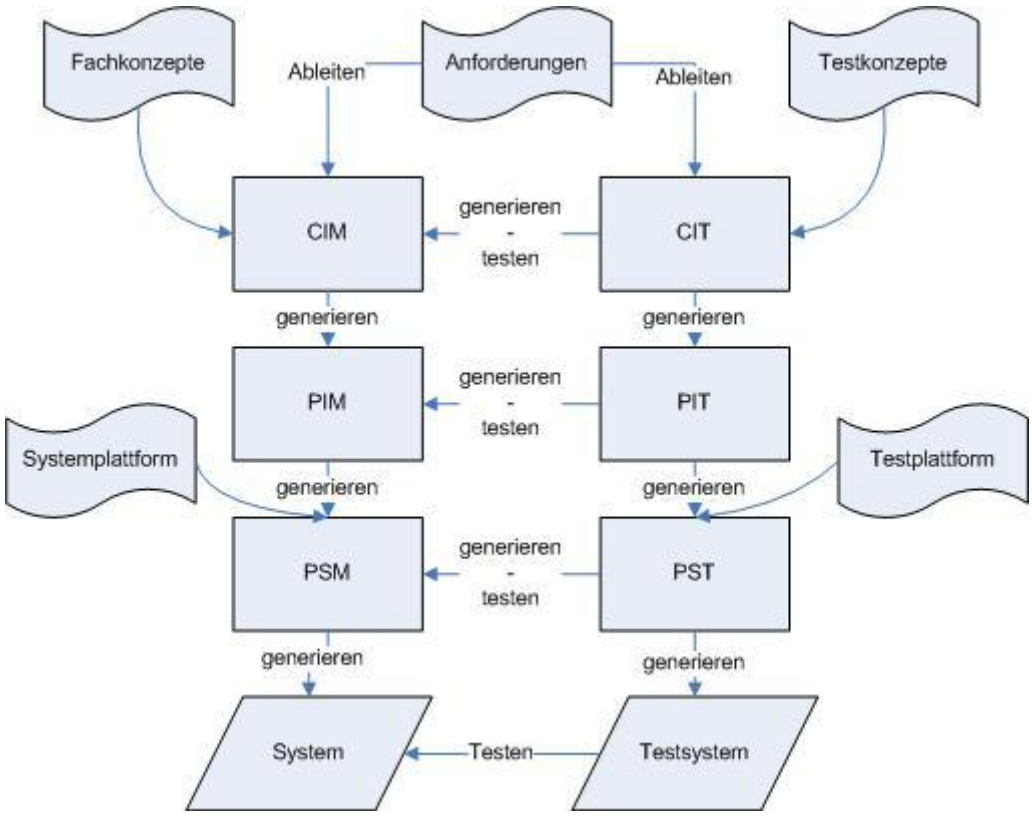

Abbildung 2.2.: Transformation der Modelle

2.6. Einschränkungen des modellbasierten Testens

Auch wenn MDT wie gezeigt viele Vorteile bietet den Testentwurfsprozess effizienter zu machen, hat das MDT ein paar Einschränkungen welche beachtet werden müssen.

- Bei vielen Informationssystemen ist es Aufgrund von Komplexitätsproblemen oftmals kaum möglich das System oder das Testsystem komplett zu modellieren.
- Die aufwändige Testmodellierung erzeugt einen Arbeitsoverhead, der sich nicht für jede Anwendung lohnt.
- Die Testsystemmodellierung erzeugt einen zeitlichen Overhead, der nur aufholbar ist, falls die durch MDT erreichten Automatisierungsmöglichkeiten genutzt werden.
- Für den Entwurf des Testmodells ist ein Testorakel [3] nötig, wessen Erzeugung unter Umständen sehr problematisch sein kann. Falls das Testorakel ein System ist, welches die vom SUT erwartete Antwort auf die entsprechende Eingabe liefert ist wäre das zu entwickelnde System quasi obsolet. Somit sind die meisten Aussagen über das erwartete Systemverhalten sehr wage.
- Nicht jede Modellierungssprache eignet sich für ein modellbasiertes Vorgehen und Testvorgehen.

Prinzipiell sind dies jedoch Probleme die entweder generelle Schwierigkeiten eines modellbasierten Entwurfsansatzes sind (Punkte 1,2,4), oder eine generelle Testproblematik (Punkt 3).

[3] Ein Testorakel ist ein System oder Dokument aus welchem hervor geht, welches Systemverhalten für eine bestimmte Eingabe erwartet wird

3. Das UML Testing Profile

Dieses Kapitel stellt das UML Testing Profile als Modellierungssprache für Testmodelle vor.

3.1. UML Testing Profile

Die Unified Modelling Language (UML) ist die für MDA typischerweise eingesetzte Modellierungssprache. Wie bereits genannt, müssen jedoch die Artefakte der MDA mit Testaspekten angereichert werden um eine systematische Ableitung der Testfälle zu ermöglichen. Es fehlt zudem die Verknüpfung zwischen den Modell und den für die Testdurchführung wichtiger Testtechnologien. Aus diesem Grund startete die Object Management Group (OMG) 2001 einen Request for Papers (RFP) für ein UML Testing Profile (UTP), welches die folgenden Randbedingungen erfüllen sollte: Ein UML Testing Profile, basiert auf dem UML Metamodell oder einem MOF basierten Metamodell für UML Testing , welches die Spezifikation von Tests für strukturelle (statische) oder verhaltens (dynamische) Aspekte von UML Modellen ermöglicht. Zudem sollte es fähig sein mit existierenden Testtechnologien für Black-Box-Testing (wie wie JUnit und TTCN-3 [1]) zu interoperieren. (aus OMG (2008)). 2003 wurde dann das UTP in die UML2-Spezifikation aufgenommen. Weiterhin bietet UTP Mappings für unterschiedliche Testtechnologien.

UML-Profile können als Spezialisierung von UML gesehen werden. Ein Profil erweitert und beschränkt die ursprüngliche Sprache. Mit der UTP-Erweiterung ist es nun möglich die Konzepte Testarchitektur, Testdaten, Testverhalten, und Testzeit abzudecken. (Baker u. a. (2007)) Anhand eines beispielhaften “MobileGaming”-Systems werden in 3.1.1 einige Begriffe und Möglichkeiten von UTP vorgestellt. UTP definiert jedoch noch viele weitere Modellierungsmöglichkeiten beispielsweise für die Testdaten. Wie oben genannt beinhaltet es bereits Mappingprofile für verschiedene Black-Box-Test-Technologien wie TTCN-3 und JUnit. (OMG (2007))

[1] TTCN-3 (Testing and Test Control Notation) ist eine genormte Testprogrammiersprache, die vornehmlich für die Automatisierung von Tests für kommunikations-basierte Systeme verwendet wird

3.1.1. Testarchitektur

Mit Hilfe der Testarchitektur (Abb. 3.1) werden die strukturellen Voraussetzungen für die Ausführung der Tests definiert. Sie enthält das Testprofil für den Zugriff auf vorgefertigte Testkonzepte, zu importierende Schlüsselklassen des *System under Test* (SUT), und zu das zu entwerfende Testpackage. (Baker u. a. (2007))

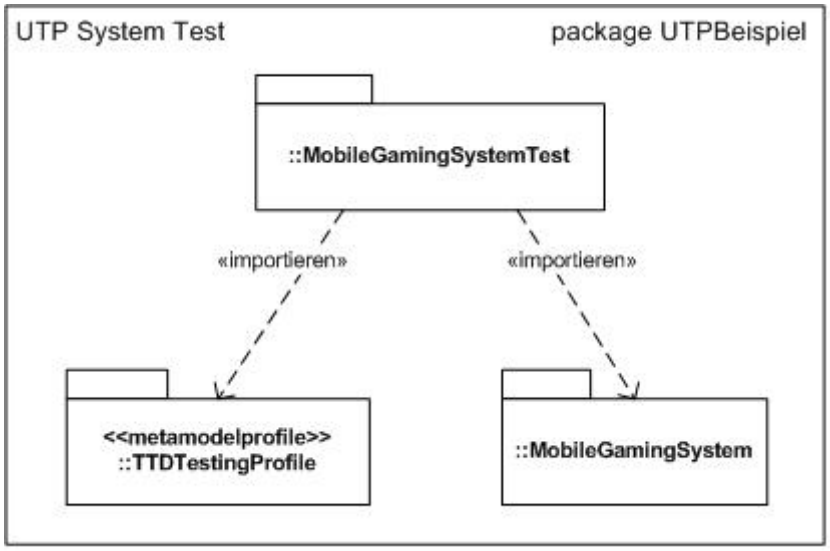

Abbildung 3.1.: Testarchitektur

3.1.2. Testpackage

Das *Testpackage* (siehe Abb. 3.2) beinhaltet nun die Testkonfiguration, welche nun die Komponenten des SUTs mit den Testkomponenten verknüpft sowie den *Testkontext*. Die Testkomponenten realisieren hier das Testverhalten. Der *Testkontext* (Abb. 3.3 erlaubt die Tests zu konfigurieren, eine *Testkontrolle* zu definieren, und Testfälle zu gruppieren. (Baker u. a. (2007))

3.1.3. Testkomponenten

Nachdem die Teststruktur definiert ist, bestimmt das Testverhalten der Testkomponenten die Aktionen und Evaluierungen des Testziels, welches beschreibt was getestet werden soll. Zum Beispiel können UML Interaktionsdiagramme, Zustandsautomaten, Aktivitätsdiagramme, und Sequenzdiagramme benutzt werden um Aktionen, Invocations und Koordination zu

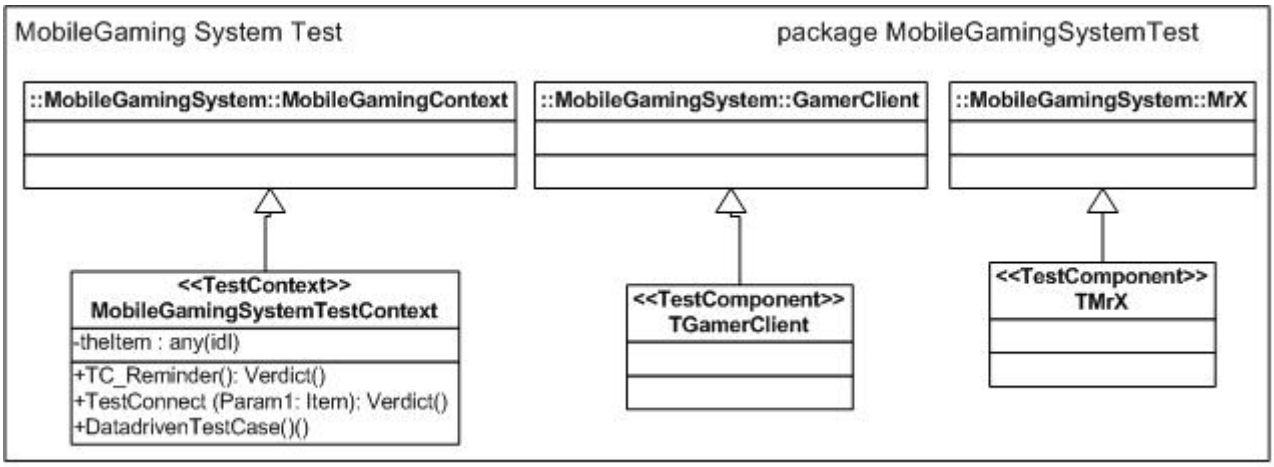

Abbildung 3.2.: Test Package

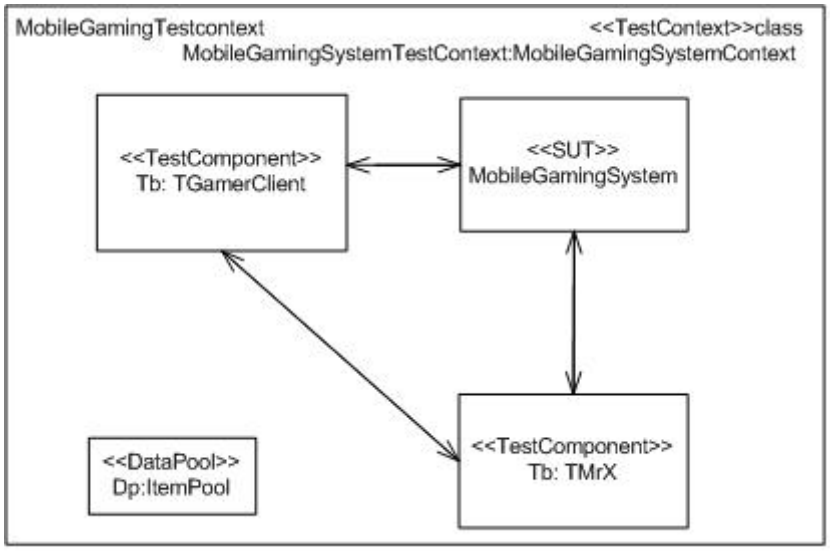

Abbildung 3.3.: Testkontext

testen. Vor allem die Verhaltensdiagramme eignen sich hierfür gut. Es ist jedoch aus möglich aus jedem UML-Diagramm Testfälle zu generieren (siehe Anhang A). Um nun das Testverhalten zu spezifizieren, stellt das Testprofil diverse Konzepte zu Verfügung. (OMG (2007))

- *Test objective* welche die Intention des Tests ausdrücken
- *Test case* sind Operationen, welche beschreiben wie Objekte mit dem SUT interagieren um eine *Test objective* zu erfüllen
- *Defaults* vervollständigen die Verhaltesbeschreibung in dem Situationen beschrieben werden, in denen die beschriebene Sequenz nicht stattfindet
- *Verdict* ist eine vordefinierte Enumeration welche mögliche Testresultate definiert: pass, inconclusive, fail, und error
- Ein erweitertes Timerkonzept

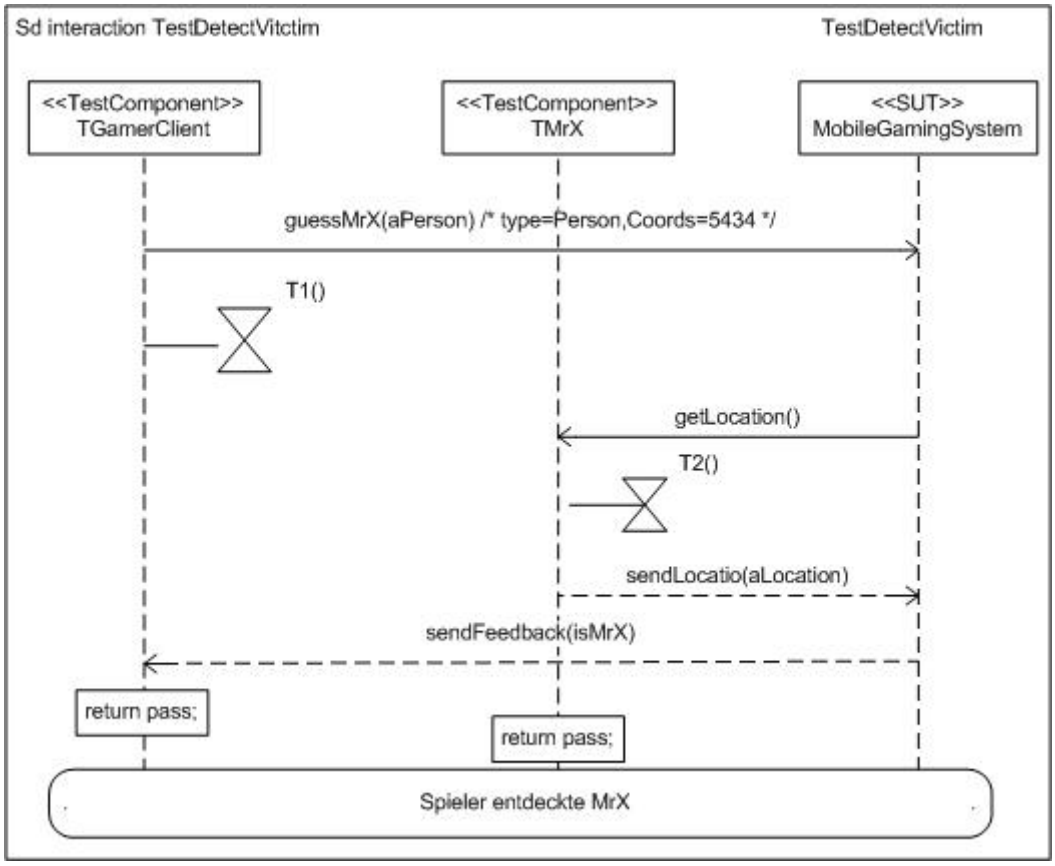

Abbildung 3.4.: Beispiel Aktivitätsdiagrammtest

Der Abbildung zeigt exemplarisch einen Testfall “TestdetectVictim” bei dem die Testkomponenten Funktionen des SUT prüfen. Bei positivem Verlauf geben die Testfälle laut Testprofil Verdict:pass zurück, und das *Test objective* wurde erreicht.

4. Zusammenfassung und Ausblick

Folgend die Arbeit kurz zusammengefasst. Anschließend wird kurz erläutert wie das MDT im kommenden Semester im Projekt eingesetzt werden kann, und welche Punkte dabei noch tiefer gehend zu untersuchen sind.

4.1. Zusammenfassung

Diese Arbeit gibt einen klassifizierenden Überblick über existierende Ansätze und Methoden modellbasierten Testens sowie über das UTP zur Umsetzung dieser Ansätze. Es wurde gezeigt welche Arten von Testverfahren mit einem modellbasierten Testansatz umgesetzt werden können. Zudem wurden die Zusammenhänge zwischen den Modellen, der konkreten System und den Anforderungen deutlich gemacht. MDT kann in unterschiedlichen Varianten umgesetzt werden. Die Kategorisierung dieser Varianten fand unter dem Aspekt statt, aus welchem Modell oder welchen Modellen die konkreten Systeme generiert werden. Anschließend orientiert sich die Arbeit an den von der OMG empfohlenen Systemartefakten für MDA um zu zeigen, wie die Testmodelltransformation und Testsystemmodellierung in einem modellbasiertes Entwicklungsvorgehen aussehen kann. (OMG (2003)) In Kapitel 3 wurde mit dem UTP eine für das Vorgehen passende Modellierungssprache vorgestellt. Beispielhaft wurde einige Konzepte des UTP aufgeführt und kurz erläutert, um dem Leser einen Überblick über die konkrete Umsetzung zu bieten.

In Schieferdecker (2007) werden einige vorrangig wissenschaftlich ausgerichtet Konferenzreihen aufgezählt, die sich weiterführend mit dem Thema beschäftigen. Auch die überschaubare Anzahl an Werkzeugen für ein modellbasiertes Testvorgehen macht deutlich, dass MDT den industriellen Durchbruch noch nicht geschafft hat. Dieser kann jedoch nach Schieferdecker (2007) erwartet werden. Das Engangement von Unternehmen wie DaimlerChrysler, LogicaCMG, und Nokia in Schaffung eines einheitlichen Standards belegen das wirtschaftliche Interesse. TT-Medal-Project (2008)

4.2. Anwendungsbeispiel

Diese Arbeit verdeutlicht die vielfältigen Möglichkeiten modellbasierten Testens. In dem für das nächstes Semester geplante Projekt soll ein Framework für Pervasive-Gaming-Anwendungen modellbasiert entwickelt werden. Dies ist eine gute Grundlage um einen modellbasierten Testansatz zu realisieren. Falls die von der OMG für MDA empfohlenen Systemmodellstufen umgesetzt werden, wird versucht eine systemmodell- und testmodell-getriebener Testansatz umzusetzen. Denkbar wären hier im ersten Schritt eine unabhängige Erstellung eines Testmodells, oder die Ableitung des Testmodells aus dem Systemmodell auf CIM- oder PIM-Ebene. Welches Vorgehen hier die größeren Vorteile bietet ist zu untersuchen. Weiterhin sollen anschließend statische Testmethoden auf Modellebene ausgeführt werden. Insbesondere die Modellsemantik (vgl 2.2), sprich Modellkonsistenz und Vollständigkeit sind hier zu untersuchende Punkte. Nach der Modellierung des Frameworks soll mit modellbasiertem Vorgehen ein konkrete Anwendung entwickelt werden. Für diese Anwendung kann dann ein PST-Modell erstellt werden um schließlich die Testfälle zu generieren. Bevor dies geschieht, muss geprüft werden welches Werkzeug sich dafür anbieten würde. Die Verwendung eines Ecplise-Plugins, welches vom TT-Medal-Zusammenschluss für die für eine UTP zu TTCN-3 Transformation entwickelt wurde, wird dabei geprüft. (Dai (2004)) Ob das Systemmodell zumindest in Teilen ausführbar sein wird, um das Systemverhalten auf Modellebene zu testen ist ebenfalls zu prüfen. Weitere zu untersuchende Punkte wären die Automatisierungsmöglichkeiten für statische Semantiktests auf Modellebene, oder eine Aufstellung von formalen Anforderungen an das Systemmodell damit es ausreichend testbar ist.

Ein Risiko ist die Abhängigkeit vom Systemmodell. Falls dieses nicht ausreichend modelliert ist, ist nur wenig testbar. Zudem sind Werkzeuge zur Testfallgenerierung oder dem Testen von statischen und dynamischen Aspekten auf Modellebene oft kommerziell. Inwiefern Lizenzen an der HAW im nächsten Semester verfügbar sein werden, oder ob man alternative Open-Source-Lösungen findet, kann noch nicht definitiv beantwortet werden.

Literaturverzeichnis

[Baker u. a. 2007] BAKER, Paul ; DAI, Zhen R. ; GRABOWSKI, Jens ; HAUGEN Øystein ; SCHIEFERDECKER, Ina ; WILLIAMS, Clay: *Model-Driven Testing: Using the UML Testing Profile*. 1. Springer, Berlin, 2007. – URL http://dx.doi.org/10.1007/978-3-540-72563-3. – ISBN 3540725628

[Dai 2004] DAI, Z.: *Model-Driven Testing with UML 2.0*. 2004

[Dai 2005] DAI, Z.: *Analysis of the methods and processes for UML test profiles*. TT-Medal Consortium. 2005

[Kim u. a. 1999] KIM, Y.G. ; HONG, H. S. ; CHO, S. M. ; BAE, D. H. ; CHA, S. D.: *Test cases generation from UML state diagrams*. IEE Proc. Software, Vol. 146, No. 4. 1999

[Moll 2004] MOLL, Linard: *Testing-Möglichkeiten für MDD/MDA*. Software Evolution and Architecture Lab Universität Zürich. 2004

[OMG 2003] OMG: *MDA Guide Version 1.0.1*. 2003

[OMG 2007] OMG: *Object Management Group - UML 2.0 testing profile specification*. 2007

[OMG 2008] OMG: *Object Management Group Webseite*. 2008. – http://www.omg.org

[Sarstedt u. a. 2005] SARSTEDT, Stefan ; GESSENHARTER, Dominik ; KOHLMEYER, Jens ; RASCHKE, Alexander ; SCHNEIDERHAN, Matthias: *ActiveChartsIDE - An Integrated Software Development Environment comprising a Component for Simulating UML 2 Activity Charts*. The European Simulation and Modelling Conference (ESM05), pages 66-73. 2005

[Schieferdecker 2007] SCHIEFERDECKER, Ina: *Modellbasiertes Testen*. OBJEKT spektrum 03/2007. 2007

[TT-Medal-Project 2008] TT-MEDAL-PROJECT: *Test and Testing Methologies For Advanced Languages Consortium*. 2008. – http://www.tt-medal.org

[Unhelkar 2005] UNHELKAR, B.: *Verification and Validation for Quality of UML 2.0 Models*. 2005

A. Übersicht über die UML-Diagramme

...-diagramm der UML	Teststufe				Testart		Testverfahren	
	Unit	Komponente	Integration	System	funktional	nicht-funktional	statisch	dynamisch
Struktur								
Klassen- *(class)*	x	x	x	x	x		x	x
Kompositions-struktur- *(composite structure)*		x	x		x		x	x
Komponenten- *(component)*		x	x	x	x		x	x
Verteilungs- *(deployment)*			x	x	x		x	
Objekt *(object)*		x	x		x		x	x
Paket- *(package)*			x		x		x	
Verhalten								
Anwendungsfall- *(use case)*			x	x	x	x[1)]	x	x
Aktivitäts- *(actitiy)*			x	x	x	x[1)]	x	x
Sequenz- *(sequence)*			x	x	x	x[1)]	x	x
Kommunikations- *(communication)*		x	x		x	x[1)]	x	x
Interaktions-übersichts- *(interaction overview)*			x	x	x	x[1)]	x	x
Zeitverlaufs- *(timing)*		x	x			x[1)]	x	x
Zustands- *(state machine)*	x	x	x	x	x	x[1)]	x	x

Abbildung A.1.: Übersicht über die UML-Diagramme nach Schieferdecker (2007)

1) Oftmals unter Nutzung von UML-Profilen für nicht-funktionale Konzepte, z.B. dem SPT-Profil