Lehrstuhl für Softwaretechnik
Prof. Dr. Stephan Diehl
Universität Trier
Fachbereich IV - Informatik

Diplomarbeit

Collaborative Requirements Engineering with Wiimotes (CREWW)

Kollaborative Erstellung von Anforderungsanalysen

vorgelegt von
Felix Bott
Matrikelnummer: 701087

Trier im Dezember 2009

Erklärung zur Diplomarbeit

Hiermit erkläre ich, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Diplomarbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Felix Bott
Trier, den 21. Dezember 2009

Inhaltsverzeichnis

Glossar

Bezeichnung	Beschreibung	
API	Application Programming Interface	24
CASE	Computer Aided Software Engineering	37
CRC	Classes – Responsibilities – Collaborators	4
CSCW	Computer Supported Collaborative Work – Forschungsgebiet, das sich mit computergestützter kollaborativer Arbeit befasst	17
DLL	Dynamically Linked Library – Dynamisch gebundene Bibliothek	24
Groupware	Software, die kollaboratives Arbeiten unterstützt	17
GUI	Graphical User Interface – Grafische Benutzeroberfläche	20
HID	Human Interface Device – Eingabegerät	23
ICR	Intelligent Character Recognition – Die automatische Erkennung von Hand-Blockschrift auf Basis von Rastergrafiken	18
IEEE	Institute of Electrical and Electronical Engineers – Weltweiter Berufsverband aus den Bereichen Elektrotechnik und Informatik	8

Bezeichnung	**Beschreibung**	
JNI	Java Native Interface – Technik zur Einbindung von dynamischen Bibliotheken in Java	24
JVM	Java Virtual Machine – Java-Laufzeitumgebung	58
OCR	Optical Character Recognition – Die automatische Erkennung von Maschinenschrift auf Basis von Rastergrafiken	18
SDK	Software Development Kit	21
TWAIN	Standard zum Datenaustausch zwischen Bildeingabegeräten und PCs (auch: Technology without an interesting Name)	39
UDP	User Datagram Protocol – Verbindungsloses Netzwerkprotokoll, das die Transportschicht der IP-Protokollfamilie darstellt	25
UML	Unified Modeling Language	8
VC60	Microsoft Visual C Compiler 6.0	55
VC70	Microsoft Visual C Compiler 7.0	55
XMI	XML Metadata Interchange Format	14
XML	Extendable Markup Language	14

Kapitel 1

Einleitung

Zur Einführung in das zunächst wenig griffige Themengebiet der Anforderungsanalyse soll an dieser Stelle mit einer kleinen Geschichte begonnen werden:

Fünf Pilger wandern durch die Steppe. Dort treffen sie auf eine hohe Mauer, an der sie einige Tage entlang laufen. Am dritten Tag kommen sie an ein Tor. Das Tor ist mit einem schweren Schloss versehen und daher überlegen sie, was sie machen können. Vier von ihnen beginnen das Schloss zu bearbeiten. Sie versuchen, es zu knacken oder zu zerstören, aber ohne Erfolg. Der Fünfte Pilger sieht sich das einige Stunden an, geht in ein nahe gelegenes Wäldchen und kommt mit einer langen Stange zurück. Er nimmt Anlauf, springt mit der Stange über das Tor und setzt seinen Weg fort.

Was die Geschichte zum Ausdruck bringen soll, ist Folgendes: Die Arbeit der ersten vier Pilger war nicht etwa erfolglos, weil sie sich nicht genug angestrengt haben oder ihnen nicht die richtigen Hilfsmittel zur Verfügung standen. Auch mit dem langen Stock hätten sie das Schloss nicht entfernen können. Ihr Fehler war, dass sie das Problem falsch definiert haben. Sie versuchten sofort, das Schloss zu öffnen, dabei bestand das eigentliche Problem nicht im Öffnen des Schlosses, sondern in der Überwindung des Tors.

Überträgt man das Beispiel ins Allgemeine, so lässt sich feststellen, dass die Lösung eines Problems *immer* eine konkrete Beschreibung des Problems voraussetzt. Versäumt man es, sich die Zeit zu nehmen, das Problem zu analysieren und zu beschreiben, kann das Finden einer Lösung erschwert, im ungünstigen Falle gar unmöglich gemacht werden.

> *„Das Problem erkennen ist wichtiger als die Lösung finden, denn die genaue Darstellung des Problems führt fast automatisch zur richtigen Lösung “*
> ——(Alberst Einstein)

Diese Feststellung lässt sich auf viele allgemeine und technische Bereiche übertragen, so auch auf die Softwaretechnik, die Entwicklung von Software nach ingenieurwissenschaftlichen Verfahrensweisen. Den Vorgang des „Erkennens“ des Problems bezeichnet man hier als Anforderungsanalyse.

Die Softwaretechnik kennt verschiedene Methoden zur Durchführung solcher Anforderungsanalysen, eine davon, die sogenannte CRC-Methode[1], wird in der vorliegenden Arbeit vorgestellt. Es wird beschrieben, wie mit dieser Methode, nur durch Einsatz von Papier und Bleistift, in Gruppenarbeit Probleme analysiert und definiert werden können. Große Stärke dieser Methode ist neben der objektorientierten Vorgehensweise die Integration der verschiedenen am Softwareentwurf beteiligten Interessengruppen in dieses frühe Stadium der Entwicklung, doch der Low-Tech-Ansatz hat auch Schwächen.

Gerade hier setzt die vorliegende Diplomarbeit an: Eine Software zur Durchführung besagter Methode mittels Computerunterstützung wird entwickelt und vorgestellt, was insofern ein gewagtes Unterfangen darstellt, als deren Erfinder Technikeinsatz eigentlich von vornherein untersagt haben. Somit tun sich an dieser Stelle direkt ganz unterschiedliche Problemfelder auf:
Wie kann eine Benutzerschnittstelle für eine kollaborative Software aussehen?
Wie funktioniert eigentlich Gruppenarbeit und welche psychologischen Prozesse müssen beachtet werden?
Wie lassen sich die Nachteile der traditionellen CRC-Methode bei Beibehaltung der Vorteile verbessern?
Und nicht zuletzt: Wie kann man überhaupt einen PC mit mehreren Benutzern kollaborativ ansteuern, welche technischen Möglichkeiten gibt es?

All diese Probleme mussten durch die implementierte Software CREWW gelöst werden, und im Verlauf der vorliegenden Arbeit wird begründet, warum dieser Weg gewählt wurde und wie die scheinbaren Unvereinbarkeiten von Methode und Technikeinsatz durch die Implementierung eines Tools, das *echtes kollaboratives Arbeiten* am PC ermöglicht, aufgelöst wurden.

Abschließend wird die ebenfalls im Rahmen dieser Diplomarbeit durchgeführte Benutzerstudie vorgestellt, in der das Erreichen der gesteckten Ziele durch Konfrontation mit der harten End-User-Realität abgeprüft wurde.

[1] CRC – Classes, Responsibilities, Collaborators

Kapitel 2

Grundlagen

Wie sich im Verlauf der vorliegenden Arbeit zeigen wird, vereint CREWW sowohl hardwareseitig als auch konzeptionell Techniken aus den verschiedensten Bereichen – hier seien z. B. Softwaredesign, optische Zeichenerkennung und Bluetooth-Drahtlosdatenübertragung genannt. Um dem Leser ein Grundverständnis der benutzten Konzepte zu vermitteln, werden in diesem Kapitel die theoretischen Grundlagen zu CREWW vorgestellt.

Zunächst erfolgen die Betrachtungen zu den eher konzeptionellen Bereichen Softwareentwicklung und kollaboratives Arbeiten. In den darauf folgenden Unterkapiteln werden die technischen Bereiche Handschrifterkennung und Wii-Anbindung erläutert.

2.1 Objektorientierte Analyse und Design

Betrachtet man ein Ranking aktuell genutzter Programmiersprachen [TIO], so lässt sich feststellen, dass objektorientierte Programmiersprachen acht der zehn ersten Plätze belegen. Somit lässt sich sagen, dass Objektorientierung für *das* Programmierparadigma der heutigen Zeit steht. Dies gilt nicht nur für die Implementierung, sondern für den gesamten Softwarelebenszyklus, somit auch für Analyse und Design.

Dieses Kapitel bietet zunächst einen kurzen Abriss der Geschichte des Softwaredesigns, mit der Entwicklung vom imperativen zum objektorientierten Modell, um dann auf drei Aspekte des objektorientierten Designs näher einzugehen. Dies wird zum Ersten die CRC-Methode sein, eine Methode, die zwischen Analyse- und Designphase angesiedelt ist und das Herzstück von CREWW darstellt. Zum Zweiten wird die *Unified Modeling Language* (UML) vorgestellt, die mit ihren verschiedenen Diagrammtypen den aktuellen Standard für die Notation objektorientierter Designs darstellt. Als Drittes wird die Meta-Modellierungssprache XMI eingeführt,

mit der UML-Modelle beschrieben werden können und die als Exportformat für CREWW gewählt wurde.

2.1.1 Softwareentwicklung im Laufe der Zeit

> „*Einen Spaten zum Blumenumtopfen einzusetzen ist so fehl am Platz wie ein Teelöffel zum Ausheben einer Baugrube. Auf das richtige Werkzeug und die richtige Methode kommt es an.*“ [Oes04]

So schreibt Bernd Oestereich in seinem Buch „Objektorientierte Softwareentwicklung“ von 2004. Diese Einsicht, die so klar zu sein scheint, hat in dem Bereich der Softwareentwicklung lange auf sich warten lassen. Während die Grundlagen zu den ersten objektorientierten Programmiersprachen bereits in den 1970er Jahren am *Xerox Palo Alto Research Center* mit Smalltalk-71 bis Smalltalk-80 gelegt wurden, erschienen die ersten Publikationen zu objektorientierter Analyse und Design ca. 15 Jahre danach in den späten 1980er Jahren (siehe [WBW89, BC89, CY91a, CY91b]).

Geht man vom einfachen Wasserfallmodell aus, in dem die einzelnen Phasen der Softwareentwicklung iterativ durchlaufen werden [Roy87], besteht der Softwarelebenszyklus, stark vereinfacht, aus fünf Phasen: *Anforderungsanalyse*[1], *Design*, *Implementierung*, *Test* und *Betrieb & Wartung*. Es folgt eine kurze Vorstellung der einzelnen Phasen, wobei der Fokus hier auf den ersten beiden liegen soll, da die in der vorliegenden Arbeit zentrale CRC-Methode ein Bindeglied zwischen diesen Phasen darstellt.

Anforderungsanalyse: Ziel der Anforderungsanalyse ist die Erstellung einer Anforderungsspezifikation, auch Pflichtenheft genannt. Diese muss das System hinsichtlich seiner Funktionalität abgrenzen, also sowohl beschreiben, was das Programm leisten soll, als auch explizit abgrenzen, was es *nicht* leisten soll. Diese Beschreibung muss bestimmten Qualitätskriterien genügen, die da wären: Vollständigkeit, Korrektheit, Konsistenz, Prüfbarkeit, Eindeutigkeit, Verfolgbarkeit und bestehende Bewertung [IEE98]. Pohl et al. unterteilen die Anforderungen außerdem noch bezüglich ihrer *Art* und ihrer *Stärke* [PR09]:

[1] Die Begriffe Anforderungsanalyse, Anforderungserhebung und Requirements Engineering werden im folgenden synonym benutzt. Laut IEEE ist die Anforderungsanalyse zwar nur ein Teil der Anforderungserhebung, allerdings kann diese Ungenauigkeit in Hinsicht auf den Fokus der vorliegenden Arbeit toleriert werden.

- Art
  - Funktionale Anforderung: Definiert eine vom System gewünschte Funktionalität
  - Qualitätsanforderung: Gewünschte Qualitätskriterien bezüglich Performance, Skalierbarkeit, etc.
  - Randbedingungen: Bedingungen, die den Entwickler einschränken, wie Fertigstellungstermin, Zielsystem oder zu benutzende Programmiersprache
- Stärke
  - Muss-Anforderung: Anforderung ist zwingend zu erfüllen
  - Soll-Anforderung: Anforderung ist erwünscht, aber nicht zwingend erforderlich
  - Kann-Anforderung: Anforderung ist optional zu erfüllen

Ein typisches Vorgehen bei der Erhebung der Anforderungen ist die Identifikation von Use-Cases, wobei ein Use-Case ein konkreter Anwendungsfall des zu implementierenden Systems ist. Durch systematisches Durchspielen dieser Use-Cases schließlich lassen sich die Systemanforderungen identifizieren.

Es sollte noch erwähnt werden, dass nach Boehm 60 % von Fehlern in der Softwareentwicklung von Analysefehlern ausgehen und nur 40 % von Design- und Implementierungsfehlern [Boe81]. Daraus folgt, dass die Phase der Anforderungsanalyse richtungsweisend für den Verlauf und auch den Erfolg des gesamten Projektes ist.

Design: In der Designphase werden die Anforderungsspezifikationen in einen Rohentwurf umgewandelt. Während die Designphase zu Zeiten prozeduraler Programmierung eher kurz gehalten und ein früher Beginn der Implementierungsphase angestrebt wurde [WBWW90], hat sie mit dem Paradigmenwechsel zur Objektorientierung hin einen erheblichen Bedeutungszuwachs erhalten:

> „*Object-oriented design aims for robust software, that can easily be reused, refined, tested, and maintained...*“ [WBWW90]

Objektorientiertes Design zielt auf die Identifikation von Objekten mit genau definierten Zuständigkeiten ab. Die identifizierten Objekte können in einem nächsten Schritt eingeordnet und Klassen herausgearbeitet werden – jedes Objekt versteht sich hier als Instanz einer Klasse. Auf der Menge der ermittelten Klassen kann nun eine Hierarchie definiert und Klassen zu Subsystemen oder Paketen zusammengeschlossen werden.

Eine Notationsmöglichkeit für die Schritte und Ergebnisse der Designphase bieten die verschiedenen Diagrammtypen der in Kapitel 2.1.3 vorgestellten Modellierungssprache UML.

Implementierung: In der Implementierungsphase wird das in der Designphase entwickelte strukturelle Modell in ein lauffähiges Programm umgewandelt. Bei der prozeduralen Programmierung ist die Implementierungsphase der Kern der Softwareentwicklung, in der objektorientierten nimmt ihre Bedeutung ab. Genauer gesagt zeigt sich in der Implementierungsphase, wie gewissenhaft die Phasen Anforderungsanalyse und Design abgearbeitet wurden: Die Auswirkungen einer lückenhaften Anforderungsspezifikation oder eines ungenauen Designs manifestieren sich meist erst im Verlauf der Implementierungsphase.

Test: In der Testphase wird das Programm, bzw. werden die einzelnen Programmteile, getestet. Mit dem hierfür nötigen Aufwand verhält es sich ähnlich wie mit dem für die Implementierungsphase nötigen: Sind die ersten beiden Phasen gewissenhaft durchlaufen worden, wird sich herausstellen, dass die Testphase mit weniger Problemen absolviert werden kann. Denn wenn die Programmteile sinnvoll in autarke Module unterteilt sind, lassen sich diese besser getrennt testen und die Ursachen von Fehlern lassen sich leichter identifizieren.

Betrieb & Wartung: Der Softwarelebenszyklus endet nicht mit der Auslieferung des Produktes. Die Wartung umfasst sowohl die Korrektur auftretenden Fehlverhaltens der Software als auch die Integration von Erweiterungen. Tatsächlich ist diese Phase die längste, für gewöhnlich wird ein fertiges Softwareprodukt jahrelang genutzt und entsprechend lang laufen auch die Wartungsverträge.

Auch diese Phase profitiert vom Einsatz objektorientierter Vorgehensweise: Ein gutes objektorientiertes Design ermöglicht den Austausch von Komponenten und die Integration neuer Schnittstellen bei Bedarf.

2.1.2 CRC-Karten

Nachdem im vorangegangenen Kapitel die Phasen der Softwareentwicklung im Allgemeinen angesprochen wurden, folgen nun Erläuterungen zu CRC-Karten und eine Einordnung der Methode in ebendiese Phasen.

CRC-Karten: Erstmalig wurden CRC-Karten von Kent Beck und Ward Cunningham auf der International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA) 1989 vorgestellt. Während das ursprüngliche Anwendungsgebiet der Karten die Lehre objektorientierten Designs

und Denkens war [BC89], zeigte sich in den nachfolgenden Jahren, dass sie ebenfalls für das Requirements Engineering genutzt werden können (siehe [BS97, Wil99]). Methodisch sind CRC-Karten in den Übergang von der Analyse- zur Designphase einzuordnen. Während die klassische Anforderungsspezifikation ein Dokument in Fließtext ist, ermöglicht die Anforderungsanalyse mithilfe von CRC-Karten die Erstellung eines objektbezogenen Anforderungsmodells. Dies erleichtert den Designschritt, da bereits Kandidatenklassen für das Programmdesign existieren.

CRC-Karten modellieren Klassen sowie deren Verantwortlichkeiten und Abhängigkeiten. Eine CRC-Karte hat in etwa die Größe 10*15 cm und auf Ihrer Vorderseite befinden sich mindestens drei Felder: Im ersten Feld (Classname), der ersten Zeile der Karte, befindet sich der Klassenname. Im zweiten Feld (Responsibilities), das die linke Hälfte der Karte einnimmt, befinden sich die Verantwortlichkeiten der Klasse. Genauer umfasst dies die Aufgaben der Klasse sowie die Kenntnisse, die sie benötigt, um ihre Aufgaben zu erfüllen. Für den Fall, dass die Klasse im weiteren Verlauf der Softwareentwicklung beibehalten wird, bilden die Verantwortlichkeiten die späteren Methoden und Variablen ab. Auf der rechten Hälfte der Karte (Collaborators) werden alle Klassen notiert, deren „Hilfe“ die aktuelle Klasse benötigt, um ihre Verantwortlichkeiten zu erfüllen. Dies sind z. B. solche Klassen, an die die aktuelle Klasse Aufgaben delegiert.

Diese Felder muss jede CRC-Karte enthalten, wobei weitere optionale Felder, z. B. für Superklassen und abgeleitete Klassen [BS97], möglich sind. Weiterhin kann die Klasse auf der Rückseite der Karte verbal beschrieben werden. Tatsächlich lässt sich feststellen, dass für CRC-Karten kein einheitliches Layout existiert, eher ist es so, dass die Karten dem jeweiligem Anwendungsgebiet angepasst werden. Die von CREWW genutzten CRC-Karten beschränken sich auf die drei notwendigen Felder, da somit ein kompaktes Design der Karten möglich ist. Die Rückseite bleibt leer, so dass die Karten in einem Schritt eingescannt werden können (siehe Abb. 2.1).

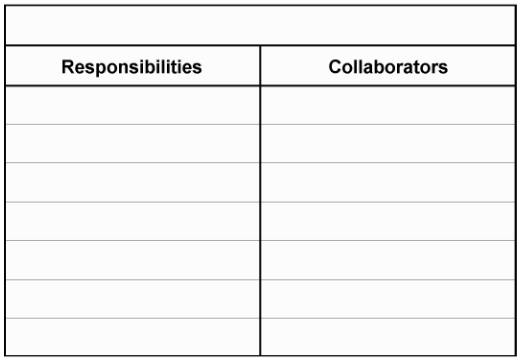


Abbildung 2.1: Von CREWW genutztes CRC-Format

Bezug zur Softwareentwicklung: Eine genaue Beschreibung des Ablaufs einer CRC-Arbeitssitzung liefert Kap. 3, an dieser Stelle erfolgt zunächst nur eine Einordnung der CRC-Methode in die klassischen Phasen der Softwareentwicklung.

Die CRC-Methode kann an verschiedenen Stellen der Analysephase einsetzen: Entweder existiert schon ein grober Anforderungsentwurf in Textform, aus dem die Kandidatenklassen abgeleitet werden können. Es ist aber auch möglich, und gerade hier liegt die Stärke der CRC-Methode, die komplette Analysephase in einer Teamsitzung durchzuführen. Diese Sitzung wird dann meist in Zusammenarbeit mit dem Kunden, der die Software in Auftrag gibt, durchgeführt.

Das Ziel der CRC-Methode geht allerdings insofern über die reine Anforderungsanalyse hinaus, als mit der CRC-Methode bereits ein erster Modellentwurf entwickelt wird. Somit lässt sich sagen, dass die CRC-Methode sowohl Analyse- als auch Designelemente enthält, was sie zu einem Bindeglied zwischen diesen beiden Phasen macht.

2.1.3 UML-Modellierung

Mit der Einführung objektorientierter Methoden im Softwaredesign wurde schnell ein Missstand deutlich: Das Fehlen einer einheitlichen Notation für objektorientierte Designs. Die allgemeinsprachliche Beschreibung stößt in diesem Bereich schnell an ihre Grenzen, so werden unterschiedliche Konzepte mit gleichen Begrifflichkeiten besetzt und ungleiche Begrifflichkeiten mit demselben Konzept verbunden. Da Softwareentwicklung heutzutage praktisch nur noch im Team stattfindet, können Mehrdeutigkeiten und Ungenauigkeiten so zu Fehlern führen, die erst im späteren Verlauf der Softwareentwicklung zu Tage treten und dann nur mit hohem Aufwand zu beheben sind.

Zur Behebung dieses Missstandes stellt die Unified Modeling Language (UML) verschiedene Diagrammtypen zur Verfügung, die eine einheitliche Notation sowohl der Struktur als auch des Verhaltens von Software ermöglichen. Entwickelt wurden die UML und die dazugehörenden Methoden für objektorientiertes Design Mitte der neunziger Jahre von den später als „*die Amigos*“ bekannt gewordenen Entwicklern Grady Booch, Ivar Jacobson und James Rumbaugh. Zum Quasi-Standard wurde die UML schließlich mit der Akzeptierung der Version 1.1 durch die Object Management Group (OMG) im Jahre 1997. Bis heute wurde sie ständig erweitert, der aktuelle und hier vorgestellte Stand ist Version 2.2 vom Februar 2009 [UML]. Diese umfasst neben den erwähnten Diagrammtypen (Infrastructure Specification) auch ein Metamodell (Superstructure Specification).

Die UML umfasst insgesamt 15 Diagrammtypen, neun davon beschreiben die Struktur und sechs das Verhalten von Programmen. Bei den Strukturdiagrammen steht die Architektur im Vordergrund: Mit diesen können unter anderem Klassen, Komponenten und Subsysteme sowie deren gegenseitige Abhängigkeiten beschrie-

ben werden. Die Verhaltensdiagramme dagegen richten den Fokus auf Interaktionen, zeitliche Abläufe sowie Zustände und deren Änderungsverhalten. Etwas genauer sollen nun zwei Diagrammtypen beschrieben werden, die sowohl für die CRC-Methode als auch für CREWW von Bedeutung sind: Dies sind die zu den Strukturdiagrammen gehörenden Diagrammtypen *Klassendiagramm* und *Anwendungsfalldiagramm*. Für einen tieferen Einstieg in die UML sei hier auf [Oes04] verwiesen.

Klassendiagramm: Das Klassendiagramm stellt den meistgenutzten Diagrammtyp und somit den Kern der UML dar. Es beschreibt Klassen, deren Attribute, Operationen und Eigenschaften sowie Abhängigkeiten zwischen den Klassen.

In Abb. 2.2 ist eine UML-Klasse dargestellt. Die Attribute beschreiben spätere Membervariablen und die Operationen Methoden, womit das Klassendiagramm sehr implementierungsnah ist. Gängige UML-Tools stellen meist eine Code-Generierungs-Funktion zur Verfügung, die aus einer UML-Klasse fertigen Code – mit Methoden- und Klassenköpfen – erzeugt. Für die Modellierung von Klassenabhängigkeiten stehen in CREWW als wichtigste die Beziehungstypen Assoziation, Vererbung und Aggregation zur Verfügung. Weitere Arten von Abhängigkeiten können in UML modelliert werden, stehen jedoch in CREWW (noch) nicht zur Verfügung. Für die grundlegende UML-Modellierung sind diese drei Typen ausreichend, somit ist deren Auswahl ein guter trade-off zwischen Implementierungsaufwand und Usability. Es ist jedoch angestrebt, weitere Beziehungstypen in CREWW zu integrieren.

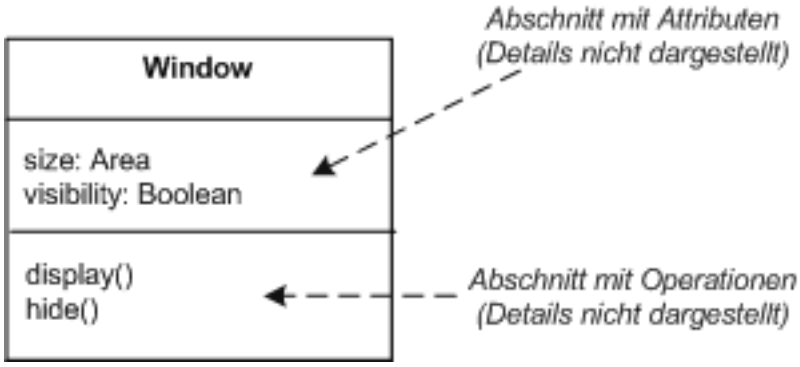


Abbildung 2.2: UML-Klasse

Anwendungsfalldiagramm: Das Anwendungsfalldiagramm (engl. *Use Case Diagram*) beschreibt die konkreten Anwendungsfälle, die in einem System oder einer Systemkomponente auftreten können und die daran beteiligten Akteure. Im

Granulat unterscheidet man hier Geschäftsanwendungsfälle (business use cases), die auf Geschäftsebene ablaufen – unabhängig von jeglicher Umsetzung durch das System – von Systemanwendungsfällen (system use cases), die konkrete Systemanfragen darstellen. Ein Geschäftsanwendungsfall zerfällt hier meist in mehrere Systemanwendungsfälle. Ein Anwendungsfall seinerseits kann verschiedene Ausprägungen annehmen, diese werden dann als *Szenarien* bezeichnet.

Die Identifizierung von Anwendungsfällen findet in einer frühen Phase der Anforderungsanalyse statt. Mit Anwendungsfällen lassen sich Kandidatenklassen und –verantwortlichkeiten identifizieren. Sollte bereits ein Systementwurf bestehen, kann dieser durch das Durchspielen von Anwendungsfällen auf Fehler geprüft werden (zum Durchspielen der Anwendungsfälle mehr in Kap. 3). Ein Anwendungsfalldiagramm besteht aus einem System, den beteiligten Akteuren und den Anwendungsfällen (siehe Abb. 2.3). Das System und seine Grenzen wird durch ein Rechteck, die Anwendungsfälle durch Ellipsen innerhalb und die Akteure durch „Strichmännchen“ außerhalb des Systems dargestellt.

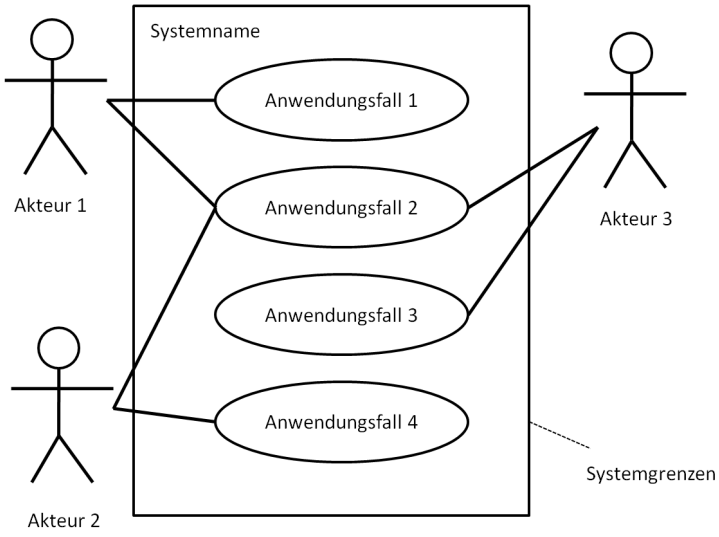


Abbildung 2.3: Anwendungsfalldiagramm

2.1.4 Exkurs: XMI als Meta-Modellierungs-Sprache

Wie zu einem späteren Zeitpunkt (siehe Kap. 4) noch herausgearbeitet werden wird, ist eines der Ziele von CREWW die Verbindung der beiden Techniken CRC und UML. Hieraus ergibt sich die Problematik eines Austauschformats, da es nötig ist, die Daten einer CRC-Karte in ein UML-Modell zu übertragen. Gängige UML-

Tools (FuJaba, ArgoUML, UMLed) bieten zwar proprietäre Speicherformate an, diesen mangelt es aber an Transparenz und Portabilität. Daher fiel die Wahl des Austauschformats auf das XML Metadata Interchange Format (XMI).

XMI ist ein Standard zur Beschreibung von Meta-Metamodellen in XML-Format und somit geeignet zum Austausch von beliebigen objektbezogenen Daten. Weiterhin genügt XMI der Meta Object Facility (MOF), einer ebenenbezogenen Metadatenarchtitektur. Diese unterteilt Daten in vier Ebenen:

- **Ebene 3 :** allgemeines Meta-Metamodell (MOF-Modell)
- **Ebene 2 :** Metamodell des Ebene-1-Modells (besteht aus Meta-Metadaten, z. B. UML2.2 Superstructure)
- **Ebene 1 :** jedes Modell der Ebene-0-Daten, sei es in UML oder einer anderen Modellierungssprache (bestehend aus Metadaten)
- **Ebene 0 :** beschreibt die konkreten Daten

Die obige Einteilung führt zu folgenden Schlüssen: Das Objekt einer Ebene ist immer die Instanz eines Modells der übergeordneten Ebene. Im Umkehrschluss ist ein Objekt aber auch Modell für Daten der untergeordneten Ebene. Im konkreten Fall wird ein Austauschformat für UML-Modelle benötigt, folglich das Modell der Ebene 2 für UML (UML2.2 Superstructure). Die aktuelle Version von XMI nach der OMG ist 2.1.1, da die o. g. Tools dieses jedoch nicht unterstützen, fiel die Wahl auf die XMI-Spezifikation 1.2 [XMI]. Diese wird zumindest von „ArgoUML“ [Arg], einem verbreiteten Freeware-UML-Tool, unterstützt.

Eine Klasse wird in XMI 1.2 durch das Tag `UML:Class` modelliert, welches wiederum eine beliebige Anzahl an Element-Tags `UML:Operation` aggregiert.Die Beziehungen werden durch die Tags `UML:Association` und `UML:Generalization` repräsentiert, wobei die Aggregation eine Spezialisierung von `UML:Association` darstellt. Auf weitere Details und Unter-Tags soll hier nicht eingegangen werden, zur genaueren Spezifikation des Austauschformats siehe [XMI].

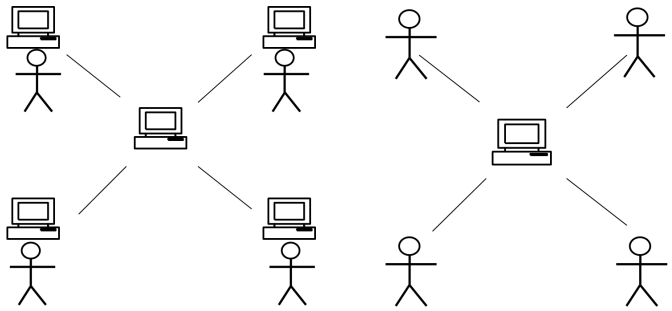
2.2 Kollaboratives Arbeiten

„*Viele Köche verderben den Brei*“ (Redensart)

„*Viele Hände - Schnelles Ende*“ (Redensart)

„Ja was denn nun?“ So mag sich der kritische Leser fragen. Wird die Effizienz gesteigert oder herabgesetzt, wenn eine Aufgabe von mehreren Akteuren erledigt wird?

Diese Frage beschäftigt auch die Informatik, wobei hier in den letzten Jahren eine Akzentverschiebung stattgefunden hat: Waren Neuentwicklungen früher eher auf verteiltes kollaboratives Arbeiten ausgerichtet (n User arbeiten an $n+1$ Rechnern, Abb. 2.4(a)) steht heute immer mehr die kollaborative Arbeit an einem Arbeitsplatz (n User arbeiten an *einem* Rechner, Abb. 2.4(b)) im Mittelpunkt. So hat der weltweite Marktführer im Softwarebereich Microsoft im Oktober 2009 mit Windows 7 das erste multitouch-fähige Betriebssystem auf den Markt gebracht.



(a) Client-Server-basierte verteilte kollaborative Arbeit

(b) Einzelplatzbasierte kollaborative Arbeit

Abbildung 2.4: Modelle computerunterstützter kollaborativer Arbeit

Um die obige Frage zu klären, werden im Folgenden einige Fragestellungen zum Thema „Kollaboratives Arbeiten“ aufgezeigt und diskutiert. Hierbei wird zunächst auf allgemeine Problematiken beim kollaborativen Arbeiten – wie z. B. Koordination und Kommunikation – eingegangen, um schließlich zu speziellen Problemstellungen bei der kollaborativen Arbeit am PC zu gelangen.

2.2.1 Grundsätzliche Problematiken der kollaborativen Arbeit

Allgemein beschrieben ist kollaborative Arbeit eine auf gemeinsame Ziele ausgerichtete Gruppenarbeit. Als solche bietet sie sowohl Chancen – so ist es gerade in kreativen Prozessen möglich, den Einzelnen über seine Leistungsfähigkeit beim isolierten Arbeiten hinauszuführen (z. B. *Brainstorming-Methode*) – hält aber als sozialer Prozess auch besondere Anforderungen bereit: Gruppendynamik muss berücksichtigt werden, die Zusammensetzung der Individuen muss geeignet und Regeln und Vorgehensweise für das gemeinsame Arbeiten müssen definiert sein.

Seitz definiert in diesem Zusammenhang Voraussetzungen für qualifizierte Gruppenarbeit [Sei93]. Im Folgenden werden diese erklärt und daraufhin ihr Bezug zur computergestützten kollaborativen Arbeit (CSCW) erläutert.

(A) Funktions- und Aufgabenintegration: Die in Gruppenarbeit durchgeführte Aufgabe muss in sich hierarchisch vollständig und abgeschlossen sein. Sie umfasst somit Planung, Prozessvorbereitung und –sicherung, Ausführung und Kontrolle. Dies ist sowohl nötig, um eine Autonomie der Gruppe zu erreichen, als auch, um die Zufriedenheit und somit die Leistungsfähigkeit der Gruppenmitglieder zu fördern. Wenn möglich, soll außerdem eine Rotation dieser Aufgabenfelder innerhalb der Gruppe erfolgen.

(B) Selbstregulation: Die Steuerung der Gruppenarbeit soll nicht von außen erfolgen, sondern aus der Gruppe heraus. Hierfür muss die Gesamtaufgabe für jedes Gruppenmitglied überschaubar und in sich zusammenhängend sein. Somit ist Bedingung (A) bereits eine Vorbedingung für (B).

(C) Kooperation und Kommunikation: Gruppeninterne Kommunikation und wechselseitige Unterstützung der Gruppenmitglieder müssen durch das Kooperationsmodell unterstützt werden.

(D) Qualifikatorische Integration Die Qualifikationsstruktur der Gruppe muss der Aufgabe angepasst sein. Es existiert auch die Forderung nach homogenem Qualifikationsniveau aller Beteiligten, allerdings wird diese komplexen Aufgaben mit hochgradigen Spezialisierungsanteilen nicht gerecht.

2.2.2 Computer Supported Collaborative Work

Computer Supported Collaborative Work (CSCW, auch : Computer Supported *Cooperative* Work) bezeichnet das Teilgebiet der Informatik, das sich mit Gruppenarbeit und deren soft- und hardwaretechnischen Unterstützungsmöglichkeiten befasst. Dabei ist das Selbstverständnis des Forschungsgebiets interdisziplinär, starke Bezüge bestehen u. a. zu Psychologie und Soziologie. Der Ursprung dieses Forschungsgebiets liegt in der Mitte der 80er Jahre, so wurde der Begriff erstmals 1984 als Name eines Workshops benutzt, und seit 1986 findet zweijährlich die ACM Conference on Computer Supported Cooperative Work statt. Außerdem besteht seit 1995 die Fachgruppe CSCW der deutschen Gesellschaft für Informatik e. V. .

Hasenkamp et al. unterscheiden in der CSCW drei Grundsätzliche Fachrichtungen [HS94]: Grundlagenforschung zu Kooperation und Kommunikation, Entwicklung von Werkzeugen und Konzepten – häufig als *Groupware* bezeichnet – sowie als drittes die Bewertung ebenjener Werkzeuge und Konzepte. Von Interesse ist hier vor allem der zweite Bereich, da das Ziel der vorliegenden Arbeit die Entwicklung eines solchen Tools war.

Zunächst zur Einbindung von CREWW in den Kontext Groupware: Man unterscheidet hier im zeitlichen Zusammenspiel der Akteure zwischen synchronen und asynchronen Formen. Synchrone Groupware verlangt, dass alle Akteure das Programm gleichzeitig ausführen, bei asynchroner kann dies zeitversetzt stattfinden. Zweites Unterscheidungskriterium ist die räumliche Komponente, so kann Groupware sowohl zentralisiertes (alle Akteure in einem Raum) als auch verteiltes (Akteure in unterschiedlichen Räumlichkeiten) Arbeiten unterstützen. CREWW ist als nicht-verteilte, synchrone Groupware realisiert, was sich, wie im Folgenden erläutert wird, auf die Anforderungen auswirkt.

Einen guten Überblick zum Bereich Groupware liefert die Website des Fachs Psychologie der Universität Freiburg [Fac], erläutert werden nun die dort formulierten Anforderungen an Groupware, an denen sich CREWW messen lassen muss:

- **Awareness:** Bezeichnet die Information des Nutzers über die anderen Nutzer. Dies umfasst das Wissen über das Handeln der Anderen und eventuell genutzte oder reservierte Artefakte.

- **Kommunikationsunterstützung:** Die Groupware muss den Austausch von Informationen zwischen den beteiligten Akteuren ermöglichen. Diese Anforderung ist bei einer synchronen, nicht-verteilten Anwendung zu vernachlässigen, da hier verbale Kommunikation zur Verfügung steht.

- **Kooperationsunterstützung:** Kooperation bezeichnet in diesem Fall den gemeinsamen Zugriff auf Daten und Funktionen. Groupware muss den gemeinsamen Zugriff auf Daten ermöglichen und sinnvoll kontrollieren.

- **Konsensfindungs- und Entscheidungsunterstützung:** Konsensfindung meint hier die Entscheidungsfindung in einer Gruppe. Sie kann durch explizite Aushandlung oder implizite Akzeptierung von getroffenen Entscheidungen zustande kommen. Groupware soll diesen Prozess fördern.

2.3 Handschrifterkennung

Zentraler Punkt dieses Kapitels ist die optische Zeichenerkennung. Zur Verbesserung der Usability soll in CREWW eine Handschrifterkennung auf CRC-Karten durchgeführt werden, daher folgt nun eine kurze Einführung in diese Thematik. Hierfür werden zunächst die Begriffe OCR und ICR voneinander abgegrenzt, um daraufhin einige frei erhältliche Opensource-Tools vorzustellen, die für die Integration in CREWW in Frage kamen.

2.3.1 OCR / ICR

Der Begriff optische Zeichenerkennung beschreibt die automatische Texterkennung auf Basis von Bilddaten. Grundlegend unterscheidet man hier zunächst *Online*- von *Offline*-Zeichenerkennung. Bei der Offline-Zeichenerkennung stehen dem Erkennungsprogramm nur Bilddaten in Form einer Rastergrafik zur Verfügung,wie dies z. B. bei eingescannten Bildern der Fall ist. Die Online-Zeichenerkennung zeichnet sich dadurch aus, dass zusätzlich zum Bildmaterial die zeitliche Abfolge des Eingabeflusses vorliegt (Touchscreen, PDA), wodurch diese eine deutlich höhere Erkennungsgenauigkeit liefert. Die folgenden Betrachtungen beschränken sich auf die Offline-Zeichenerkennnug, da bei CREWW keine Online-Eingabe zur Verfügung steht.

Allgemein ist der Ablauf bei der optischen Zeichenerkennung wie folgt: Zunächst findet eine Layouterkennung statt, hier werden Textblöcke identifiziert und von Bildblöcken abgegrenzt. Danach werden die Textblöcke in Zeilen und diese wiederum in einzelne Buchstaben unterteilt. Auf den einzelnen Buchstaben wird nun eine Mustererkennung durchgeführt.

Zu den Begrifflichkeiten lässt sich sagen, dass diese nicht trennscharf sind: Die Begriffe optische Zeichenerkennung, Texterkennung, ICR (Intelligent Character Recognition) und OCR (Optical Character Recognition) werden sowohl synonym als auch homonym benutzt. In der Wissenschaft stellt Texterkennung meist den Oberbegriff dar, während OCR für die Fehlerkorrektur auf Pixelebene und ICR für die kontextbezogene Fehlerkorrektur auf Buchstabenebene steht. OCR erkennt lediglich Buchstaben und fügt diese aneinander, ICR arbeitet zusätzlich mit Wörterbüchern und Vorkommenshäufigkeiten.

Die Begriffe werden aber auch genutzt, um Maschinenschrifterkennung (OCR) von Handschrifterkennung (ICR) abzugrenzen. In der vorliegenden Arbeit soll die zweite Definition Verwendung finden, somit bezeichnet OCR in diesem Kontext das Erkennen von Maschinenschrift – ob nur auf Pixel- oder auch auf Buchstabenebene soll hier nicht berücksichtigt werden – während ICR die automatische Erkennung von Hand-Blockschrift beschreibt.

2.3.2 Opensource-Tools

Anfängliche Zielsetzung war, in CREWW eine Opensource-Handschrifterkennung zu integrieren. Opensource-Tools im Allgemeinen bieten die Vorteile, dass der Code zum einen kostenlos erhältlich, zum anderen frei zugänglich ist. Ersteres verringert die Anschaffungskosten für den Enduser, Letzteres vereinfacht die Einbindung in CREWW für den Entwickler. Vorab lässt sich sagen, dass kein OpenSource-Tool gefunden werden konnte, welches die gewünschte Erkennungsgenauigkeit erzielt. Die gefundenen Tools beschränken sich entweder auf OCR oder können Handschrift

nur mit unzureichender Treffergenauigkeit erkennen. Nun eine kurze Vorstellung der getesteten OCR/ICR-Werkzeuge:

Tesseract : Tesseract [Tes] ist eine von Hewlett Packard (HP) entwickelte OCR-Engine, die nach dem Ausstieg von HP aus dem OCR-Sektor 1995 zu freier Software wurde. Seit 2006 wird sie von Google als Opensource-Software weiterentwickelt und ist über GoogleCode [Goo] frei erhältlich.

Tesseract basiert auf C++ und lässt sich sowohl als Komponente integrieren als auch als stand-alone-Anwendung auf Kommandozeilenebene ansprechen. Es unterstützt keine Layouterkennung, beherrscht also nur Zeilen- und Buchstabentrennung. Die Erkennungsgenauigkeit bei Maschinenschrift ist sehr gut, die bei Handschrift eher mangelhaft, womit eine Einbindung in CREWW für Tesseract nicht in Frage kommt.

Ocropus : Ocropus [OCR] ist ein modular aufgebautes, freies Text- und Dokumentenlayouterkennungssystem. Es wird entwickelt und betreut von Thomas Breuel am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI Kaiserslautern) und ist ebenfalls über GoogleCode erhältlich. Ocropus arbeitet auf Kommandozeilenebene und bietet, wie der Name sagt, sowohl Text- als auch Layouterkennung an. Allerdings beinhaltet Ocropus keine eigene Texterkennungsengine: Vielmehr bietet es die Möglichkeit, eine beliebige Engine als Modul zu wählen, wobei aktuell einzig Tesseract (s. o.) als Modul zur Verfügung steht.

GOCR : GOCR [GOC] ist wiederum ein vollständiges OCR-Paket. Es integriert Layout- und Texterkennung und bietet sowohl Kommandozeilen- als auch GUI-Funktionalität. Während es bei Maschinenschrift akzeptable Erkennungsergebnisse liefert, sind diese bei Handschrift stark fehlerbehaftet. Wegen der hohen Fehlerrate kommt auch GOCR für CREWW nicht in Frage.

STIHRS : STIHRS [STI], die letzte vorgestellte Software, bietet als einziges eine ICR-optimierte Erkennungsengine. So beinhaltet STIHRS z. B. die Möglichkeit, die Engine zu trainieren und erzielte Trainingsergebnisse in einem proprietären Format abzuspeichern. Dies geschieht bei STIHRS über eine GUI, hier lassen sich „Trainingsgrafiken“ importieren oder online erfassen.

Obgleich STIHRS bei Handschrifterkennung im Vergleich zu den anderen Opensource-Tools die besten Ergebnisse liefert, sind auch diese für die Verwendung in CREWW qualitativ nicht ausreichend.

Zusammenfassend lässt sich somit sagen, dass die zum aktuellen Zeitpunkt erhältlichen Opensource-Tools den Anforderungen hinsichtlich Erkennungsqualität

nicht genügen. Die Anwendungsgebiete von CREWW sollen in der Lehre und im Requirements Engineering liegen, beides Fachgebiete, in denen die Benutzerfreundlichkeit in hohem Grad die Akzeptanz bestimmt.

Aus diesem Grund wurde im Prototyp auf die Testversion einer kommerziellen ICR-Software zurückgegriffen: Omnipage Capture SDK. Dieses liefert akzeptable Erkennungsergebnisse bei guter Integrierbarkeit. Ausdrücklicher Dank geht an dieser Stelle an die Firma Nuance Communications Germany GmbH, die so freundlich war, kostenlose Lizenzschlüssel zur Probeevaluation zur Verfügung zu stellen.

2.4 Hardware

Zum Abschluss der Grundlagen folgen nun die Betrachtungen für Technikinteressierte: Die Anbindung der Wiimote an CREWW (siehe Abb. 2.5) wird beschrieben und die hierfür nötigen Techniken erläutert. Zunächst erfolgt die Vorstellung der Wii-Technologie und des zur Datenübertragung genutzten Bluetooth-Standards, um dann auf die von CREWW genutzte Client/Server-Architektur „SIGMAii“ näher einzugehen.

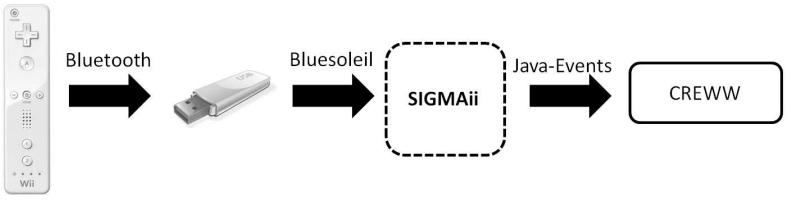


Abbildung 2.5: Anbindung der Wiimote an CREWW

2.4.1 Wii-Technologie

Die Spielebranche ist seit jeher nicht nur ein wichtiger Bereich der Softwarebranche, sondern auch Antreiber für die gesamte Computerindustrie. Schnellere Prozessoren, größere Speicherkapazitäten – Entwicklungen, deren Ursachen weniger in der steigenden Anforderung von Expertensystemen und Desktopanwendungen, sondern vor allem in der immer anspruchsvoller werdenden Architektur von Computerspielen liegen. Leistungsfähige, teure Rechner werden nicht von Firmen zum Einsatz am Arbeitsplatz gekauft, sondern von „Zockern“, die endlich ihr neuestes Computerspiel ohne störende Verzögerungen spielen wollen. Manchmal dient die Spielewelt aber auch als Ideengeber, so bei der *Nintendo Wii* (Abb. 2.6).

Die Spielekonsole „Nintendo Wii“ wurde zuerst auf der Tokyo Games Show am 18.09.2005 vorgestellt und erfreut sich seitdem ständig wachsender Beliebtheit. Grund hierfür ist wohl vor allem die innovative Steuerung über die Wii-Remote, im Folgenden kurz als Wiimote bezeichnet: Ein einhändiger, drahtloser Controller, der neben verschiedenen digitalen Knöpfen zusätzlich über eine Infrarotkamera und sechs Beschleunigungssensoren verfügt. Die Anbindung an die Wii erfolgt über Bluetooth, somit verfügt der Controller über eine PC-taugliche Schnittstelle. Diese Komponenten ermöglichen dem Controller sowohl eine dreidimensionale Lagebestimmung im Raum als auch eine hochauflösende sechsachsige Bewegungserkennung. Anhand dieser Möglichkeiten lassen sich Bewegungsabläufe der realen Welt auf Computerspiele abbilden: Vor allem Sportarten, z.B. Boxen, Golf oder Tennis seien hier genannt.

Abbildung 2.6: Nintendo Wii mit Wiimote

Zunächst einige technische Details zu den Komponenten:

Infrarotkamera : Die Integrierte Infrarotkamera kann bis zu vier Infrarotpunkte in einer Auflösung von 1024 * 768 mit 100Hz erfassen. Zusammen mit der mitgelieferten LED-Bar mit dem irritierenden Namen „Sensorbar“[2], die gut sichtbar aufgestellt wird, ist somit die dreidimensionale Lagebestimmung möglich. Als wichtig hat sich hier eine eine gute Signalqualität der LED-Bar gezeigt: Problematisch können improvisierte und batteriebetriebene Lösungen sein. Abfragen lassen

[2]Die *Sensorbar* enthält selbstverständlich keine *Sensoren* – schließlich ist sie nicht mit dem PC verbunden – sondern vier Infrarot-Leuchtdioden

sich von der Kamera sowohl zweidimensionale Positionen der erkannten LEDs als auch die dreidimensionale Lageposition der Wiimote. Diese wird errechnet durch Interpolation der Positionen der sichtbaren LEDs und Vergleich dieser Werte mit den gemessenen Beschleunigungswerten.

Beschleunigungssensor ADXL330 : Der im Controller angebrachte Beschleunigungssensor erfasst die Bewegungen der Wiimote in sechs Achsen (siehe Abb. 2.7: links/rechts (X) ,vor/zurück (Y), auf/ab (Z), Rollen (Roll), Neigen (Yaw) und Gieren (Pitch). Durch die hohe Empfindlichkeit des Sensors ist eine sehr genaue Auswertung der Handbewegung des Benutzers möglich.

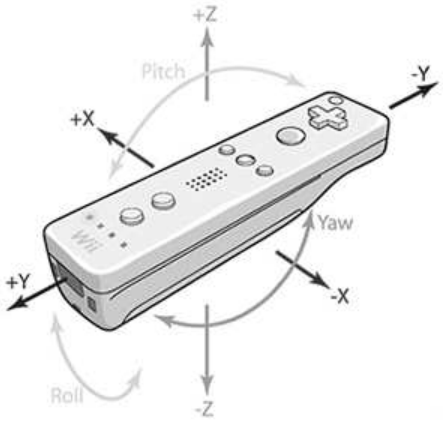


Abbildung 2.7: Achsen der Wiimote

Diese Merkmale lassen sich auch für traditionelle Anwendungen nutzen. So kann die Wiimote als Eingabegerät (HID)[3] genutzt werden, aber gerade die Infrarotkamera an der Vorderseite des Controllers bietet Möglichkeiten weit darüber hinaus. Hier sei z. B. auf die Arbeiten von Johnny Chung Lee verwiesen [Joh].

CREWW benutzt die Wiimote als HID. Wie bei einem Laserpointer zeigt der Benutzer auf eine Stelle am Bildschirm (bzw. beim Beamer auf die projizierte Fläche) und dort erscheint ein Cursor, der Interaktion ermöglicht. Die Kommunikation mit der Wiimote erfolgt bei CREWW über eine Client/Server-Architektur, die auf einem Bluetooth-Stack aufsetzt. Beide werden in den folgenden Unterkapiteln näher beschrieben.

[3]HID – Human Interface Device

2.4.2 Bluetooth

Die bei der Wiimote eingesetzte Bluetooth-Technik beschreibt einen Industriestandard zur drahtlosen Datenübertragung mittels Funk [4]. Mit Bluetooth können bis zu 255 Geräte vernetzt werden, von denen acht gleichzeitig aktiv sein können. Genutzt wird Bluetooth im PC-Bereich (Tastatur, Maus, etc.), in neueren Audio-Playern, PDAs und in Handys, z. B. zur Anbindung an Freisprecheinrichtungen. Bluetooth-Chips unterscheiden sich in Protokollversion, diese bestimmt die Datenübertragungsrate, und Senderstärke („Klasse"), die wiederum die Übertragungsreichweite bestimmt.

Der in der Wiimote eingesetzte Bluetooth-Chip BCM 2042 der Klasse 2 unterstützt Bluetooth 2.0 und bietet eine Datenübertragungsrate von bis zu 2,1 MB/s bei einer theoretischen Reichweite bis zu 10m. Zur Anbindung an den PC wird ein sogenannter Blutooth-Dongle (Empfänger) benötigt, z. B. in Form eines USB-Sticks oder einer PCMCIA-Card. Die betriebssystemseitige Kommunikation mit dem Dongle erfolgt über einen Protokollstapel (Bluetooth-Stack). Die im Windows-XP-Umfeld gebräuchlichsten sind die der Hersteller Bluesoleil, Widcomm und Toshiba, wobei auch Microsoft in seinem Service Pack 2 für Windows XP einen Bluetooth-Stack anbietet.

Für die Entwicklung von CREWW fiel die Wahl auf den Bluesoleil-Stack, wobei dies keine Auswirkungen auf die Anwendung hat: Der Benutzer kann von einem Bluetooth-Stack seiner Wahl Gebrauch machen. Abgeraten sei an dieser Stelle vom Windows-XP-Stack. Besagter fiel durch schlechtes Handling und häufige Übetragungsfehler auf.

2.4.3 SIGMAii Client/Server-Architektur

Mit der Vorstellung der von Sabine Müller im Rahmen ihrer Diplomarbeit am Lehrstuhl für Softwaretechnik entwickelten Client/Server-Architektur, die in CREWW Verwendung findet, erfolgt der Abschluss dieses Kapitels. Die Architektur und ihre API soll hier nur in Grundzügen beschrieben werden, für nähergehende Informationen sei an dieser Stelle auf die Diplomarbeit von Frau Müller [Mül08] verwiesen.

Das Backend des SIGMAii-Projektes bildet ein Server, der die Anbindung an den Bluetooth-Stack umsetzt, das Frontend wird durch einen Java-Client realisiert, der sich in ein beliebiges Java-Projekt integrieren lässt (siehe Abb. 2.8).

Server: Der Server übernimmt die Kommunikation mit der Hardware und liegt in zwei Varianten vor: Zum Einen als dynamische Bibliothek (DLL) in der Datei ServerConnect.dll, zum Anderen als C-basiertes externes Programm in der Datei WiiServer.exe. Für CREWW wurde die Variante des externen Servers gewählt.

[4] Benutzte Frequenzen: ISM-Band 2,402 - 2,480 GHz

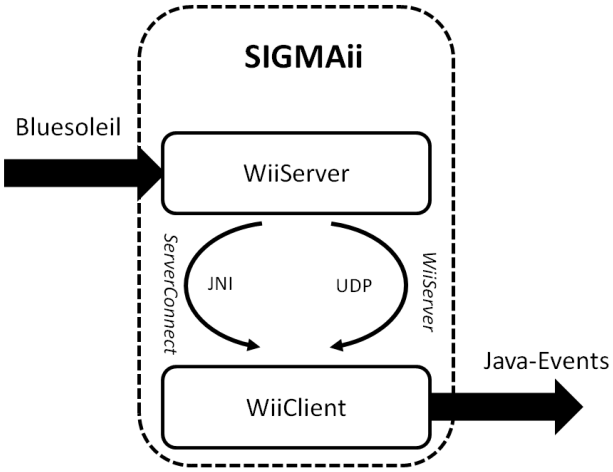


Abbildung 2.8: SIGMAii-Architektur

Der WiiServer seinerseits stellt einen Wrapper für die freie Bibliothek *wiiuse* [Wii]dar. Diese arbeitet als nicht-blockierendes Polling-System und bietet Methoden zum periodischen Abruf der Wiimote-Register sowie zur Anmeldung, Entfernung und Verwaltung angeschlossener Wiimotes. So lassen sich die Sensoren der einzelnen Wiimotes sowie deren Rumblefunktion an- und abstellen.

Die Kommunikation mit dem Client erfolgt beim WiiServer über UDP auf Port 1234. Hierfür sendet ein verbindungswilliger Client seine Bildschirmauflösung an den Server, der sich zu Beginn im Empfangsmodus befindet. Diese wird an die Bibliothek wiiuse weitergegeben, sodass wiiuse beim Auslesen der Wiimote-Register die richtigen Werte für die Cursorpositionen ermitteln kann. Daraufhin wechselt der Server in den Sende- und der Client in den Empfangsmodus. Zunächst sendet der Server eine Nachricht, die die Anzahl angeschlossener Wiimotes enthält, dann sendet er bei jeder Änderung die aktuellen Sensordaten der Wiimotes. Hierbei gibt es drei Nachrichtentypen:

- **CURSOR :** Lagedaten der erfassten LEDs
- **ACC :** Daten der Beschleunigungssensoren
- **BUTTON :** Gedrückte (oder losgelassene) Knöpfe

Der Client seinerseits kann dem Server ebenfalls Nachrichten schicken, diese sind:

- **LEDS :** Setzen der Wiimote LEDs
- **RUMBLE :** Rumble An- und Ausschalten

- **ACCS :** Beschleunigungssensoren An- und Ausschalten
- **IRDOT :** IR-Sensoren An- und Ausschalten
- **WAIT :** Server in Sleep-Modus setzen
- **SHUTDOWN :** Server herunterfahren

Client : Der in Java implementierte WiiClient ist nach dem *Observer Pattern* realisiert (siehe Abb. 2.9). Das Observer Pattern ist ein Entwurfsmuster aus der Gruppe der Verhaltensmuster, bei dem eine Anzahl von Objekten über Zustandsänderungen eines Subjektes wacht (siehe auch [Gam05]). Ein gängiges Beispiel hierfür ist die in GUI-Anwendungen gebräuchliche Listener/Event-Architektur.

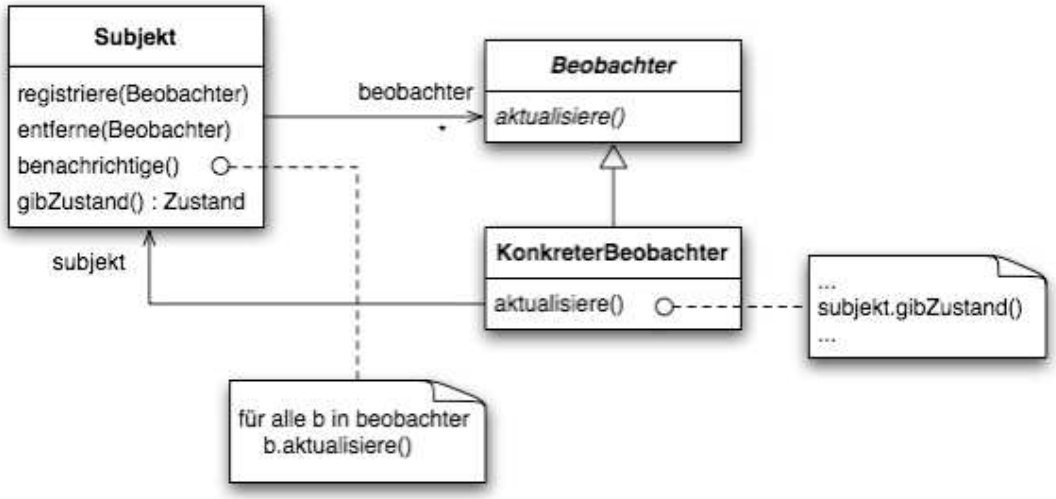


Abbildung 2.9: Observer Pattern

Ähnlich ist der WiiClient realisiert: Der Client selbst (WiiClient) stellt das Subjekt dar und die Listener (WiiAccelerationlistener, WiiListener, WiiRotationListener, WiiMotionListener) die Observer (siehe Abb. 2.10). Eine Klasse, die auf Wii-Aktionen reagieren will, muss somit die jeweiligen Listener implementieren. Der WiiClient erzeugt hierbei WiiEvents, die von den Klassen, welche die jeweiligen Listener implementieren, verarbeitet werden können.

Im Hauptprogramm selbst wiederum muss zu Beginn eine Instanz des WiiClient erzeugt und zu einem späteren Zeitpunkt die Listener-Klasse an den WiiClient übergeben werden. Hierfür existiert die Klasse WiiCenter, sie stellt den zentralen Zugriffspunkt für den WiiClient dar und bietet die *addListener*-Methoden an.

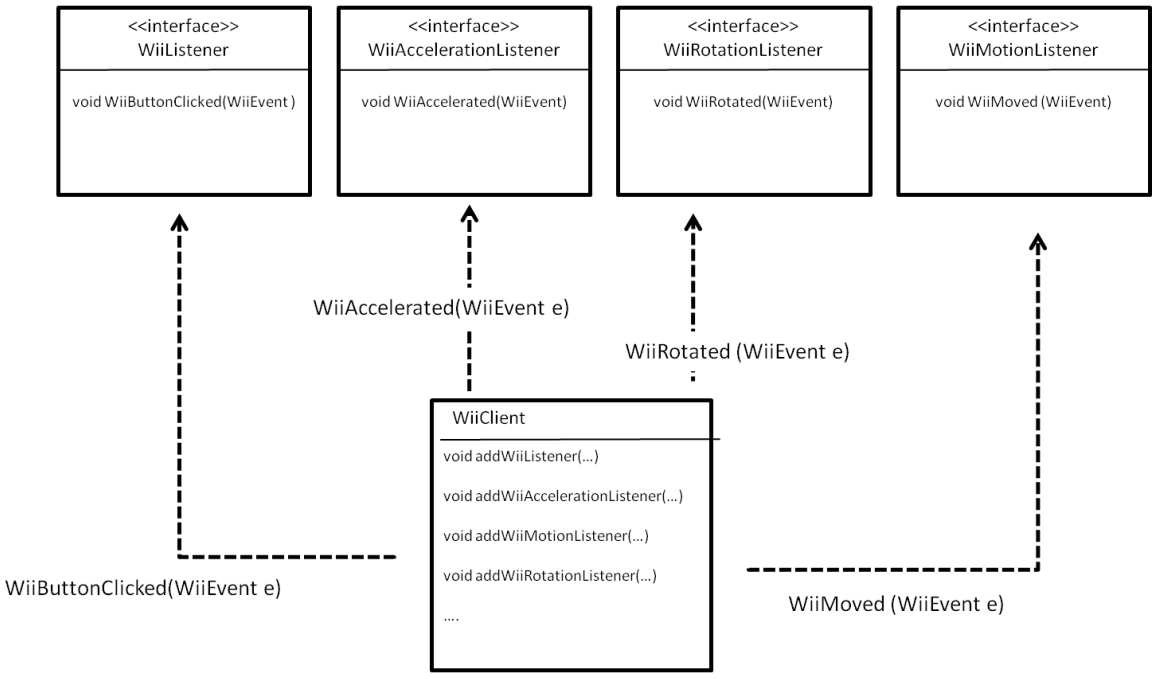


Abbildung 2.10: WiiClient-Architektur

Mit den dargelegten Betrachtungen zur SIGMAii-Architektur soll das Grundlagenkapitel abgeschlossen werden. Es folgt im nächsten Kapitel ein tieferer Einstieg in die CRC-Methode sowie die Vorstellung einer computergestützten Alternative.

Kapitel 3

Traditionelle Entwicklung mit CRC-Karten

> *„Grau, mein teurer Freund, ist alle Theorie und grün des Lebens goldner Baum.“*
> — J.W. von Goethe, Faust I

So soll nun, nachdem das Prinzip der CRC-Methode in Kap. 2.1.2 eher formal eingeführt wurde, ganz praktisch beschrieben werden, wie eine solche CRC-Arbeitssitzung ablaufen kann. Diese Beschreibung ist angelehnt an den von Bellin et al. geschilderten Ablauf einer solchen Sitzung [BS97]. Hierfür werden zunächst die einzelnen Phasen genauer beschrieben, um schließlich auf Vor- und Nachteile der traditionellen CRC-Methode einzugehen.

3.1 Ausgangssituation

Zunächst einmal eine Beschreibung der Ausgangssituation: Es soll eine Software entworfen werden, für die noch keine genaue Anforderungsspezifikation besteht. Der Anforderungskatalog existiert vorerst nur in den Köpfen der Auftraggeber, aus denen dieser mühsam „extrahiert“ werden muss. Hier zeigen sich bereits zwei Problemfelder:

Zum Ersten erfüllt dieser Katalog selten die Qualitätsanforderungen aus Kap. 2.1.1, sondern existiert eher als vage Vorstellung des Umfangs der Software: Wer soll mit ihr arbeiten? Was soll sie leisten? Was soll sie nicht leisten?

Zum Zweiten haben Auftraggeber (User) und Auftragnehmer (Designer, Programmierer) noch kein gemeinsames Vokabular. Der Auftraggeber hat kein softwaretechnisches Wissen und der Auftragnehmer kein Fachwissen über das Spezialgebiet des Auftraggebers.

Hier ist ein Ansatzpunkt für die CRC-Methode. Es wird ein Team aus vier bis sieben Teilnehmern zusammengestellt, dieses sollte je mindestens einen Fachexperten (Domain Expert), einen Programmierer, einen Analysten, einen Designer und einen Moderator enthalten[1].

Der Fachexperte bringt das Spezialwissen des Fachgebietes ein und ist für das Team von zentraler Bedeutung, z. B. für die Festlegung von Begrifflichkeiten und die Identifikation von Anwendungsfällen. Die Integration des Programmierers in dieses frühe Stadium der Softwareentwicklung ermöglicht einen praxisnahen Systementwurf, ist allerdings auch kritisch zu betrachten, da die Gefahr besteht, sich in Implementierungsdetails zu verlieren. Analyst und Designer werden wegen ihres methodischen Wissens in das Team integriert, ihre Funktion besteht in der Identifikation von strukturellen Komponenten und Mustern. Dem Moderator schließlich kommt die wichtige Funktion der Leitung der Sitzung zu. Wichtig ist an dieser Stelle zu erwähnen, dass mit Leitung hier nicht Führung gemeint ist. So soll der Moderator technische Entscheidungen nicht selbst treffen, sondern eher Diskurs und Konsensfindung in der Gruppe fördern sowie den Gesamtverlauf der CRC-Sitzung im Auge behalten und notfalls eingreifen, wenn die Gruppe abzuschweifen droht.

Hierbei muss folgendes beachtet werden: Die beschriebenen Funktionen der Gruppenmitglieder, außer der Moderation, sind keineswegs explizite Tätigkeitsaufforderungen. Vielmehr beschreiben sie sie Herangehensweisen der unterschiedlichen Fachgebiete, deren Zusammenführung gerade die Stärke der CRC-Methode ausmacht.

Soweit zur Zusammenstellung der Gruppe und Gruppengröße. Für die Sitzung an sich werden nur Stifte und ausreichend Karteikarten[2] benötigt. Nun kann die eigentliche CRC-Sitzung beginnen.

3.2 Phase I : Identifizieren von Use-Cases

Wenn sich die Gruppe zusammengefunden hat, ist der erste Schritt die Durchführung eines Brainstormings zur Identifikation von Use-Cases und Akteuren. Mögliche Systemanwendungsfälle werden erkannt und die beteiligten Akteure festgehalten. Dies kann z. B. mit UML-Use-Case-Diagrammen geschehen (siehe Kap. 2.1.3). Je nach Anwendung können die Anwendungsfälle jetzt schon kategorisiert und einzelnen Systemteilen zugeordnet werden. Dies hängt vom Umfang des Zielsystems ab: Bei kleineren Projekten genügt eine Hauptsystem, bei umfangreicheren Softwa-

[1] Dies ist nur eine von vielen möglichen Gruppenzusammensetzungen – So finden sich im Internet diverse Anleitungen zur Durchführung einer CRC-Arbeitssitzung, mit oder ohne expliziter Moderation durch einen Teilnehmer

[2] Größe: ca. 10 * 15 cm

relösungen hingegen zeichnet sich meist frühzeitig eine Unterteilung in Subsysteme ab, denen dann spezielle Anwendungsfälle zugeordnet werden können.

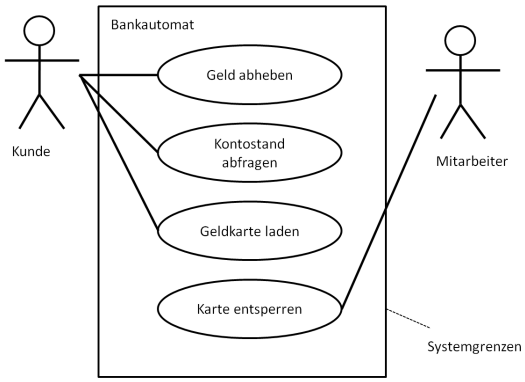


Abbildung 3.1: Beispiel: Use-Cases

Ist sich die Gruppe einig, dass alle relevanten Anwendungsfälle identifiziert wurden, kann in die nächste Phase übergegangen werden.

Eine interessante Vorstufe zur Identifikation von Anwendungsfällen ist die von Oestereich vorgeschlagene Anfertigung einer verbalen Produktbeschreibung in Form eines Produktkartons [Oes04]. Diese soll den Produktumfang – und damit die Anforderungen – grob umschreiben und die Identifikation der Teilnehmer mit dem Produkt erhöhen.

> „*Der Karton, als materialisiertes Ziel, schafft eine Identifikationsmöglichkeit und hat einen hohen symbolischen Wert.*“[Oes04]

3.3 Phase II : CRC-Karten entwerfen

Sind alle relevanten Use-Cases identifiziert, müssen zunächst Kandidatenklassen gefunden werden. Diese ergeben sich zu einem großen Teil aus den Anwendungsfällen.

Eine mögliche Methode hierfür ist die sog. *noun extraction*. Hier werden alle Substantive und substantivierten Verben aus den Anwendungsfällen und einer vorliegenden Produktbeschreibung als Kandidatenklassen übernommen. Danach wird auf diesen Kandidatenklassen eine Filterung vorgenommen, um offensichtlich unsinnige Kandidatenklassen und Duplikate zu entfernen. Hierbei gilt die Regel: Lieber am Anfang zu viele Klassen als zu wenige. Das nachträgliche Löschen nicht benötigter Klassen stellt bei der CRC-Methode keinen großen Aufwand dar.

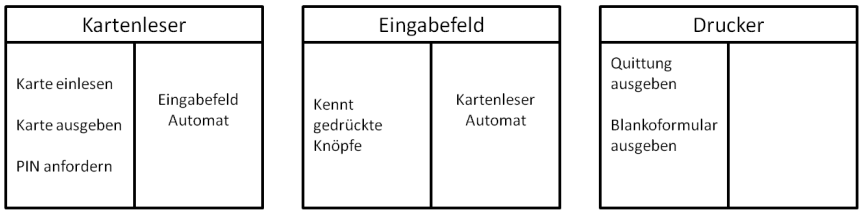


Abbildung 3.2: Beispiel: CRC-Karten

Wenn alle Kandidatenklassen identifiziert sind, wird jeder Klassenname auf eine CRC-Karte geschrieben. Den nächsten Schritt stellen Identifikation und Zuordnung der Verantwortlichkeiten dar. Analog zur *noun extraction* gibt es hier die Methode der *verb extraction*. Hier werden Verben zu Kandidatenresponsibilities, diese können dann den Kandidatenklassen zugeordnet werden. Dafür wird die Verantwortlichkeit in das Responsibility-Feld der CRC-Karte der Kandidatenklasse geschrieben. Eine vollständige Erfassung aller Verantwortlichkeiten ist nicht nötig, fehlende Responsibilities werden in Phase III erfasst.

Abschließend werden nun alle Klassen und Verantwortlichkeiten erneut geprüft: Ist die Bezeichnung eindeutig und gehört die Verantwortlichkeit tatsächlich in diese Klasse? In diesem Schritt können auch erste offensichtliche Collaborators identifiziert werden. Falls ersichtlich ist, dass Klasse A eine Anfrage an Klasse B stellen muss, um eine Responsibility zu erfüllen, wird Klasse B im Collaborators-Feld von Klasse A notiert.

3.4 Phase III : Use-Cases durchspielen

Nun kann Phase III starten. In dieser Phase werden die Use-Cases in einer Art Rollenspiel durchgespielt. Hierbei übernimmt jedes Gruppenmitglied eine Menge von Klassen und agiert im Verlauf des Rollenspiels für diese.

Zu Beginn des Rollenspiels werden zunächst die Klassen verteilt. Dies sollte ungefähr gleichmäßig geschehen, wobei wichtig ist, dass jeder Teilnehmer im Besitz mindestens einer Klasse ist. Nun wird ein Anwendungsfall gewählt und die beginnende Klasse identifiziert. Dies ist die Klasse, die den Anwendungsfall als Verantwortlichkeit hat. Damit ist der Teilnehmer, der diese Klasse besitzt, an der Reihe. Wenn ein Teilnehmer an die Reihe kommt, gibt es grundsätzlich zwei Möglichkeiten: Entweder kann er seine Verantwortlichkeit erfüllen oder er muss zu dieser Erfüllung auf eine oder mehrere andere Klassen (spätere Collaborators) zurückgreifen.

Kann er seine Verantwortlichkeit ohne fremde Hilfe erfüllen, ist er fertig und die

Klasse an der Reihe, die die Anfrage an ihn gestellt hat. Im zweiten Fall – der Teilnehmer kann die Verantwortlichkeit nicht selbstständig erfüllen – wird bei den anderen Klassen nach der Responsibility gesucht, die er benötigt, um die eigene zu erfüllen. Auch hier gibt es wieder zwei Möglichkeiten. Steht die Responsibility bei einer der anderen Klassen, wird diese aufgerufen und ist somit an der Reihe. Ist die Responsibility noch nicht erfasst, muss sie in das CRC-Modell aufgenommen werden. Hierfür wird zunächst geprüft, ob die Responsibility in eine der vorhandenen Klassen passt und wenn ja wird sie in diese übernommen. Passt sie in keine der vorhandenen Klassen, wird eine neue Klasse angelegt und die Responsibility an diese Klasse übertragen. Immer dann, wenn eine Klasse eine andere aufruft, kann der Name der aufgerufenen Klasse in das Collaborators-Feld der aufrufenden Klasse eingetragen werden.

Somit wird ein Stapel von gegenseitigen Anfragen aufgebaut, der genau zu dem Zeitpunkt wieder leer ist, wenn der Anwendungsfall beendet ist. Auf diese Art und Weise werden alle Anwendungsfälle durchgespielt. Ist dies bei allen der Fall, ist davon auszugehen, dass die wichtigsten Klassen und Verantwortlichkeiten identifiziert wurden. Wahlweise kann aber auch erneut in Phase I begonnen und mit dem im Rollenspiel erworbenen Wissen zusätzliche Use-Cases identifiziert werden: Phasen I bis III können beliebig oft durchlaufen werden.

3.5 Transfer nach UML

Ist Phase III abgeschlossen und sind somit alle Klassen und deren Verantwortlichkeiten gefunden, kann das so erstellte rudimentäre Modell in eine mächtigere Modellierungssprache übertragen werden. Der Umfang der CRC-Karten ist für die Modellierung einer komplexen Klassenstruktur bei weitem nicht ausreichend, dies kann z. B. mit UML-Klassendiagrammen (siehe Kap. 2.1.3) geschehen. Dort können auch weitergehende Konzepte wie Generalisierung, Spezialisierung, Interfaces, Attribute usw. realisiert werden.

3.6 Vor- und Nachteile der traditionellen Methode

CREWW bietet die Möglichkeit, den beschriebenen Prozess computergestützt zu durchlaufen. Doch ist diese technische Unterstützung überhaupt notwendig, wenn nicht gar kontraproduktiv? Zur Beantwortung dieser Frage folgt nun die Analyse der beschriebenen Methode einschließlich einer Bewertung deren Stärken und Schwächen.

Zunächst einmal zu den Vorteilen der traditionellen Methode. Als erstes wäre hier das Abhandensein jedweder technischer Abhängigkeiten zu nennen. Da die

CRC-Methode keinerlei elektronischer Hilfsmittel bedarf, ist sie zum Einen unabhängig von Rechnerabstürzen, Kompatibilitätsproblemen, leeren Batterien etc. – jeder, der schon einmal einen Vortrag von einem fremden Laptop halten musste, weiß, wovon die Rede ist. Zum Anderen ist die Einstiegsmöglichkeit für technophobe Nutzer einfach, es muss keine Steuerung oder Ähnliches erlernt werden.

Eine zweite Stärke ist der Gruppenarbeits-Aspekt. Zwar lassen sich CRC-Karten auch von einzelnen Anwendern zur Anforderungsanalyse nutzen, doch der Nutzen der Karten ist in der Gruppe deutlich höher. Für CREWW bedeutet dies, dass kollaborative Steuerung der Anwendung möglich sein muss.

Die größte Stärke der CRC-Methode ist letztlich die Identifiaktion der Teilnehmer mit ihren Klassen[3]. Die Karten kann man in der Hand halten oder auf dem Tisch bzw. der Tafel herumschieben und gerade diese Erfahrung ermöglicht laut Beck et al. die Identifikation [BC89] und somit eine neue, objektorientierte Betrachtungsweise des Problems: Was muss ich tun oder wissen, um die an mich gestellte Anfrage zu beantworten? Manche Autoren gehen an dieser Stelle so weit, den Technikeinsatz bei dieser Methode aus diesem Grund ganz zu untersagen. Hieraus leitet sich schon die erste Anforderung an CREWW ab: Steuerung und Interaktion müssen gewährleisten, dass die Benutzer das Gefühl haben, die Karten tatsächlich in der Hand zu halten. Dies erfordert eine gute Usability und intuitive Steuerungsmöglichkeiten.

Doch nun zu den Schwächen der traditionellen CRC-Methode. Hier wäre zunächst einmal die fehlende Flexibilität der Karten zu nennen. Die Größe ist fest vorgegeben und kann während der Session schlecht verändert werden. Weiterhin können Fehler nicht korrigiert werden: Ein Durchstreichen ist zwar möglich, allerdings werden die Karten somit schnell unleserlich. Das heißt, dass Karten neu geschrieben werden müssen, was sowohl Zeitaufwand als auch eine mentale Hürde für etwaige Änderungen darstellt – Ist der Änderungsaufwand zu hoch, kann es vorkommen, dass eventuell sinnvolle Änderungen nicht durchgeführt werden.

Ein weiterer Schwachpunkt ist die fehlende Schnittstelle zu UML. Nach der CRC-Sitzung werden die Karten entsorgt und somit Arbeit zunichte gemacht. Dies bedeutet für CREWW, dass zumindest Grundlegende UML-Konzepte implementiert werden sollten. Außderem soll eine Möglichkeit bestehen, die auf den CRC-Karten gespeicherten Informationen in ein UML-Modellierungs-Tool zu übertragen.

Zentraler Kritikpunkt jedoch ist die Tatsache, dass sich mit der CRC-Methode der Informationsfluss nicht immer einwandfrei nachvollziehen lässt. Steigt die Verschachtelungstiefe der gegenseitigen Aufrufe, müssen sich die Teilnehmer bei der traditionellen Methode merken, wer wen in welcher Reihenfolge aufgerufen hat.

[3] Dies wird z. T. auch anders gesehen – Börstler kritisiert diese Identifikation und beschreibt die CRC-karten als „möglicherweise schädlich“, da sie auch keine Unterscheidung zwischen Klassen und Objekten machen [Bör05].

Eine Software zur Unterstützung der CRC-Methode sollte diese Zusammenhänge visualisieren und eine komfortable Navigation durch den kompletten Anwendungsfall ermöglichen. Implementiert sein müsste auch die Möglichkeit, Szenarien zu Speichern und zu Laden, um ein zuvor unterbrochenes Szenario wieder aufzunehmen oder ein durchlaufenes Szenario zu einem späteren Zeitpunkt (z.B. in der Implementierungsphase) noch einmal durchzugehen.

Zur Veranschaulichung der aufgezählten Punkte und der daraus zu folgernden Anforderungen an die Software erfolgt nun eine tabellarische Auflistung der Vor- und Nachteile der traditionellen CRC-Methode.

Tabelle 3.1: Vor- und Nachteile der traditionellen CRC-Methode und Folgerungen

Vorteil/Nachteil	**Eigenschaft der traditionellen Methode**	**Folgerung**
+	keine technischen Mittel nötig	sichere, ausgereifte Technik nötig
+	einfacher Einstieg	Software darf nicht zu kompliziert sein;gute Erlernbarkeit, intuitive Steuerung wichtig
+	Gruppenarbeit ermöglicht kreatives Arbeiten	Software muss kollaborativ gesteuert werden können
+	starke Identifikation	Software muss Identifikation erhalten; „Anfassen“, Reservieren, Verschieben der Karten muss über intuitive Metapher möglich sein
-	fehlende Flexibilität der Karten	Karten müssen ohne Aufwand und jederzeit veränderbar sein
-	fehlende Schnittstelle zu UML	Bereitstellung einer Schnittstelle zu einem UML-Tool
-	Vor- und Zurückgehen im Use-Case schwierig	Implementierung einer Undo/Redo-Funktion im Tool
-	Informationsfluss nicht immer nachvollziehbar	Entwicklung einer Metapher zur Visualisierung der Aufrufe

Kapitel 4

Computerunterstützte Entwicklung mit CREWW

Im vorangegangenen Kapitel wurde die CRC-Methode ausführlich beschrieben und deren Stärken und Schwächen diskutiert. Aus diesen wurden die Anforderungen abgeleitet, die an ein Tool zur CRC-Unterstützung zu stellen sind.

Darauf basierend soll nun gezeigt werden, wie eine solche Sitzung mit Computerunterstützung aussieht: Das im Rahmen der vorliegenden Arbeit entwickelte CASE[1]-Tool CREWW wird vorgestellt. Hierbei werden die Funktionen des Tools in der Reihenfolge beschrieben, in der sie in der CRC-Sitzung eingesetzt werden. Falls für den Leser eher eine Auflistung der Funktionen aus Architektursicht interessant ist, sei an dieser Stelle auf Kap. 5.3 verwiesen.

4.1 Ausgangssituation

Zur Ausgangssituation lässt sich zunächst sagen, dass die Aufgabenstellung identisch ist und für die Gruppenzusammenstellung dieselben Bedingungen gelten. Einzig für die Gruppengröße gilt eine Einschränkung: CREWW kann kollaborativ von bis zu vier Anwendern gesteuert werden, daher wird hier als Gruppengröße vier empfohlen. Größere Gruppen sind möglich, allerdings müssten in diesem Fall Teilgruppen gebildet werden, wobei in jeder Gruppe ein Mitglied die Rolle des Koordinators einnehmen und für die Wiimote-Steuerung verantwortlich sein müsste. Ebenfalls möglich ist eine Erweiterung der Software – in einem Bluetooth-Netzwerk (Piconet) können bis zu acht Geräte zur gleichen Zeit aktiv sein – hierfür wären allerdings Anpassungen der genutzten Client/Server-Architektur SIGMAii nötig.

Da sich der Methodeneinsatz von dem zuvor beschriebenen unterscheidet, gelten hier einige zusätzliche Bedingungen. Zunächst einmal werden ein Rechner und ein

[1] Computer Aided Software Engineering

Beamer benötigt. CREWW lässt sich auch ohne Beamer betreiben, allerdings wäre dann nicht gewährleistet, dass alle Teilnehmer eine gute Sicht auf den Bildschirm haben. Weiterhin wird eine der Gruppengröße entsprechende Anzahl Wiimotes benötigt, da mit diesen die Steuerung der Software durch die Benutzer erfolgt. Durch die kollaborative Steuerung kann gemeinsam modelliert oder ein Anwendungsfall durchgespielt werden.

Sowohl Soft- als auch Hardware müssen zuvor geprüft worden sein, um unnötige Verzögerungen zu vermeiden. Zuerst wird der WiiServer und dann CREWW gestartet, wobei automatisch eine neue Sitzung angelegt wird. Ebenfalls möglich ist die Fortführung einer zuvor gespeicherten Sitzung. Nun werden die Namen der Gruppenmitglieder erfasst, so lässt sich während des Betriebs besser verfolgen, welcher Cursor zu welchem User gehört. Weiterhin erhält jedes Gruppenmitglied eine kurze Einführung in CREWW, so dass alle zumindest die Grundfunktionen beherrschen. Im Rahmen dieser Einarbeitung kann auch die Nutzung der Wiimote als Zeigegerät gelernt werden.

Werden im Verlauf der Sitzung Hardcopy-CRC-Karten benötigt, können diese mit CREWW ausgedruckt werden. Laufen Soft- und Hardware und sind alle Teilnehmer eingearbeitet, kann die Sitzung beginnen.

4.2 Phase I : Identifizieren von Use-Cases

CREWW bietet zum aktuellen Zeitpunkt keine Unterstützung zum Entwerfen von Use-Case-Diagrammen. Allerdings ist die Implementierung einer Anwendungsfallverwaltung durchaus in Betracht zu ziehen. In einer solchen Komponente müssten sich Systeme und Anwendungsfälle modellieren und zu den Anwendungsfällen verschiedene Szenarien abspeichern lassen.

Da diese Möglichkeit zur Zeit jedoch noch nicht besteht, erfolgt die Identifizierung der Use-Cases wie bei der traditionellen Methode (siehe Kap. 3.2). Auch hier wird die Benutzung von UML Use-Case Diagrammen empfohlen (Siehe Abb. 3.1). Ist eine ausreichende Menge von Use-Cases identifiziert, können die CRC-Karten[2] entworfen werden.

4.3 Phase II : CRC-Karten entwerfen

Von der Vorgehensweise her unterscheidet sich Phase II kaum von der der traditionellen Methode, nur werden die Karten nun digital erfasst. Das Auffinden von

[2]Die Begriffe CRC-Karte, CRC-Klasse und CRC-Komponente werden im Folgenden synonym verwendet

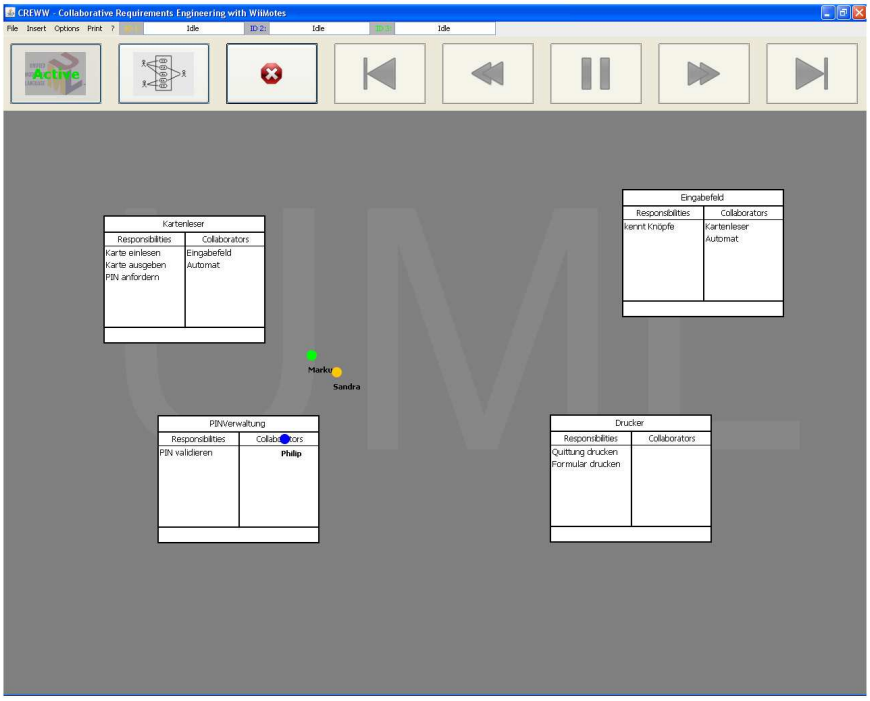


Abbildung 4.1: CRC-Karten entwerfen mit CREWW

Kandidatenklassen und Verantwortlichkeiten geschieht wie in Kap. 3.3 beschrieben, zur Erfassung bietet CREWW zwei Möglichkeiten:

Erstens können die Daten der Karten von Hand eingetragen werden. Dies kann entweder zur Bearbeitungszeit geschehen, sodass physische Hardcopy-CRC-Karten überhaupt nicht mehr benutzt werden oder aber die Karten können komplett erfasst werden, nachdem der Entwurf steht. Hierfür wird über das Menü (oder per Wiimote) eine neue Karte angelegt und die Kartendaten (Responsibilities, Collaborators) per Hand eingetragen.

Zweitens besteht die Möglichkeit, die Karten einzuscannen. CREWW verfügt über eine TWAIN-Anbindung, sodass ein beliebiger Windows-kompatibler Scanner zur Erfassung der Karten benutzt werden kann. Hierfür steht eine Funktion zur Verfügung, die nach dem Scanvorgang automatisch eine neue CRC-Karte angelegt und zusätzlich noch eine Handschrifterkennung auf der Karte durchführt, falls ein Handschrifterkennungsmodul vorhanden ist.

Die angelegten digitalen Karten können nun beliebig auf dem Bildschirm verteilt werden. Sinnvoll ist hier eine Anordnung entweder nach Komponenten, falls solche schon identifiziert wurden, oder nach Benutzern. Hierbei gilt: Zur gleichen Zeit kann nur ein Benutzer mit einer Karte interagieren, während der Interaktion ist diese für andere Nutzer gesperrt. Sind alle angelegten Karten erfasst und wie

erwünscht auf der Oberfläche angeordnet, kann das Rollenspiel beginnen.

4.4 Phase III : Der Use-Case-Mode

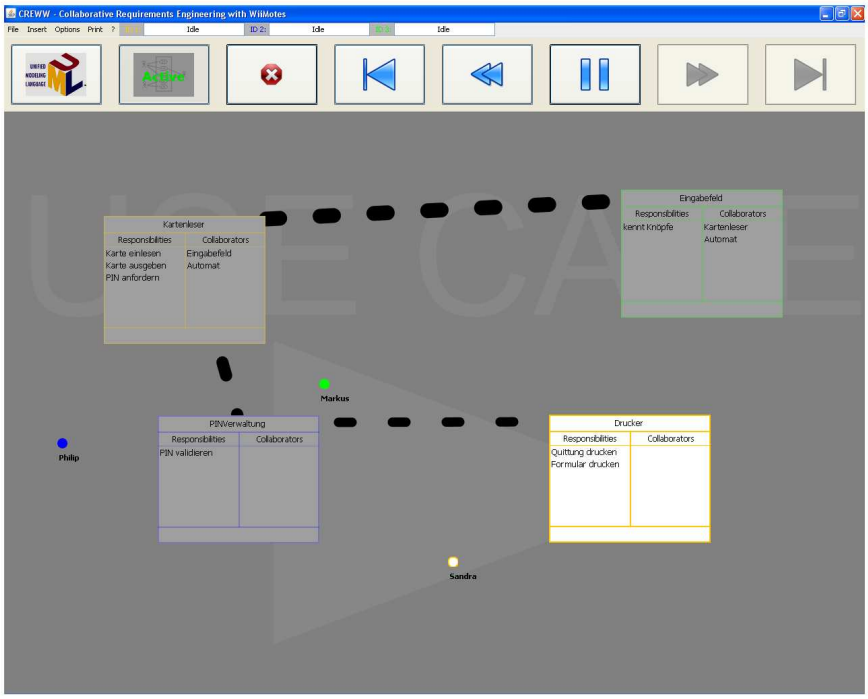


Abbildung 4.2: Anwendungsfälle durchspielen mit CREWW

Der zentrale Aspekt der CRC-Methode ist auch der von CREWW, nämlich das Rollenspiel. Die Use-Cases werden durchgespielt und hierbei Fehler und Lücken im Design aufgedeckt.

Hierfür wird zunächst in den Use-Case-Mode gewechselt. Daraufhin werden die Klassen unter den Benutzern aufgeteilt. Hierbei gelten dieselben Bedingungen wie bei der traditionellen Methode: Die Klassen sollten gleichmäßig verteilt sein und jeder Benutzer muss mindestens eine Klasse besitzen. Zu Beginn des Use-Case-Mode sind alle Klassen ohne Besitzer. Bei Inbesitznahme durch einen Nutzer wird die Klasse in der Cursorfarbe des Nutzers eingefärbt, und andere Benutzer können mit dieser Klasse nicht mehr interagieren.

Sind alle Klassen verteilt, kann der Anwendungsfall gestartet werden, das Programm wechselt in den Play-Modus. Im Play-Modus steuert nur ein Spieler den Kontrollfluss, dies ist der Besitzer der aktiven Klasse. Die aktive Klasse ist hierbei diejenige, auf deren Verantwortlichkeit gerade zugegriffen wird. Wie auch in der

traditionellen Variante ist zu Beginn des Rollenspiels die Klasse aktiv, in deren Verantwortlichkeit der Use-Case fällt.

Analog zu Kap. 3.4 bestehen nun zwei Möglichkeiten: Entweder ist die Verantwortlichkeit erfüllt oder die Klasse muss eine Teil-Verantwortlichkeit an eine andere Klasse delegieren. Für beide Zwecke muss eine andere Klasse aktiviert werden, was jeweils mit einem Knopfdruck der Wiimote des entsprechenden Benutzers möglich ist

Dieser Vorgang wird so lange wiederholt, bis die zuerst aufgerufene Klasse ihre Verantwortlichkeit erfüllt hat und somit der Anwendungsfall beendet ist. Treten im Verlauf des Rollenspiels Missverständnisse auf, ist eine Navigation im Anwendungsfall (Redo/Undo) möglich.

Nach Beendigung dieses Anwendungfalles können auf dieselbe Art und Weise beliebig viele weitere Anwendungsfälle durchgespielt werden. Sind alle das System betreffenden Anwendungsfälle durchgespielt, kann mit der UML-Modellierung begonnen werden.

4.5 CRC-UML-Modellierung: Der UML-Mode

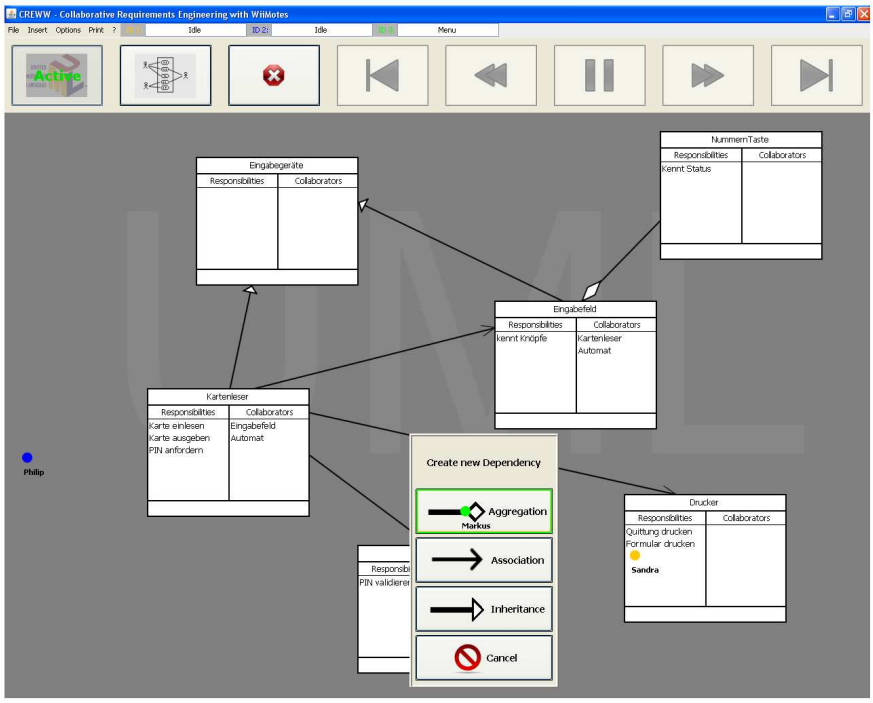


Abbildung 4.3: UML-Modellierung mit CREWW

Die eigentliche CRC-Sitzung ist nun abgeschlossen, jedoch bietet CREWW, um den Übergang zu UML zu vereinfachen, die Möglichkeit, grundlegende UML-Konzepte wie Vererbung und Aggregation zu modellieren. Hierfür wird nach Abschluss von Phase III erneut in den UML-Modus gewechselt. Nun können Beziehungen zwischen den Klassen hergestellt werden, indem je zwei Klassen angewählt und dann eine Beziehungsart gewählt wird. Über ein Popup-Menü kann der Benutzer zwischen den Beziehungsarten *Aggregation*, *Inheritance* (Vererbung) und *Association* (Assoziation) wählen.

Ist die UML-Modellierung in CREWW abgeschlossen, steht dem Benutzer ein Export nach XMI 1.2 zur Verfügung. Das Exportformat ist optimiert für den späteren Import in ArgoUML, ein verbreitetes Opensource-UML-Modellierungs-Tool. Somit besteht die Möglichkeit, die Ergebnisse der CRC-Sitzung festzuhalten und weiterzuverwenden, eine Funktion, die noch kein existierendes CRC-Tool bietet.

Kapitel 5

Design

In Kap. 4 wurde die Bedienung von CREWW auf chronologische Art und Weise anhand des Verlaufs einer CRC-Sitzung erklärt. Der Vollständigkeit halber erfolgt nun eine Beschreibung der Funktionen von CREWW aus Architektursicht. Hierfür werden zunächst die Anforderungen zusammengestellt, um daraus dann sowohl das benötigte Modell als auch die zu implementierenden Funktionen abzuleiten.

5.1 Anforderungsspezifikation

5.1.1 Funktionale Anforderungen

In Tabelle 3.1 wurden die Anforderungen bereits stichpunktartig hergeleitet. Zur Veranschaulichung werden diese im Folgenden in einem zusammenhängenden Kontext verbalisiert.

CREWW soll als kollaboratives CASE-Tool realisiert werden. Ziel ist es, die Möglichkeit zu bieten, eine Gruppe bei der Durchführung einer CRC-Arbeitssitzung zu unterstützen, wobei die Anwendung von allen Gruppenmitgliedern kollaborativ gesteuert werden kann. Im Mittelpunkt steht hier das Durchspielen eines Anwendungsfalles im Rollenspiel. Hierbei ist es wichtig, dass zu jedem Zeitpunkt klar ist, wer aktuell die Steuerung hat – „an der Reihe“ ist – und in welcher Reihenfolge diese Steuerung übergeben wurde. Für die Verdeutlichung dieser Reihenfolge muss eine sinnvolle Metapher entwickelt werden. Während des Rollenspiels muss für den Fall eines Fehlaufrufs oder eines Designfehlers die Möglichkeit der Navigation durch den Anwendungsfall, bildlich ein Vor- und Rücklauf, gegeben sein.

Damit diese im Rollenspiel genutzt werden können, soll die Möglichkeit bestehen, CRC-Karten von Hand einzugeben oder einzuscannen. Wenn möglich soll eine Handschrifterkennung auf eingescannten Karten durchgeführt werden.

Darüber hinaus soll CREWW als Schnittstelle zwischen CRC-Karten und der

UML dienen und somit in begrenztem Umfang UML-Funktionalität bieten. Die Konzepte Vererbung und Aggregation müssen in CREWW modelliert werden können, weitere sind möglich, aber nicht zwingend. Damit das mit den CRC-Karten entwickelte Modell weiter genutzt werden kann, muss eine Export-Funktion implementiert werden, die in ein gängiges UML-fähiges Format exportiert.

5.1.2 Nicht-funktionale Anforderungen

Bei der Implementierung ist die Benutzerfreundlichkeit von größter Bedeutung. Dies hat zwei Gründe: Zum einen ist die große Stärke der CRC-Methode das Rollenspiel – dieses soll durch den Technikeinsatz nicht gestört werden. Zum anderen sind die Einsatzgebiete der CRC-Methode die Lehre und das Requirements Engineering, beides Bereiche, in denen man es oft mit unerfahrenen Anwendern zu tun hat.

Weiterhin muss das System in Java implementiert werden, da der vorhandene WiiClient ebenfalls in Java implementiert ist.

5.2 Modell

Aus den zuvor gestellten Anforderungen ergibt sich zunächst folgende Notwendigkeit: Es müssen zwei verschiedene Modi implementiert werden, wobei ein Umschalten zwischen diesen jederzeit möglich sein muss. Dies ist zum Einen der UML-Modus, zur Modellierung von rudimentären UML-Klassendiagrammen. Zum Anderen ist dies der Use-Case-Modus, in dem das Durchspielen der Use-Cases ermöglicht und unterstützt wird.

Weiterhin ergibt eine Analyse der gestellten Anforderungen, dass das für die Realisierung benötigte Datenmodell die Struktur eines Graphen G hat.

$G = (V, E)$
$V = \{Menge\ der\ CRC - Komponenten\}$
$E = \{Menge\ der\ Beziehungen\ zwischen\ Komponenten\}$

Die Knotenmenge V ist hier die Menge der CRC-Karten, auf diese wird in Abschnitt 5.2.1 eingegangen. Die Kantenmenge E wiederum ist die Menge der Beziehungen zwischen den CRC-Karten, diese werden in Abschnitt 5.2.2 vorgestellt.

5.2.1 CRC-Knoten

Wie schon weiter oben erwähnt werden die CRC-Karten als Knoten eines Graphen repräsentiert. Diese speichern als zentrale Elemente Klassenname, Verantwortlich-

keiten und kollaborierende Klassen. Weiterhin wird optional, falls die Karte eingescannt wurde, die Originalkarte als Bild gespeichert. Zur Unterstützung des Rollenspiels wird außerdem noch der Besitzer der Karten gespeichert. Für die Knoten stehen drei grundsätzliche Darstellungsarten zur Verfügung: digitale CRC-Karte, eingescannte CRC-Karte und zusammengeklappte Darstellung.

Die Darstellung der Karten variiert aber auch mit dem Modus: Im UML-Modus wird unterschieden zwischen „in Gebrauch“ und „verfügbar“. Verfügbare Karten erhalten einen schwarzen Rahmen und Karten im Gebrauch erhalten die Farbe des Benutzers, in dessen Gebrauch die Karte ist – dies ist z. B. beim Draggen der Fall.

Im Use-Case-Modus können Karten in Besitz genommen werden, dies geschieht beim Verteilen der Klassen. Karten ohne Besitzer werden hier schwarz dargestellt und bereits verteilte Karten in der Farbe des Benutzers. Hierdurch ist eine klare Zuordnung der Karten immer möglich. Im „Play“-Modus werden außerdem alle nicht aktiven Klassen leicht ausgegraut.

5.2.2 Kanten

Die Kanten wiederum werden genutzt, um sowohl UML-Beziehungen (UML-Kanten) als auch den Verlauf des Anwendungsfalles (Use-Case-Kanten) darzustellen. Existiert eine Use-Case-Kante von Knoten A noch Knoten B, bedeutet dies, dass Knoten A eine Verantwortlichkeit von Knoten B aufgerufen hat und diese noch nicht abgearbeitet ist.

Allgemein speichert eine Kante Ziel- und Endknoten sowie den Kantentyp. Die Darstellung der Kanten variiert mit Kantentyp und Modus. Im UML-Modus werden die UML-Kanten hervorgehoben dargestellt und die Use-Case-Kanten werden ausgegraut. Umgekehrt verhält es sich beim Use-Case-Modus: Hier werden die Use-Case-Kanten deutlich und die UML-Kanten ausgegraut dargestellt. Zusätzlich unterscheidet der Use-Case-Modus noch zwischen „Play“ und „Pause“ (näheres in Kap. 5.3.3). Im Play-Modus werden die Use-Case-Kanten animiert dargestellt, sie „fließen“ vom Start- zum Zielknoten. Im Pause-Modus wird die Animation gestoppt.

5.3 Interaktionsmögichkeiten

Mit einer Aufstellung der Interaktionsmöglichkeiten soll das Architekturkapitel abgeschlossen werden. Unterteilt in die einzelnen Modi wird nun die eigentliche Benutzerschnittstelle beschrieben. Begonnen werden soll hierbei mit den allgemeinen Funktionen, um von diesen schließlich auf die spezielle Funktionalität der implementierten Modi zu gelangen.

5.3.1 Allgemein

Die Steuerung von CREWW erfolgt über die Eingabegeräte Maus, Tastatur und Wiimote. Über Maus und Tastatur können das Hauptmenü angesprochen und CRC-Komponenten manipuliert werden, während über die Wiimotes die kollaborative Steuerung des Programms realisiert wird. Die Wiimote wird hier ähnlich wie ein Laserpointer eingesetzt, zeigt man auf den Bildschirm – bzw. beim Beamer auf die projizierte Fläche – erscheint dort ein Cursor (`WiiCursor`), der mittels Knopfdruck Interaktion ermöglicht.

Im Folgenden werden nun die Funktionen beschrieben, die in beiden Modi verfügbar sind. Hierbei gilt lediglich die Einschränkung, dass im Use-Case-Mode CRC-Karten-bezogene Wiimote-Funktionen (Löschen, Verschieben, Auf-/Zuklappen) nur dem Besitzer der jeweiligen Karte gestattet sind.

Zunächst erfolgt die Beschreibung der Maus- und Tastaturfunktionen.

`Save:` Diese Funktion ermöglicht das Abspeichern des aktuellen Modells inklusive eines eventuell noch laufenden oder abgeschlossenen Use-Case zur späteren Fortsetzung oder Bearbeitung. Die genaue Umsetzung dieser Funktion wird in Kap. 6 näher erläutert.

`Load:` Hiermit ist das Laden eines zuvor abgespeicherten Szenarios möglich. Wurde ein aktiver Use-Case mit abgespeichert und waren an diesem Use-Case mehr Wiimotes beteiligt als zum Ladezeitpunkt vorhanden, wird der Use-Case zurückgesetzt.

`Export to XMI:` Eine Forderung an CREWW ist die Bereitstellung einer Schnittstelle zu UML. Hierfür bietet diese Funktion den Export nach XMI 1.2 . Solcherhand exportierte Dateien im XML-Format können dann in einer aktuellen Version von ArgoUML importiert werden.

`Empty CRC Card:` Mit dieser Funktion lässt sich eine neue CRC-Karte anlegen. Diese ist zu Beginn ohne Inhalt und kann mit den weiter unten beschriebenen Funktionen zur Manipulation der Responsibilities/Collaborators verändert werden. Diese Funktion ist zugänglich sowohl über das Hauptmenü als auch über das Drücken des Wiimote-Knopfes (+) über mindestens eine Sekunde. Wird die neue Karte über das Hauptmenü erzeugt, erscheint sie oben links auf der Arbeitsfläche, wird sie über die Wiimote erzeugt, erscheint sie an der Stelle, an der sich der auslösende Wiimote-Cursor befindet.

`Insert from Scanner:` Die Erstellung einer neuen CRC-Karte ist auch durch den Scanner möglich. Dies geschieht wie das Erstellen einer leeren CRC-Karte

über das Hauptmenü oder den (+)-Knopf der Wiimote. Ist eine Handschrifterkennung installiert, wird diese automatisch auf die neue Karte angewendet und das Erkennungsergebnis angezeigt.

`Adjust Smoothing:` Über diese Option lässt sich die Glättung der Wiimote-Bewegungen justieren. Die Glättung ist nötig, da die Wiimote feinste Handbewegungen erfasst und somit ohne Glättung ein starkes Zittern der Cursor zu sehen wäre. Die Glättung wird realisiert durch die Berechnung eines gleitenden Mittelwerts, genaueres hierzu findet sich in Kap. 6.6.2.

`Set User Names:` Zur besseren Identifizierbarkeit der Wiicursor lassen sich mit dieser Funktion die Namen der Benutzer angeben. Über einen Druck auf die Nach-Unten-Taste der Wiimote wird der Name des Benutzers unter dem zugehörigen Cursor eingeblendet.

`Reactivate Wiimotes:` Zum Zeitpunkt der Entwicklung von CREWW war der WiiServer noch fehleranfällig, so lieferten nicht immer alle Wiimotes auswertbare Infrarotdaten. Mit dieser Funktion lassen sich die Sensoren der Wiimotes erneut aktivieren, wodurch in manchen Fällen eine Behebung des beschriebenen Fehlers möglich ist.

`Print Set of CRC Cards:` Mit dieser Funktion lässt sich ein (oder mehrere) DIN A4-Blatt mit je vier Blanko-CRC-Karten ausdrucken. Diese genormten Karten werden benötigt, wenn mit Handschrifterkennung gearbeitet werden soll, da in diesem Fall das Layout fest vorgegeben ist.

So viel zu den allgemein über Maus und Tastatur zugänglichen Funktionen, nun zur Wiimote-Funktionalität.

Löschen einer CRC-Karte: Angelegte CRC-Karten lassen sich selbstverständlich auch löschen, dies geschieht durch langes Drücken der Wiimote-Taste (–) auf der Komponente.

Ist die Komponente Teil eines aktiven Use-Case, erscheint vor dem Löschen eine Sicherheitsabfrage, da der aktive Use-Case durch Löschen einer beteiligten Komponente beendet wird.

Komponentenansicht ändern: Mit der Taste (A) der Wiimote lässt sich die Komponentenansicht ändern. Umgeschaltet werden kann zwischen zwei bis drei Ansichten: Vollständige Ansicht, minimierte Ansicht (nur Klassenname sichtbar) und Bildansicht (wenn die Karte gescannt wurde).

Verschieben der Komponenten: Um die CRC-Komponenten auf dem Bildschirm anzuordnen, hat jeder Benutzer die Möglichkeit, die Karten mit Taste (B) zu verschieben. Hierbei gilt: Hat ein Benutzer eine Komponente „in der Hand“ (Drag aktiv), ist sie für andere Benutzer gesperrt.

Desweiteren gibt es bezüglich Verschieben eine Einschränkung im Use-Case-Modus: Hier können nur Komponenten verschoben werden, die bereits vom Benutzer in Besitz genommen wurden.

Panel einblenden: Über Taste (Home) lässt sich ein von der Wiimote bedienbares Menü (`OptionPanel`) am oberen Bildschirmrand einblenden. Dieses Menü erlaubt das Umschalten der Modi sowie die Navigation im Use-Case (Vorlauf, Rücklauf, Start, Pause, ...).

Modus umschalten: Im Folgenden werden die beiden von CREWW unterstützten Modi beschrieben. Neben den Knöpfen im OptionPanel ist das Umschalten auch mittels Drücken der Taste (2) der Wiimote möglich.

5.3.2 UML-Mode

Soweit zu den Funktionen, die den Benutzern jederzeit zur Verfügung stehen. Einige Funktionen jedoch beschränken sich auf einen einzelnen Modus. Nun soll näher auf die im UML-Modus zur Verfügung stehenden Funktionen eingegangen werden. Beim Anlegen und Löschen der UML-Kanten gelten für alle Benutzer folgende Einschränkungen: Hat ein Benutzer eine Komponente zum Löschen oder Erzeugen einer Kante angewählt, ist diese Komponente bis zum Abschluss des Vorgangs für alle anderen Benutzer gesperrt. Ebenso ist der Cursor des Benutzers, der die Komponente gewählt hat, bis zur Beendigung des Vorgangs für andere Aktionen gesperrt.

UML-Kante anlegen / ändern: Der UML-Modus erweitert die CRC-Methode um einige Aspekte der UML. Ähnlich wie in einem UML-Klassendiagramm (siehe Kap. 2.1.3) können Beziehungen zwischen CRC-Komponenten dargestellt werden. Hierfür werden nacheinander mit Taste (+) zwei Komponenten markiert. Daraufhin erscheint ein Menü, in dem der gewünschte Typ (Assoziation, Aggregation, Vererbung) gewählt werden kann.

Hierbei gilt einschränkend, dass zwischen je zwei Klassen nur eine Beziehung möglich ist. Existiert schon eine Kante zwischen den gewählten Klassen, erfolgt eine Abfrage, die die Änderung des Kantentyps oder der Kantenrichtung erlaubt.

UML-Kante löschen: Die erzeugten UML-Kanten lassen sich wenn nötig auch wieder löschen: Mittels Auswahl der beiden beteiligten Komponenten durch die Taste (–) der Wiimote kann die Kante wieder aus dem Modell entfernt werden.

5.3.3 Use-Case-Mode

Zentraler Nutzen von CREWW ist wie an voriger Stelle bereits erwähnt das Durchspielen einer CRC-Arbeitssitzung. Die hierfür benötigten Funktionen leiten sich zum einen aus der CRC-Methode an sich, zum anderen aus den in Kap. 3.6 herausgearbeiteten Vor- und Nachteilen der traditionellen CRC-Methode ab.

Zunächst einmal unterteilt CREWW den Use-Case-Mode in Start- und Pause-Modus. Dies liegt darin begründet, dass es im Verlauf des Rollenspiels öfter vorkommt, dass Klassen hinzugefügt werden müssen. Allerdings sollte die Steuerung während des Anwendungsfalles nur bei einer Person liegen und möglichst keine verteilte Modellierung möglich sein. Dieses Problem löst der Pause-Modus. Im Pause-Modus können Klassen erzeugt und verteilt werden und im Play-Modus findet ausschließlich das Durchspielen des Use-Case statt.

Klassen in Besitz nehmen: Im Pause-Modus kann eine Klasse ohne Besitzer von einem Benutzer in Besitz genommen werden, indem er sie mit (+) anwählt. Die Karte wird dem Benutzer zugeordnet und diese Zuordnung bleibt auch bei einem Moduswechsel sowie beim Laden und Speichern erhalten. Visualisiert wird die Zuordnung, indem die Karte in der Farbe des Cursors des Benutzers eingefärbt wird.

Klasse abgeben: Das Abgeben der Klasse ist ebenfalls nur im Pause-Modus möglich und wird erreicht durch Auswählen der Karte mit der Taste (–). Die Karte wird nun wieder neutral in schwarz dargestellt. Ist die Karte Teil eines aktiven Use-Case erfolgt hier eine Sicherheitsabfrage, da der Use-Case im Falle der Abgabe gelöscht werden muss.

Anwendungsfall Vor- und Rücklauf: CREWW bietet die Möglichkeit, in aktiven Anwendungsfällen zu navigieren. So ist der Vorlauf über die Wiimote-Taste (Rechts) und der Rücklauf über die Taste (Links) möglich. Beide Funktionen können auch über das `OptionPanel` aufgerufen werden. Zusätzlich bietet dieses die Möglichkeit, direkt an das Ende oder den Anfang des Use-Case zu navigieren. Die beschriebenen Funktionen stehen dem Benutzer sowohl im Play- als auch im Pause-Modus zur Verfügung.

Anwendungsfall Reset: Das Zurücksetzen des Anwendungsfalles ist einzig über das OptionPanel und wie die zuvor beschriebenen Funktionen in beiden Modi (Play und Pause) möglich. Mit Zurücksetzen des Use-Case wird der für den aktiven Anwendungsfall erzeugte Stapel gelöscht und der zurücksetzende Benutzer ist an der Reihe.

Verantwortlichkeit weitergeben / erfüllt: Die einzige für den Play-Modus zusätzlich nötige Funktion ist die Delegation einer Verantwortlichkeit, bzw. die Signalisierung der kompletten Erfüllung einer Verantwortlichkeit. Beides geschieht über Auswahl einer Komponente mit der Taste (+). Soll eine Verantwortlichkeit delegiert werden, wird die Klasse, die die Verantwortlichkeit erfüllen soll, mit (+) angewählt. Entscheidet sich der Benutzer, die Verantwortlichkeit allein zu erfüllen, wählt er die eigene Klasse mit (+) an, und die ihn aufrufende Klasse kommt erneut an die Reihe.

5.3.4 Übersicht

Zum Abschluss dieses Kapitels nun noch eine Übersicht der in den jeweiligen Modi möglichen Aktionen. Die mit „B“ markierten Felder bedeuten, dass die Aktion nur von der Wiimote ausgeführt werden kann, die im Besitz der CRC-Karte ist.

Tabelle 5.1: Die Funktionen von CREWW

Funktionen	UML	Use-Case	
		Pause	Play
`File->Save`	✓	✓	✓
`File->Load`	✓	✓	✓
`File->Export to XMI`	✓	✓	✓
`Insert->Empty CRC Card`	✓	✓	×
`Insert->From Scanner`	✓	✓	×
`Options->Adjust Smoothing`	✓	✓	✓
`Options->Set User Names`	✓	✓	✓
`Options->Reactivate Wiimotes`	✓	✓	✓
`Print->Set of CRC Cards`	✓	✓	✓
CRC-Karte anlegen	✓	✓	×
CRC-Karte löschen	✓	B	×
Ansicht ändern	✓	B	B
CRC-Karte verschieben	✓	B	B
Panel einblenden	✓	✓	✓
Modus umschalten	✓	✓	✓
UML-Kante erstellen	✓	×	×
UML-Kante löschen	✓	×	×
Use-Case starten / anhalten	×	✓	✓
Use-Case Vorlauf	×	✓	✓
Use-Case Rücklauf	×	✓	✓
Use-Case Reset	×	✓	✓
CRC-Karte in Besitz nehmen	×	✓	×
CRC-Karte abgeben	×	B	×
Verantwortlichkeit delegieren	×	×	B

Kapitel 6

Implementierung von CREWW

Grundlagen erarbeiten, Methoden erklären, Anforderungen formalisieren: Fleißarbeit muss auch sein, doch ist sie nicht des Pudels Kern. Die wirklich interessanten Fragen lauten doch: Wie funktioniert das? Welche Tricks und Kniffe wurden angewendet? Wo liegen die wahren Knackpunkte?

All das wird in diesem Kapitel herausgearbeitet. Die komplette Implementierung wird vorgestellt, wobei zunächst ihre zentralen Aspekte beschrieben werden: Wo lagen die Schwierigkeiten der Implementierungsphase? In den darauf folgenden Abschnitten werden die Komponenten von CREWW und die Anbindung der externen Bibliotheken beschrieben.

6.1 Zentrale Aspekte der Implementierung

Ohne bereits auf Details einzugehen erfolgt an dieser Stelle zunächst eine Einführung in die zentralen Problembereiche, die sich im Verlauf der Implementierungsphase in den verschiedenen Programmteilen zeigten. Erneut soll an dieser Stelle darauf hingewiesen werden, dass die Zusammenführung von sich bis dahin fremden Methodenbereichen – CRC, UML, CSCW – *das* Alleinstellungsmerkmal von CREWW darstellt. Es wird keine neue Methode entwickelt, sondern existierende Methoden in einen zuvor nicht bestehenden Kontext gesetzt. Aus den Schnittstellen zwischen diesen Methoden ergeben sich die meisten Problembereiche.

Zugriffsprotokoll: Einer dieser Bereiche war die Kollaborationsunterstützung durch die Software in einer computerunterstützten CRC-Sitzung. Hier mussten von der Wiimote steuerbare Komponenten mit Zugriffssynchronisation entwickelt werden. Denn anders als im herkömmlichen (1 User – 1 Cursor) müssen im kollaborativen Fall (n User – n Cursor) Vorkehrungen getroffen werden, für den Fall, dass ein Benutzer eine Komponente manipulieren will, die bereits von einem anderen

Benutzer gesteuert wird.

Hierfür wurden in der Klasse `CRCComponent`, die die CRC-Karten realisiert, die Attribute `locked`, `owner` und `activeUser` eingeführt. In diesen wird vermerkt, in wessen Besitz die Komponente ist, wer aktuell versucht, die Komponente zu manipulieren und ob sie für andere Benutzer zur Verfügung steht. Durch Abfrage dieser Variablen vor einer Operation wird somit die Einführung eines Protokolls ermöglicht, das festlegt, *wer* zu *welchem* Zeitpunkt *welche* Rechte auf einer Komponente hat. Eine genauere Erklärung dieses Protokolls und weiteres zur Klasse `CRCComponent` findet sich in Kap. 6.5.1.

Anwendungsfallsteuerung: Ebenfalls angesprochen werden muss in diesem Kapitel die Steuerung des Kontrollflusses im Anwendungsfall. Hier soll ein Token-basiertes Verfahren Anwendung finden, in dem immer ein Benutzer „an der Reihe" ist. Eine Weitergabe des Tokens sowie eine Rückgabe an den Vorbesitzer soll implementiert werden. Weiterhin soll ein Undo/Redo, folglich eine Navigation im aktiven Use-Case, möglich sein.

Umgesetzt wurde dies in der Klasse `UseCaseStack`. Diese verwaltet den aktuellen Anwendungsfall mittels zweier ArrayLists, in denen die am Use-Case beteiligten CRC-Klassen gespeichert werden. Weiterhin stellt sie Methoden zum Vor- und Rücklauf im Anwendungsfall zur Verfügung. Für eine genaue Beschreibung dieser Methoden sowie der Datenstruktur „UseCaseStack" sei hier auf Kap. 6.5.3 verwiesen.

Scanneranbindung mit JTwain: Ein weiterer kritischer Punkt ist die gewünschte Scan-Funktion: Das Einscannen zuvor angefertigter CRC-Karten soll in CREWW möglich sein. Dies erweist sich insofern als problematisch, als die Kommunikation mit Bildeingabegeräten in Java nur über Umwege möglich ist. Letztendlich wird die gewünschte Funktionalität mittels der Schnittstelle JTwain realisiert, die ihrerseits wiederum einen Wrapper für die TWAIN-Schnittstelle darstellt. Wie hierbei vorgegangen wird und Lehrreiches zu dynamischen Bibliotheken unter Windows findet sich in Kap. 6.3.

WiiServer-Reparaturen: Nicht immer – aber immer öfter – liegen Probleme da, wo man sie am wenigsten vermutet. In diesem Fall wurde von einer lauffähigen Version des WiiServer, der das softwareseitige Backend zur Kommunikation mit den Wiimotes darstellt, ausgegangen. Allerdings waren hier Instandsetzungsarbeiten in größerem Umfang von Nöten, da es gerade bei Anbindung mehrerer Wiimotes häufig zu Fehlern kam. Um diese zu beheben mussten wiederum zuerst Versionskonflikte beim benutzten Compiler erkannt und beseitigt werden. Ein Vorgang, der sich entgegen der Annahme als kompliziert herausstellte und in Kap. 6.2

näher beschrieben wird.

6.2 Vorarbeiten

Wie an voriger Stelle bereits erwähnt, nutzt CREWW die von Sabine Müller im Rahmen ihrer Diplomarbeit implementierte Client/Server-Architektur SIGMAii. Allerdings zeigte sich diese bei ersten Tests noch fehleranfällig. So gab es Probleme bei der Verbindung mit mehreren Wiimotes: Ein oder mehrere Cursor konnten nicht durch den Benutzer bewegt werden, obwohl die Wiimotes durch den Server erkannt wurden. Somit waren, bevor die eigentliche Implementierung begonnen werden konnte, zunächst Reparaturarbeiten am Server fällig.

Hierbei zeigte sich direkt ein grundlegendes Problem: Der C-basierte Server wurde mit Microsoft Visual Studio.NET 2003 (VS.NET2003) entwickelt und die älteste durch die Universität erhätliche Version dieser Entwicklungsumgebung ist VS.NET2003 Professional. Allerdings ließ sich der Server mit dieser Version nicht kompilieren. Nach eingehender Recherche konnten einige der Fehlerquellen identifiziert werden:

Die verschiedenen Versionen von Visual Studio nutzen selbstverständlich verschiedene VisualC-Compiler. Bei diesen ist interessanterweise keineswegs von einer Abwärtskompatibilität auszugehen. Tatsächlich benutzt VS.NET2003 den VisualC6.0-Compiler (VC60), während VS.NET2003 Professional den neueren VisualC7.0-Compiler (VC70) benutzt und gerade bei diesem Versionssprung gibt es einige gravierende Änderungen. So arbeitet der VC70 mit grundsätzlich anderen C-Strings als der VC60, und einige sicherheitskritische Pointer-Operationen sind beim neueren VC70 nicht mehr erlaubt.

Weiterhin ist der Server in C geschrieben und im Include-Bereich wird ein C++-Header (`<iostream>`) eingebunden. Dies führt zwangsläufig zu einem Fehler, da C++ entgegen weitläufiger Meinung *keine* bloße Erweiterung von C, sondern eine eigenständige Programmiersprache mit Gemeinsamkeiten zu C ist. Wie der Server zuvor kompiliert werden konnte, ist nicht ermittelbar – möglicherweise war der ältere VC60 in dieser Hinsicht fehlertoleranter.

Außerdem finden sich im Code diverse z. T. gravierende Fehler, an einigen Stellen stimmt z. B. die Anzahl der Dereferenzierungen nicht oder Felder wurden nicht ausreichend groß alloziert, so dass es zu Speicherüberläufen kam. Auch hier scheint die einzig mögliche Erklärung für die mit VC60 erfolgte Kompilierung zu sein, dass dieser eine erheblich höhere Fehlertoleranz hat.

Nach Feststellung dieser Mängel zeigten sich drei mögliche Auswege:

1. **Auftreiben einer Entwicklungsumgebung, die den VC60-Compiler unterstützt**
   Für diesen Weg würde der wegfallende Implementierungsaufwand sprechen. Der Server muss nicht angepasst, sondern es können direkt die gewünschten Codezeilen geändert werden. Gegen diesen Weg spricht jedoch zum einen die Tatsache, dass auf einen veralteten Compiler zu setzen nicht als zukunftssicher anzusehen ist. Möglicherweise wird der veraltete Compiler von neuen Betriebssystemen (z. B. Windows 7) gar nicht mehr unterstützt. Zum anderen würden hierdurch die bestehenden Implementierungsfehler nicht behoben, und es ist noch nicht geklärt, inwieweit diese das Fehlverhalten des Servers beeinflussen. Somit überwogen die Nachteile die Vorteile der beschriebenen Lösung, weshalb dieser Weg ausgeschlossen wurde.

2. **Neuimplementierung des Servers**
   Ein weiterer Ausweg wäre die Neuimplementierung des WiiServer in einer aktuellen Entwicklungsumgebung. Hiernach könnte zumindest von einer sauberen, lauffähigen Architektur ausgegangen werden. Gegen diese Lösung spricht allerdings der erhebliche Implementierungsaufwand, zentrales Thema der vorliegenden Arbeit sollte schließlich nicht der WiiServer, sondern die Entwicklung einer eigenständigen Software sein. Damit entfällt auch der zweite Lösungsansatz.

3. **Debugging und Kompilierung mit VC70-Compiler**
   Der dritte und schließlich eingeschlagene Weg ist die Anpassung des WiiServer an einen aktuell erhältlichen Compiler. Dieser Weg beinhaltet die Auflösung der bestehenden Fehler und Header-Konflikte, erscheint aber dennoch als guter trade-off zwischen nötigem Implementierungsaufwand und erzieltem Mehrwert.

Als Entwicklungsumgebung wurde VS.NET2003 Professional gewählt. Das Bugtracking zeigte sich weniger umfangreich als zunächst angenommen, so stand schon nach einem Tag eine kompilier- und lauffähige Version des Servers zur Verfügung. Nun konnten die eigentlichen Programmanpassungen vorgenommen werden.

Erste Schwierigkeit war die Eingrenzung des Fehlers: Wie in Kap. 2.4 beschrieben sind die Wiimotes durch eine Kette von Komponenten (Wiimote -> Bluetooth-Stack -> Wiiuse -> WiiServer -> WiiClient) an die Software angebunden, wobei zunächst nicht klar war, wo in dieser Kette der Fehler lag. Als Erstes mussten also möglichst viele Kandidaten für Fehlerquellen ausgeschlossen werden. Die Wiimotes wurden vertauscht, es wurden unterschiedliche Bluetooth-Stacks benutzt (Windows-XP-Stack und Bluesoleil) und an verschiedenen Stellen im WiiClient wurden Testausgaben eingebaut. Die beschriebenen Komponenten zeigten keine Auffälligkeiten, womit klar war, dass das Problem am Server oder an der freien

Bibliothek Wiiuse lag. Für die Bibliothek Wiiuse stand kein Ersatz zur Verfügung, daher wurde die Fehlersuche auf den WiiServer eingeschränkt.

Hier wurde das Problem ebenfalls mittels einer Testausgabe eingegrenzt. Es zeigte sich, dass die fehlerhaften Wiimotes durchaus Daten sendeten, welche allerdings keine Informationen der Infrarotsensoren enthielten. Daher wurde in der Initialisierungsphase des WiiServer eine Schleife eingebaut, die erneut eine Nachricht zum Anschalten der Infrarot- und Beschleunigungssensoren an die Wiimotes sendet. Außerdem wurde der WiiClient um die Nachricht SHUTDOWN – und entsprechend um die öffentliche Funktion `shutdown_server()` – erweitert, die ein geregeltes Herunterfahren des Server ermöglicht. Dieses geregelte Herunterfahren schließt die Deaktivierung der Wiimote-Sensoren sowie den Befehl `wiiuse_cleanup()` zum Abmelden der Wiimotes mit ein. Zuvor konnte der Server nur von Hand (Strg + C) heruntergefahren werden, dies ohne eine Abmeldung der Wiimotes.

Durch die beschriebenen Änderungen konnte der Fehler zwar nicht vollständig behoben werden, so tritt er noch sporadisch auf, wenn alle vier Wiimotes mit dem Server verbunden werden. Allerdings bilden die Änderungen, zusammen mit der Möglichkeit, die IR-Sensoren zur Laufzeit erneut zu aktivieren, ein akzeptables Workaround.

Abschließend sei hier gesagt, dass die Implementierung eines robusteren Servers für die Zukunft angestrebt werden sollte. Ansatzpunkte für die Behebung des Fehlers sind zum einen die Befehle zum An- und Abschalten der Wiimotes. Diese können, wenn mehrere Wiimotes verbunden sind, nicht immer von allen Wiimotes richtig verarbeitet werden. Zum anderen ist es die Tatsache, dass fast immer die Wiimote mit der ID „4“ nicht richtig angesprochen werden kann. Da die Initialisierung der Wiimotes immer mit ID „1“ beginnt, könnte dies darauf hinweisen, dass an einer Stelle die Zeit bis zur nächsten zu verarbeitenden Nachricht zu kurz bemessen ist.

6.3 Dynamische Bibliotheken

Zur Anbindung externer Funktionalität gibt es in Programmiersprachen, aber auch ganz allgemein in Betriebssystemen, unterschiedliche Konzepte. Eines davon sind Bibliotheken, Sammlungen von Funktionen und Strukturen.

Man unterscheidet hier zwischen statischen Bibliotheken und dynamischen Bibliotheken. Statische Bibliotheken werden beim Kompilieren in den Programmcode eingebunden: Findet der Compiler im Code eine Stelle, an der auf eine statische Bibliotheksfunktion zugegriffen wird, so fügt er den Bibliothekscode an die entsprechende Stelle im Quellcode ein. Dynamische Bibliotheken werden nicht beim Kompilieren, sondern zur Laufzeit gebunden. Beim Starten des Programms, bzw. im Falle der „faulen“ Bindung (*lazy binding*) bei Aufruf der Bibliotheksfunktion,

werden die benötigten dynamischen Bibliotheken, falls noch nicht vorhanden, in den Speicher geladen und dem Programm die Eintrittspunkte (Speicheradressen) übergeben. Über den Eintrittspunkt und den Funktionsnamen kann das Programm dann eine Funktion der Bibliothek aufrufen.

Der Vorteil solcher dynamischer Bibliotheken liegt auf der Hand: Da der Bibliothekscode nicht in den Quellcode hinein kopiert wird, sind die kompilierten Dateien kleiner, und da der Bibliothekscode im Normalfall nicht mehr verändert werden darf erhält man, gerade in Systemen mit virtueller Speicherverwaltung, zusätzlichen Performancegewinn. Denn wenn mehrere Prozesse die Bibliothek benötigen, reicht es aus, diese einmal physisch im Speicher zu halten und in die Adressräume der einzelnen Prozesse einzublenden.

Im Folgenden wird nun erklärt, wie diese dynamische Bindung im Falle von Java aussieht und wie die in CREWW genutzten Bibliotheken implementiert wurden.

6.3.1 Einbinden von dynamischen Bibliotheken mittels JNI

Die Programmiersprache Java benötigt an sich keine expliziten dynamischen Bibliotheken, da Bindung in Java immer dynamisch, also zur Laufzeit, erfolgt. Benötigte Klassen werden von der *Java Virtual Machine* (JVM) bei Bedarf entweder direkt oder aus Klassenbibliotheken (.jar-Dateien) nachgeladen. Der Einsatz beliebiger dynamischer Bibliotheken ist in Java nicht möglich, da Java Eintrittspunke, aber nicht den zum Aufruf nötigen Funktionsnamen kennt.

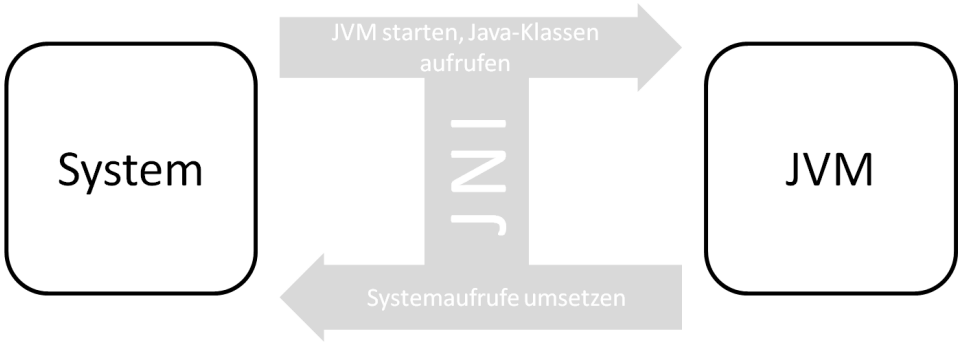


Abbildung 6.1: Das Java Native Interface

Wegen der Einschränkung auf die JVM werden manche systemspezifischen Funktionen – wie Zugriff auf den Scanner – allerdings nicht von Java unterstützt. Für solche Fälle fordert Java von den die JVM implementierenden Betriebssystemen die Schnittstelle zu dynamischen Bibliotheken *Java Native Interface* (JNI). Diese Schnittstelle funktioniert in beide Richtungen (siehe Abb. 6.1), so kann eine Sys-

temfunktion (z. B. in C++) über eine dynamische Bibliothek (im Falle Windows: *jvm.dll*) eine JVM starten und in dieser Java-Klassen aufrufen. In der anderen Richtung muss es einer Java-Klasse möglich sein, über eine dynamische Bibliothek eine Systemfunktion aufzurufen.

Im vorliegenden Fall ist die zweite Variante vonnöten. Hierfür wird eine Java-Klasse angelegt, die dann für andere Klassen den Eintrittspunkt für die Bibliothek darstellt, das dynamische Laden bewerkstelligt und die Funktionsnamen bereitstellt. Im Folgenden soll kurz die Herstellung einer solchen dynamischen Bibliothek, zunächst im Allgemeinen, danach im Speziellen für Scanner und Handschrifterkennung, skizziert werden (siehe Abb. 6.2).

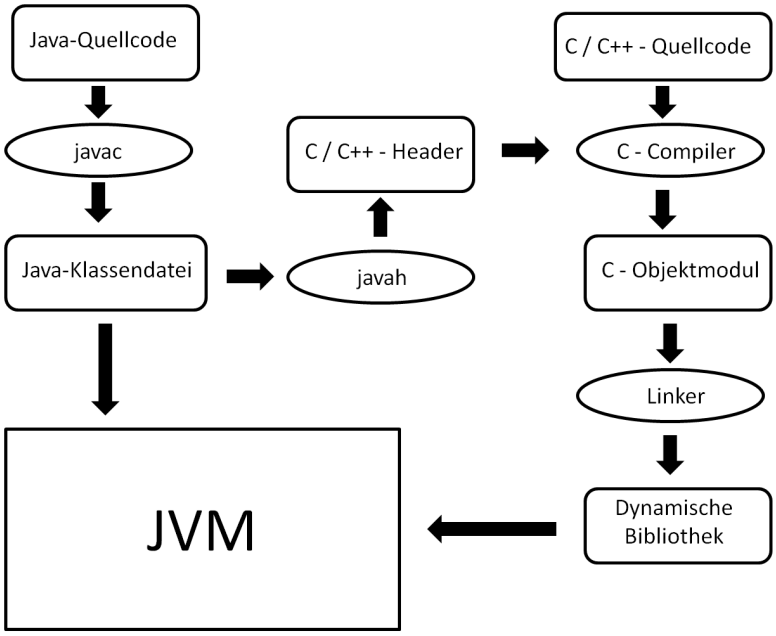


Abbildung 6.2: Erstellen einer DLL für Java

6.3.2 Erstellung von dynamischen Java-Bibliotheken

Um eine dynamische Bibliothek in C oder C++ zu erzeugen, benötigt man zunächst eine Java-Wrapper-Klasse. Die Wrapper-Klasse enthält sämtliche Bibliotheksfunktionen, auf die der Zugriff möglich sein soll, als Klassenmethoden. Diese Methoden müssen mit dem Schlüsselwort „`native`“ gekennzeichnet werden. Weiterhin wird für gewöhnlich in der Wrapper-Klasse eine statische Methode `init()` zur Verfügung gestellt, die für das Laden der dynamischen Bibliothek

verantwortlich ist, was wiederum durch den Systemaufruf `System.loadLibrary()` realisiert wird.

```
1  public class Recognition
2  {
3    public static boolean init ()
4    {
5          try
6          {
7              System.loadLibrary ("recognition");
8              return true;
9          }
10         catch (UnsatisfiedLinkError e)
11        {
12             return false;
13         }
14   }
15
16   public static native String acquireDialog();
17 }
```

Listing 6.1: Wrapper-Klasse – Recognition.java

Die Wrapper-Klasse wird nun kompiliert und so die zugehörige .class-Datei erzeugt. Aus dieser kann wiederum mittels Kommandozeilenaufruf

```
javah -o <Bibliotheksname> <Klassenname>
```

die Headerdatei erzeugt werden. Die im Header erzeugten Methoden haben folgende Signatur:

```
JNIEXPORT <Rückgabetyp> JNICALL
Java_<Paketname>_<Klassenname>_<Funktionsname>(JNIEnv*, jclass,
<evtl. Parameter>);
```

Die Headerdatei kann auch manuell erstellt werden, allerdings ist hier die Einhaltung der beschriebenen Namenskonvention von größter Wichtigkeit, da Java sonst keinen Zugriff auf die Funktionen der Bibliothek erhält.

`JNIEnv` ist ein Pointer auf die aufrufende JVM, über diese kann die C-Funktion auf die gesamte Java-API zugreifen. Als Rückgabetypen stellt JNI zunächst den Standardbasistypen entsprechend `jboolean`, `jint`, `jbyte` etc. zur Verfügung. Für erweiterte Datentypen existieren die Abstraktionen `jobject` für einzelne Objekte und `jobjectArray` für Felder von Objekten.

In der zum Header gehörenden Quellcodedatei können nun die Funktionen implementiert werden. Aus Quellcode- und Headerdatei können dann erst Objektdatei und schließlich die dynamische Bibliothek erzeugt werden.

Möchte man die Bibliotheksfunktionen im Java-Programm aufrufen, so muss zunächst sichergestellt werden, dass die Bibliothek – z. B. über die init()-Funktion – geladen wurde. Ist dies der Fall, ist über die Wrapper-Klasse der Aufruf sämtlicher Bibliotheksfunktionen, die wie oben beschrieben deklariert wurden, möglich. Somit erhalten wir eine Java-Klasse, die mittels JNI auf beliebige in C implementierte Funktionen zugreifen kann.

6.3.3 Die Scanneranbindung `JTwain`

> „*Oh, East is East, and West is West, and never the* ***twain*** *shall meet!*“
>
> ——Rudyard Kipling, The Ballad of East and West

„... und niemals werden die Zwei sich treffen!“ So in etwa ließe sich der zweite Teil des obigen Zitats übersetzen. Dafür, dass sie zumindest kommunizieren können, sorgt auf Windows und Macintosh PCs seit 1992 TWAIN [TWA], eine Schnittstelle zum Datenaustausch zwischen Bildeingabegeräten und Anwendungssoftware.

„Twain“ steht eigentlich altenglisch für „zwei“, soll in Bezug auf obiges Zitat aber auf die Schwierigkeit der Kommunikation zwischen solchen Geräten auf der einen und Software auf der anderen Seite hinweisen. Realisiert wird sie bei TWAIN durch das Zusammenspiel von drei Komponenten: Anwendung, Gerätemanager (data source manager) und Datenquelle (data source). Hierbei realisiert TWAIN nicht die eigentliche Steuerung der Komponenten, sondern liefert das Protokoll, um zwischen diesen zu vermitteln. Die Software für die Datenquelle (Treiber) wird vom Hersteller geliefert und die Gerätemanagersoftware wird entweder vom Betriebssystem oder ebenfalls vom Hersteller zur Verfügung gestellt. Hierbei verwaltet der Treiber das Gerät, ist somit für den eigentlichen Scanvorgang und Einstellungen am Gerät verantwortlich, während die Gerätemanagersoftware die Schnittstelle zur Anwendung darstellt und die Verwaltung mehrerer Datenquellen ermöglicht (siehe Abb. 6.3).

Die Vorgehensweise beim Scannen eines Dokumentes ist hierbei folgende:

Zunächst wird der Gerätemanager geöffnet, dieser kann dann den Namen des Standardeingabegerätes beim System erfragen. Mit dem Gerätenamen lässt sich dann eine Datenquelle für das Gerät anlegen und öffnen. Ist die Datenquelle geöffnet, können am Gerät Einstellungen vorgenommen oder es kann mit dem Gerät gescannt werden. Nach Beendigung des Vorgangs muss die Datenquelle wieder geschlossen werden, ein Schließen des Gerätemanagers direkt nach dem Scanvorgang ist optional, muss jedoch zumindest bei Beendigung des Programms erfolgen.

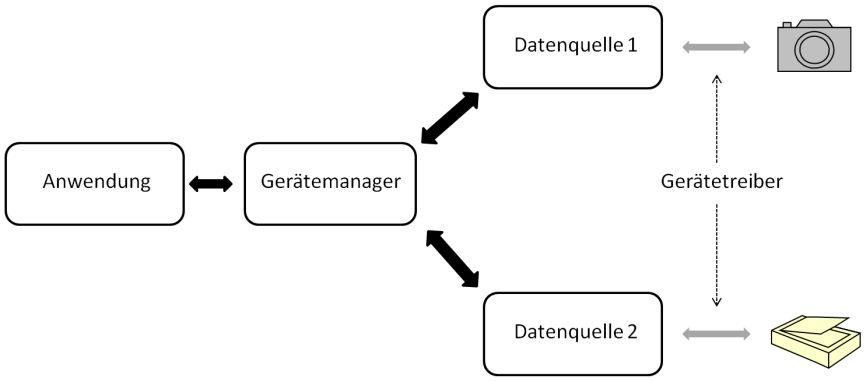


Abbildung 6.3: Die Komponenten von TWAIN

Die Funktionalität von TWAIN ist in Windows-Systemen in der dynamischen Bibliothek *twain.dll* zu finden, allerdings lässt sich auf diese nicht direkt in Java zugreifen, da kein Eintrittspunkt für Java-Klassen vorhanden ist (siehe Kap. 6.3.1). Diese Lücke überbrückt die Scannerschnittstelle JTwain [JTw], ein von Jeff Friesen entwickelter Wrapper für die Windows TWAIN-Schnittstelle. Auch wenn JTwain nicht die volle TWAIN-Funktionalität bietet, so unterstützt es zumindest die Grundfunktionen zum Scannen einzelner oder mehrerer Bilder sowie zum Öffnen und Schließen von Datenquellen und Gerätemanager.

Auf der Webseite [JTw] findet sich ein Tutorial, wie sich JTwain leicht auf den eigenen Rechner portieren lässt, allerdings zeigte sich hier ein Problem: Es existiert eine ältere Version von JTwain mit begrenztem Funktionsumfang, diese bietet folgende Methoden an:

- `public static boolean init()`
  Lädt jtwain.dll in den Speicher

- `public static native Image acquire()`
  Öffnet das Menü für Scannereinstellungen, aus dem heraus auch das Scannen möglich ist

- `public static native void selectSourceAsDefault()`
  Öffnet das Menü zur Auswahl des Standardeingabegerätes

Diese Funktionen erlauben zwar Betrieb, jedoch keine komfortable Handhabung des Scanners. Um ein Bild zu Scannen, muss jedes mal das Scanmenü mit den Einstellungen (Helligkeit, Kontrast usw.) aufgerufen werden.

Dieses Problem sollte mit der nächsten Version von JTwain gelöst werden, hier wurden einige zusätzliche Funktionen bereitgestellt, wobei die folgenden von Interesse sind:

- `public static native Image [] acquire()`
  Scannt eines oder mehrere Bilder ein
- `public static native void closeDS()`
  Schließt die aktuell offene Datenquelle
- `public static native void closeDSM()`
  Schließt den Gerätemanager
- `public static native String getDefaultDS()`
  Erfragt das Standardeingabegerät
- `public static native String getFirstDS()`
  Liefert das erste vorhandene Eingabegerät
- `public static native String getNextDS()`
  Liefert das nächste vorhandene Eingabegerät
- `public static native void openDS(String srcName)`
  Öffnet das Eingabegerät mit Namen „srcName“
- `static native void openDSM()`
  Öffnet den Gerätemanager

Positiv fällt hier auf, dass das Menü für Scannereinstellungen nicht mehr bei jedem Scanvorgang geöffnet und mit den Funktionen zum Öffnen und Schließen von Datenquelle und Gerätemanager ein feingranulareres Vorgehen beim Einscannen ermöglicht wird. Negativ ist allerdings anzumerken, dass weder Funktionen zum Ändern von Geräteeinstellungen noch eine Funktion zum Öffnen des Einstellungsmenüs zur Verfügung steht. Und da mit den so einzig zur Verfügung stehenden Standardeinstellungen kein akzeptables Scanergebnis erzielt werden kann, schien die neuere API zunächst unbrauchbar.

Die Lösung zeigt sich in einem Mittelweg: Teile der alten `acquire`-Funktion (mit Dialog) werden als `acquireDialog`-Funktion in die neue API übernommen. In der Anwendung wird dann sichergestellt, dass vor dem Aufruf der neuen `acquire`-Funktion zunächst die `acquireDialog`-Funktion aufgerufen wird, um dort die Einstellungen vorzunehmen. Die vorgenommenen Einstellungen bleiben dann erhalten und beim nächsten Aufruf steht zusätzlich die neue `acquire`-Funktion (ohne Dialog) zur Verfügung.

Um diese Funktionalität zu gewährleisten sind bei der alten `acquire`-Funktion einige Anpassungen nötig. Als einzig aufgerufene Funktion ist diese sowohl für das Öffnen als auch für das Schließen von Datenquelle und Gerätemanager zuständig. Allerdings werden alle vorgenommenen Einstellungen am Scanner beim Schließen des Gerätemanagers verworfen, weswegen das Öffnen und Schließen des Gerätemanagers aus der acquire-Funktion ausgelagert werden muss. Hierfür wurden die Codeblöcke zum Öffnen und Schließen des Gerätemanagers in der Datei jtwain.cpp gelöscht (siehe Listings 6.2 und 6.3).

```
246
247   rc = (*g_pDSM_Entry) (&g_AppID,
248                         0,
249                         DG_CONTROL,
250                         DAT_PARENT,
251                         MSG_OPENDSM,
252                         (TW_MEMREF) &hwnd);
253
254   // If data source manager could not be opened, throw exception. Because
          the
255   // exception is not actually thrown until execution returns to Java, we
256   // first exit current block to destroy previously-created window and
          return
257   // a value (which isn't seen in the Java code).
258
259   if (rc != TWRC_SUCCESS)
260   {
261       throwJTE (env, "Unable to open data source manager (acquire)");
262       EXIT_CURRENT_BLOCK
263   }
```

Listing 6.2: Codeblock zum Öffnen des Gerätemanagers – jtwain.cpp

```
527 // Close the data source manager.
528
529   (*g_pDSM_Entry) (&g_AppID,
530                    0,
531                    DG_CONTROL,
532                    DAT_PARENT,
533                    MSG_CLOSEDSM,
534                    (TW_MEMREF) &hwnd);
```

Listing 6.3: Codeblock zum Schließen des Gerätemanagers – jtwain.cpp

Nach Entfernen der beschriebenen Codeblöcke kann die neue JTwain-Version eingesetzt werden. Allerdings sollte in der Anwendung zu einem möglichst frühen Zeitpunkt geprüft werden, ob Gerätemanager und Datenquelle zur Verfügung stehen, da es sonst bei Aufruf der `acquireDialog`-Funktion zum Programmabsturz kommen kann.

6.3.4 Die Handschrifterkennung `Recogtest`

Als zweite ausgelagerte Komponente soll nun die Handschrifterkennung beschrieben werden. Die im Prototyp eingesetzte Erkennungssoftware entstammt dem OCR/ICR-Paket „Omnipage Capture SDK“, welches die Firma Nuance Deutschland als Evaluierungsversion zur Verfügung stellt. Die Funktionalität steht als C-Quellcode zur Verfügung, nötig sind noch die Umsetzung als Java-kompatible dynamische Bibliothek und die Einbindung mittels JNI. Da diese Vorgehensweisen in Kap. 6.3.2 und Kap. 6.3.1 bereits ausführlich diskutiert wurden, folgt nun die Beschreibung der eigentlichen Handschrifterkennung.

Da die Vorstellung des vollen Produktumfangs der Erkennungssoftware den vorliegenden Rahmen sprengen würde, sollen hier nur die wichtigsten Schritte angesprochen werden. Der Vorgang der Schrifterkennung setzt sich aus sechs Phasen zusammen, welche im Folgenden kurz beschrieben werden.

1. **Übergebenen Java-String konvertieren**
   Zu Beginn der Erkennungsfunktion wird der Übergebene Java-String, der den Dateinamen des eingescannten Bildes angibt, in einen C-String (`src_name`) umgewandelt. Dies ist nötig, da die aufzurufende Unterfunktion keine Java- sondern nur C-Strings akzeptiert. Außerdem wird der Name der nach erfolgter Schrifterkennung anzulegenden Textdatei (`dest_name`) festgelegt. Hierzu wird der Name der Bilddatei um die Erweiterung gekürzt und dafür „.txt“ angehängt.

```
62  // uebergebene Java-Strings konvertieren
63
64  int sz=env->GetStringUTFLength(img);
65  const char* fn=env->GetStringUTFChars(img,0);
66
67  char src_name[50];
68  char dest_name[50]="";
69  sprintf(src_name,"%s",fn);
70  strncat(dest_name,src_name,sz-4);
71  strcat(dest_name,".txt");
72  env->ReleaseStringUTFChars(img,fn);
```

Listing 6.4: Java-String konvertieren – Recogtest.cpp

2. **Kernel initialisieren**
   Nach der Konvertierung des Übergabeparameters erfolgt die Initialisierung des Kernels durch Aufruf der Funktion `kRecInit(char*,char*)`. Diese überprüft, ob die benötigten Bibliotheken erfolgreich geladen wurden und ein gültiger Lizenzschlüssel zur Verfügung steht. Im Fehlerfall bricht die Erkennung ab.

```
79   rc = kRecInit(YOUR_COMPANY,YOUR_PRODUCT); // use your company and
        product name here
80    if ((rc != REC_OK) && (rc != API_INIT_WARN))
81    {
82        printf("Error code = %X\n", rc);
83        kRecQuit();
84        return 0;
85    }
```

Listing 6.5: Kernel initialisieren – Recogtest.cpp

3. **Erkennungsmodul laden**
   Nun muss die eigentliche Erkennungsengine geladen werden. Die Schrifterkennung findet in zwei Phasen statt: erst Layouterkennung, dann Zeichenerkennung. Zur Zeichenerkennung stehen bei Omnipage Capture SDK verschiedene Engines, z. B. zum Erkennen von Maschinenschrift, Handschrift, Barcodes usw., zur Verfügung. Im vorliegenden Fall muss die Handschrifterkennungsengine RER eingebunden werden. Hierfür wird zunächst geprüft, ob das Handschriftmodul zur Verfügung steht.

```
88    rc = kRecGetModulesInfo(&pModules, &size);
89    if (rc != REC_OK)
90    {
91        printf("Error code = %X\n", rc);
92        kRecQuit();
93        return 0;
94    }
95    if (pModules[INFO_RER].Version <= 0)
96    {
97        printf("The RER handprint recognition module is not installed
              .");
```

```
98        kRecQuit();
99        return 0;
100   }
```

Listing 6.6: Handschrifterkennungsmodul laden – Recogtest.cpp

4. **Eingescanntes Bild laden und Preprocessing durchführen**

   Ist das Erkennungsmodul vorhanden, kann das Bild geladen und für die Schrifterkennung vorbereitet werden. Das Preprocessing umfasst Erkennung der Bildparameter (Auflösung, Größe, Farbmodus) sowie je nach Einstellung Rotation, Entzerrung und Konvertierung in ein Schwarz/Weiß-Bild.

```
103   rc = kRecLoadImgF(SID, src_name, &hPage, PAGE_NUMBER_0);
104   if (rc != REC_OK)
105   {
106       printf("Error code = %X\n", rc);
107       kRecQuit();
108       return 0;
109   }
```

Listing 6.7: Bild laden – Recogtest.cpp

```
142   printf("Get information about the specified image --
          kRecGetImgInfo()\n");
143   rc = kRecGetImgInfo(SID, hPage, II_CURRENT, &ii);
144 printf("Preprocessing the image -- kRecPreprocessImg()\n");
145 rc=kRecPreprocessImg(SID,hPage);
146 if(rc!=REC_OK)
147 {
148   printf("Preprocessing failed !\nError code : %X",rc);
149   kRecFreeImg(hPage);
150   kRecQuit();
151   return 0;
152 }
```

Listing 6.8: Preprocessing durchführen – Recogtest.cpp

5. **Erkennungsbereich festlegen und Schrifterkennung durchführen**
   Im nächsten Schritt werden jetzt zunächst der zu erkennende Bildbereich (`zone`) festgelegt und die Einstellungen für den Bereich vorgenommen. Dies sind:

- Filling method(`zone.fm`): Methode, nach der Leerstellen zwischen Zeichen gesucht werden (`FM_HANDPRINT` = Handschrifterkennung)
- Erkennungsengine(`zone.rm`): zu benutzende Erkennungsengine (`RM_RER` = Handschrifterkennung)
- Zeichenfilter(`zone.filter`): Auswahl des Filters, der zur Erkennung der Zeichen angewendet wird(`FILTER_ALPHA` = nur Buchstaben)
- Zonentyp(`zone.type`): Typ der zu erkennenden Zone, z. B. Grafik, Tabelle usw. (`WT_FLOW` = Fließtext)

Danach kann die eigentliche Schrifterkennung gestartet werden.

```
154    kRecInitZone(&zone);     // set all fields to default, then
           modify some
155    zone.rectBBox.left = 0;
156    zone.rectBBox.top = 0;
157    zone.rectBBox.right = ii.Size.cx;
158    zone.rectBBox.bottom = ii.Size.cy;
159    zone.fm       = FM_HANDPRINT;
160    zone.rm       = RM_RER;
161    zone.filter  = FILTER_ALPHA;
162    zone.type = WT_FLOW;
163
164    printf("Inserting a zone -- kRecInsertZone()\n");
165    rc = kRecInsertZone(hPage, II_CURRENT, &zone, PAGE_NUMBER_0);
166
167    printf("Recognize DEMOICR2.TIF file -- kRecRecognize()\n");
168    unlink(dest_name);
169    rc = kRecRecognize(SID, hPage, PAGE_NUMBER_0);
170    if (rc != REC_OK)
171    {
172        printf("Error code = %X\n", rc);
173        kRecFreeImg(hPage);
174        kRecQuit();
175        return 0;
176    }
```

Listing 6.9: Schrifterkennung durchführen – Recogtest.cpp

6. **Erkannten Text in Zieldatei abspeichern**
   Nach der Erkennung muss der aus dem Bild extrahierte Text abgespeichert werden. Hierfür wird zunächst der zu erzeugende Filetyp (.txt) gewählt und

dann der Text in die Zieldatei (`dest_name`) geschrieben.

```
179     rc = kRecSetDTXTFormat(SID, DTXT_TXTS);
180     if (rc != REC_OK)
181     {
182         printf("Error code = %X\n", rc);
183         kRecFreeImg(hPage);
184         kRecQuit();
185         return 0;
186     }
187
188     unlink(dest_name);
189     printf("Conversion to output text file -- kRecConvert2DTXT()\n");
190   rc = kRecConvert2DTXT(SID, &hPage, 1, dest_name);
191     if (rc != REC_OK)
192         printf("Error code = %X\n", rc);
```

Listing 6.10: Erkannten Text in Zieldatei ausgeben – Recogtest.cpp

Mit der Freigabe der beteiligten Ressourcen ist die Handschrifterkennung abgeschlossen. Das Ergebnis des Erkennungsvorgangs ist in einer Textdatei abgespeichert, deren Name den Rückgabewert der Erkennungsfunktion darstellt. Über diesen kann die aufrufende Java-Klasse dann das Erkennungsergebnis einlesen.

6.4 Wii-Bedienelemente

Nachdem im vorangegangenen Abschnitt sowohl Realisierung als auch Anbindung der externen Komponenten ausführlich behandelt wurden, folgt in den nächsten Kapiteln die Beschreibung der Funktionsweise des in Java implementierten Hauptteils der Anwendung. Begonnen wird an dieser Stelle mit der Erläuterung der per Wiimote zu bedienenden Steuerelemente.

Da ein Teil der Steuerung von CREWW über die Wiimote erfolgen soll, können einige Swing-Komponenten nicht direkt aus der Standard-API übernommen werden, sondern müssen um Wiimote-Funktionalität erweitert werden: Ein Knopfdruck der richtigen Wiimote-Taste soll den selben Effekt haben, wie ein Druck der Maustaste.

Hierfür wurden drei Klassen implementiert, deren Realisierung im Folgenden beschrieben wird.

6.4.1 Wii-Interaktion mit `MyJButton`

Um dem Benutzer eine Menüsteuerung über die Wiimote zu ermöglichen, wird zunächst einmal etwas benötigt, was man drücken kann, also ein Knopf. Dies realisiert in CREWW die Klasse MyJButton.

Die Klasse MyJButton ist, wie der Name schon sagt, abgeleitet von der Swing-Komponente JButton, erbt also Ihre Funktionalität bezüglich Darstellung und Containereigenschaften. Dies erleichtert die Implementierung, da zum einen die Funktionen zum Ändern von Größe, Position und Aussehen schon vorhanden sind. Zum anderen können die MyJButton-Instanzen so problemlos jedem Swing-Container hinzugefügt werden oder selbst als Container für Swing-Komponenten dienen. Gedrückt werden kann der MyJButton mit Wiimote-Taste (A), hier gibt es jedoch eine Einschränkung:

Der Hauptunterschied zur Basisklasse besteht nämlich nicht in der Wii-Funktionalität, sondern in der Personalisierbarkeit. Jeder MyJButton besitzt eine Membervariable owner, in der der Besitzer des Knopfes als Integerwert gespeichert ist. Ist `owner==0`, ist der Knopf nicht personalisiert, also von allen Benutzern klickbar, im anderen Fall kann der Knopf nur vom Benutzer mit der `ID==owner` bedient werden. Zur Realisierung des Buttonclicks, siehe Listing 6.11.

Kern der Funktion sind die Zeilen 181-185. Hier wird für jeden angehängten ActionListener ein ActionEvent erzeugt, wodurch der Programmierer die Möglichkeit erhält, die Funktionalität des Knopfes wir gewohnt über das ActionListener-Interface anzusprechen.

```
161  public void WiiButtonClicked(WiiEvent w) {
162    if(!this.isShowing())
163      return;
164    int id=w.getCtl();
165    int x=w.ir_data[id][1];
166    int y=w.ir_data[id][2];
167    Point p=this.getParent().getLocationOnScreen();
168
169    //wenn komp getroffen und knopf "a" geklicked
170    if(this.getParent().getComponentAt(x-p.x, (y-p.y)+27)==this&&w.
         but_data[id][1]==WiiClient.BUTTON_A)
171    {
172      if(!this.isEnabled())
173        return;
174      if(this.cursorOver==id) //variante: knopf kann nur drÃ¼cken, wer
           zuerst drauf war
175        //     if(id==this.owner||this.owner==0) // variante
176      {
```

```
177         if(!this.isSelected())
178         {
179           //knopf selektieren
180           this.setSelected(true);
181           //actionevent erzeugen
182           ActionListener[] l=this.getActionListeners();
183           for(int i=0;i<l.length;i++)
184             l[i].actionPerformed(new ActionEvent(this, ActionEvent.
                  ACTION_PERFORMED,""+id ));
185         }
186       }
187     }
188     // bei optionpanel ausschalten: tooltip lÃ¶schen
189     if(w.but_data[id][1]==WiiClient.BUTTON_HOME)
190     {
191
192       ((CREWWWindow)(this.getTopLevelAncestor())).getLayeredPane().remove(
             this.toolTipPanel);
193       ((CREWWWindow)(this.getTopLevelAncestor())).getLayeredPane().repaint
             ();
194     }
195   }
```

Listing 6.11: Die Methode WiiButtonClicked – MyJButton.java

Einen weiteren interessanten Aspekt bei der Realisierung des MyJButton stellen die Tooltips dar. Diese sollen, wie bei Swing gewohnt, angezeigt werden, sobald ein `WiiCursor` sich über dem Knopf befindet. Versucht man nun an dieser Stelle, die Tooltips von Swing zu benutzen, so ergibt sich ein Problem: Jede Instanz von JButton, somit auch jede Instanz von MyJButton, registriert sich bei der Maus als Listener, um so MouseOverEvents verarbeiten zu können. Benutzt man nun für die Wiimote den Swing-Tooltip und aktiviert diesen, sobald eine Wiimote über den Knopf bewegt wird, so wird der Tooltip nur angezeigt, solange die Maus nicht bewegt wird. Dies liegt an der Implementierung der Swing-Tooltips, die automatisch gelöscht werden, sobald die Maus sich nicht mehr über dem zugehörigen Knopf befindet.

Dieses Problem wird gelöst, indem die Tooltips (Text, Platzierung, Hintergrundfarbe) von der Klasse MyJButton verwaltet werden. Jeder neu angelegte Knopf wird als WiiMotionListener an den WiiClient angehängt und empfängt somit alle Wii-Bewegungsevents. Durch Vergleich der Positionsabfrage des WiiMovedEvents mit der Position des Knopfes ist das An- und Abschalten der Tooltips zum richtigen Zeitpunkt möglich.

```
218   public void WiiMoved(WiiEvent w) {
219     if(!this.isDisplayable()||!this.isShowing())
220       return;
221
222     int id=w.getCtl();
223     int x=w.getX(id);
224     int y=w.getY(id);
225     // wenn tooltip angezeigt ->tooltip verschieben
226     if(this.toolTipPanel.isShowing()&&id==this.cursorOver)
227     {
228       this.toolTipPanel.setLocation(((CREWWWindow)(this.
              getTopLevelAncestor())).getCursor(id-1).getX()+CREWWWindow.
              CURSOR_WIDTH+5, ((CREWWWindow)(this.getTopLevelAncestor())).
              getCursor(id-1).getY());
229       ((CREWWWindow)(this.getTopLevelAncestor())).getLayeredPane().repaint
              ();
230       ((CREWWWindow)(this.getTopLevelAncestor())).contentpane.repaint();
231     }
232     Point p=this.getParent().getLocationOnScreen();
233     //wenn komp getroffen
234     if(this.getParent().getComponentAt(x-p.x, (y-p.y)+27)==this&&(id==this
          .owner||this.owner==0))
235     {
236       // wenn button noch ncht in besitz -> in besitz nehmen
237       if(this.cursorOver==0)
238       {
239         this.cursorOver=id;
240         //       tooltip anzeigen
241         if(this.toolTipText.getText()!="")
242           this.showToolTip(id);
243         // wenn enabled -> Rahmen einfärben
244         if(this.isEnabled())
245           this.setBorder(new CompoundBorder(this.getBorder(), new
                LineBorder(WiiCursor.COLORS[id], 2, true)));
246         MouseListener[] ml=this.getMouseListeners();
247         // mouseentered-event feuern
248         for(int i=0;i<ml.length;i++)
249           ml[i].mouseEntered(new MouseEvent(this, MouseEvent.MOUSE_ENTERED
                ,System.currentTimeMillis(),0, x-this.getX(), y-this.getY(),x
                ,y, 0, false,MouseEvent.NOBUTTON));
250       }
251     }
```

```
252     else
253     {
254       // wenn button in besitz war -> rahmen zurÃ¼cksetzen
255       if(this.cursorOver==id)
256       {
257         this.hideToolTip();
258         this.cursorOver=0;
259         if(this.isEnabled())
260           this.setBorder(new LineBorder(this.isEnabled()?Color.black:Color
                .gray, 1, true));
261         MouseListener[] ml=this.getMouseListeners();
262         for(int i=0;i<ml.length;i++)
263           ml[i].mouseExited(new MouseEvent(this, MouseEvent.MOUSE_EXITED,
                System.currentTimeMillis(),0, x-this.getX(), y-this.getY(),
                0, false));
264       }
265     }
266   }
```

Listing 6.12: Tooltip-Erzeugung – MyJButton.java

6.4.2 Das Infofenster `InfoPanel`

Weiterhin soll, in Anlehnung an die Swing-Klasse `JOptionPane`, eine Klasse implementiert werden, die statische Methoden zur Anzeige von Systemnachrichten anbietet. In CREWW leistet dies die Klasse `InfoPanel`. Angeboten werden folgende Funktionen:

- `public static void showInformationMessage(JLayeredPane layeredpane,String message,int millis,boolean circles)`
  Zeigt ein Panel mit einer Nachricht (`message`) für `millis` Millisekunden an (siehe Abb. 6.4). Die boolean-Variable `circles` gibt an, ob zur Visualisierung der Restzeit bis zum Ausblenden des Panel Kreise in der unteren Hälfte des Panel gezeichnet werden sollen. Der Parameter `layeredpane` gibt die `JLayeredPane` an, auf die Swing das `InfoPanel` zeichnen soll.

- `public static void showOKMessage(final JLayeredPane layeredpane,String message, final int id)`
  Zeigt ein Panel mit der Nachricht `message` und einem „OK“-Knopf an (siehe Abb. 6.5). Das Panel bleibt eingeblendet, bis der Benutzer mit `ID == id` den „OK“-Knopf drückt, ist somit personalisiert für einen Benutzer.

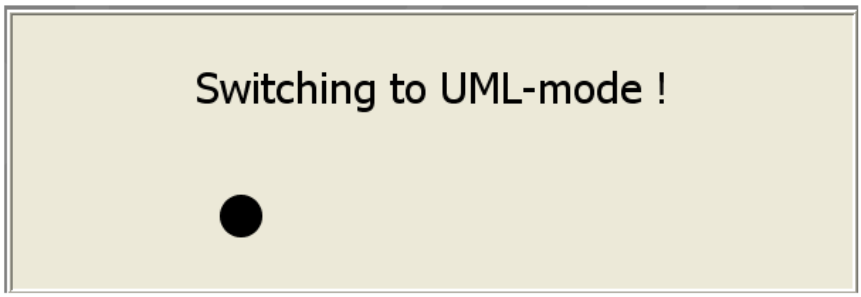


Abbildung 6.4: Das showInformationMessage-Panel

Abbildung 6.5: Das showOKMessage-Panel

- `public static void showConfirmMessage(final CRCComponent component,final JLayeredPane layeredpane,String message, final int id,final int option)`
  Zeigt ein Bestätigungsfenster mit einem „OK“- und einem „Cancel“-Knopf an (siehe Abb. 6.6) und wird bei der Manipulation von CRC-Komponenten eingesetzt. `component` ist die bei Bestätigung zu manipulierende Komponente, `message` die anzuzeigende Nachricht und `option` gibt die Aktion an, die bei Bestätigung ausgeführt werden soll. Für die möglichen Aktionen sind in der Klasse `InfoPanel` Integer-Konstanten definiert. Mit dem Parameter `id` wird das Panel wie in der vorangegangenen Methode personalisiert.

Abbildung 6.6: Das showConfirmMessage-Panel bei Komponentenmanipulation

- `public static void showConfirmMessage(final int edgeNr,final JLayeredPane layeredpane,String message, final int id,final int option)`
  Zeigt analog zur vorigen Methode ein Bestätigungsfenster an (siehe Abb. 6.7), nur dass diese Methode beim Löschen von Kanten aufgerufen wird. Die Konstanten für die Operationen sind ebenfalls in der Klasse `InfoPanel` definiert.

Die beschriebenen Methoden lagern die Funktionalität „Infofenster anzeigen“ wie gewünscht aus. Weiterhin lässt sich die Klasse durch die Definition neuer Konstanten für Operationen oder die Implementierung neuer Methoden auf einfachem Wege erweitern.

Was an der Signatur der Methoden auffällt, ist die Tatsache, dass jeder Methode ein `JLayeredPane` übergeben wird. Anstatt auf die ContentPane des Hauptfensters zu zeichnen, wird das anzuzeigende Infofenster auf diese gezeichnet. Für eine genauere Beschreibung der Anordnung der Darstellungskomponenten in der z-Achse sei an dieser Stelle auf Kap. 6.6.2 verwiesen.

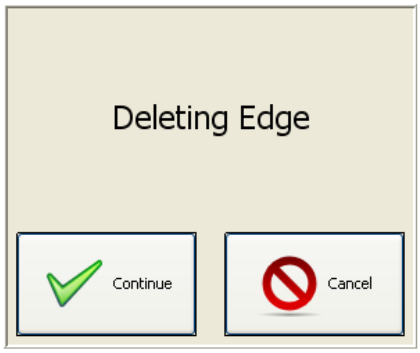


Abbildung 6.7: Das showConfirmMessage-Panel bei Kantenmanipulation

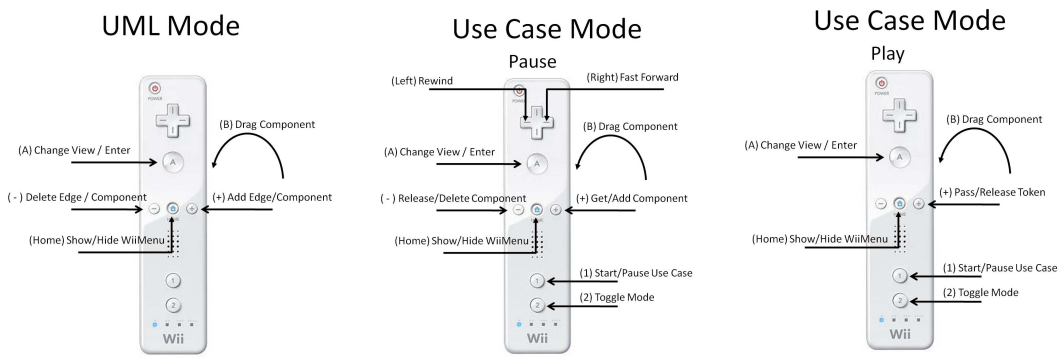
6.4.3 Steuerung des Use-Case mit `OptionPanel`

Wie in Kap. 5.1 gefordert, soll CREWW im Use-Case-Modus ein Undo/Redo, also eine Navigation im Use-Case, erlauben. Für die Umsetzung stehen hier bezüglich der Nutzersteuerung zwei Möglichkeiten zur Verfügung:

Erste Möglichkeit ist die Steuerung über Knöpfe der Wiimote und die zweite ist die Steuerung mittels für alle Benutzer verfügbare Knöpfe auf dem Bildschirm. Vorteile der ersten Variante sind zum einen die schnelle Zugreifbarkeit für alle Benutzer, zum anderen die Tatsache, dass kein Darstellungsplatz für Knöpfe auf dem Bildschirm geopfert werden muss. Der Nachteil liegt ebenfalls auf der Hand: Zusätzlich belegte Knöpfe der Wiimote bedeuten zusätzlichen Lernaufwand.

Da keine der beschriebenen Varianten einen „Goldstandard“ darstellt, werden in CREWW beide Möglichkeiten angeboten, die Steuerung ist sowohl über die Knöpfe der Wiimote (siehe Abb. 6.8) als auch über ein optional einzublendendes Wiimenu möglich.

Das Wiimenu ist in der Klasse `OptionPanel` als Erweiterung von `JPanel` realisiert und wurde nach dem Singleton-Entwurfsmuster implementiert. Das Singleton-Pattern garantiert einerseits, dass von einer Klasse nur eine Instanz angelegt werden kann, andererseits ermöglicht es den Zugriff auf die Klasse aus anderen Klassen heraus.



(a) Knopfbelegung im UML-Modus (b) Knopfbelegung im Use-Case-Modus (Pause) (c) Knopfbelegung im Use-Case-Modus (Play)

Abbildung 6.8: Use-Case-Navigation über die Wiimote

Wie in Listing 6.13 dargestellt, geschieht dies in der Klasse `OptionPanel` durch die öffentliche Methode `getinstance(JLayeredPane)`.Weiterhin wird der einzige Konstruktor geschützt, indem dieser als `private` deklariert wird. Wurde bereits eine Instanz von `OptionPanel` erzeugt, liefert `getinstance()` eine Referenz darauf, wurde noch keine erzeugt, wird über den Konstruktor eine neue Instanz angelegt und diese übergeben.

```
30  public static OptionPanel getInstance(JLayeredPane lpane)
31  {
32    return(instance==null)?new OptionPanel(lpane):instance;
33  }
```

Listing 6.13: Die getInstance-Methode – OptionPanel.java

Das `OptionPanel` wird bei Bedarf über die (Home)-Taste der Wiimote eingeblendet und beinhaltet acht MyJButtons (siehe Abb. 6.9). Anders als in den `InfoPanels` sind die Knöpfe nicht personalisiert und somit von allen Benutzern klickbar. Die beiden linken Knöpfe dienen zum Umschalten zwischen den Modi und der dritte bewirkt ein Rücksetzen des Use-Case. Die fünf restlichen Knöpfe gehören zur eigentlichen Navigation im Use-Case, hier wurde auf die z. B. von Audio-Playern bekannte Metapher „Vor-/Zurückspulen“ zurückgegriffen.

Implementierungsseitig bietet das `OptionPanel` die Methoden `validateOptions()` und `setVisible(boolean)` an. Die `setVisible()`-Methode zeigt das `OptionPanel` an und die validateOptions()-Methode wird vom Hauptprogramm nach erfolgten Aktionen aufgerufen und aktiviert bzw. deaktiviert die Knöpfe des Panels. Dies ist nötig, da nicht zu jedem Zeitpunkt alle Knöpfe aktiv sein sollen,

Abbildung 6.9: Use-Case-Navigation über das Wiimenu

so ergibt ein Redo z. B. erst nach einem Undo Sinn.

6.5 Modell

In Kap. 5.2 wurde bereits angedeutet, dass die von CREWW zu modellierende Struktur einem Graphen ähnelt. So soll nun die Implementierung des beschriebenen Modells dargelegt werden.

Zunächst wird auf die CRC-Karten als zentrale Komponenten eigegangen, um dann die Verbindungen zwischen diesen mittels der Klasse `Edge` zu beschreiben. Schließlich wird die Klasse `UseCaseStack` vorgestellt, die für die Verwaltung des eigentlichen Use-Case-Szenarios verantwortlich ist.

6.5.1 Die zentrale Klasse `CRCComponent`

Die CRC-Karten werden mittels der Klasse `CRCComponent` modelliert. Diese ist, um den Implementierungsaufwand gering zu halten, von der Swing-Klasse `JComponent` abgeleitet. Hierdurch können deren Methoden zum Ändern der Darstellung sowie die Containerfunktionalität genutzt werden.

Die Klasse `CRCComponent` verwaltet nicht nur die CRC-Daten der Karten – Klassenname, Verantwortlichkeiten, Hilfsklassen – sondern ist auch verantwortlich für

die Darstellung der Karten, weiterhin stellt sie die Wiimote-Schnittstelle für die Karten dar. In letzteren Bereich fällt auch die Eingangs erwähnte Zugriffsverwaltung der Benutzer: Welcher Nutzer darf zu einem bestimmten Zeitpunkt welche Aktionen durchführen? Im Folgenden sollen nun die aufgezählten Funktionen beschrieben werden.

Verwaltung und Darstellung der CRC-Daten: Für die Darstellung und Verwaltung der Daten nutzt CRCComponent für den Klassennamen ein Textfeld (`JTextField`) und für die Responsibilities und Collaborators Swing-Listen (`JList`). Diese haben den Vorteil, dass sie bereits nach dem Model-View-Controller-Konzept implementiert sind und somit bereits eine Trennung zwischen Benutzerinteraktion, Darstellung und Datenhaltung beinhalten.

Die Visualisierung der Daten erfolgt durch die `JList`, die eigentliche Datenhaltung geschieht in einem Listenmodell (`DefaultListModel`) und Datenänderungen und Nutzereingaben werden durch entsprechende Controller überwacht (`ListSelectionModel`). Zur Aufteilung der angezeigten CRC-Komponenten siehe Abb. 6.10.

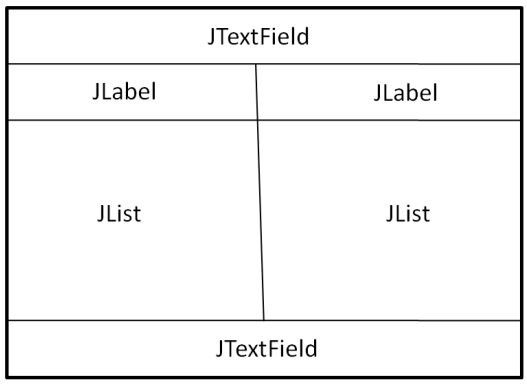


Abbildung 6.10: Aufteilung der CRC-Karte

Das zusätzliche Feld am unteren Rand ist zum Editieren der Listeneinträge für Responsibilities und Collaborators. Die Klasse JList unterstützt kein direktes Editieren in der Liste und die Datenänderung in einem ausgelagerten Editierfeld erscheint hier als sinnvollste Lösung.

Benutzerinteraktion und –koordination: Die Benutzerinteraktion mit Maus und Keyboard geschieht, wie in Swing gewohnt, über Mouse- und Keyboard-Listener. Hierfür werden Standardmethoden benutzt und da diese Eingabegeräte nur einmal zur Verfügung stehen, ist keine Koordination der Benutzer vonnöten.

Aus diesem Grund soll auf die üblichen Interaktionsmethoden hier nicht näher eingegangen werden, interessanter erscheint an dieser Stelle die Interaktion mit der Wiimote.

Zur Koordination der Benutzer verwaltet die Klasse CRCComponent die Variablen `owner` und `activeUser`. Die Variable `owner` speichert den Besitzer der CRC-Karte, falls im Use-Case-Modus ein solcher gewählt wurde, und in der Variable activeUser wird der Benutzer vermerkt, welcher zur Zeit mit der Komponente interagiert. In beiden Fällen gilt: Ist der Wert der Variablen null, so ist kein Benutzer im Besitz der Komponente bzw. keiner interagiert mit ihr.

Im UML-Modus ist eine Interaktion mit der Komponente nur möglich, wenn `activeUser==0`, im USE-Case-Mode müssen beide Variablen null sein. Außerdem wird noch abgefragt, ob der Benutzer zum aktuellen Zeitpunkt bereits eine andere Aktion durchführt, auch in diesem Fall ist eine Interaktion mit der Komponente nicht möglich. Näheres zur nutzerbezogenen Speicherung der aktuell durchgeführten Aktion durch die Klasse `WiiCursor` folgt in Kap. 6.6.3.

Erfolgt nun ein Tastendruck werden also folgende Aktionen durchlaufen (siehe Listing 6.14): Zunächst werden die Daten des Wiimote-Events (Position, gedrückte Taste) in lokalen Variablen gespeichert. Dann wird überprüft, ob die Komponente getroffen wurde und zum aktuellen Zeitpunkt eine Interaktion möglich ist. Daraufhin erfolgen mittels zweier verschachtelter switch/case-Blöcke die Auswertung des Tastendrucks und der Aufruf entsprechender Funktionen, abhängig von Modus und gedrückter Taste.

```
979     public void WiiButtonClicked(WiiEvent w) {
980       // ID und Cursorposition abfragen
981       int id=w.getCtl();
982       int x=w.getX(id);
983       int y=w.getY(id)-20;
984       if(y<=0)
985         y=1;
986 
987       // abfrage, ob komponente getroffen (evtl. Verdeckung durch
              Optionpanel)
988       if(this.cw.getContentPane().getComponentAt(x, y)==this&&(!this.cw.
              isShowingOptions()||this.cw.getLayeredPane().getComponentAt(x, y
              ).getClass()!=OptionPanel.class))
989       {
990         this.editField.setText("");
991         this.editField.setEnabled(false);
992         this.responsibilities_controller.clearSelection();
993         this.collaborators_controller.clearSelection();
```

```
994         switch(this.cw.getMode())
995         {
996         // UML MODE !!!
997         case CREWWWindow.MODE_UML:
998           switch(w.but_data[id][1])
999           {
1000          case WiiClient.BUTTON_A:
1001            if(this.locked||this.cw.getCursor(id-1).getAction()!=WiiCursor.
                    ACTION_IDLE)
1002              break;
1003            this.cw.setChanged(true);
1004            this.changeView();
1005            break;
1006            //         mit dem "+"-Knopf wird eine Kante hinzugefï¿½gt
1007          case WiiClient.BUTTON_PLUS:
1008            if(this.cw.getCursor(id-1).getAction()!=WiiCursor.ACTION_ADD&&
                    this.cw.getCursor(id-1).getAction()!=WiiCursor.ACTION_IDLE)
1009              break;
1010            this.addEdge(id);
1011            break;
1012
1013          case WiiClient.BUTTON_MINUS:
1014            if(this.cw.getCursor(id-1).getAction()!=WiiCursor.ACTION_DELETE
                    &&this.cw.getCursor(id-1).getAction()!=WiiCursor.
                    ACTION_IDLE)
1015              break;
1016            this.deleteEdgeOrComponent(id);
1017            break;
1018          } //switch
1019          break;
1020
1021          //USE-CASE-MODE !!
1022        case CREWWWindow.MODE_USE_CASE:
1023          if(this.cw.getCursor(id-1).getAction()!=WiiCursor.ACTION_IDLE)
1024            break;
1025          switch(w.but_data[id][1])
1026          {
1027          case WiiClient.BUTTON_A:
1028            if(this.locked||this.owner!=id||this.cw.getCursor(id-1).
                    getAction()!=WiiCursor.ACTION_IDLE)      // auzuklappen nur
                     durch besitzer
1029              break;
1030            this.cw.setChanged(true);
```

```
1031            this.changeView();
1032            break;
1033          case WiiClient.BUTTON_PLUS:
1034            if(stopTimer()>CREWWWindow.PRESS_DURATION)
1035              break;
1036            if(this.cw.getCursor(id-1).hasToken()&&this.owner!=0)
1037            {
1038              // Aufgabe weiterdelegieren oder 'aufgabe erfÃ¼llt' an
                     Aufrtraggeber senden
1039              this.cw.setChanged(true);
1040
1041              if(this.active)
1042                this.cw.popActive();
1043              else
1044                this.cw.pushActive(this);
1045            }
1046            if(this.owner==0&&!this.cw.isUseCasePlay())
1047            {
1048              // Komponente in Besitz nehmen
1049              this.cw.setChanged(true);
1050              this.owner=id;
1051              this.col=WiiCursor.COLORS[this.owner];
1052            }
1053            break;
1054          case WiiClient.BUTTON_MINUS:
1055            if(this.cw.isUseCasePlay())
1056              break;
1057            this.releaseOrDeleteComponent(id);
1058            break;
1059          } //ende switch(w.but_data[id][1])
1060        } // ende switch(cw.getmode())
1061      } // if (komp getroffen)
1062
1063    }
```

Listing 6.14: Die WiiButtonClicked()-Methode – CRCComponent.java

6.5.2 Wer mit wem - Edge und ihre Unterklassen

Wie zuvor schon erwähnt, muss in CREWW eine Datenstruktur zur Modellierung von Beziehungen zwischen CRC-Karten zur Verfügung gestellt werden. Dies können zum einen UML-Relationen sein, zum anderen wird die Abfolge der Delegationen im aktuellen Use-Case ebenfalls als eine Abfolge von Kanten gespeichert.

Wurden die Klassen A, B, C, D nacheinander aufgerufen, so wird dies durch die Use-Case-Kanten (A,B), (B,C), (C,D) ausgedrückt.

Benötigt werden somit Kanten unterschiedlichen Aussehens und Typs, wobei ihnen gleich ist, dass sie sich über Start- und Zielknoten definieren. Zur Abbildung dieser Sachverhalte wird in CREWW die abstrakte Basisklasse `Edge` zur Verfügung gestellt. In dieser werden die Daten gespeichert, die alle Kantenarten gemein haben, als da wären: Ursprungsknoten, Zielknoten und die Variablen, in denen die Kantenorientierung gespeichert ist. Weiterhin dient sie als Container für die konkreten Implementierungen und bietet die Methoden an, die von allen Kanten gebraucht werden.

Die spezialisierten Methoden und Attribute finden sich in den konkreten Ableitungen `AggregationEdge`, `AssociationEdge`, `InheritanceEdge` und `UseCaseEdge`. Dort finden sich zum Beispiel die Kantensymbole und die spezialisierten paint()-Methoden.

In diesem Abschnitt werden nun eine Funktion der Basisklasse sowie eine Funktion aus einer der abgeleiteten Klassen etwas genauer beschrieben. Aus den Methoden der Basisklasse wurde die Berechnung von Start- und Endpunkt gewählt, in der Gruppe der konkreten Klassen wird dies die Animation der Use-Case-Kanten sein.

Edge – Berechnung von Start- und Endpunkt: Erwähnenswert unter den Funktionen der Basisklasse ist z. B. die Funktion zur Berechnung von Start- und Endpunkt der Kante. Ziel bei der Implementierung der Kanten ist eine saubere adaptive Kantenzeichnung, die die Endpunkte der Kante beim Verschieben der beteiligten Komponenten über die Außenlinien der Komponenten wandern lässt. Als Metapher kann man sich hier ein „Gummiseil“ vorstellen, das auf der Rückseite der CRC-Karten befestigt ist und diese verbindet. Im Folgenden soll nun beschrieben werden, wie beim Zeichnen der Kanten vorgegangen wird.

Zunächst einmal müssen virtueller Start- und Endpunkt (S_v,E_v) der Kante – in der Metapher die Punkte, an denen das Gummiseil befestigt ist – festgelegt werden. Diese sind nicht sichtbar, da die Kante erst ab den Kartenrändern gezeichnet wird und somit nur für die Berechnung relevant. Die virtuellen Endpunkte liegen im Zentrum der Felder, in denen die Klasssennamen angezeigt werden (siehe Abb. 6.11), da sich diese beim Auf- und Zuklappen der Karten nicht verändern. Hätte man die Endpunkte in der Kartenmitte platziert, wäre ein Verschieben der Endpunkte beim Auf- und Zuklappen der beteiligten Karten nötig gewesen. Für die Benutzer hätte dies den unerwünschten Effekt ergeben, dass sich Kanten bewegt hätten, ohne dass eine beteiligte Komponente verschoben worden wäre.

Die beim Zeichnen relevanten Punkte (S,E) befinden sich in den Schnittpunkten von den Karten und der Verbindungslinie zwischen den Punkten S_v und E_v. Auf

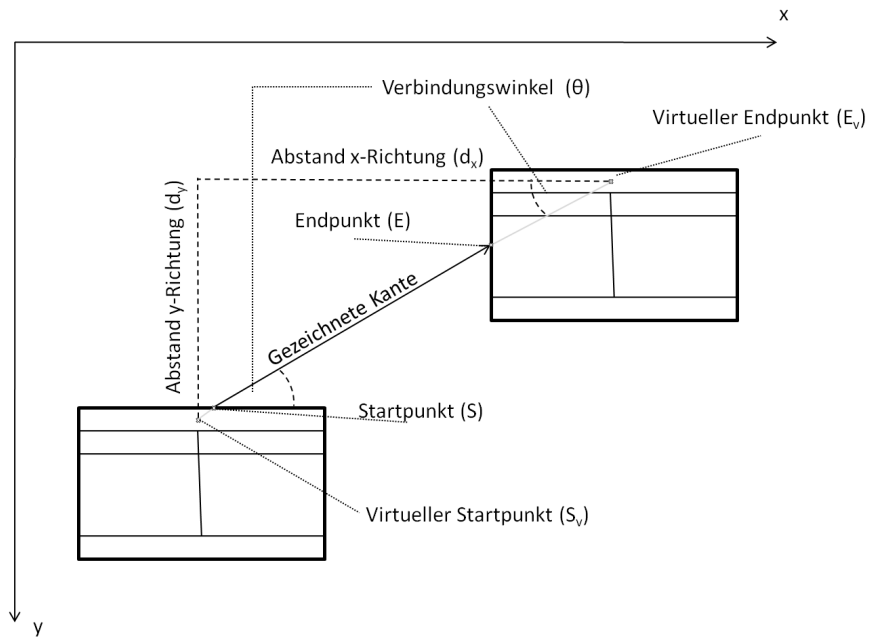


Abbildung 6.11: Zeichnen von Kanten – Übersicht

die Berechnung dieser Punkte wird nun im Näheren eingegangen.

Zunächst soll hier darauf hingewiesen werden, dass die Java-API interessanterweise keine Methoden anbietet, um den Schnittpunkt eines Rechtecks und einer Geraden zu berechnen. Es existiert zwar eine `boolean`-Methode, die berechnet, ob ein solcher Schnittpunkt existiert, allerdings lässt sich dieser nicht ausgeben. Aus diesem Grund muss die Berechnung der Schnittpunkte von Hand geschehen.

Als Erstes wird bestimmt, welchen Kartenrand die Verbindungslinie (S_v,E_v) schneidet: oben, unten, links oder rechts. Hierfür werden trigonometrisch Grenzwinkel (topRight, topLeft, bottomLeft, bottomRight)[1] bestimmt (Tangenssatz), die dann mit dem Richtungswinkel (θ) der Verbindungslinie verglichen werden (siehe Abb. 6.12, Listing 6.15). Liegt θ z. B. zwischen topRight und bottomRight, dann schneidet die Verbindungslinie den rechten Kartenrand und auf diesem liegt auch der Startpunkt. Der Schnittpunkt von rechtem Kartenrand und Verbindungslinie kann dann ebenfalls mittels des Tangenssatzes ermittelt werden.

```
38 private static final double bottomRight=Math.atan((CREWWWindow.
       CLASS_HEIGHT-CREWWWindow.CLASS_FIELD_HEIGHT/2.0)/(CREWWWindow.
       CLASS_WIDTH/2.0));
```

[1] Anmerkung: Falls eine der Karten – oder beide – zusammengeklappt sind, müssen andere Grenzwinkel zur Berechnung der Austrittsseite herangezogen werden. Dies wird bei den folgenden Betrachtungen außer Acht gelassen.

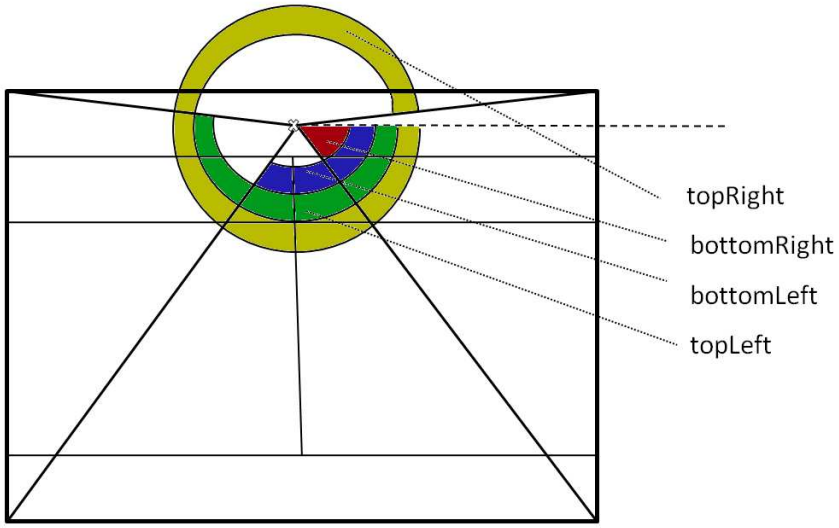


Abbildung 6.12: Zeichnen von Kanten – Grenzwinkel

```
39  private static final double bottomLeft=Math.PI-bottomRight;
40  private static final double topLeft=Math.PI+Math.atan((CREWWWindow.
        CLASS_FIELD_HEIGHT/2.0)/(CREWWWindow.CLASS_WIDTH/2.0));
41  private static final double topRight=2*Math.PI-Math.atan((CREWWWindow.
        CLASS_FIELD_HEIGHT/2.0)/(CREWWWindow.CLASS_WIDTH/2.0));
```

Listing 6.15: Berechnung der Grenzwinkel – Edge.java

Zunächst aber zur Bestimmung von θ. Wünschenswert für θ sind Werte zwischen 0 und 2π, wobei hier im Uhrzeigersinn gedreht wird, da die y-Achse bei Bildschirmkoordinaten von oben nach unten verläuft. Prinzipiell erfolgt die Berechnung von θ über den Tangenssatz:

$$\theta = arctan(\mathrm{d_y}/\mathrm{d_x})$$

Allerdings gilt dies nur für den Fall $\mathrm{d_x}$,$\mathrm{d_y} > 0$. Ist einer der beiden Werte < 0, wird bei θ eine Korrektur nötig, womit sich für die Berechnung von θ folgende Fälle ergeben:

1. $\mathrm{d_x}$,$\mathrm{d_y} > 0 \Rightarrow \theta = arctan(\mathrm{d_x}/\mathrm{d_y})$
2. $\mathrm{d_x} > 0$,$\mathrm{d_y} < 0 \Rightarrow \theta = arctan(\mathrm{d_x}/\mathrm{d_y}) + 2\pi$
3. $\mathrm{d_x} < 0 \Rightarrow \theta = arctan(\mathrm{d_x}/\mathrm{d_y}) + \pi$

Der so errechnete Winkel θ kann nun mit den Grenzwinkeln verglichen werden, um die Austrittsseite der Karte zu bestimmen. Anhand der Austrittsseite lässt sich dann entweder die horizontale Differenz `diffX` (Austrittsseite oben oder unten) oder die vertikale Differenz `diffY` (Austrittsseite links oder rechts) errechnen, mit der der Schnittpunkt zur Lotrechten verrechnet werden muss, um Punkt S bzw. E zu erhalten. Exemplarisch soll dies hier für den Fall $0 < \theta < bottomRight$ geschehen, das heißt die Verbindungslinie (S_v,E_v) tritt aus der rechten Seite der Startkomponente aus.

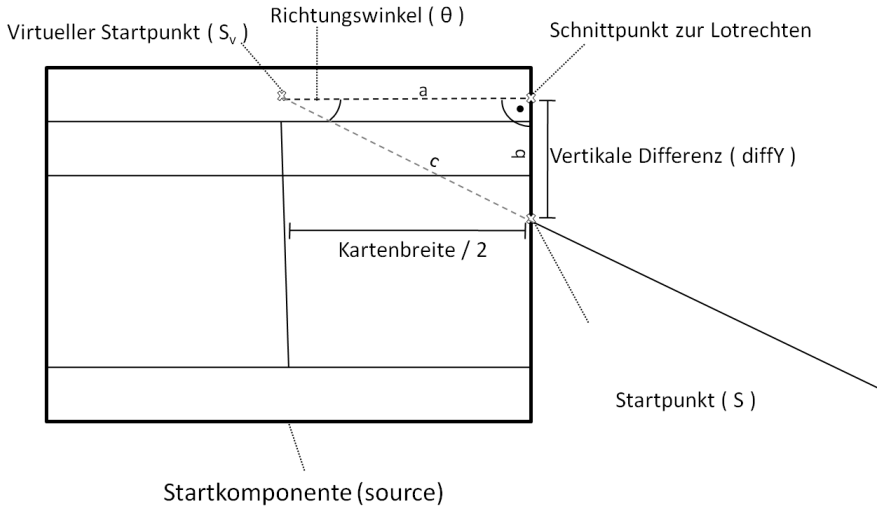


Abbildung 6.13: Zeichnen von Kanten – Bestimmung von `diffY`

Wie in Abb. 6.13 zu sehen, stellt sich folgendes Problem: Gegeben ist ein Dreieck (a,b,c), wobei eine Kantenlänge ($a = Kartenbreite/2$) sowie die drei Winkel $(\theta, 90°, 90° - \theta)$ gegeben sind. Gesucht ist `diffY`, dies entspricht der Länge der Kante b.

b lässt sich nun erneut über den Tangenssatz berechnen:

$$tan(\theta) = b/a = diffY/(Kartenbreite/2)$$

Da $\theta = arctan(d_y/d_x)$ (s.o.) ergibt sich so leicht:

$$diffY = d_y * Kartenbreite/(2 * d_x)$$

Die so ermittelte vertikale Abweichung muss nun noch mit dem Schnittpunkt der Lotrechten verrechnet werden, um den Startpunkt S (im Programm: `this.pSource`) der Kante zu erhalten (siehe Listing 6.16).

```
134     // punkt auf source-komponente bestimmen
135     if(this.theta<=bottomRight||this.theta>=topRight)
136     {
137       diffY=directionY*this.source.getWidth()/(2*directionX);
138       this.pSource=new Point(this.source.getX()+this.source.getWidth(),
              this.source.getY()+CREWWWindow.CLASS_FIELD_HEIGHT/2+diffY);
139     }
```

Listing 6.16: Berechnung der Grenzwinkel – Edge.java

Mit Listing 6.16 sollen die Erläuterungen zur Ermittlung von Start- und Endpunkt der Kante abgeschlossen werden. Das Vorgehen bei der Errechnung des Endpunktes E erfolgt analog, nur dass hier anstatt mit θ mit $\gamma = \pi - \theta$ gerechnet werden muss.

Animation der Use-Case-Kanten: Im zweiten Abschnitt dieses Kapitels wird nun die Animation der Use-Case-Kanten beschrieben. Die Animation veranschaulicht die Dynamik im aktuellen Use-Case und ermöglicht ein schnelles Rekapitulieren der gegenseitigen Aufrufe im `UseCaseStack`.

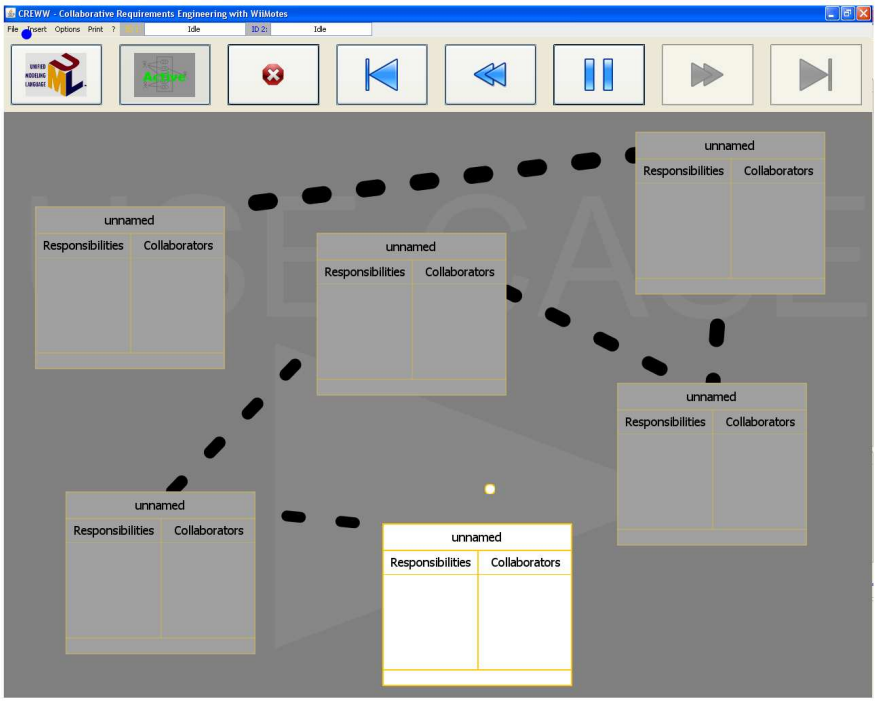


Abbildung 6.14: Die Kanten im Use-Case-Modus

Gefragt war eine Möglichkeit, die gegenseitigen Aufrufe beim Durchspielen des Use-Case zu visualisieren, wobei gleichzeitig noch – allerdings eher im Hintergrund – die UML-Kanten dargestellt werden sollten. Dies wird in CREWW durch die Klasse `UseCaseEdge` realisiert. Die Use-Case-Kanten sind animiert, sie „fließen" gewissermaßen vom Ursprungs- zum Zielknoten (siehe Abb. 6.14). Die Animation läuft allerdings nur während des Use-Case-Play-Modus, sowohl im Pause- als auch im UML-Modus stoppt sie.

Neben der Aufrufrichtung visualisiert eine Use-Case-Kante noch die Tiefe des Stapelaufrufs: Mit jedem Unteraufruf nimmt die Kantenstärke um den in der Klasse `CREWWWindow` definierten Wert `STEPSIZE_THICKNESS_UC_EDGE` ab. Dies ist nötig, um die Reihenfolge mehrerer von einer Klasse ausgehender Unteraufrufe nachzuvollziehen.

Die Animation der Use-Case-Kanten wird in CREWW durch eine fortlaufende Manipulation der Phase der `UseCaseEdge` erreicht. Phase bezeichnet in diesem Zusammenhang den Versatz, mit dem die Kante beim Aufruf der paint()-Methode gezeichnet wird. Hierfür wird zusammen mit der ersten `UseCaseEdge` ein Klassen-Thread phaseCorrector (siehe Listing 6.17) initiiert, der die Klassenvariable `phase` alle 25 Millisekunden um den ebenfalls in der Klasse `CREWWWindow` festgelegten Wert `ANIMATION_SPEED` verringert. Die Phase beginnt bei 80, was der Länge eines Elements der Kante entspricht, und endet bei 0. Danach wird der Wert vom Thread wieder auf 80 gesetzt. Desweiteren ruft der Thread noch die asynchrone repaint()-Methode auf, um den Phasenversatz auch sichtbar zu machen.

```
21  transient private static Runnable phaseCorrector=new Runnable() {
22
23    @Override
24    public void run() {
25      while(true)
26      {
27        try {
28          Thread.sleep(25);
29        } catch (InterruptedException e) {
30          e.printStackTrace();
31        }
32        if(!pause)
33        {
34          phase=phase-CREWWWindow.ANIMATION_SPEED;
35          if(phase<0)
36            phase=80;
37          MyContentPane.getInstance().repaint();
38        }
39      }
```

```
40      }
41    };
```

Listing 6.17: Der phaseCorrector-Thread – UseCaseEdge.java

In der paint()-Methode wird die Kante durch einen simplen Aufruf der drawLine()-Methode gezeichnet. Der Trick liegt darin, dass zuvor mit der setStroke()-Methode die Art des Strichs verändert wird (siehe Listing 6.18).

```
78      // linienstärke bestimmen
79      Stroke stroke=new BasicStroke(this.strength,BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_BEVEL,0,new float[]{20,60},phase);
80      g2d.setStroke(stroke);
```

Listing 6.18: Die setStroke-Methode – UseCaseEdge.java

Der erste Parameter bestimmt die Dicke der Kante und hängt von der Stapeltiefe ab. Parameter zwei, drei und vier beziehen sich auf die Form der einzelnen Kantensegmente und der fünfte Wert steht für die Länge der einzelnen Segmente und deren Versatz zueinander. Der sechste Parameter schließlich bestimmt die Phase, mit der begonnen wird, die Linie zu zeichnen.

Über diese Methode ist eine Animation mit geringem Rechenaufwand möglich, was im vorliegenden Fall auch nötig ist, da die paint()-Methode vom System synchron aufgerufen wird und somit bei hohem Rechenaufwand die komplette Programmausführung ausbremsen könnte.

6.5.3 Der aktuelle Anwendungsfall: Die Klasse `UseCaseStack`

Das Durchspielen von Anwendungsfällen nimmt bei CREWW, wie zuvor bereits an mehreren Stellen erwähnt, eine zentrale Rolle ein. Neben der Aufgabendelegation zwischen den CRC-Klassen wird eine Undo/Redo-Funktion unterstützt, auf deren Implementierung im Folgenden näher eingegangen werden soll.

Die Verwaltung des aktuellen Anwendungsfalles findet in CREWW in der Klasse `UseCaseStack` statt. Diese ist als Singleton implementiert, Einsatzgebiet sowie Vorteile des besagten Entwurfsmusters wurden schon an anderer Stelle (sieh Kap. 6.4.3) erwähnt. Nach außen hin bietet sie die für einen Stack unabdingbaren Methoden `push()`, `pop()` und `top()` sowie darüber hinaus solche zur Verwaltung der intern mitgeführten History – hier seien z. B. die Methoden `forward()`, `back()`, `toStart()` und `toEnd()` genannt.

Um diese Funktionalität gewährleisten zu können, verwaltet die Klasse `UseCaseStack` intern einen Stack aus CRC-Karten für den eigentlichen Stapel (`activeStack`), je eine ArrayList von CRC-Karten (`redoList`)und Integerwerten (`operations`) sowie einen Zeiger auf die aktuelle Position (`redoPosition`) für die Undo/Redo-Funktion, weiters noch einen Stack aus Kanten (`edges`), in dem

die Kanten im aktuellen Stapel gespeichert werden. In der Integer-Liste sind analog zur Redo-Liste die Operationen gespeichert, da die Redo-Liste keinen Unterschied zwischen `push()` und `pop()` macht. Für die Undo/Redo-Funktion werden `ArrayLists` benötigt, da bei dieser, anders als beim eigentlichen Stack, *nicht* ausschließlich Zugriffe auf das oberste Element nötig sind. Ebenfalls in den Zuständigkeitsbereich der Klasse `UseCaseStack` fällt das Aktivieren/Deaktivieren der Klassen, was insofern Sinn ergibt, als besagte Funktionen ausschließlich bei Stack-Operationen nötig sind.

Es werden nun die Aktionen betrachtet, die eine Anfrage an die Klasse `UseCaseStack` intern auslöst, anders ausgedrückt, welche Manipulationen der internen Datenstrukturen die externe Funktionalität gewährleisten.

Delegation einer Verantwortlichkeit (`push()`) : Gibt eine Klasse eine Teilverantwortlichkeit an eine andere Klasse ab, muss die neue Klasse zunächst auf dem aktiven Stapel abgelegt werden. Im selben Arbeitsschritt wird auch die aufrufende Klasse deaktiviert und die aufgerufene aktiviert. Weiters muss eine Kante von der aufrufenden zur aufgerufenen Klasse angelegt und diese auf den Kanten-Stapel gelegt werden.

Um ein Rückgängigmachen zu gewährleisten, werden die aufgerufene Klasse an das Ende der Redo-Liste und der Integer-Wert für OPERATION_PUSH an das der Operationsliste angehängt sowie der Zähler für die Redo-Liste um den Wert eins erhöht.

Abb. 6.15 veranschaulicht diese Operationen. Hier wird davon ausgegangen, dass zuvor die Karten A, B, C aufgerufen, somit gepushed wurden.

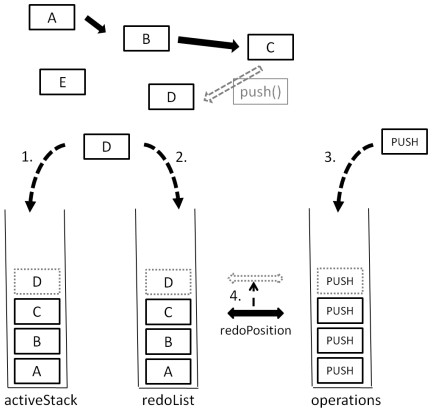


Abbildung 6.15: Die Aktionen der push()-Operation – Der UseCaseStack

Rückgabe der Verantwortlichkeit (`pop()`) : Bei der Rückgabe der Verantwortlichkeit aufgrund der Erfüllung der Aufgabe werden folgende Aktionen durchlaufen (siehe Abb. 6.16). Die abgebende Klasse wird vom aktiven Stack genommen und die entsprechende Kante abgebaut. Um die Redo-Funktionalität zu gewährleisten, muss sie aber dennoch der Redo-Liste hinzugefügt werden. Außerdem wird der Operationsliste der Wert für OPERATION_POP hinzugefügt und der Redo-Zähler erhöht.

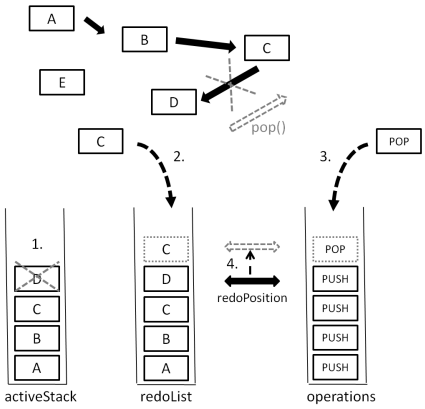


Abbildung 6.16: Die Aktionen der pop()-Operation – Der UseCaseStack

Eine Aktion zurück gehen (`back()`): Im Falle des Undo wird zunächst anhand der Operationsliste überprüft, welche Aktion als letztes erfolgte. War die letzte Aktion ein `push()`, muss nun ein `pop()` ausgeführt werden und umgekehrt. Allerdings wird dies nicht über die öffentlichen Operationen ausgeführt, hierfür sind interne `push()`- und `pop()`-Operationen implementiert, die die Undo/Redo-Behandlung aussparen. War die letzte Aktion ein `pop()`, so muss beispielsweise das zweit-oberste Element der Redo-Liste auf den activeStack abgelegt werden. Außerdem wird der Redo-Zeiger eine Position nach unten verschoben, auf diese Art wird der Redo-Aufruf ermöglicht.

Abb. 6.17 zeigt dies beispielhaft für den Fall, dass zuletzt eine `pop()`-Operation erfolgte.

Aktion wiederherstellen (forward()): Ein Redo – also ein Wiederherstellen der Aktion – ist nur möglich, wenn zuvor ein Undo erfolgte. In diesem Fall wird erneut über die Operationsliste ermittelt, für welche Aktion zuvor ein Undo erfolgte. War dies ein `pop()`, muss nun ebenfalls ein `pop()` erfolgen, analog im umgekehrten Falle.

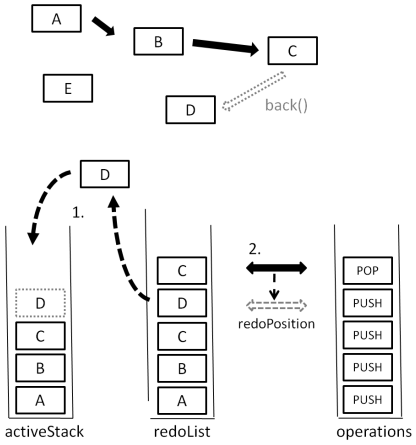


Abbildung 6.17: Die Aktionen der back()-Operation – Der UseCaseStack

Der Redo-Zeiger wird eins nach oben verschoben und die entsprechende Aktion ausgeführt. Graphisch dargestellt ist dies in Abb. 6.18 für den Fall, dass ein Undo für ein `pop()`, also ein internes `push()` erfolgte.

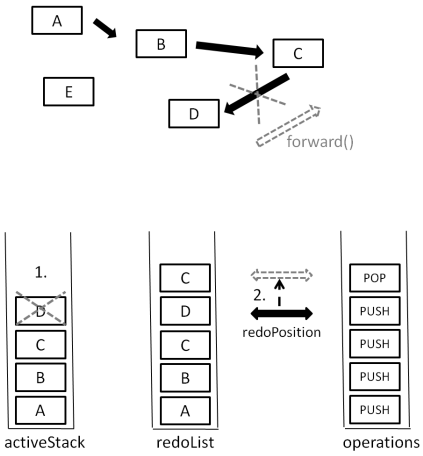


Abbildung 6.18: Die Aktionen der forward()-Operation – Der UseCaseStack

Soweit zur Funktionalität der Klasse `UseCaseStack`. Zusätzlich zu den dargestellten Aktionen muss bei der Abarbeitung der Operationen selbstverständlich eine ausführliche Ausnahmebehandlung erfolgen. Auf die Beschreibung der Grenzfälle soll an dieser Stelle allerdings verzichtet werden, um den Umfang der vorliegenden Arbeit in einem akzeptablen Rahmen zu halten. Die Nachvollziehbarkeit ist an dieser Stelle durch ausführliche Dokumentation im Quellcode gegeben.

6.6 Zentrale Benutzerschnittstelle

Thema des vorletzten Unterkapitels des Bereichs "Implementierung" stellt die zentrale Benutzerschnittstelle dar, welche in der noch vorzustellenden Klasse `CREWWWindow` implementiert ist. Außerdem findet noch die Klasse `WiiCursor` Erwähnung, die jedem Benutzer einen Cursor zur Interaktion mit der Software zur Verfügung stellt.

Begonnen werden soll jedoch zunächst mit der Beschreibung der Komponentenverwaltung, die eine sinnvolle Anordnung der Interaktionseinheiten von CREWW, z. B. Modellkomponenten, Infofenster, Dialoge, ermöglicht.

6.6.1 Komponentenanordnung

Die Klasse `CREWWWindow` ist ein von der Swing-Klasse `JFrame` abgeleiteter Top-Level-Container und bietet somit als allgemeine Zeichenfläche eine `JRootPane` an. Diese beinhaltet wiederum neben der `GlassPane`, die standardmäßig durchsichtig ist und z. B. zum Abfangen von Events genutzt werden kann, eine `JLayeredPane`, die die Komponenten des Fensters beinhaltet.

Die `JLayeredPane` ist – wie der Name schon sagt – in Schichten oder auch Ebenen unterteilt, die durch Integer-Werte repräsentiert werden. Im Folgenden sollen nun die Schichten und ihre Inhalte bei CREWW aufgelistet werden:

- **DEFAULT_LAYER (0):** Auf dieser Ebene befindet sich die `ContentPane`, die in CREWW durch eine Instanz der Klasse `MyContentPane` repräsentiert wird. Diese beinhaltet die Modellkomponenten, also die Instanzen von `CRCComponent` und die der konkreten Ableitungen von `Edge`. Hierbei liegen die Komponenten über den Kanten, was durch die paint()-Methode der Klasse `MyContentpane` (siehe Kap. 6.6.4) gewährleistet wird.

- **PALETTE_LAYER (100):** Diese Ebene enthält das `OptionPanel` mit den Knöpfen zur Steuerung des Anwendungsfalles mit den Wiimotes.

- **POPUP_LAYER (300):** Hier werden die von der Klasse `InfoPanel` (siehe Kap. 6.4.2) angezeigten Informations- und Bestätigungsnachrichten angezeigt.

- **DRAG_LAYER (1000):** Diese Ebene wird über alle anderen gezeichnet. Sie beinhaltet die zur Benutzerinteraktion notwendigen `WiiCursor` (siehe Kap. 6.6.3).

6.6.2 Die Hauptklasse `CREWWWindow`

`CREWWWindow` stellt die zentrale Klasse in CREWW dar und zu ihren Aufgaben gehören neben der Darstellung des Hauptfensters unter Anderem auch die Steuerung des Kontrollflusses, die Verwaltung des Hauptmenüs und die Bereitstellung der allgemeinen Konstanten, mit denen sich die Funktionalität von CREWW anpassen lässt.

Auch wenn die zentrale Klasse eine breite Funktionalität bietet, sind die meisten dieser Funktionen eher Java-Standardfunktionen, deren Beschreibung an dieser Stelle weniger von Interesse ist. Zum Teil mussten jedoch auch eigene Lösungen implementiert werden, wofür nun zwei Beispiel folgen.

Glättung der Cursorbewegung: Begonnen werden soll mit der Beschreibung folgenden Problems: Zu Beginn der Implementierungsphase zeigte sich bei der Steuerung über die `WiiCursor` ein starkes Zittern der Cursor selbst. Ein störender Effekt, der sowohl bei der Menüauswahl als auch bei der Interaktion im Allgemeinen hinderlich war. Grund dieses Zitterns ist die hohe Genauigkeit der Wii-Sensoren, gepaart mit der hohen Aktualisierungsfrequenz. Eine auf dem Tisch liegende Wiimote registriert über die Beschleunigungssensoren, deren Daten in die Lagedaten miteingehen, schon ein Klopfen auf die Tischplatte.

Zur Lösung dieses Problems bietet CREWW eine justierbare Glättungsfunktion, die einen gleitenden Mittelwert aus der alten und der neuen Cursorposition berechnet (siehe Listing 6.19). Der gleitende Mittelwert wird mittels der Funktion

$$f : x \rightarrow t* \mathrm{x}_0 + (1-t) * x$$

berechnet, wobei der Faktor t für die Gewichtung des alten Cursor-Wertes steht. Der Faktor t lässt sich in CREWW über die Menüfunktion „Adjust Smoothing“ justieren. Bei $t = 0$ ist die Glättung ausgestellt und das Zittern nach obiger Gleichung maximal, bei t=0.99 ist die Glättung maximal und der Cursor eher träge. Voreingestellt ist ein Wert von 0.7 . Es folgt nun das Listing zur Cursorglättung.

```
2587    // Cursorbewegung glaetten
2588    this.coord_x_alt[id]= (this.smooth*this.coord_x_alt[id]+(100-this.
            smooth)*x)/100;
2589    this.coord_y_alt[id]=(this.smooth*this.coord_y_alt[id]+(100-this.
            smooth)*y)/100;
2590    this.cursors[id].setLocation(this.coord_x_alt[id],this.coord_y_alt[id
            ]);
```

Listing 6.19: Glättung der Cursorbewegung – CREWWWindow.java

Speichern und Laden: Ein noch nicht angesprochener Punkt bei der Implementierung ist das Abspeichern, sowohl von CRC-Sessions als auch von UML-Modellen. Hierfür bietet die Klasse `CREWWWindow` die Möglichkeit, mittels `save()`- und `load()`-Funktionen eine Session (inklusive History) zu speichern und zu einem beliebigem Zeitpunkt fortzuführen. An dieser Stelle soll nun der Speichermechanismus angesprochen werden, das in CREWW benutzte Speicherformat wird in Kap. 6.7.1 beschrieben.

Zur persistenten Speicherung von Objekten bietet Java das Serializable-Interface an. Die Instanzen implementierender Klassen können in einen ObjectOutputStream serialisiert und dieser wiederum in einen FileOutputStream geschrieben werden. Hierbei werden sämtliche Member der Objekte mitgespeichert. Im umgekehrten Fall der `load()`-Methode können die so gestreamten Objekte wieder deserialisert werden.

Diese Funktionalität konnte in CREWW integriert werden, hier waren lediglich einige Änderungen nötig. Da die zu speichernden CRC-Komponenten Listener zur Maus- und Tastaturverwaltung beinhalteten, ließen sich diese nicht vollständig serialisieren. Threads und von diesen abgeleitete Klassen (hier: Listener) sind per se nicht serialisierbar. Weiterhin ist die automatische Serialiserung von Java-Images durch das Serializable-Interface nicht möglich.

Um die CRC-Komponenten dennoch zu serialisieren, werden die nicht serialisierbaren Member (Listener, Images, usw.) mit dem Schlüsselwort `transient` versehen. Hierdurch „weiß“ die writeObject()-Methode, die zum Serialisieren eines Objekts aufgerufen wird, dass ein bestimmtes Attribut aus dem Serialiserungsvorgang auszuschließen ist. Die so ausgeschlossenen Member müssen dann entweder „von Hand“ gespeichert oder nach der Deserialisierung neu erzeugt werden.

Die erste Methode wird bei den Images angewendet: Hintergrundimages von eingescannten CRC-Karten erhalten als Wrapper ein `ImageIcon`, das wiederum eine serialisierbare Klasse darstellt. Beim Laden (Deserialiseren) müssen die abgespeicherten Images dann lediglich aus dem ImageIcon ausgepackt werden.

Letztere Methode findet bei den in der Klasse `CRCComponent` untergebrachten Listenern Verwendung. Wurde eine Karte erfolgreich deserialisiert, wird der Konstruktor der Klasse für diese Karte aufgerufen, um die Listener erneut an die Karte zu binden. Im Anschluss daran müssen allerdings die Verweise auf die Karte aktualisiert werden, da der Kontruktor eine neue Instanz anlegt.

6.6.3 Die Steuerung über `WiiCursor`

Als Cursor bezeichnet man, allgemein gesprochen, eine graphische Komponente, die die aktuelle Bearbeitungsposition des Benutzers visualisiert. In grafischen Benutzeroberflächen ist dies für gewöhnlich ein Mauszeiger.

Da CREWW kollaborativ gesteuert wird und der Java-Standard-Cursor nicht

multiuser-fähig ist, war die Implementierung einer solchen Cursor-Klasse vonnöten, in CREWW leistet dies die Klasse `WiiCursor`.

Um deren Swing-Funktionalität nutzen zu können, ist der `WiiCursor` von der Klasse `JComponent` abgeleitet. So lassen sich die `WiiCursor` auf die oberste Ebene der `JLayeredPane` aufbringen (siehe Kap. 6.6.1) und sind somit immer sichtbar. Zu den Attributen, die die `WiiCursor` verwalten, gehört neben ihren graphischen Eigenschaften auch ihr aktueller Arbeitszustand. So merkt sich ein Cursor immer die Aktion, die der Benutzer aktuell durchführt. Dies ist nötig für die Kollaborations-Steuerung, z. B. sind manche Funktionen, wie das Scannen von CRC-Karten, nur möglich, wenn alle Wiimotes `IDLE` sind, die Benutzer also gerade keine Aktion durchführen.

Hierfür bietet die Klasse auch die Objektmethoden `lock()` und `unlock()` an. Mit diesen lässt sich ein `WiiCursor` von außen für weitere Interaktionen sperren und entsperren.

6.6.4 Zeichnen der Komponenten mit `MyContentPane`

In CREWW sollte gewährleistet sein, dass die Kanten immmer hinter den Komponenten gezeichnet werden, da die Komponenteninformation die wichtigere der beiden darstellt. Weiterhin sollte der Hintergrund bei einem Moduswechsel zur Visualisierung dessselben verändert werden.

Realisiert wurde dies durch die Klasse `MyContentPane`. Genutzt werden in dieser hauptsächlich Standardfunktionen, der Vollständigkeit halber soll sie jedoch an dieser Stelle erwähnt werden. Wie das Standard-ContentPane von Swing ist `MyContentPane` von `JPanel` abgeleitet, anders als dieses jedoch als Singleton implementiert.

Die wichtigste Methode der Klasse ist die `paint()`-Methode. In dieser wird die Zeichenreihenfolge festgelegt und der Hintergrund entsprechend dem Modus eingefärbt und beschriftet bzw. mit einem Symbol versehen (Pause/Play). Auf das zugehörige Listing soll an dieser Stelle verzichtet werden, da die beschriebenen Funktionen wie erwähnt Swing-Standardfunktionen darstellen und somit wohl keiner näheren Erklärung bedürfen.

6.7 In and Out - Die Datenschnittstelle und das Paket `io`

So ausführlich die Beschreibung der Implementierung bis zu diesem Punkt war, so knapp soll die der Datenschnittstelle erfolgen. Hier werden ausschließlich Standardmethoden zum Dateihandling (Dateien öffnen, schreiben, lesen usw.) eingesetzt,

von Interesse sind lediglich die Ein- und Ausgabeformate, welche der Vollständifkeit halber im Folgenden beschrieben werden sollen.

6.7.1 Das Speicherformat

In Kap. 6.6.2 wurden bereits die technischen Aspekte beim Abspeichern diskutiert, darum wird sich hier auf die Beschreibung des Formats selbst beschränkt. Dieses könnte bei einer eventuellen Erweiterung des Programms interessant sein. In den von CREWW erzeugten Speicher-Dateien – zu erkennen an der Erweiterung „.crw" – werden die Daten in folgender Reihenfolge abgelegt:

- **mode (int)**
  Aktueller Modus während des Abspeicherns
- **isUseCasePlay (boolean)**
  Play- oder Pause-Modus
- **countEdge (int)**
  Anzahl der abgespeicherten `Edge`s
- **countComponent (int)**
  Anzahl der abgespeicherten `CRCComponent`s
- **countHistory (int)**
  Anzahl der abgespeicherten `CRCComponent`s im aktuellen Use-Case
- **redoList (serialisiert)**
  Redo-List des `UseCaseStack`, zu jeder Komponente werden die ID und die zugehörige Operation (PUSH/POP) gespeichert
- **edges (serialisiert)**
  Kanten (Use-Case und UML)
- **crccomponents (serialisiert)**
  CRC-Komponenten, für gescannte Komponenten wird zusätzlich das Bild in einem `ImageIcon` gespeichert

6.7.2 Exportieren mit `XMIExport`

Nach der Beschreibung des proprietären Speicherformats erfolgt nun die des Datenaustauschformats. Dies genügt dem in Kap. 2.1.4 vorgestellten XMI1.2-Standard. Der Export ist angepasst an das Opensource-UML-Tool ArgoUML und beschreibt die CRC-Komponenten sowie die UML-Kanten. Von diesen werden allerdings nur

Modelldaten exportiert, XMI unterstützt keine Layoutinformationen, weshalb das Layout im UML-Tool erneut festgelegt werden muss.

Wie im vorangegangenen Abschnitt wird sich im Folgenden auf das Format an sich beschränkt, da die Technik der Ausgabeerstellung weniger von Interesse ist. Eine solche XMI-Datei ist wie folgt aufgebaut:

Im Header befinden sich Informationen über die exportierende Software und den benutzten XMI-Standard. Danach folgt der Hauptteil mit dem eigentlichen UML-Modell. Zunächst werden die CRC-Komponenten in die XMI-Datei geschrieben, wobei für jede CRC-Komponente eine beliebige Menge an Responsibilities in die Datei geschrieben werden. Von ArgoUML werden diese beim späteren Import als Attribute interpretiert, ein Typ für diese wird beim Export nicht angegeben.

Nach dem Export der Komponenten erfolgt der der Kanten. Hierbei wird zwischen den einzelnen Kantenarten unterschieden und jeweils ein Verweis auf die beteiligten Klassen angelegt.

Beim XMI-Export ist von größter Wichtigkeit, dass jedes zu exportierende Element über eine dateiweit eindeutige XMI.id verfügt. Über diese ist auch die spätere Referenzierung, z. B. beim Übertragen der Kanten nach ArgoUML, möglich.

Mit den dargelegten Betrachtungen soll nun das Kapitel „Implementierung“ und somit der Hauptteil der vorliegenden Arbeit abgeschlossen werden. Trotz des beträchtlichen Umfangs stellt dieses jedoch keinen Anspruch auf Vollständigkeit, es wurden lediglich einige interessant erscheinende Stellen ausgewählt und näher vorgestellt. Noch offene Fragen werden bei Lektüre des ausführlich dokumentierten Quellcodes sicherlich zur Genüge beantwortet. Mit Goethe wurde begonnen, mit Wittgenstein soll nun abgeschlossen werden:

> „*Dagegen scheint mir die Wahrheit der hier mitgeteilten Gedanken unantastbar und definitiv. Ich bin also der Meinung, die Probleme im Wesentlichen endgültig gelöst zu haben.*“
> — Ludwig Wittgenstein, Tractatus logico-philosophicus

Kapitel 7

Evaluation

Ziel der vorliegenden Arbeit war nicht nur die Implementierung eines stabilen Prototypen, sondern darüber hinaus die Durchführung einer Benutzerstudie zur Evaluation dessen grundlegender Funktionalität. Neben der allgemeinen Nutzerfreundlichkeit stand hier die Frage im Vordergrund, inwiefern die CRC-Methode durch Computerunterstützung profitiert oder ihre Durchführung im Gegenteil durch den Technikeinsatz erschwert wird.

Die Studie wurde im Wintersemester 2009/2010 mit Studenten der Informatik der Universität Trier durchgeführt, eine genaue Beschreibung ihrer Konzeption und Durchführung sowie die Auswertung der Ergebnisse finden sich in den nun folgenden Abschnitten.

7.1 Konzeption

Begonnen werden soll an dieser Stelle mit der Beschreibung der Konzeption der Studie. Bei Usability-Studien wird allgemein zwischen formativ und summativ angelegten unterschieden. Formative Studien dienen der Verbesserung der Gebrauchstauglichkeit und bedienen sich überwiegend qualitativer Methoden, während summative Studien auf die Erlangung empirischer Werte abzielen und somit eher quantitative Methoden nutzen. Die durchgeführte Studie war rein formativ angelegt, methodisch bestand sie aus einem Usability-Test mit anschließendem Fragebogen.

Zum Usability-Test: Im Usability-Test sollten Kleingruppen von je drei bis vier Versuchsteilnehmern mit CREWW eine CRC-Session durchspielen. Vorgegeben waren den Teilnehmern die Anforderungsspezifikation in Form einer Produktbeschreibung (siehe Anhang A.1) und die Systemanwendungsfälle als Use-Case-Diagramm (siehe Anhang A.2). Das anzufertigende Produkt bestand in einer Software für einen Bankautomat mit den gängigen Funktionen. Mittels dieser Vorgaben soll-

ten dann zunächst Kandidatenklassen herausgearbeitet und schließlich ein CRC-Szenario durchgespielt werden.

Getestet wurden insgesamt 14 StudentInnen der Informatik, die alle an der Vorlesung „Softwaretechnik“ teilnahmen. Die CRC-Methode wurde in einer Sitzung , die zwei Wochen vor Durchführung der Studie stattfand, behandelt, wodurch allen Teilnehmern die Vorgehensweise der CRC-Modellierung noch präsent war.

Die Durchführung erfolgte in zwei Vierer- und zwei Dreiergruppen, wobei die Versuchsbedingungen in den Gruppen variiert wurden. Variiert wurden folgende Parameter:

- **Gruppenleitung:** mit/ohne Moderation durch den Versuchsleiter
- **Bedienung im Use-Case:** Navigation im Use-Case und Umschalten der Modi über das `OptionPanel` oder per Wiimote-Button
- **Granulat der Use-Cases:** Vorgabe der Use-Cases oder Vorgabe der Arbeitsschritte in einem Use-Case (siehe Anhang A.3)

Der zeitliche Ablauf des Tests sah folgendermaßen aus: Zunächst wurde den Teilnehmern eine Einführung in CREWW gegeben, hierfür wurden ca. zehn Minuten benötigt. Dann wurden die Blätter mit der Aufgabenstellung (Produktbeschreibung und Anwendungsfälle) verteilt und aufkommende Fragen beantwortet. Nun hatten die Probanden 40 Minuten Zeit, um aus den Vorgaben Klassen zu erarbeiten und mit diesen dann einen Anwendungsfall durchzuspielen. Im Anschluss daran wurde ein eigens für die Studie entworfener Fragebogen bearbeitet, der im Folgenden kurz vorgestellt werden soll.

Zum Fragebogen: Fragebögen zur Evaluierung der Usability von Benutzerschnittstellen (z.B. Isometrics [Dev], SUMI [SUM], Isonorm [Fra])orientieren sich meist an der EN ISO 9241-110, die allgemeine Anforderungen an die Dialoggestaltung festlegt, und enthalten somit meist Fragen aus den folgenden Kategorien:

- **Aufgabenangemessenheit:** Sind die Funktionen des Programms der Lösung der Aufgabe angemessen?
- **Selbstbeschreibungsfähigkeit:** Ist die Benutzung des Programms verständlich (durch geeignete Hilfen, Rückmeldungen usw.)?
- **Steuerbarkeit:** Lässt sich das System gut steuern (auch: Rückgängigmachen/Abbrechen von Aktionen)?
- **Erwartungskonformität:** Verhält sich das System bei Eingaben wie erwartet (konsistente Benutzerführung)?

- **Fehlerrobustheit:** Wo treten Fehler auf und welche Auswirkungen hat dies auf den Programmablauf?
- **Individualisierbarkeit:** Lässt sich das Programm an die Bedürfnisse des Benutzers anpassen ?
- **Erlernbarkeit:** Wie gut lässt sich die Nutzung der Software erlernen?

Bei der Evaluierung von CREWW stand die erste Kategorie im Mittelpunkt. Aus diesem Grund wurden in den Fragebogen hauptsächlich Fragen aus dieser Kategorie aufgenommen. Hier sollte außerdem ein Vergleich mit der traditionellen CRC-Methode (siehe Kap. 3) erfolgen: Wo sahen die Probanden Schwächen oder Stärken der beiden Varianten? Weiterhin musste, da CREWW in den Bereich der Groupware fällt und die angesprochene Norm keine Kollaborationsunterstützung berücksichtigt, eine zusätzlicher Bereich „Kollaborationsunterstützung“ in den Fragenkatalog aufgenommen werden.

Der vollständige Fragebogen befindet sich in Anhang A.4. Er umfasst 17 offene Fragen zu den genannten Kategorien sowie eine Auswahlfrage zu den genutzten Features.

7.2 Durchführung

Die Durchführung der Studie erfolgte im Tagesrhythmus, was ein Anpassen der Versuchsbedingungen nach den Testgruppen vereinfachte. Zusammenfassend lässt sich bereits sagen, dass die Gruppen in der Dynamik stark variierten, dabei jedoch vergleichbare Ergebnisse erzielten.

Auch wenn keine statistische Auswertung angestrebt ist, soll doch festgehalten werden, dass alle Gruppen nur einen der angebotenen Anwendungsfälle bearbeiten konnten. Bei der Hälfte geschah auch dies nur zum Teil, die Gruppen III und IV haben den bearbeiteten Anwendungsfall abgeschlossen.

Dies lag zum einen an der kurzen zur Verfügung stehenden Zeitspanne, zum anderen an der Unerfahrenheit der Probanden. Wie sich im Laufe der Studie zeigte, hatten die wenigsten ein genaues Verständnis der CRC-Methode, daher wäre es von Interesse, den Test zu einem anderen Zeitpunkt mit einer Gruppe von CRC-erfahrenen Anwendern zu wiederholen.

Im Folgenden soll nun auf die Durchführung in den einzelnen Gruppen eingegangen werden.

Gruppe I: Die erste Gruppe bestand aus vier Teilnehmern und bei ihr wurde noch kein Moderator eingesetzt. Dies war insofern nicht notwendig, als sich schnell zwei Wortführer herauskristallisierten, die die Diskussion am Laufen hielten. Zu

den weiteren Parametern ist festzuhalten, dass die Steuerung über das OptionPanel erfolgte und die Use-Cases auf Systemebene angegeben waren.

Als zu bearbeitenden Use-Case wählte Gruppe I den Anwendungsfall „Geldkarte von Konto laden“. Die Gruppe verbrachte viel Zeit (23 Minuten) mit der Identifikation erster Klassen, weswegen in der nächsten Gruppe ein Moderator eingeführt wurde, der die Gruppe zu einem schnelleren Einstieg ins Rollenspiel führen sollte.

Bei Gruppe I zeigte sich bereits ein Effekt, der in der Auswertung erneut angesprochen werden soll: Hier kamen zwei starke Redner, ein mittelstarker und ein eher zurückhaltender Proband zusammen. Bei einer solchen Gruppenzusammenstellung besteht die Gefahr, dass ein Teilnehmer in der Gruppe „untergeht“, also nicht dazu kommt, sich zu äußern.

Daher wäre ein Verbesserungsvorschlag für CREWW die automatische Verteilung der Klassen bei der Generierung. Hiermit wäre zumindest gewährleistet, dass alle Teilnehmer nahezu gleich viele Klassen haben.

Der Anwendungsfall wurde von Gruppe I nicht abgeschlossen, es fehlten noch ca. zwei Bearbeitungsschritte zum Abschluss des Use-Case.

Gruppe II: Wie zuvor angedeutet wurde bei dieser Gruppe, ebenfalls eine Vierergruppe, ein Moderator eingesetzt, dessen Rolle vom Versuchsleiter übernommen wurde. Dieser gab der Gruppe die Vorgabe, vier initiale Klassen zu identifizieren, so dass jedes Gruppenmitglied eine Klasse übernehmen und die Gruppe zügig mit dem Use-Case beginnen konnte.

Der beschriebene Eingriff in den Versuchsaufbau erwies sich als Erfolg: Alle folgenden Gruppen begannen nach ca. zehn Minuten mit dem Rollenspiel.

Bewertet man die Dynamik der Diskussion, so zeigte sich Gruppe II etwas schwächer als die erste. Ein Teilnehmer übernahm hier die Führung und stellte Fragen in den Raum.

Die Steuerung im Use-Case-Modus erfolgte in dieser Gruppe mittels Wiimote. Dies wurde von den Teilnehmern gut angenommen, keiner hatte Probleme mit der Steuerung.

Wie in Gruppe I waren auch hier ausschließlich Use-Cases vorgegeben. Allerdings hatten die Teilnehmer Probleme beim Nachvollziehen der Teilschritte im Use-Case. Es war nicht immer klar, wo genau im Use-Case sie sich befanden und welcher Schritt der nächste war.

Daher wurden in den darauf folgenden Gruppen die Teilschritte in zwei Use-Cases vorgegeben. Eine andere Möglichkeit wäre gewesen, die Probanden als erste Teilaufgabe die Teilschritte herausarbeiten zu lassen. Allerdings hätte dies zusätzlich Zeit gekostet und hinsichtlich des eng gesteckten Zeitrahmens von 40 Minuten und dem Fokus der Untersuchung, der wie schon erwähnt auf dem Tool und nicht auf der CRC-Methode an sich liegt, wurde erstere Lösung bevorzugt.

Gruppe III: In der ersten Dreiergruppe wurde ebenfalls ein Moderator eingesetzt, was in diesem Fall von Vorteil für den Verlauf der Sitzung war. Die Gruppe zeigte sich eher zurückhaltend und musste des öfteren zur Diskussion ermuntert werden. Wenn die Beteiligung der Gruppe insgesamt auch geringer war, so war sie jedoch in dieser Gruppe am ausgeglichensten. Alle Teilnehmer haben sich gleich viel an der Diskussion beteiligt.

Wie zuvor beschrieben wurden hier erstmals die Teilschritte zweier Anwendungsfälle vorgegeben. Hierdurch war es der Gruppe möglich, den Use-Case „Geld abheben“ ganz durchzuspielen.

Von der vorigen Gruppe wurde die Steuerung beibehalten: Auch dieser Gruppe stand nur die Wiimote-Steuerung zur Verfügung und die Teilnehmer kamen ebenfalls gut mit der Steuerung zurecht.

Gruppe IV: Die letzte Gruppe, wie die vorangegangene eine Dreiergruppe, fiel durch eine rege Gesamtbeteiligung auf. Der Versuchsleiter übernahm zwar erneut die Rolle des Moderators, auch hier war wieder die Identifizierung dreier initialer Klassen gefordert, allerdings konnte sich dieser meist im Hintergrund halten. Die Gruppe konnte die Entscheidungen selbst herbeiführen, und wenn ein Zwischenschritt fehlte, wurde dies meist von einem der Probanden kurze Zeit später registriert.

Diese Gruppe konnte wie die erste mit dem `OptionPanel` arbeiten und auch hier ergaben sich keine Probleme.

Der Gruppe standen die Zwischenschritte der Use-Cases zur Verfügung und die Probanden entschieden sich für den Anwendungsfall „Geld abheben“, welcher auch zu Ende geführt wurde.

7.3 Auswertung

Zum Abschluss erfolgt nun die Auswertung der im Fragebogen erhobenen qualitativen Daten. Dies erfolgt zusammenfassend, die exakten Ergebnisse der Befragungen befinden sich in Anhang B. Hierbei soll zuerst die Evaluierung der allgemeinen Gebrauchstauglichkeit (siehe Kap. 7.1: Fragekategorien) erfolgen, um dann den Fokus auf das zentrale Thema der Studie zu richten. Es wird geprüft, inwiefern sich die These bestätigt, dass CREWW die Nachteile der CRC-Methode, bei Beibehaltung der Vorteile, mittels Computerunterstützung aufhebt.

7.3.1 Usability

Den letzten Abschnitt zusammenfassend lässt sich an dieser Stelle zunächst einmal sagen, dass das Tool gut von den Probanden angenommen wurde. Während der

Durchführung gab es keine größeren Probleme hinsichtlich der Steuerung mittels Wiimote und der Gesamtbedienung der Software.

Dies zeigt sich auch in den Antworten im Fragebogen, die zumeist in der Bewertung des Tools positiv ausfallen. Es gab jedoch auch einige Kritikpunkte seitens der Probanden, die zum teil interessante Denkanstöße für mögliche Programmverbesserungen lieferten. Im Folgenden sollen sowohl die hervorzuhebenden negativen als auch die positiven Punkte angeführt werden.

- **Bereich: Selbstbeschreibungsfähigkeit**
  Zu diesem Bereich wurden von den Probanden kaum Angaben gemacht. Ein Hauptgrund hierfür ist wahrscheinlich das zehnminütige Tutorial, das jeder Proband durchlaufen hat. Hier konnten offene Fragen geklärt werden, wodurch die Nutzung der Software soweit ohne Hilfefunktionen möglich war.
  Ein Teilnehmer machte hier den Vorschlag, auch für die CRC-Karten Tooltips mit dem Karteninhalt zu implementieren. Dies wäre z. B. bei zusammengeklappten Karten sinnvoll.

- **Bereich: Steuerbarkeit**
  Auch zur Steuerbarkeit wurden nicht viele Angaben gemacht. Einige Probanden machten den Vorschlag einer Ein-Knopf-Steuerung mit Toolbar, ähnlich einem klassischen Bildbearbeitungsprogramm. Die Umsetzung eines solchen Modus wäre durchaus interessant.
  Eine weitere Forderung war die nach mehr Tastaturen, so dass jeder Benutzer seine eigene Eingabemöglichkeit hätte. Zu überlegen wäre an dieser Stelle die komplette Ersetzung von Maus und Tastatur, allerdings müsste die Wiimote hierfür um grundlegende Funktionen, wie die Bearbeitung von Text, erweitert werden.
  Außerdem gab ein Proband nach der Studie Schmerzen im Handgelenk an. Da das Problem bei keinem anderen Teilnehmer auftrat, wird angenommen, dass diese höchstwahrscheinlich von einer Fehlhaltung bei der Bedienung der Wiimote herrührten.

- **Bereich: Erwartungskonformität**
  Der Bereich „Erwartungskonformität“ wurde von den Probanden positiv bewertet. Hier gab es wenig Kritikpunkte, einer Bestand in der Tatsache, dass sich die Menüs nicht mit der Wiimote-Taste bestätigen ließen, welche die Menüs öffneten. Dieser Kritikpunkt würde mit der zuvor beschriebenen Ein-Tasten-Steuerung wegfallen.

- **Bereich: Fehlerrobustheit**
  Diese Fragenkategorie wurde ebenfalls von fast allen Benutzern ausgelassen. Wenn hier etwas erwähnt wurde, dann ging es meist um Detailfragen. So

hat ein Proband bemängelt, dass sich eine in den Use-Case eingebundene Klasse nicht ohne Zurücksetzen des kompletten Use-Case löschen lässt. Dies ist jedoch de facto nicht möglich, da die Klasse nun einmal im Use-Case enthalten ist und der Anwendungsfall dann ohne diese Klasse keinen Sinn ergibt.

- **Bereich: Individualisierbarkeit**
  Hier wurde von den Probanden bemängelt, dass sich die Kartengröße nicht anpassen lässt. Außerdem wurde von vielen die Möglichkeit gewünscht, die Farben der Cursor einzustellen. In der Tat ergäbe eine Erweiterung von CREWW um die genannten Einstellmöglichkeiten Sinn, lediglich der zeitliche Rahmen hat verhindert, dass diese bereits im Prototyp enthalten sind.
  Ein Proband schlug außerdem vor, die erzeugten Klassen automatisch zu verteilen. Hierdurch könnten zurückhaltende Teilnehmer mehr in das Rollenspiel integriert werden.

- **Bereich: Erlernbarkeit**
  Mit diesem Bereich verhält es sich ähnlich wie mit dem ersten. Da die Teilnehmer zuvor in die Software eingewiesen wurden, hatten die Probanden hier keine Probleme, was sich auch in den Antworten auf diesen Fragenbereich zeigte. Die Teilnehmer empfanden die Software als intuitiv steuerbar und leicht erlernbar.

- **Bereich: Kollaborationsunterstützung**
  Die Kollaborationsunterstützung stellte einen zentralen Bereich der Untersuchung dar. Auch hier wurden nur wenige Mängel festgestellt. So war im Use-Case-Modus nicht immer allen ersichtlich, wer an der Reihe war. Hierfür sollte eine eindeutigere Metapher eingeführt werden.
  Weiterhin wurde, wie im Bereich Steuerung schon erwähnt, von einigen Teilnehmern kritisiert, dass nur eine Tastatur zur Verfügung stand.
  Ein Proband aus einer der Gruppen, welche die Wiimote zur Navigation im Use-Case nutzen durften, kritisierte außerdem die Umsetzung des Vor- und Zurückspringens im Use-Case.
  Die Auswertung der Auswahlfragen zur Kollaborationsunterstützung fiel positiv aus. Fast alle Probanden fanden, dass die Verteilung der Rechte im Use-Case-Modus die Durchführung des Rollenspiels unterstützten.

Soweit zu den Usability-Fragekategorien. Nun folgt noch eine kurze Zusammenfassung der allgemeinen Stellungnahmen, die von den Teilnehmern in den letzten beiden Fragen erwartet wurden.

Die Teilnehmer zeigten auch in diesen letzten Fragen eine breite Zustimmung zum Tool. Einige der Aussagen waren : „sehr intuitive Steuerung“, „hoher Spaßfak-

tor", „übersichtlich und teamfördernd" oder „lädt zum Experimentieren ein". Somit bestätigt sich mit der Auswertung der Fragebögen der Eindruck, der sich bereits bei Durchführung der Studie einstellte: CREWW lässt sich intuitiv steuern und ist optisch ansprechend gestaltet. Selbstverständlich gibt es diverse Möglichkeiten die Software noch zu verbessern – schließlich hat CREWW noch Prototypenstatus – sie stellt aber auf jeden Fall einen Schritt in die richtige Richtung dar.

7.3.2 Gegenüberstellung traditionelle vs. computerunterstützte CRC-Methode

Abschließend soll nun noch auf die zentrale Frage der Evaluation eingegangen werden, ob Computerunterstützung den Einsatz der CRC-Methode vereinfacht oder erschwert.

In Tabelle 3.1 wurden die zu verbessernden Nachteile der traditionellen CRC-Methode angesprochen, und zunächst einmal ist an dieser Stelle zu erwähnen, dass alle dort angesprochenen Punkte von den Probanden aufgezählt wurden. Von der Veränderbarkeit der Karten, über die UML-Schnittstelle bis hin zur Use-Case-Navigation und der Visualisierung des Informationsflusses durch die Use-Case-Kanten: Alle zentralen Features von CREWW wurden als Vorteil der computerunterstützten CRC-Methode aufgezählt.

Gerade die Möglichkeit, CRC-Karten während des Szenarios zu bearbeiten, wurde von vielen Probanden als Vorteil von CREWW gesehen. Die Erfahrung im CRC-Rollenspiel bestätigt dies, da ein Ziel der Anforderungsanalyse mit CRC-Karten ja gerade die Festlegung von Bezeichnungen für zu modellierende Entitäten ist: Die Identifizierung eindeutiger Namen für Objekte ist nicht etwa trivial, sondern zentraler Bereich der Analysephase. Daher kommt es im Laufe des Rollenspiels öfter zu Namensänderungen, die nötig werden, um Mehrdeutigkeiten zu vermeiden oder Zuständigkeiten klarer auszudrücken. Dies führt bei der traditionellen Methode zwangsläufig zu Mehrarbeit oder eben zu unleserlichen Karten.

Weiterhin empfanden die meisten Probanden die computerunterstützte Methode insofern übersichtlicher, als eine Änderung an einer Karte direkt allen Teilnehmern visuell zur Verfügung stand. Ein Proband erwähnte, dass somit durchaus auch sehr viele Personen („20 Mann") an einer CRC-Session teilnehmen könnten. Die Methode bietet aber auch eine bessere Übersicht bezüglich der Zugehörigkeit der Klassen: Durch die Farbgebung ist eine klare Zuordnung der Klassen zu den Benutzern möglich.

Betrachtet man nun die Punkte, die von den Probanden als Vorteile der traditionellen Methode genannt wurden, so ergeben sich, wie erwartet, auch hier Überschneidungen mit Tabelle 3.1.

So wurde von einigen Teilnehmern der größere technische Aufwand als Nachteil

der computerunterstützten Methode gesehen. Bei der traditionellen Methode wird im Grunde nichts außer Papier und Bleistift benötigt und dies stellt durchaus einen Vorteil dar.

Ebenfalls hervorgehoben wurde der zeitliche Aspekt: Das Schreiben der Karten von Hand geht zunächst einmal schneller vonstatten, als dies beim Eintippen über die Tastatur der Fall ist. Hier ist jedoch anzumerken, dass für Verbesserungen der Karten wiederum mehr Zeit benötigt wird. Daher ist davon auszugehen, dass die für Notation benötigte Zeit in beiden Methoden in der Summe etwa gleich ist.

Die meisten Probanden sahen jedoch die dort nötige Einarbeitungszeit als Manko der computerunterstützten Methode. Diese entfällt bei der traditionellen Methode, da anzunehmen ist, dass den meisten Menschen das Arbeiten mit Papier und Bleistift vertraut ist. Einschränkend soll hier jedoch erwähnt werden, dass alle Probanden nach einer Einarbeitungszeit von nur zehn Minuten gut mit der Software zurecht kamen, womit dieser Kritikpunkt als zu vernachlässigend betrachtet werden kann.

Zusammenfassend soll an dieser Stelle darauf hingewiesen werden, dass die zwei zentralen Punkte „Gruppenarbeit“ und „starke Identifikation“, die gerade die Stärke der CRC-Methode darstellen, von den Probanden als Vorteil sowohl der traditionellen als auch der computerunterstützten Methode angesehen wurden. Dies ist ein Hinweis darauf, dass die Aspekte Kollaboration und Identifikation durch CREWW erhalten werden konnten, was eine zentrale Forderung darstellt.

Somit sind die zwei Anforderungen an CREWW– Ausgleich der Schwächen sowie Beibehalten der Stärken der traditionellen CRC-Methode – weitgehend erfüllt, eine abschließende Betrachtung und Bewertung der durchgeführten Arbeiten liefert das nächste Kapitel.

Kapitel 8

Fazit & Ausblick

Rückblickend erfolgt nun eine bewertende Gesamtbetrachtung der durchgeführten Arbeiten: Welche Erkenntnisse konnten gewonnen werden und wurden die gesteckten Ziele erreicht?

Mit Blick auf die im vorangegangenen Kapitel vorgestellte Studie lässt sich sagen, dass das Ziel, den Prototyp für eine kollaborative Software zur Durchführung einer CRC-Sitzung zu implementieren, erreicht wurde. Selbstverständlich lässt die Studie ob ihrer methodischen Orientierung keine statistische Aussage in Form einer zahlenmäßigen Bewertung zu, doch deutet sie in eine klare Richtung. Die CRC-Methode lässt sich sehr wohl computerunterstützt durchführen, nur muss die Bedienung der Software stimmen.

Als ausgesprochen gute Entscheidung hat sich hier die Wahl der Wiimote als Eingabegerät gezeigt. Das zentrale Element der CRC-Methode ist eben das Rollenspiel und die Wiimote trägt entscheidend zum Erhalt des Spiel-Charakters der Methode bei. Dies konnte sowohl in der Studie als auch bei Test-Usern im Verlauf der Entwicklung beobachtet werden: Die Benutzer hatten Spaß, ein sehr wertvoller Effekt, der mit klugen Algorithmen und hübschen Designs schwer zu erreichen ist.

Auch implementierungsseitig gehört die Einbindung der Wiimote-Steuerung zu den interessantesten Bereichen von CREWW. Die Entwicklung einer komplett neuartigen Steuerung – einschließlich einer Zugriffssynchronisation auf reservierte GUI-Komponenten – stellte eine interessante Herausforderung dar, die mit den in Kap. 6 vorgestellten Strukturen jedoch gut bestanden wurde.

Die größte Herausforderung bei der Implementierung bestand jedoch in der Zusammenführung der verschiedenartigen Themenbereiche. Sowohl in den zu Beginn der vorliegenden Arbeit aufgezeigten Fragestellungen (siehe Kap. 1) als auch in den vorgestellten Grundlagen (siehe Kap. 2) zeigt sich die Vielfalt der zu bearbeitenden Themengebiete. Und dies ist nicht nur im theoretischen (Softwardesign, CRC, Kollaboration), sondern auch im praktischen Bereich der Fall: Hier etwas C, dort ein wenig C++, dann wieder Java – eine breite Palette an Programmiersprachen

musste zur Lösung der Probleme eingesetzt werden.

Zusammenfassend lässt sich nun sagen, dass auch diese Herausforderung angenommen und die gestellten Probleme weitgehend gelöst wurden. Jedoch ergeben sich durchaus noch einige Verbesserungsmöglichkeiten, auf die im Folgenden eingegangen werden soll.

Einige der in Betracht zu ziehenden Arbeiten beziehen sich auf die Benutzerschnittstelle, andere auf den Programmablauf und die integrierten Komponenten. Zunächst soll hier auf Erstere eingegangen werden, bevor dann zum Abschluss des Kapitels die aufwändigeren Verbesserungen der Programmstruktur angesprochen werden sollen.

Benutzerschnittstelle und –interaktion

Farbwahl: Die Benutzer sollten die Möglichkeit bekommen, über ein Menü die Farbe ihres Cursors einzustellen.

Dynamische CRC-Karten: Die Größe der CRC-Karten sollte sich dynamisch anpassen lassen.

Panels in Farbe des Benutzers: Wenn personalisierte Panels eingeblendet werden, sollte, z. B. über ihre Hintergrundfarbe, eine einfache Zuordnung zu ihrem Besitzer möglich sein.

Ein-Knopf-Steuerung: Zu überlegen ist die Einführung einer Toolbar, so dass auf eine Ein-Knopf-Steuerung umgestiegen werden kann. Hierfür müssten verschiedene Cursor-Visualisierungen implementiert werden, die dann für die aktuell ausgewählte Aktion stehen würden.

Ersetzung von Maus und Keyboard: Erstrebenswert, aber auch mit hohem Aufwand verbunden, wäre die vollständige Ersetzung von Maus und Tastatur durch die Wiimote. Möglich wäre dies unter Umständen durch das Einführen von Wii-Gesten (siehe nächster Punkt)

Einführung von Wii-Gesten: Gesten sind eine intuitive Möglichkeit der Benutzerinteraktion. Eine Integration in CREWW wäre in Betracht zu ziehen.

Vibration stärker nutzen: Mit der Rumble-Funktion bietet die Wiimote die Möglichkeit, dem Benutzer ein haptisches Feedback zu geben. Dies sollte noch öfter bei nicht-erlaubten Aktionen genutzt werden.

Programmablauf

Liste von Teilschritten im Use-Case mitführen: Beim Durchspielen eines Anwendungsfalles kam es hin und wieder zu Verwirrungen über die genaue Aktionsabfolge im Use-Case. Hierfür könnte eine Liste der zu vollziehenden Teilschritte (siehe Anhang A.3) mitgeführt werden, wodurch den Benutzern zu jedem Zeitpunkt der aktuelle Bearbeitunsgstanmd des Use-Case zur Verfügung stünde.

Modus zur Modellierung von Use-Cases: In einem dritten noch zu implementierenden Modus könnte die Erstellung und Verwaltung von Use-Cases ermöglicht werden. Für jeden Use-Case könnten dann mehrere Szenarien angelegt und durchgespielt werden.

Automatische Verteilung neuer Klassen: Wie schon in der Auswertung der Studie vorgeschlagen, könnte eine Funktion implementiert werden, die neu erstellte Klassen automatisch dem Benutzer zuweist, welcher zum aktuellen Zeitpunkt die wenigsten Klassen besitzt. Hierdurch würde eine gleichmäßige Verteilung der Klassen erzwungen.

Automatische Verwaltung der Collaborators: Collaborators könnten von CREWW automatisch bei der Delegation einer Aufgabe erfasst werden. Wenn Klasse A an Klasse B delegiert, könnte B im Collaborators-Feld von A notiert werden.

Komponenten

Opensource-ICR finden und integrieren: Zu einem späteren Zeitpunkt wäre die feste Integration einer Opensource-ICR-Lösung durchaus von Interesse, da das Erkennen von handschriftlichen Karten die Benutzerfreundlichkeit erheblich steigern würde.

Server optimieren: Der Server zeigte sich auch nach den vorgenommenen Verbesserungen noch fehleranfällig. Wiimotes wurden z.T. nicht erkannt und vereinzelt kam es noch zu Serverabstürzen. Da eine Forderung an CREWW (siehe Tab. 3.1) eine einwandfrei funktionierende Technik war, ist die Implementierung eines robusten Servers durchaus anzustreben.

Komfortableres Scannerinterface: Mit einem tieferen Einstieg in die TWAIN-API verbunden wäre die Implementierung eines komfortableren Scannerinterfaces. Die aktuelle Vorgehensweise stellt ein akzeptables Workaround dar, ist jedoch verbesserungswürdig.

Mit den hier dargelegten Betrachtungen soll die vorliegende Arbeit nun abgeschlossen werden. Was bleibt also, wenn man die gerade aufgeführten Punkte in ein Verhältnis zu angestrebten und erreichten Zielen setzt?

CREWW ermöglicht die Durchführung der CRC-Methode mit modernen Mitteln, wobei die Stärken der traditionellen Methode erhalten bleiben und die Schwächen, wenn auch nicht aufgehoben, so doch zumindest nivelliert werden. Einschränkend muss allerdings auf den Prototypenstatus der Software hingewiesen werden: CREWW kann nur einen ersten Schritt darstellen, doch erfolgt dieser auf jeden Fall in die richtige Richtung.

Anhang A

Materialien: Studie

A.1 Produktbeschreibung: Software für Bankautomat

Ziel des heutigen Treffens ist der Softwareentwurf für einen Bankautomat. Mit diesem sollen die Funktionen Ein- und Auszahlung, Anzeigen des Kontostands, Ausdrucken der Kontoauszüge sowie Aufladen der Geldkarte möglich sein. Das Aufladen der Geldkarte soll wahlweise vom Konto oder von einem manuellen Geldscheinslot erfolgen können. Die Funktionen „Auszahlung“ und „Geldkarte aufladen“ sollen nur über vorhergehende Authentifizierung durch PIN-Eingabe möglich sein.

Bei dreimaliger Falscheingabe der PIN wird die Karte automatisch gesperrt. Entsperrt werden kann sie von einem Servicemitarbeiter der Bank.

Während des gesamten Vorgangs soll der Nutzer durch geeignete Bilschirmnachrichten über die anstehende Aktion, bzw. mögliche Optionen informiert werden.

Die Konto- und PIN-Verwaltung sind nicht Teil des zu modellierenden Systems. Diese stehen dem System als fertige Komponenten zur Verfügung.

A.2 Use-Case-Diagramm Bankautomat

Das Use-Case-Diagramm zu dem in Anhang A.1 beschriebenen System sieht wie folgt aus:

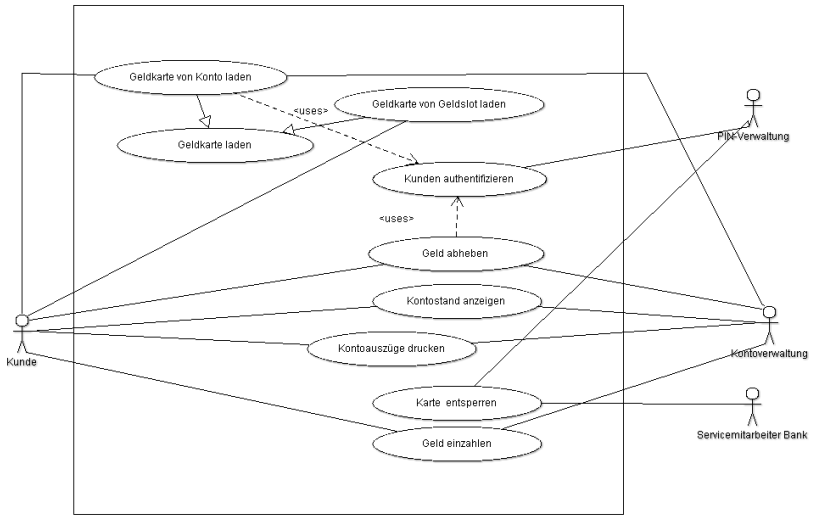


Abbildung A.1: Use-Case-Diagramm Bankautomat

A.3 Ablauf der Anwendungsfälle

Use Case: Geld abheben

- Karte in Automat stecken
- Optionen anzeigen
- „Von Konto abheben“ wählen
- Mögliche Beträge anzeigen
- Betrag wählen
- PIN verifizieren
- Prüfung: Betrag möglich
- Karte ausgeben

- Geld ausgeben
- Willkommensbilschirm für nächsten Kunden anzeigen

Use Case: Gedlkarte von Geldslot laden

- Karte in Automat stecken
- Optionen anzeigen
- „Geldkarte laden“ wählen
- Ladeoptionen anzeigen
- „Von Geldslot laden“ wählen
- Geldscheine akzeptieren
- Taste „Fertig“
- Geldkartenstand aktualisieren
- Karte ausgeben
- Willkommensbilschirm für nächsten Kunden anzeigen

A.4 Fragebogen

Aufgabenangemessenheit

Frage 1

Welche Funktionen des Programms wurden von Ihnen genutzt?

Einscannen der Karten	( )	Szenario laden	( )
Karteneingabe von Hand	( )	Anlegen von UML-Kanten	( )
Karten verschieben	( )	Ändern von UML-Kanten	( )
Karten löschen	( )	Löschen von UML-Kanten	( )
Szenario exportieren	( )	Komponenten auf-/zuklappen	( )
Szenario speichern	( )		

Frage 2

Welche Funktionen haben Ihnen zur Lösung der Aufgabe gefehlt?

Frage 3

Wo sehen Sie die Vorteile der computerunterstützten Methode?

Frage 4

Wo sehen Sie die Nachteile der computerunterstützten Methode?

Frage 5

Wo sehen Sie die Vorteile der CRC-Methode ohne Computerunterstützung?

Frage 6

Wo sehen Sie die Nachteile der CRC-Methode ohne Computerunterstützung?

Selbstbeschreibungsfähigkeit

Frage 7

Haben Sie Hilfefunktionen in Anspruch genommen, haben Hilfefunktionen gefehlt, haben Sie Meldungen nicht verstanden, ... ?

Steuerbarkeit

Frage 8

An welchen Stellen des programmablaufs hätten Sie sich mehr Interaktivität gewünscht oder diesen abbrechen wollen, ... ?

Erwartungskonformität

Frage 9

An welchen Stellen hat sich die Software anders verhalten, als von Ihnen erwartet, z.B. Knöpfe, die eine andere Funktionalität hatten, als erwartet, Meldungen, die an der „falschen“ Stelle auftauchten, ... ?

Fehlerrobustheit

Frage 10

Wo traten Programmabstürze auf, welche Fehlermeldungen waren missverständlich, haben Sie Fehler gemacht, die Sie nicht mehr rückgängig machen konnten (versehentlich ausgelöste Aktionen etc), ... ?

Individualisierbarkeit

Frage 11

Welche Einstellungsmöglichkeiten haben Sie vermisst?

Erlernbarkeit

Frage 12

Was hätte Ihnen die Erlernbarkeit der Software erleichtert?

Kollaborationsunterstützung

Frage 13

An welchen Stellen traten Probleme mit den anderen Benutzern auf?

Frage 14

War Ihnen immer klar, welcher Benutzer welche Rechte hat?

ja ( ) nein ( )

Frage 15

Sollte man die Rechte im Programmablauf weniger einschränken?

ja ( ) nein ( )

Allgemeine Stellungnahme

Frage 16

Was hat Ihnen bei der Benutzung von CREWW gut gefallen und passte bis jetzt in keine Fragenkategorie?

Frage 17

Was hat Ihnen bei der Benutzung von CREWW nicht gefallen und passte bis jetzt in keine Fragenkategorie?

Anhang B

Auswertungsergebnisse

Antworten zu Frage 1

- *Einscannen* genutzt : 0
- *Karteneingabe von Hand* genutzt: 10
- *Karten verschieben* genutzt: 13
- *Karten löschen* genutzt: 10
- *Szenario exportieren* genutzt: 0
- *Szenario speichern* genutzt: 0
- *Szenario laden* genutzt: 0
- *Anlegen von UML-Kanten* genutzt: 2
- *Ändern von UML-Kanten* genutzt: 2
- *Löschen von UML-Kanten* genutzt: 2
- *Komponenten auf/-zuklappen* genutzt: 9

Antworten zu Frage 2

- Mehrere Eingabemöglichkeiten
- Mehr Tastaturen
- Beim Löschen einer Karte sollte nicht der Ganze Ablauf gelöscht werden, sondern nur die Karte mit ihren Verbindungen

Antworten zu Frage 3

- Intuitive Steuerung mittels der Wii-Controller und die Interaktion mit den Mitteilnehmern, da hier leicht Dinge verbessert werden können
- Leichte Veränderung
- Abarbeitungsstapel durch animierte Linien nachvollziehbar und durch Player jederzeit wiederholbar
- Fehler können bequem berichtigt werden
- Übersichtlichkeit: selbst wenn von 20 Mann nur einer mit der Jedi-Fernbedienung arbeiten würde, die restlichen 19 Mann würden alles mitbekommen. Beim Skizzieren auf einem DIN A3 Blatt wäre das nicht der Fall
- Einfaches und schnelleres Anlegen von karten
- Einfaches Anpassen der CRC-Karten
- Einfaches Editieren am Bildschirm
- Hat man sicht erstmal eingearbeitet, ist es übersichtlicher und strukturierter als Skizzen auf Papier
- Das Ganze ist übersichtlicher als auf papier
- Daten sind besser änderbar
- Gleichzeitiges Arbeiten an einer Modellierung
- Es ist wesentlich übersichtlicher als alles per Hand zu machen und man kann einfacher Dinge verändern
- Leicht zu verändern
- Gute Visualisierung des Use Case Mode Klare Verantwortlichkeiten für CRC-Karten
- Übersichtlich
- Alle Teammitglieder sehen auf einen Blick die Änderungen
- Schnelles Übertragen an Kunden möglich, z. B. Email
- Änderung von Parametern der Klassen zum Zeitpunkt des Szenarios
- Geht schneller und sieht sofort gut aus
- Es ist einfach die Karten zu verschieben oder zu löschen
- Umweltfreundlicher (weil kein Papier)
- Klares Rollenspiel (es ist auch nach der Diskussio noch bewusst, welche Klasse aktiv ist)

Antworten zu Frage 4

- Man muss sich zunächst in die Arbeitsweise der Controller gewöhnen
- Eingabe über Tastatur etwas umständlich
- Verlauf der Prozesse ist anfangs nicht direkt sichtbar
- Jeder Teilnehmer kann nur seine Karte verschieben
- Braucht PC
- Einarbeitungszeit ist nicht ohne. Wenn nur eine simple Lösung gesucht ist, ist man mit einem größeren Blatt Papier schneller fertig
- Evtl. zu viele Klassen, zu viele Möglichkeiten um unnötige Szenarien auszugestalten
- Es werden spezielle Software benötigt und spezielle Einarbeitungsphasen in die Software nötig
- Wenn alle gleichzeitig am Bildschirm interagieren wird es schnell unübersichtlich
- Nicht intuitiv
- Man braucht viel Platz und gutes Material
- Unter Umständen weniger flexibel
- Zusätzlicher Einarbeitungsaufwand
- Die Bedienung muss erstmal eingeübt werden
- Jetzt mit diesem Tool konnte immer nur einer beschriften. Das könnte die Entwicklung evtl. verlangsamen. Aber sonst...
- Braucht mehr zeit als auf einen Zettel zu schreiben

Antworten zu Frage 5

- Niedrigerer Aufwand (kein Beamer, keine Konsole etc.)
- Handschriftliche Erstellung schnell und natürlicher
- Man hat was in der Hand Man denkt vielleicht schärfer nach um unnötige Arbeit zu sparen (Erstellung CRC-Karten)
- Aktivierte Beteiligung aller Gruppenmitglieder am Entwurf -> mehr ideen und größeres Verständnis des Modells
- Erst denken dann schreiben „Kern“ des Prozesses wird besser bearbeitet
- Einfache Einarbeitung
- Anwendungsfälle werden von mehreren Personen gleichzeitig durchgespielt Unvollständige Klassen und fehlende Klassen werden schnell gefunden und ergänzt
- Einigwerden über Begriffe, Akteure, Funktionen, Abhängigkeiten...
- Erste Szenarios durchspielen
- Überblick verschaffen Diskusione erstmal ohne Zwang
- „Intuitive“ Bedienung von Papier und Stift
- Einbindung aller Beteiligten
- Ortsunabhängig

Antworten zu Frage 6

- Änderungen können nicht so schnell und einfach realisiert werden
- Ablauf der Anwendungsfälle nicht einfach, man muss sich alles im Kopf behalten
- Ablauf kann nur mit unbequemen Methoden aufgezeichnet werden
- Ist überall und zu jedem Zeitpunkt durchführbar
- Jedes Teammitglied stellt sich evtl. den Ablauf etwas anders vor
- Es wird nicht ganz klar wie das Zusammenspiel ablaufen soll
- Aufwendiges Verfahren
- Nicht so einheitlich
- Nicht sehr schön, da man Sachen streichen muss und nicht wie auf dem PC einfach löschen kann
- Papierschlacht bei großen Projekten
- Ändern der karten zu pr*****, ist nicht **** und nicht so unmittelbar

Antworten zu Frage 7

- Keine Programmhilfe in Anspruch genommen
- War im Use Case nicht ganz klar wie man eine Verantwortlichkeit abgeben kann
- Nein, keine Hilfefunktion nötig
- Benutzt: Zusammenklappen der Tabellen
- Fehlt: Zoom in/Zoom out Selbstbeschreibungsfähigkeit nicht groß
- Alles verstanden, da uns alles erklärt wurde
- Nein
- Keine Hilfefunktion benutzt

Antworten zu Frage 8

- Im USE CASE Modus wäre es schöner, alle Karten verschieben zu können, als nur die eigenen
- Software war leicht steuerbar
- Verbindungen zwischen den Karten evtl. durch Pfeile ersetzen
- Buttens zur Erstellung von Klassen/Relationen
- Beim Erstellen der Karte automatisch Feld zur Namenseingabe
- Alle Teilnehmer sollten die Möglichkeit haben die Klassen zu beschreiben
- Evtl. wäre dies erst nach intensiverer Nutzung zu erfahren

Antworten zu Frage 9

- Neu erstellte karten werden nicht dort angelegt, wo man sie mit dem Cursor platziert hatte
- Durch die bildhaften Beschreibungen der Wii-Bedienung hat sich die Software auch für einen nicht Wii-Benutzer wie erwartet verhalten
- Beim Aufruf des Menüs zur Erstellung einer neuen Karte mit „+“ hätte ich auch erwartet, dass ich mit „+“den entsprechenden Eintrag auswählen kann
- Karten erscheinen nicht an der Stelle wo man sie erstellt und verdecken ggfls. Andere
- Knöpfe waren gut gewählt
- Mir hat die Möglichkeit gefehlt, eine Entscheidung direkt rückgängig zu machen, da das nur über den Button oben ging

Antworten zu Frage 10

- Keine Abstürze
- Keine
- Maus hat nicht funktioniert für einige Zeit
- Wir wollten den Kunden löschen, war aber zu umständlich weil er schon eingebunden war
- Keine Auffälligkeiten
- Wenn man z. B. eine neue Karte anlegen will und das Popup-Menü die Menüleiste überlappt, werden 2 Funktionen ausgeführt, statt nur die des obersten Buttons
- Beim einscannen

Antworten zu Frage 11

- Änderung der Kartengrößen (für Karten mit vielen Aufgaben)
- Zufällige Zuweisung der Karten, um alle Teilnehmer gleichberechtigt einzubinden
- Anpassung der Spaltenbreite der CRC-Karten
- Farben anpassen, um die Erkennbarkeit zu erleichtern bzw. Rot/Grün-Schwäche
- Farbwahl für Teilnehmer und eindeutige Farben zu generien
- Es gab keine Einstellungsmöglichkeiten

Antworten zu Frage 12

- Die vorliegende Beschreibung reichte vollkommen aus!
- Software war sehr intuitiv und daher leicht erlernbar
- Tutorial
- Der Zwang, dass jeder 1-2 Klassen selbst anlegen und verschieben soll
- War bereit einfach zu bedienen nach Einführung Langes bzw. kurzes drücken eines Knopfes wäre durch einen alternativen Knopf besser
- Tooltips mit Infos bei längerem Rollover über die Karten
- Einstieg war so ziemlich gut
- Ein Beispiel am Anfang
- Besser als „persönlich erklärt“ ist nur schwer möglich
- Anderer Controller - Schmerzen im Handgelenk

Antworten zu Frage 13

- In der Play-Phase der USE CASES war es nicht immer ersichtlich, welcher Teilnehmer an der Reihe ist
- Keine
- Es hat sich nicht immer jeder getraut Eingaben per Tastatur zu machen bzw. Vorschläge zum weiteren Vorgehen
- beim USE CASE Mode
- Bei der Farbwahl der Punkte, gelb und orange sind kaum voneinander zu unterscheiden
- Beim Vor-/Zurückspringen während der Simulation
- Farbwahl der CRC-Karten könnte besser angepasst werden
- Weil man nicht genau erkannte wer im Play-Modus dran war
- Tastatureingabe

Antworten zu Frage 14

- **Ja**: 11 **Nein**: 3

Antworten zu Frage 15

- **Ja**: 3 **Nein**: 11

Antworten zu Frage 16

- Sehr Intuitive Steuerung
- Hohes Zusammenspiel mit den anderen Teilnehmern
- Höherer Spaßfaktor
- Erfrischendes medium zum lernen
- weckt Interesse (Spieltrieb)
- Geniale Steuerung
- CREWW lädt zum experimentieren ein, da alles rückgängig gemacht werden kann
- Fühlt sich gut in der Hand an
- Vibration beim Player
- Sieht nicht schlecht aus
- Sehr gute Darstellung
- gute Möglichkeit im Team an einem Prozess zu arbeiten, sodass jeder seine Ideen/Anregungen einbringen kann
- Visualisierung im Play-Modul
- Bessere Interaktivität mit Benutzern als im nicht computergesteuerten Fall
- Übersichtlich
- Teamfördernd
- Relativ intuitiv

Antworten zu Frage 17

- Zu viele verschiedene Tasten, eventuell auf wenige Haupttasten beschränken und die Funktion vom Kontext abhängig machen
- Maus nicht nötig (zu viele Eingabegeräte)
- Die Einarbeitungszeit ist nicht zu vernachlässigen
- Es wäre von Vorteil, Relationen während der Simulation noch zu erstellen, d.h. das Projekt direkt während der Simulation zu entwickeln
- Das man schlecht die Farben der Karten im Play-Modul erkannte
- Teilweise ungenaue Steuerung über die WiiMote
- Teilweise musste die WiiMote relativ unbequem hoch gehalten werden, damit eine Bewegung bemerkt wurde

Literaturverzeichnis

[Arg] ArgoUML 0.28.1. http://argouml.tigris.org/.

[BC89] Beck, Kent und Ward Cunningham: *A Laboratory For Teaching Object Oriented Thinking*. In: *OOPSLA*, 1989.

[Boe81] Boehm, Barry: *Software Engineering Economics*. Prentice Hall, 1981.

[Bör05] Börstler, Jürgen: *Improving CRC-Card Role-Play with Role-Play Diagrams*. In: *OOPSLA*, Seiten 356 – 364, 2005.

[BS97] Bellin, David und Susan Suchman Simone: *The CRC Card Book*. Addison-Wesley, 1997.

[CY91a] Coad, Peter und Edward Yourdon: *Object-Oriented Analysis*. Yourdon Press, 1991.

[CY91b] Coad, Peter und Edward Yourdon: *Object-Oriented Design*. Yourdon Press, 1991.

[Dev] Development of a software usability instrument. http://www.isometrics.uni-osnabrueck.de/.

[Fac] Fach Psychologie der Universität Freiburg. http://www.psychologie.uni-freiburg.de/Members/rummel/wisspsychwiki/wissenspsychologie/CSCW/.

[Fra] Fragebogen Isonorm. http://www.ergo-online.de/site.aspx?url=html/software/verfahren_zur_beurteilung_der/fragebogen_isonorm_online.htm.

[Gam05] Gamma, Erich: *Design Patterns - Elements of Reusable Object Oriented Software*. Addison Wesley, 2005.

[GOC] GOCR – GNU Optical Character Recognition. http://www.gocr.de/.

[Goo] Google Codebase. http://code.google.com/intl/de-DE/.

[HS94] Hasenkamp, Ulrich und Michael Syring: *CSCW in Organisationen - Grundlagen und Probleme.* In: *CSCW - Computer Supported Cooperative Work*, Seiten 13–35. Addison-Wesley, 1994.

[IEE98] IEEE830: *IEEE Std. 830 - IEEE Recommended Practice for Software Requirements Specifications*, 1998.

[Joh] Johnny Chung Lee – Wii-Projects. http://johnnylee.net/projects/wii/.

[JTw] JTwain – Acquire Images with TWAIN and SANE. http://today.java.net/article/2004/11/16/java-tech-acquire-images-twain-and-sane-part-1/.

[Mül08] Müller, Sabine: *Java-API für Mehrbenutzerinteraktion mit Wii-Remotes.* Diplomarbeit, Universität Trier, 2008.

[OCR] OCRopus. http://code.google.com/p/ocropus/.

[Oes04] Oestereich, Bernd: *Objektorientierte Softwareentwicliung - Analyse und Design mit der UML 2.0.* Oldenbourg Verlag München Wien, 2004.

[PR09] Pohl, Klaus und Chris Rupp: *Basiswissen Requirements Engineering - Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level.* dpunkt.verlag, 2009.

[Roy87] Royce, W. W.: *Managing the Development of Large Software Systems: Concepts and Techniques.* In: *ICSE*, Seiten 328–339, 1987.

[Sei93] Seitz, D.: *Gruppenarbeit in der Produktion - Ein Beitrag zur Systematisierung von Entwicklungsstand und Perspektiven.* In: *Entwicklung der Gruppenarbeit in Deutschland*, Seiten 32–74. Campus-Verlag, 1993.

[STI] STIHRS – Scanner/Touchscreen Input Handwriting Recognition Software. http://stihrs.sourceforge.net/.

[SUM] SUMI Questionnaire Homepage. http://sumi.ucc.ie/.

[Tes] Tesseract OCR. http://code.google.com/p/tesseract-ocr/.

[TIO] TIOBE Programming Community Index for October 2009. http://www.tiobe.com/index.php/content/paperinfo/tpci/.

[TWA] TWAIN – Standard for image acquisition devices. http://www.twain.org/.

[UML] UML Specification 2.2 - 2009. http://www.omg.org/spec/UML/2.2/.

[WBW89] Wirfs-Brock, Rebecca und Brian Wilkerson: *Object-Oriented Design: A Responsibility-Driven Approach.* In: *OOPSLA '89*, 1989.

[WBWW90] Wirfs-Brock, Rebecca, Brian Wilkerson und Lauren Wiener: *Designing Object Oriented Software.* Prentice Hall, 1990.

[Wii] Wiiuse – The Wiimote C Library. http://www.wiiuse.net/.

[Wil99] Wilkinson, Nancy M.: *Using CRC cards : an informal approach to object-oriented development.* Cambridge University Press, 1999.

[XMI] XMI Specifications 1.2 - 2002. http://www.omg.org/cgi-bin/doc?formal/2002-01-01.

Abbildungsverzeichnis

Listingverzeichnis