Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Oliver Köckritz

WeFace: Eine Software für soziale Interaktion mit nahtloser Komposition verteilter Arbeitsflächen.

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Oliver Köckritz

WeFace: Eine Software für soziale Interaktion mit nahtloser Komposition verteilter Arbeitsflächen.

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke

Abgegeben am 18. November 2010

Oliver Köckritz

Thema der Masterarbeit

WeFace: Eine Software für soziale Interaktion mit nahtloser Komposition verteilter Arbeitsflächen.

Stichworte

Co-located Collaborative Workspace, soziale Interaktion, Koordinationsdesign, Multi-Display Environment, multiple Oberflächen, Ambient Intelligence, Seamless-Interaction, Selbst-Konfiguration

Kurzzusammenfassung

Die wachsende Anzahl und Vernetzung von Rechnern und die gemeinsame Interaktion bei der Nutzung dieser wirft in Bezug auf die etablierte Desktopmetapher viele Fragen auf. Für diese Fragen sind im letzten Jahrzehnt viele problemspezifische oder abstrakte Lösungsansätze entstanden. Für die Erarbeitung einer allgemein gültigen Lösung wird in dieser Arbeit der Sachverhalt des Co-located Collaborative Workspaces aus verschiedenen Perspektiven neu analysiert und strukturiert. Die daraus folgende Kategorisierung von Handlungen und aggregierten Anforderungen bilden die Grundlage für das daraus entwickelte Software-Design für eine verteilte Interaktionsschicht. Der zur Evaluation des Designs entstandene Prototyp WeFace wird vorgestellt und reflektiert. WeFace ermöglicht es Entwicklern und Benutzern sozial zu interagieren und auf einer gemeinsamen Arbeitsoberfläche zu arbeiten, die aus einzelnen Arbeitsoberflächen zu einer Collage zusammengefügt wurde.

Oliver Köckritz

Title of the paper

WeFace: A software for social interaction with a seamless composition of spreaded surfaces.

Keywords

co-located collaborative workspace, social interaction, coordination design, multi-display environments, multi-surface, ambient intelligence, seamless-interaction, self-configuring

Abstract

The increasing number of computers, its networking and the collaborative interaction of users raises many questions concerning the established desktop metaphors. Within the last decade many suggested solutions revolving around one special problem or abstract suggested solutions dealing with these questions have been developed. In order to find a generally valid solution I am going to analyse the facts of Co-located Collaborative Workspaces from different perspectives and structure them in a new way. The resulting categorisation of actions and aggregated requirements form the basis of this newly developed software-design of a spreaded layer of interaction. The prototype WeFace, which was created to evaluate the designs, is introduced and reflected. It enables designers and users to socially interact and work on one surface, which was joined together out of different surfaces.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Interaktion mit elektronischen Verarbeitungsmaschinen verändert sich stetig und passt sich der aktuellen Technik und Problematik an. Bis Mitte der 80er Jahre wurde noch über einen Command-Prompt mit dem Rechner interagiert. Ab 1984 etablierte sich die WIMP-Oberfläche als Desktopmetapher, mit der wir heute noch Leben, auch wenn es seit der letzten Decade Bemühungen gibt, diese abzulösen. Interessanterweise beschreibt 10 Jahre zuvor Alan Kay in (Kay, 1972) ein Konzept für „A Personal Computer for Children of All Ages“ und gibt ihm den Namen „DynaBook“. Sein Konzept beschreibt indirekt die letzten 25 Jahre der Computerentwicklung. Betrachten wir den Übergang zwischen Command-Prompt und Desktop, so erkennen wir zwar dass der Desktop den Command-Prompt abgelöst hat, wir aber bis heute in den gängigen Desktopsystemen einen Command-Prompt öffnen und nutzen können. Ebenfalls geben wir immer noch Texte mit der Tastatur ein. In der neuen Metapher gibt es also Enklaven, in der wir die vorherige Metapher oder Teile dieser wiederfinden.

Mark Weiser stellt in (Weiser, 1991) eine Vision vor, in der der Personal Computer als Gerät verschwunden ist und durch allgegenwärtige Gegenstände mit integriertem Computer ersetzt wird. In (Weiser, 1994) argumentiert er, dass viele vorstellbare Metaphern für den Computer von Morgen nicht wahr werden, weil die Computer in diesen Metaphern als Computer „sichtbar“ sind. Wahr ist, dass heute die meisten Computer nicht mehr „sichtbar“ sind. Schauen wir heute auf unsere Umgebung, so sehen wir mehr Computer denn je, auch wenn sie sich verändert haben. Sie haben unterschiedliche Bildschirme und Eingabegeräte, kommen meistens in einem Stück und sind mobil. Sie heißen Mediacenter, Notebook, Netbook, Tablet oder Smartphone. Bei den einen werden die Bildschirme immer größer, bei den anderen immer kleiner. Auf die großen Bildschirme schauen verschiedene Menschen und mehrere kleine müssen sich die Aufmerksamkeit eines einzigen Menschen teilen. Diese Veränderung und der Arbeitsalltag in den technologischen Kreativbüros hat schon Ende der 90er dazu geführt, dass angefangen wurde, Konzepte für eine visuell nachvollziehbare Vernetzung zwischen großen, kleinen, öffentlichen und privaten Bildschirmen und deren dazugehörigen Rechnern und Eingabegeräten zu entwickeln.

Warum haben sich hauptsächlich die technischen aber nicht die funktionalen Voraussetzungen weiterentwickelt? Die Nutzung der heutigen Computer haben ihren Ursprung in der Nutzung des Personal Computers (PC) und dieser wiederum in starkem Maße in der Schreibmaschine. Im letzten Jahrhundert wurden die Geräte wegen ihrer Immobilität und wegen der stärker

vorgegebenen Arbeitsabläufe meist nur individuell genutzt. In Großraumbüros galt ein Arbeitsplatz, ein Arbeitnehmer und ein PC als Produktionseinheit. Diese Vorgabe galt über Jahrzehnte und spiegelt sich auch in der Denkweise und Entwicklung neuer Technologien. Die heutigen Arbeitsprozesse gestalten sich feingranularer und selbstbestimmter und bedürfen eines höheren Kommunikationsaufwandes, um diese dynamischeren Arbeitsprozesse zu koordinieren. (Winograd und Flores, 1987) schreiben dazu:

> „Many systems designed by computer professionals are intended to facilitate the activity of an individual working alone. Although such tools are useful, they leave out the essential dimension of collective work. In most work environments the coordination of action is of central importance. The conversational dimension permeates every realm of coordinated activity,“

Der Forderung nach mehr Vernetzungs- und Kommunikationsmöglichkeiten von Winograd und Flores kommen viele Arbeiten nach. Die meisten gehen dabei von dem bestehenden System aus und reichern diese mit kommunikationsunterstützenden Funktionen an. Die Folge davon ist, dass jedes Individuum sich mit seinen Partnern häufiger und besser austauschen kann, dies aber immer durch den „Tunnel“ des individuellen Computerarbeitsplatzes. Der „Tunnelblick“ in das System bleibt und das Gesamtsystem bleibt ein System von vielen Einzelsystemen. Besser wäre der Ansatz, nicht das alte System mit neuen Funktionen anzureichern, sondern ein neues System mit alten Funktionen zu bereichern. Neben den Versuchen kooperationsunterstützende Funktionen durch den „Tunnel“ zu verlegen gab es noch andere Versuche, die Interaktion zu verteilen. Bei der gemeinsamen Interaktion mit Hilfe der virtuellen Realität kriechen die Beteiligten sozusagen in den Tunnel hinein und bei Lösungen mit der Funktionalität des „Remote Desktop“ wird der Tunnel nur verschoben.

Bodenständig Ansätze mit denen Gesamtsysteme gebildet werden und in denen es keine Grenzen zwischen den Displays und damit zwischen den Arbeitsplätzen gibt, sind ebenfalls vielzählig. Diese begrenzen sich aber häufig auf nur einen Sachverhalt oder sie bilden Gesamtkonzepte, die nur native Anwendungen unterstützen, die zu Evaluationszwecken implementiert werden. Aus diesem Grund fordern (Wallace und Scott, 2008) einen neuen übergreifenden Blick auf das Interaktionsdesign auf technischer Seite mit gleichzeitiger Betrachtung soziologischer Aspekte. Im selben Jahr fordert auch (Nacenta, 2008) die herkömmliche Sicht auf Displays zu überdenken und die Lücke zwischen Bottom-up und Top-down Ansätzen zu schließen. Gesucht wird also ein Design für nahtlose und soziale Interaktion, welches verschiedene Rechner und deren Displays ohne Grenzen und ohne Eintauchen in eine andere Welt verbindet. Bei diesem Design sollen soziologische Aspekte zur gemeinsamen Interaktion mit einfließen. Es soll technische Grundprinzipien der gemeinsamen Interaktion berücksichtigen und für weitere Prinzipien erweiterbar sein. Ebenso soll es neue native Anwendungen unterstützen und die etablierte Technologie aus dem WIMP-Paradigma dabei nicht ausgrenzen und trotzdem einen „Tunnelblick“ vermeiden und Weitsicht fördern.

1.2 Aufbau der Arbeit

Im Anschluss an die Einleitung wird in Kapitel 2 der Diskussionsbereich und dessen Kriterien zuerst in Abschnitt 2.1 theoretisch und in den Abschnitten 2.2 und 2.3 an praktischen Beispielen eingegrenzt und aufgezeigt.

Eine Analyse von gemeinsamer Arbeit zur gleichen Zeit an gleichem Ort folgt in Kapitel 3. Diese Analyse erfolgt aus drei verschiedenen Perspektiven gefolgt von einem abschließenden Fazit. Die erste Perspektive in Abschnitt 3.1 betrachtet zunächst die Handlungräume von Personen bei Gruppenarbeit aus eher theoretischer Sicht. Aus fiktiver praktischer Sicht wird als zweites in Abschnitt 3.2 ein Szenario der gemeinsamen Präsentationserstellung und deren Kriterien beschrieben und daraus funktionale Anforderungen extrahiert und aggregiert. Die dritte Perspektive in Abschnitt 3.3 beleuchtet aus real praktischer Sicht ausgewählte Anforderungen oder Lösungen aus dem Forschungsbereich um Multi-Display-Environments (MDE). Am Ende der Analyse in Abschnitt 3.4 werden die Ergebnisse aus den drei vorangegangenen Abschnitten kategorisiert zusammen gefasst.

In Kapitel 4 wird das Software-Design für WeFace in sechs Teilen erstellt. Der erste Teile in Abschnitt 4.1 diskutiert wo und wie sich eine gemeinsame Interaktionsschicht in ein verteiltes System einbetten sollte. Die darauf folgenden Teile zwei und drei in den Abschnitten 4.2 und 4.3 beschreiben die Voraussetzungen für ein funktionales und strukturelles Modell als Leitlinie für das Design und darauf aufbauende Implementierungen. Im vierten Teil in Abschnitt 4.4 wird eine priorisierte Softwarearchitektur auf Basis der vorherigen Kapitel und Abschnitte in Form eines Komponentendiagramms entwickelt und beschrieben. Im fünften Teil in Abschnitt 4.5 wird die zuvor beschriebene Softwarearchitektur um nicht priorisierte Anforderungen aus der Analyse erweitert. Dieses erweiterte Komponentendiagramm stellt eine produktionsfähige Softwarearchitektur dar. Im sechsten und letzten Teil in Abschnitt 4.6 wird das vorgeschlagene Software-Design mit anderen Architekturen verglichen, reflektiert und erweitert.

Eine erste Evaluation und Konkretisierung des zuvor vorgestellten Software-Designs erfolgt in Kapitel 5 anhand des realisierten Prototypen zur ausschließlich gemeinsamen Interaktion. Zu Beginn wird zunächst in den Abschnitten 5.1 und 5.2 der Funktionsbereich eingegrenzt und die Auswahl der genutzten Technologien beschrieben. Der Hauptteil in Abschnitt 5.3 beschreibt die Realisierung der einzelnen Komponenten. Am Schluss des Kapitels in Abschnitt 5.4 wird der entstandene Prototyp eingeschätzt und ein weiteres Vorgehen für eine Weiterentwicklung vorgeschlagen.

Den Abschluss dieser Arbeit bildet das Fazit in Kapitel 6. Hier werden die Ergebnisse dieser Arbeit zusammengefasst und eine Einschätzung zu Nutzen und Integration in die Interfacetechnologie abgegeben.

2 Einführung in das Thema

In dieser Arbeit werden verschiedene Forschungsbereiche überschneidend bearbeitet. Das daraus folgende Ergebnis ist im Forschungsbereich der Human-Computer-Interaction (HCI), des Ubiquitous Computing (UbiComp) und Computer Supported Cooperative Work (CSCW) anzusiedeln. CSCW ist ein großer, viele Fächer beinhaltender Forschungsbereich, deshalb wird im ersten Abschnitt der Bereich der Arbeit erst einmal eingegrenzt. In dem folgenden Abschnitt werden dann ausgewählte Arbeiten aus dem eingegrenzten Bereich beschrieben, um erste einführende Überlegungen zu formulieren und die Diskussionsgrundlage plastischer zu machen.

2.1 Abgrenzung CSCW und Collaborative Workspace

CSCW bezeichnet jegliche Zusammenarbeit von Menschen, die von Computertechnologie unterstützt wird. Da bei CSCW die Mensch-Mensch-Interaktion im Vordergrund steht und nicht die Mensch-Computer-Interaktion, ist HCI nur ein Bestandteil von CSCW. In theoretisch soziologischen Betrachtungen existiert CSCW auch ohne HCI. Da die Interaktion zwischen Menschen im Vordergrund stehen soll, ist es ebenfalls nötig, dass die scheinbare Bedeutung des Computers unsichtbar wird. Wenn Computer nahezu „unsichtbar“ zwischen den Menschen und deren Zusammenarbeit stehen, es sich somit um die Schnittmenge der drei Bereiche handelt, dann sprechen wir von einer „collaborative ubiquitous computing application“ wie in Abbildung 2.1 zu sehen. Bei einem Collaborative Workspace ist genau diese Schnittmenge betroffen.

Um CSCW-Systeme zu klassifizieren, betrachten wir die zwei Dimensionen Raum und Zeit. Für die Zeit unterscheiden wir synchrone und asynchrone Arbeitsabläufe. Bei synchronen Arbeitsabläufen findet die Bearbeitung gleichzeitig statt, während asynchrone Arbeitsabläufe zeitlich versetzt stattfinden. Für den Raum unterscheiden wir zwischen zentral und dezentral. Zentrales Arbeiten bedeutet das Arbeiten am gleichen Ort und dezentral Arbeiten an verschiedenen Orten. Eine Klassifizierung von Groupware und deren Ausprägungen zeigt Abbildung 2.2.

Bei der Arbeit in einem Collaborative Workspace befinden sich normalerweise alle beteiligten Personen zur gleichen Zeit in einem Raum. Die Punkte Gruppenentscheidung und Sitzungsmoderation sind zwar ein Gruppenprozess aber eher ein Spezialfall einer Gruppenproduktivität. Bei einem Mehrautorensystem handelt es sich offensichtlich um eine Gruppenproduktion und es wäre interessant und sinnvoll, diesen von der globalen Dimension in die lokale Dimension zu transferieren. Zusammenfassend kann gesagt werden, dass die Verknüpfung der zwei

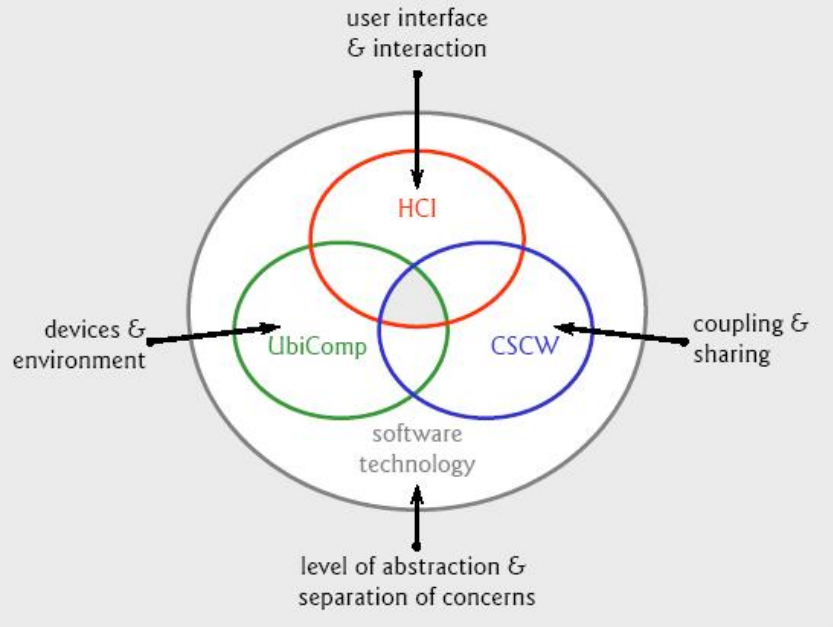

Abbildung 2.1: Drei sich überlagernde Forschungsgebiete bilden eine Schnittmenge. (Aus (Tandler, 2004b))

Computergestützte Zusammenarbeit	Zur gleichen Zeit (Synchron)	Zu verschiedenen Zeiten (Asynchron)
Am gleichen Ort (zentral, lokal)	Sitzungsmoderation Präsentationen Gruppenentscheidungsunterstützung	Terminkalendermanagement Projektmanagement
An verschiedenen Orten (dezentral, global)	Audio- und Videokonferenzen Screen-Sharing Mehrautorensysteme Remote-Support	Email Voicemail Electronic Conferencing Elektronische Bulletin Boards Shared Information Systems Workflow-Systeme

Abbildung 2.2: Eine Klassifizierung von Groupware. (Aus (Nastansky u. a., 2002))

Dimensionen lokal und synchron die höchste Komplexität in der Verwaltung und die höchste Interaktionsrate der Individuen beinhaltet.

2.2 Existierende Co-located Workspaces

Im folgenden werden exemplarisch die zwei grundlegenden Co-located Workspaces „iRoom“ aus Stanford und der Nachfolger des „i-Land“ Environments „Roomware - The Second Generation“ beschrieben. Diese werden schließlich im Hinblick auf ein später zu entwerfendes Szenario kritisch beleuchtet.

2.2.1 Roomware - 2G

Roomware - The Second Generation (Streitz u. a., 2002) von der IPSI Gruppe der Fraunhofer Gesellschaft aus Darmstadt ist der Nachfolger des i-Land Projekts (Streitz u. a., 1999). Es beinhaltet eine Reihe von Ein- und Ausgabegeräten, die sich alle im selben Raum befinden, alle miteinander vernetzt sind und auf den gleichen Datenbestand zugreifen können, wie in Abbildung 2.3 zu sehen.

Abbildung 2.3: Die zweite Generation von Roomware. (Aus (Streitz u. a., 2002))

Das Framework Beach (siehe Abbildung 2.4) ist für die Displaysteuerung sowie für die Objektdarstellung und deren jeweiliger Transformation zuständig. Für die synchrone Nutzung der Datenobjekte ist das Framework COAST (Schümmer u. a., 2000) zuständig. Die Gruppe der Ein- und Ausgabegeräte besteht aus dem ViewPort, dem CommChair, dem ConnectTable, der DynaWall und dem InteracTable.

Ein PDA stellt für den Anwender den ViewPort dar. Der CommChair ist ein Computer in Form eines Stuhls mit einem schwenkbaren Display. Der ConnecTable ist ein mit anderen ConnecTable koppelbarer Stehpult mit Display. Die DynaWall ist ein großer Wandbildschirm. Als letztes und wichtigstes Element ist der InteracTable zu nennen: ein Tisch mit eingelassenem Touchscreen. Alle Geräte bis auf die DynaWall, lassen sich im Raum bewegen und neu platzieren.

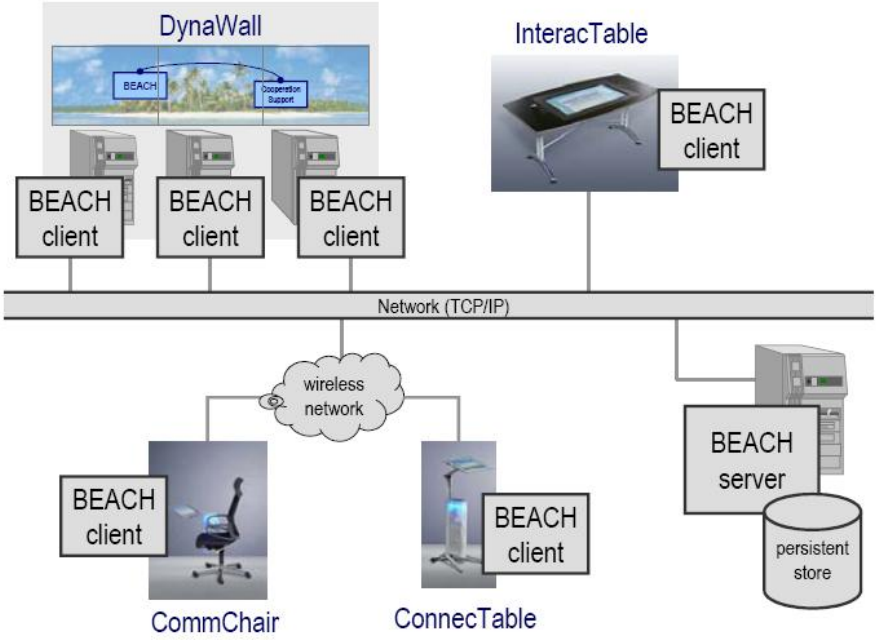

Abbildung 2.4: „Beach Clients“ werden über den „Beach Server“ synchronisiert. (Aus (Tandler, 2004a))

Unter Nutzung der Frameworks wurde die Beispielanwendung MagNet entwickelt. MagNet dient dazu, virtuelle Karteikarten hierarchisch zu sortieren. Die zwei Erweiterungen Palmbeach und BeachMap ermöglichen das autonome Erstellen von Karten mit PDAs und deren Koordination.

Roomware stellt eine Grundlage zur Entwicklung von parallel nutzbaren verteilten Anwendungen dar. Es wird keine dauerhafte Datenspeicherung unterstützt, lediglich das Datenspeichern während der Laufzeit. Das einzige Sicherheitsfeature besteht darin, die Benutzer zu identifizieren.

2.2.2 iRoom

iRoom (Johanson u. a., 2002a) von der Forschungsgruppe HCI der Stanford University besteht aus einer Reihe von Ein- und Ausgabegeräten, die sich alle im selben Raum befinden, alle miteinander vernetzt sind und auf den gleichen Datenbestand zugreifen.

Abbildung 2.5: Der iRoom der Universität Stanford. (Aus (Winograd, 2003))

Das eigene Betriebssystem iROS[1] (siehe Abbildung 2.6)unterstützt das gemeinsame Benutzen von Displays, die orts- und typenunabhängige Datenspeicherung und die Koordination von Anwendungen. Die Koordination zwischen den Anwendungen wird durch einen sogenannten EventHeap realisiert. Die Datenspeicherung ist durch einen DataHeap implementiert, welcher durch ein Transformationsframework Typenunabhängigkeit realisiert und die Daten durch Indizes auf die typenunabhängigen Metadaten zugreifbar macht. Der EventHeap ist eine Erweiterung von Tuplespaces. Tuplespace ist ein globaler Speicher, in dem Anwendungen Tuples zur Koordination mit anderen Anwendungen abspeichern können. Tuplespace ist ein ähnliches Konzept wie das Blackboard aus dem Bereich der Künstlichen Intelligenz. Ein Tuple ist eine Ansammlung von strukturierten Daten (type-value field), die von jeder Anwendung in den Tuplespace geschrieben, gelesen oder gelöscht werden können. Der iCrafter hilft bei der Instanziierung einer Benutzerschnittstelle zur verteilten Anwendung für jedes beliebige Eingabemedium. Der EventHeap ist die einzige iROS Komponente, die eine Anwendung benutzen muss, da diese die Kommunikations- und Koordinationsebene für Anwendungen innerhalb eines interaktiven Workspace ist.

Die Ein- und Ausgabegeräte des iRoom bestehen aus einem hochauflösenden Display, drei Smartboards und einem TableTop mit Touchfunktion. Weitere mobile Geräte wie Notebooks werden über Funk eingebunden. (siehe Abbildung 2.5)

Als Beispielanwendung wurde ein Präsentationsprogramm entwickelt, welches die Darstellung von Objekten auf mehreren Displays koordiniert. iRoom selbst hat keine implementierten Sicherheitsfeatures und alle Benutzer haben gleiche Rechte.

[1] interactive Room Operating System

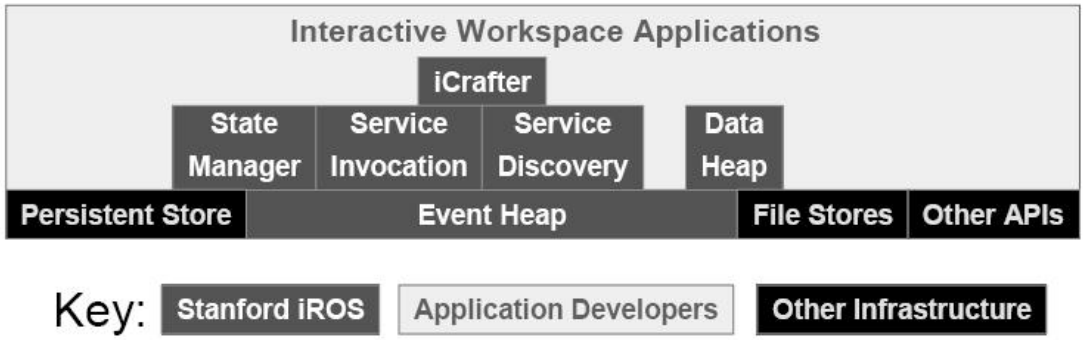

Abbildung 2.6: Struktur der iROS Komponenten. (Aus (Johanson, 2003))

Teamspace

Teamspace (Shih und Winograd, 2004) ist ein Testszenario von iRoom das unter realen Bedingungen durchgeführt wurde. Teamspace besteht aus einem großen Display mit Rechner, welcher als kooperative Arbeitsfläche genutzt werden kann. Im Teamspace können sich die Nutzer mit ihrem Laptop über Ethernet oder Wlan einloggen und sich dann Dokumente gemeinsam auf dem gemeinsamen Display anzeigen lassen und über eine Zusatzsoftware Daten austauschen. Den technischen Hintergrund liefert, genauso wie beim iRoom, das Betriebssystem für Ubiquitous Computing „iROS“. TeamSpace ist in diesem Zusammenhang erwähnenswert, da die Academic Computing Department of Stanford University Libraries dieses „Softwarepaket“ mit Hardware in ihrer Lounge mit 24-Stunden-Betrieb installiert haben. Das ist ein Beweis dafür, dass iROS über ein Prototypenstadium hinausgewachsen ist.

PointRight

Mit PointRight (Johanson u. a., 2002b) als Teil von IRoom können virtuelle Bildschirmtopologien festgelegt werden (siehe Abbildung 2.7). Auf dieser Topologie kann der Anwender über einen internen Redirect mit seinen Eingabeinstrumenten die gesamte Topologie bedienen. PointRight soll das Wandeln über die Bildschirme möglichst realitätsnah und intuitiv benutzbar machen.

UIC in LiveSpaces

Der Universal Interaction Controller (UIC) (Slay und Thomas, 2006) baut auf iROS auf und ist Teil des LiveSpaces Projekts der University of South Australia. Dieses Projekt hat sich zur Aufgabe gesetzt herauszufinden, wie (Konferenz)Räume mit Displaytechnologie, Informationen, Sprache, Sensoren und Interaktionsgeräten angereichert werden können. Dies soll intelligente Interaktion in der Zukunft sinnvoll gestalten.

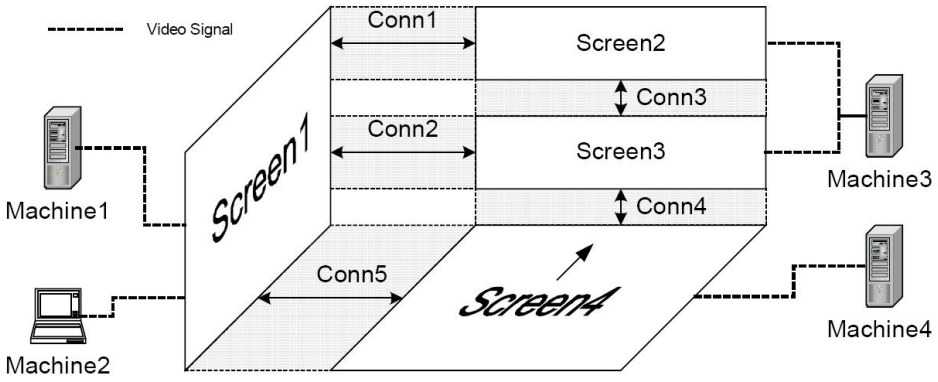

Abbildung 2.7: Screens sind in der virtuellen Topologie ähnlich der realen Welt angeordnet und die Verbindungen zwischen den Bildschirmen für den Übertritt von Mauszeigern sind zwischen den Screens definiert. (Aus (Johanson u. a., 2001))

UIC besteht aus drei Teilen, dem Interaction-Manager, dem Clipboard-Manager und dem Ukey. Der Interaction-Manager organisiert die Verbindung zwischen Displays, Anwendungen und Eingabegeräten. Der Clipboard-Manager stellt die Technik zur Verfügung, um Files oder Arrays temporär zu speichern. Es ist damit für Geräte, Anwendungen und Benutzer möglich, Daten an einem Ort zu „kopieren“ und an einem anderen wieder „einzufügen“. Der Ukey ist das physikalische Interaktionsmoment von UIC. Es ist ein kleines Gerät (PDA) mit Eingabemöglichkeit und Positionserkennung im Raum. Mit dem Wissen über die eigene Position und die Positionen der Displays im Raum ist eine Zuordnung zwischen Eingabe- und Ausgabegeräten möglich. Dies macht es möglich, die wirkliche physikalische Verteilung der Geräte zueinander zu bringen.

2.3 Aspekte von Co-Located Workspaces

Die Aspekte der nun vorgestellten Collaborative Workspaces beziehen sich auf das in Kapitel 3 vorgestellte Szenario und deren Anwendungsfälle. Die zwei ersten Projekte mit der größten Überschneidungsmenge zu der Intention des Szenarios sind der iRoom mit seinen Unterbereichen PointRight und Teamspace sowie die Sammlung von Roomware - The Second Generation. Beide Projekte beinhalten ein ähnliches Szenario, welches allgemeine Produktionsprozesse der Gruppenarbeit mit elektronischen Medien und Geräten in einem Raum zur gleichen Zeit unterstützen sollen. Des weiteren werden noch die Aspekte eines „Multispace“ und „Multiple Computer Interfaces“ betrachtet.

Roomware Komponenten unterstützen die Anwendungsfälle der Navigation innerhalb einer Geräteklasse und teilweise zwischen den Geräteklassen. Die Navigation der virtuellen Objekte zwischen den Geräteklassen wird durch zwei verschiedene Mechanismen bereit gestellt. Der erste wird mit real existierenden Stellvertretergegenständen[2] ermöglicht. Dieser muss jeweils

[2]Dieser Mechanismus wird Passage genannt. Die realen Gegenstände Passengers und die Schnittstellen

real am jeweiligen Gerät angebracht werden und der virtuelle Teil des realen Gegenstandes muss mit dem virtuellen Objekt in Verbindung gebracht werden. Dies gibt den virtuellen Objekten eine gewisse Greifbarkeit. Der zweite Mechanismus geschieht über eine explizite Handlung in Form eines lokalen Transfers zwischen privatem und öffentlichen Bereich für den globalen Transfer zwischen den Geräteklassen. Beide Transferarten setzen eine spezielle Handlung voraus und unterbrechen den Arbeitsfluss. Der Transfer innerhalb der Geräteklasse setzt immer eine bestimmte Topologie der Geräte voraus. Beim Arbeiten zweier Personen mit zwei ConnecTables ist es bei richtiger Orientierung der ConnecTables zueinander möglich, ein Objekt ohne weitere Maßnahmen über deren angrenzende Kante, dem Partner zu Verfügung zu stellen. Dies ermöglicht zwar ein ungehindertes Miteinanderarbeiten, ist aber in diesem Fall beschränkt auf zwei Teilnehmer und die Anordnung und Orientierung der Geräte und Teilnehmer ist in jedem Fall vorgeschrieben. Die Navigation unterscheidet sich somit zwischen „innerhalb einer Geräteklasse“ und „zwischen den Geräteklassen“, was einen Gedankenwechsel beim Nutzer des System erfordert.

iRoom unterstützt konzeptionell das übergangslose Verschieben von Objekten, wie laufende Anwendungen und Dateien von einem Display zum anderen innerhalb des Multi Display Environments (MDE). Die topologische Anordnung kann vor dem Start des Systems über ein Skript festgelegt werden und ist nach dem Start statisch. Sieht man von dem leichten Ruckeln und dem leicht überlappenden Erscheinungsbild (zeitlich sowie optisch) ab, welches einen technischen Vorgang im Hintergrund verrät, könnte hier von einem echten nahtlosen Übergang der Objekte innerhalb des gesamten MDEs gesprochen werden. Dies gilt nicht nur für die Objekte wie Anwendungen und Dateien sondern auch für die aktiven Eingabeobjekte. Basiert die Eingabe bei Roomware auf einer Art Touchtechnologie, die semantisch an den jeweiligen Monitor gebunden ist, so ist im iRoom die Nutzung von konventionellen Eingabegeräten wie Maus und Tastatur gewollt und wird unterstützt. Es ist somit möglich, auf dem öffentlichen Bildschirm zu arbeiten, ohne sich zu ihm zu begeben.

Leider wird die gleichzeitige Nutzung der konventionellen Eingabegeräte auf einem Display nicht unterstützt, so dass die Nutzer sich bei der Navigation gegenseitig blockieren. In einer Veröffentlichung zu Pointright (Johanson u. a., 2002b) wird die Forderung aufgestellt, dass es nötig und sinnvoll ist, eine Mehrbenutzerfähigkeit innerhalb eines Displays einzuführen, da sich bei der Nutzung des realexistierenden Teamspace (Shih und Winograd, 2004) darüber beklagt wurde, dass sie sich gegenseitig auf dem einzigen gemeinsamen Bildschirm blockieren würden. Um diesem Problem zu begegnen wurde der Teamspace II (Hebert und Chen, 2005) um einen weiteren öffentlichen Bildschirm erweitert.

Der Multispace (Everitt u. a., 2006) aus dem Mitsubishi Elektronic Research Lab (MERL) unterscheidet ebenfalls zwischen Geräteklassen. Hier allerdings sind die als öffentlich zu bezeichnenden Geräte wie Table und Wallscreen zu einer festen Einheit zusammengefasst und bieten den fast nahtlosen Übergang der Objekte zwischen den Displays über die jeweilige Bildschirmkante. Die Einschränkung „fast“ ergibt sich aus einem nicht fließenden visuellen Übergang der Objekte. Diese erscheinen einfach auf dem Bildschirm. Das heisst die Benutzer müssen

Bridges.

sich über den Vorgang des Verschiebens bewusst sein, um die Handlung nachvollziehen zu können. Bei gleichzeitigem Verschieben mehrerer Objekte kann eine klare Zuordnung nicht garantiert werden. Ebenfalls ergibt sich das Erkennen der Handlung nicht aus dem sichtbaren Geschehen. Der Übergang zwischen den privaten und öffentlichen Displays wird über explizite Übergangsflächen und auszuwählende Menüpunkte bereitgestellt. Dies ermöglicht einen ungehinderten Übergang zwischen allen Geräten, bedarf aber einer speziellen Handlung für den Übergang zwischen privatem und öffentlichem Bereich. Die physikalische Anordnung der öffentlichen Displays ist für eine intuitive Bedienung vorgegeben.

Im Interaction Laboratory von den Sony Computer Science Laboratories, Inc. arbeitete Jun Rekimoto an „Mutiple Computer Interfaces 'Beyond the Desktop'"(Rekimoto, 2000). Hier ist es möglich, mit einem „Stift" Datenobjekte aufzunehmen und wieder abzulegen. Das funktioniert auch über die Grenzen der Displays hinweg, so dass es möglich ist, seine Daten durch einfaches „Pick and Drop"(Rekimoto, 1997) anderen zur Verfügung zu stellen. Da hier mit einer Art „Cut and Paste" gearbeitet wird und nicht mit Verschieben, überbrücken die Benutzer die Topologiekomplexität und es gibt keine Unterbrechung durch die Überwindung dieser. In den „Augumented Surfaces"(Rekimoto und Saitoh, 1999) ist es mit „Hyperdragging" möglich, den Transfer vom privaten Bereich (einem automatisch in das System integriertes Laptop) in den öffentlichen Bereich zu transferieren und umgekehrt. Mit den Funktionalitäten des „Hyperdragging" ist es auch möglich, mit mehreren Personen gleichzeitig im öffentlichen Bereich zu arbeiten. Jun Rekimoto zeigt ein Szenario, in dem gleichzeitiges nahtloses Arbeiten in einem MDE möglich ist. In dieser Arbeit wird in der User-Interface-Gestaltung strikt zwischen öffentlichem und privatem Raum unterschieden. Das MDE an sich ist statisch vorgegeben und die privaten Mittel sind als Datenträger zu interpretieren. Der gleichberechtigte Transfer ist nur mit einem speziellen Tool, dem „Pen" und dessen Eigenschaften möglich.

Ein Teil der Arbeiten von (Gottwald, 2007), (Román u. a., 2002) und (Shafer u. a., 1998)[3] beinhaltet die Möglichkeit, in einem Multi-Display-Environment Objekte von einem Display auf ein anderes zu transferieren. Dies geschieht über ein Menü oder einen menüähnlichen Mechanismus, der explizit angestoßen werden muß, oder zumindest eine explizite Handlung voraussetzt. Gleichzeitig werden bei diesem Vorgang neue Instanzen der Objekte erzeugt und müssen verwaltet werden.

[3] Beim Easy Living Projekt werden hauptsächlich Desktop Ein- und Ausgaben von einem Display zum nächsten transferiert. Dies passiert ohne weiteres Zutun des Benutzers, beinhaltet aber nicht den gemeinsamen Zugang zu den Objekten. Kontrolle hat immer nur der Besitzer. Wird das Objekt transferiert, ändert sich der Besitzer und damit auch die Kontrolle.

3 Analyse

In diesem Kapitel wird der Sachverhalt für die Entwicklung einer Software für reibungsloses gleichzeitiges gemeinsames Arbeiten in einem Raum mit Hilfe von mehreren Rechnern, Monitoren und Eingabegeräten aus drei Perspektiven betrachtet. Bei dieser Betrachtung werden neben den technischen auch die menschlichen und räumlichen Ressourcen mit einbezogen.

In Abschnitt 3.1 werden zunächst die Handlungräume von Personen bei Gruppenarbeit aus verschiedenen problemspezifischen Perspektiven betrachtet und miteinander in Beziehung gesetzt. Das Ergebnis soll zur Eingrenzung und Verdeutlichung von Teilen des Handlungsraumes dienen. Im zweiten Abschnitt 3.2 wird ein Szenario der gemeinsamen Präsentationserstellung beschrieben, das Arbeitsprozesse beinhaltet, die häufig in Szenarien der gleichzeitigen ortsgebundenen Gruppenarbeit vorkommen. Dieses Szenario dient zur Erarbeitung von abstrakten Anwendungsfällen und soll später als Grundlage zur Reflexion dienen. Im dritten Abschnitt 3.3 werden ausgewählte Anforderungen oder Lösungen aus dem Forschungsbereich um Multi-Display-Environments (MDE) in Bezug auf das entworfene Szenario beschrieben. Dabei werden bisherige Bemühungen und Resultate mit in die Entwicklung einbezogen und die Betrachtung weiter konkretisiert.

Im letzten Abschnitt 3.4 werden die Ergebnisse aus den drei vorangegangenen Abschnitten zusammen gefasst. Die daraus resultierenden Anforderungen werden in priorisierte Kategorien zusammengefasst, um ein schrittweises Vorgehen bei der Lösungssuche zu unterstützen.

3.1 Handlungsräume von Gruppenarbeit

Bei der Betrachtung menschlichen Handelns und menschlicher Arbeit gehen (Winograd und Flores, 1987) aus Perspektive der Linguistik und Arbeitsorganisation vor. Der Ausgangspunkt für ihre Analyse ist Sprache. Bei der Beleuchtung der „Repräsentation“ „geistiger Modelle“ bezüglich Manipulation durch Denken und Vermittlung durch Sprache, begründen[1] sie die unterschätzte Bedeutung der Vermittlung und damit den Akt des Sprechens[2].

[1] Begründet wird dieser Umstand hauptsächlich mit der Interpretation von Aussagen von Heidegger und Maturana. Diese stellen Objektivität in Frage, in dem sie die Betrachtung des Kontextes für die Bedeutung von etwas als notwendig erachten.

[2] Die Begrifflichkeit „Akt des Sprechens“ soll hier explizit nicht gleich gesetzt werden mit dem in diesem Zusammenhang benutzten Begriff „Sprechakt“. Es soll aber ein Hinweis zum Konstrukt des „Sprechaktes“ sein.

Verbinden wir die Begrifflichkeit „Sprechen“ mit „Kommunikation“ sowie die Bedeutungen der Ausdrücke „Manipulation des geistigen Modells durch Denken“ mit „Veränderung eines Arbeitsgegenstandes“, so verbindet diese Abstraktion die Aussagen von (Winograd und Flores, 1987) mit der Unterscheidung von (Oesterreich und Resch, 1985) zwischen „kommunikativen Akten“ und „materiellen Handlungen“ innerhalb von Gruppenarbeit. Zu „materiellen Handlungen“ und „kommunikativen Akten“ schreibt (Weber, 1997):

> „Das Ziel einer materiellen Arbeitshandlung besteht in einem durch die Handlungsausführung hergestellten (Teil-) Produkt, das durch veränderte Bestandteile und neue Handlungsmöglichkeiten im Vergleich zur Ausgangssituation der Handlung gekennzeichnet ist. Zweck und Resultat eines kommunikativen Aktes bestehen nicht in einer materiellen Veränderung eines Arbeitsgegenstandes oder - abstrakter ausgedrückt - in der materiellen Veränderung von Handlungsmöglichkeiten. Stattdessen zielt die arbeitsbezogene Kommunikation darauf ab, Aktionsprogramme, deren spätere Umsetzung materielle Produktveränderungen zur Folge haben wird, aufeinander abzustimmen bzw. anzugleichen.“

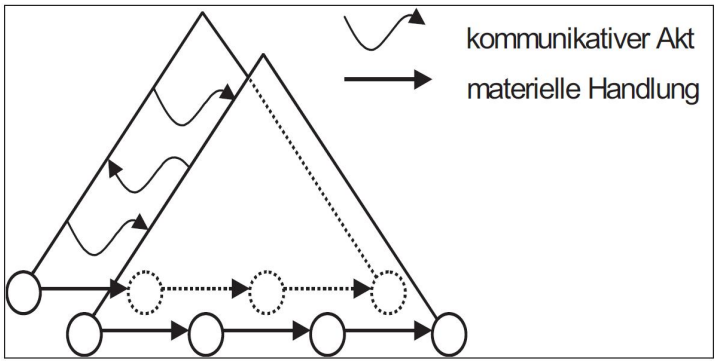

Abbildung 3.1: Handlungsregulation bei zwei Kooperationspartnern. Aus (Oesterreich und Resch, 1985)

Die Unterscheidung in „produzierendes“ Handeln („materielle Handlung“) und „sprachliches“ Handeln („kommunikativer Akt“) wird in Abbildung 3.1 gezeigt. Es zeigt sich sprachlich bei (Weber, 1997) sowie bildlich bei (Oesterreich und Resch, 1985), dass das jeweilige Handeln durch das andere unterbrochen und später unter anderen Bedingungen fortgesetzt werden kann. Gebrauchen wir den Begriff „Reflexion“ anstelle von „Kommunikation“, erweitern wir den Begriff der Kommunikation zwischen einzelnen um die Kommunikation mit sich selbst. Zur Veranschaulichung des Zusammenspiels von „Reflexion“ und „Produktion“ stellen wir dies in einem Aktivitätsdiagramm, wie in Abbildung 3.2 zu sehen, dar.

In den „Design Guidelines for Co-located Tabletop“ von (Scott u. a., 2003) wird für flüssige nachvollziehbare Übergänge bei gemeinsamer Arbeit argumentiert. Die verwendeten Begriffe „personal“ und „group“ in dem dort dargestellten Kontext werden wir nun mit den Begriffen „Produktion“ (Manipulation) und „Reflexion“ (Kommunikation) verbinden. Der „Personal“-Bereich ist, wie in Abbildung 3.3 dargestellt, der individuell erreichbare und sozial okkupierte Raum, in dem eine Person hauptsächlich individuell handelt. Deshalb übersetzen wir diesen Begriff mit

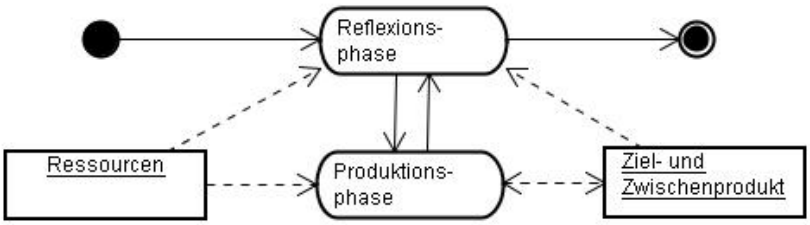

Abbildung 3.2: Einfache Darstellung des Produktionsprozesses einer Gruppenarbeit. Die Personen können einzeln oder gemeinsam in Transition gehen.

„personengebunden“. Hierbei darf nicht verschwiegen werden, dass im personengebundenen Raum eine Kommunikation zwischen den Individuen über die Wahrnehmbarkeit der Aktivitäten von Personen aus der Gruppe sowie über nicht elektronische Kommunikationskanäle statt finden kann. Diese beiläufigen Kommunikationmöglichkeiten unterstützen während des Handelns im personengebundenen Raum vorbereitend das Handeln im gemeinsamen Raum, der in Abbildung 3.3 mit „Group“ bezeichnet wird. Ebenfalls wird natürlich auch das Handeln in den anderen personengebundenen Räumen unterstützt. Da eine manipulierende Handlung eine gute Erreichbarkeit[3] voraussetzt, findet diese dominant im personengebundenen Raum statt. Weil der dargestellte „Group“-Bereich sich mit dem „Personal“-Bereich überschneidet, bilden wir für den Gruppenbereich die Differenzmenge von „Group“- und „Personal“-Bereich. Dieses Vorgehen begründet sich mit der sozialen Okkupation des personengebundenen Bereichs durch das Individuum, welches ein aktives Handeln der Gruppe in diesem Bereich einschränkt. Im nun entstandenen Gruppenbereich findet wegen der schlechten Erreichbarkeit wenig manipulierendes Handeln statt. Deshalb ist der Anteil der kommunikativen Handlungen im Gruppenbereich dominant. Aufgrund dieser Dominanzen bestimmen wir den Gruppenbereich als kommunikativen und den personenbezogenen Bereich als manipulierenden.

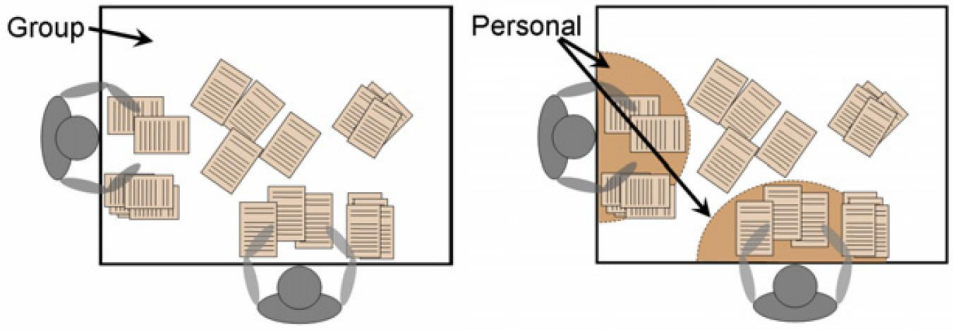

Abbildung 3.3: Aufteilung von Arbeitsbereichen auf einer gemeinsamen Ebene bei gemeinsamer Arbeit in „Personal“- und „Group“-Bereiche. (Scott, 2005)

Für abstrakte und auf Arbeitsflächen bezogene Handlungsräume folgern wir, dass sich diese in zwei Mengen von Handlungen aufteilen lassen. Die Menge der individuellen Handlungen

[3] Die Abgrenzung des „Personal“-Bereiches begründet (Scott, 2005) mit den Ergebnissen einer Studie über Aktivitätsintensivität und mit der Erreichbarkeit von Objekten.

beinhaltet die Manipulation von etwas und die Menge der kollektiven Handlungen beinhaltet die Kommunikation von etwas. Wie vorher beschrieben und in Abbildung 3.4 zu sehen überschneiden sich die beiden Mengen. In dieser Arbeit wird nicht auf die Art und Größe der Überschneidung eingegangen, weil es sich um eine rein deskriptive Analyse dieses Sachverhaltes handelt. Die Unterscheidung in „individuell“ und „kollektiv“ bleibt, auch wenn sie sich an vielen Stellen unscharf darstellt.

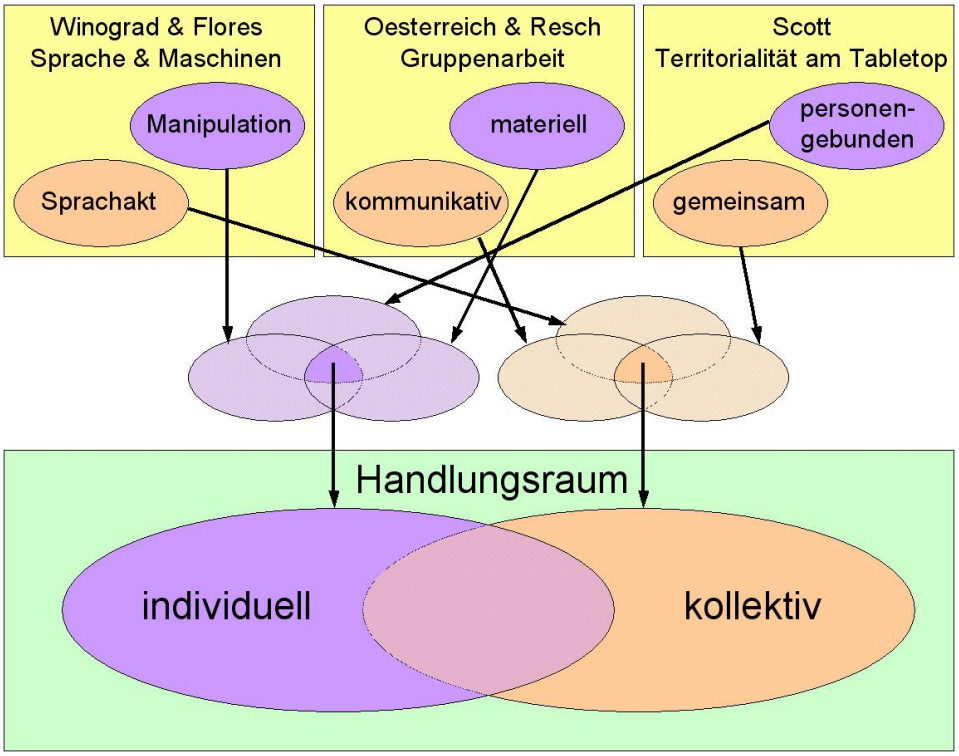

Abbildung 3.4: Bei der Analyse verschiedener Perspektiven auf Handlungsräume, teilt sich der Handlungsraum gemeinsamer Interaktion in individuell und kollektiv auf.

Weiterhin gilt, dass zwischen individuellen und kollektiven Handlungen ohne Beschränkung gewechselt werden kann, wie Abbildung 3.2 stellvertretend zeigt. Wird der Status des Bereichs außerhalb des individuellen Bereichs verändert, also manipuliert, wird dies trotzdem als kollektive Handlung bezeichnet, auch wenn es sich nicht explizit um einen kommunikativen Akt handelt, sondern um einen manipulativen. Die indirekte Gleichsetzung von den zu Anfang erwähnten Sprechakten mit dem zum Ende erwähnten kollektiven Handeln soll hier noch einmal explizit erwähnt werden, weil dies die ambitionierte Forderung von (Winograd und Flores, 1987) unterstützt, Sprechakte mit in die Ausbildung von Prozessen zu integrieren.

3.2 Szenario

Grundlage des Szenario ist eine Gruppe von zwei bis sechs Personen,[4] die eine gemeinsame Präsentation aus ihren schon vorhandenen Daten erstellen. Eine Präsentation kann aus diversen Elementen bestehen und sie vereint in der Praxis häufig Arbeiten verschiedener Individuen.[5]

Abbildung 3.5: Beispielszenario zur Veranschaulichung möglicher Gegebenheiten.

Im folgenden werden die Bedingungen dargestellt, die in diesem Szenario gelten sollen. Die Beschreibung der Handlungen der Personen im Szenario und die daraus folgenden Anwendungsfälle schließen den Abschnitt ab.

3.2.1 Bedingungen

Die hier formulierten Bedingungen sollen eine möglichst reibungslose, produktive sowie kommunikative Informationserstellung und deren Austausch unterstützen. Diese Bedingungen sollen nicht als notwendig wohl aber als hinreichend begriffen werden.

[4] In (Johansen u. a., 1991) wird eine sinnvolle Gruppengröße für ein kollaboratives Handeln mit zwei bis fünfzehn Personen angenommen. Dieser Erkenntnis folgt auch (Johanson, 2003) in seiner Arbeit, welche ein ähnliches Szenario wie das hier angenommene beinhaltet. Da die Realisierung der Ergebnisse dieser Arbeit im Ambient-Labor der HAW-Hamburg statt finden soll und die Räumlichkeiten nicht für so viele Menschen ausgelegt sind, beschränkt sich das Szenario hier auf eine Teilnehmerzahl von bis zu sechs Personen. Es soll aber davon ausgegangen werden, dass sich das Szenario ohne weiteres auf eine Anzahl bis zu fünfzehn Personen erweitern lässt.

[5] Dies können inhaltliches Layout, Grafiklayout, Kapitel, einzelne Texte und Bilder sein.

„The word processor exists as a collection of hardware or programs only when it breaks down. In its use, one is concerned with the actions of creating and modifying documents and producing physical presentations of them on a screen or printed page. The relevant domain is not a computational one, ...“ (Winograd und Flores, 1987)

„Als Ansammlung von Hardware und Software existiert das Textverarbeitungssytem nur, wenn es zusammenbricht. Solange das Gerät funktioniert, richten sich die Gedanken auf Erstellen und Modifizieren von Schriftstücken und deren physischer Darstellung auf Bildschirm oder bedrucktem Papier. Die Aufmerksamkeit richtet sich überhaupt nicht auf den Computer als solchen, sondern auf den Zusammenhang, ...“ (übersetzt aus (Winograd und Flores, 1989))

Deuten wir das Textverarbeitungssystem als Anwendung oder Betriebssystem und deuten wir das Zusammenbrechen als negativ und das nicht Zusammenbrechen als positiv, so ist wohl die beste Anwendung die, die wir nicht bemerken weil sie uns das tun lässt was wir eigentlich tun wollen.

Die Benutzer des Systems sollen sich mit möglichst wenig technischen Details und Beschränkungen während sowie vor und nach dem Arbeitsprozess auseinander setzen müssen. Hierzu seien eine möglichst kurze Einarbeitungszeit[6] in das System und ein intuitives nahtloses Interagieren[7] während der Arbeit mit dem System genannt. Ebenso soll es möglichst wenig administrativen Aufwand erfordern, wenn sich das Setting im Raum personell sowie strukturell ändert.[8]

Um die Kooperationsbereitschaft und -möglichkeiten der teilnehmenden Personen zu fördern, soll möglichst viel des Geschehens, also auch die verwendeten Daten sowie der Entwicklungsprozess an sich, transparent und gleichberechtigt für alle gehalten werden.[9] Das schafft Vertrauen, zwingt aber auch die Teilnehmer zu einem gegenseitigen vertrauensvollen Umgang. Das Vertrauen in das System und in die zusammenarbeitende Gruppe soll durch den Schutz des privaten Bereichs gewährleistet sein.[10] Das heißt, dass die nicht zum Kooperations- und Arbeitsprozess gehörenden Ressourcen geschützt sein müssen.

Neben der Notwendigkeit des gemeinsamen Handelns besteht auch die Notwendigkeit des individuellen Arbeitens. Beides soll parallel mit allen möglichen und sich verändernden Konstellationen der Gruppe möglich sein.

[6] Damit ist ein Handbuch- und einarbeitungloses Arbeiten gemeint.

[7] Seamless Interaction nach (Ishii u. a., 1994).

[8] Zero-Administration wird im Zusammenhang mit der Integration von Elementen in interactive Room (iRoom) der Stanford University genannt. Ein Vergleich zwischen inkrementeller Integration und Zero-Administaration findet sich unter (Szybalski, 2004) und ein Beispiel für inkrementelle Integration unter (Ballagas u. a., 2004).

[9] In der Produktionsphase sowie in der Reflexionsphase können die Ressourcen aller benötigt werden. Um den beteiligten Personen ein gemeinsames Arbeitsgefühl zu vermitteln und eine einfache permanente Korrektur der Prozesse zu ermöglichen, soll der ungefähre Stand der Entwicklungen allen Personen zu jeder Zeit präsent sein.

[10] Wir unterscheiden hier explizit zwischen privatem Raum bzw. Bereich und privatem Arbeitsbereich. Privater Raum/Bereich meint die nicht veröffentlichen Ressourcen des mitgebrachtem Ressourcenbestandes und deren Zugangsstrukturen jeder Person. Der private Arbeitsbereich nach (Scott, 2005) meint den Bereich während des Arbeitsprozesses, in dem die jeweilige Person aber keine andere die Daten manipuliert.

3.2.2 Handlung im Szenario

Die oben genannten zwei bis sechs Personen wollen eine gemeinsame Präsentation aus schon bestehenden Arbeitsergebnissen und Ressourcen zusammenstellen bzw. erarbeiten. Anstelle der Präsentation können auch in Teile zerlegbare informationselektronische Produkte menschlichen Handelns treten.

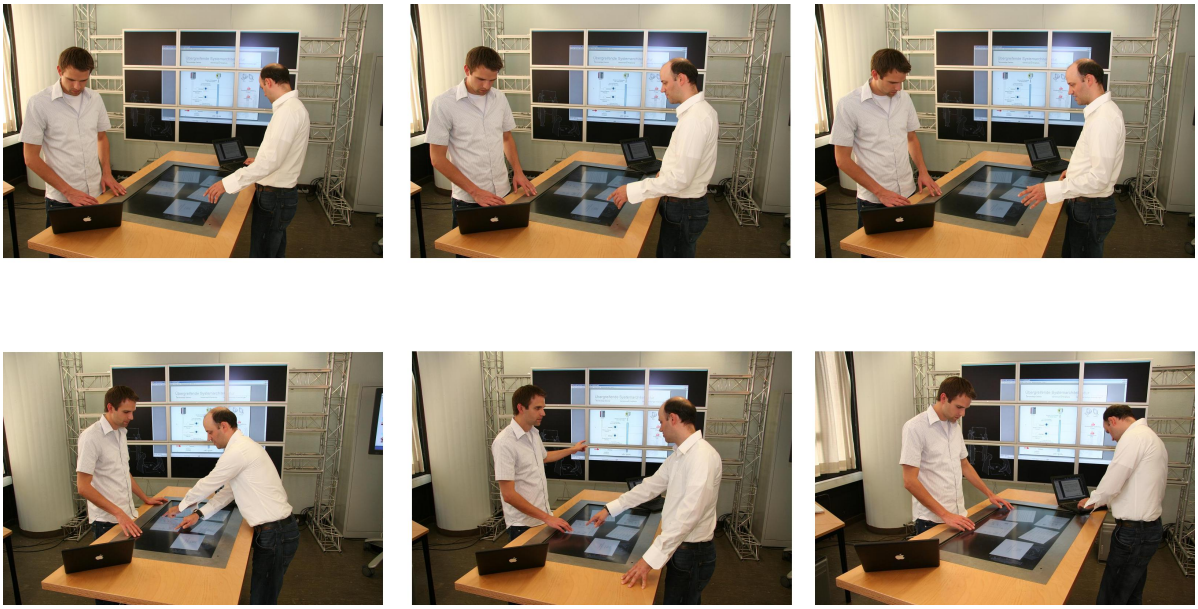

Abbildung 3.6: Ein konkretes Beispielszenario für individuelle und gemeinsame Interaktion und deren fließende Übergänge in einem Gesamtsystem mit zwei Akteuren in einer hybriden Umgebung mit mehreren Rechnern und Displays.

Die beteiligten Personen haben ihre Laptops mit ihren Daten wie Texte, Präsentationen, Filme, Grafiken oder Bilder sowie die dazugehörigen Programme zur Manipulation der Daten mitgebracht und kommen in einem Raum zusammen. Die Personen und Kollegen wollen nun eine möglichst gelungene Präsentation erarbeiten, die für die späteren Konsumenten der Präsentation und auch für sie selbst, die informationsgebenen Personen das beste Ergebnis liefert.

Um das gemeinsame Ziel definieren zu können, diskutieren und reflektieren die Personen erst einmal gemeinsam die einzelnen Standpunkte und Inhalte. Um die Argumente plastischer darstellen zu können, werden Bestandteile der individuellen Ressourcen präsentiert. Gleichzeitig werden während dieses Prozesses auch schon konkrete Lösungsansätze festgehalten und benötigte Teile aus den vorhandenen Ressourcen extrahiert und gesammelt.

Nachdem ein vorläufiges gemeinsames Ziel definiert wurde und die offensichtlichen, schon vorhandenen Bestandteile der Zielpräsentation markiert bzw. für alle beteiligten Personen zugänglich gemacht wurden, können gemeinsame und individuelle Arbeitsschritte und Aufgaben definiert werden. Die definierten Aufgaben werden alleine oder zu mehreren bearbeitet und die produzierten Ergebnisse werden dann wieder allen zur Verfügung gestellt oder direkt in das

gemeinsame Ergebnis eingebaut. Hierbei achten alle Personen darauf, dass die der Gemeinschaft zugänglichen Ressourcen für alle übersichtlich und gut einsehbar sind.

In der Produktionsphase werden Bestandteile der Präsentation von einer oder mehreren Personen so aufgearbeitet oder erstellt, dass diese nach erfolgreicher Reflexion in die Präsentation gemeinsam oder alleine eingebaut werden können. Hierbei ist zu beachten, dass die jeweilige Handhabung eines Teiles nur einer Person zufällt und die bei gemeinsamer Bearbeitung beteiligen restlichen Personen eine beratende, kontrollierende und reflektierende Rolle einnehmen.

Bei Fragen, Problemen oder Fertigstellung von Teilaufgaben treten die Teilnehmer alleine oder mit anderen wieder in die Reflexionsphase ein. Weiterhin kann es nötig sein, während des Prozesses, das gesteckte Ziel oder Teilziel zu modifizieren oder ein neues zu definieren. Die dafür nötigen oder überflüssigen Ressourcen müssen dafür in bzw. aus dem gemeinsamen Pool hinzugefügt bzw. entfernt werden.

Bei Fertigstellung nehmen eine oder mehrere Personen die Präsentation und gegebenenfalls die gesamten oder Teile der bearbeiteten Ressourcen an sich oder lagern diese auf einen der Gruppe zugänglichen Speicher. Dies geschieht im gegenseitigen Einvernehmen. Bei einer Unterbrechung der Gruppenarbeit nutzt die Gruppe den gleichen Mechanismus mit dem jeweiligen Status des Arbeitsprozesses.

3.2.3 Anwendungsfälle des Szenarios

Betrachten wir das Szenario und seine Anwendungsfälle wie in Abbildung 3.7 dargestellt in Bezug auf Produktivität fallen die vier Anwendungsfälle des Ein- und Austritts von Personen und Ressourcen in und aus dem „Raum“ heraus. Die übrigen Anwendungsfälle „Inhalte bearbeiten“ und „Inhalte diskutieren“ mit seinen beinhaltenden Anwendungsfällen können in zwei Gruppen aufgeteilt werden. Diese können dann als produktives Handeln mit sichtbaren Ergebnissen und reflexives Handeln mit Ergebnissen, die nicht konkret sichtbar sind, unterschieden werden. Ich bezeichne die Gruppe der Anwenwendungsfälle des reflexiven Handelns als „Anwendungsfälle der gemeinsamen Art“, weil die Kommunikation innerhalb der Gruppe den größten produktiven Faktor stellt. Die Gruppe der Anwendungsfälle des produktiven Handelns bezeichne ich als „Anwendungsfälle der individuellen Art“, weil hier die konkrete Manipulation des Produktes und damit die Transformation der Gedanken einer Person in die „Materie“, das fassbare Produkt einfließt.[11]

Die zu Anfang genannten vier Anwendungsfälle des Ein- und Austritts bilden die dritte Gruppe mit geringem Interaktionanteil bezogen auf die gesamte Interaktion, da diese im besten Fall

[11] Zu beachten ist, dass es sich hier um einen fließenden, mehrschichtigen Übergang handelt und die Zuordnung im konkreten Fall mehrdeutig sein kann.

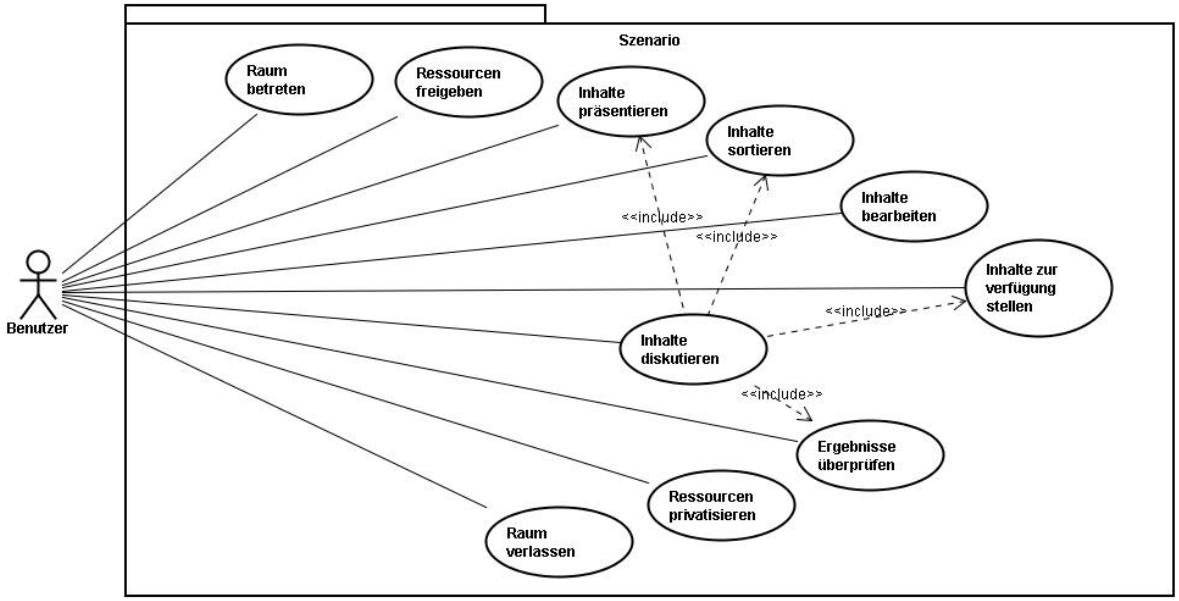

Abbildung 3.7: Use Cases innerhalb des gestellten Szenarios.

jeweils nur einmal im gesamten Prozess statt finden. Diese vier Anwendungsfälle haben einen administrativen Charakter und beteiligen sich nicht direkt am Produktionsprozess.[12]

Der individuelle Fall, der sehr komplexe Formen annehmen kann, beschränkt sich auf den ersten Blick auf den Anwendungsfall „Objekt bearbeiten“. Dieser Fall beschreibt, dass eine Datenressource wie z.B. ein Textfile oder ein Bildfile mit einer passenden Anwendung bearbeitet wird. Da diese Handlungen fast alle Anwendungsfälle, die in heutigen Systemen vorkommen, subsumiert, werden diese Anwendungsfälle während der individuellen Interaktion in dieser Arbeit nicht betrachtet. Einen Zugang zu diesen Anwendungsfällen im allgemeinen und speziellen liefern z.B. (Preece u. a., 1995) oder (Dix, 2001).

Auf das gemeinsame gleichzeitige Manipulieren einer Datenressource wird ebenfalls in dieser Analyse nicht eingegangen. Dies ist noch ein offenes Feld sowohl bei der praktischen Anwendung[13] als auch in der Forschung[14]. Die Problematiken liegen hier bei der Synchronisation und der Visualisierung der aktuellen Änderungen bzw. Arbeitsschritte. Dieses Problem zu lösen scheint nicht trivial, da die Abhängigkeiten der Inhalte innerhalb einer Ressource aufgelöst werden müssten. Um dieses zu automatisieren bedarf es auch einer Automatisierung der

[12] Die grobe Aufteilung in Navigation und Manipulation findet sich auch in anderen Arbeiten zu Interaktion wieder wie z.B. (Bowman, 1999). Hier allerdings bezieht sich die Interaktion auf eine Handlung in einem immersiven Environment, in dem sich die Person selbst im virtuellen Raum befindet und die Aufgabe darin besteht, diesen Raum zu manipulieren und nicht dessen Inhalte. Hier fügen wir einen Ebenenwechsel ein, in dem Manipulation des Raumes zu Navigation und das Ändern der Objekte an sich zur Manipulation wird. Die Navigation in erster Person im immersiven Raum fällt weg, da sich die Teilnehmer nur in zweiter Person im nicht immersiven Raum befinden.

[13] google docs... sind gute Beispiele für eine Umsetzung gleichzeitiger Dokumentenmanipulation.

[14] (revout etc.)... beschäftigen sich mit der gleichzeitigen Manipulation von Dokumenten und deren Bedingungen.

Interpretation der Inhalte und dessen Abhängigkeiten. Menschen können diese zureichend lösen, deshalb ist das gleichzeitige Manipulieren einer Ressource vor allem beim gleichzeitigen Arbeiten an verschiedenen Orten sinnvoll, da hier die Kommunikationswege der Teilnehmer eingeschränkt sind. Hier hilft eine technische Informationsanreicherung, um die kommunikativen Einschränkungen zu überbrücken.[15] Beim Arbeiten am gleichen Ort kann durch direkte Kommunikation zwischen den Teilnehmern schneller und inhaltlicher auf einen Ressourcenkonflikt eingegangen werden. Eine technische Lösung für gleichzeitiges Manipulieren ist hier nur in speziellen Fällen nötig und soll deshalb vernachlässigt werden. Diesen speziellen Fällen kann auch durch ein Auftrennen der Ressourcen begegnet werden.

Wird ein Objekt manipuliert, beschränkt sich das Gemeinsame auf das Wahrnehmen der Manipulationen. Das heißt, dass die eigentliche Manipulation einer Ressource in Ort und Zeit von erst einmal nur einer Person durchgeführt wird. Alle anderen Beteiligten sind zunächst aber gedanklich an dem Prozess beteiligt. Bei der mentalen Beteiligung handelt es sich um einen Metazustand zwischen gemeinsamem kommunikativem und individuellem Handeln. Dieser Zustand soll unter individuellem Handeln subsumiert werden, weil sich die Arbeit im späteren Verlauf auf eine technische Umsetzung bezieht und damit auf die Schnittstelle zwischen Mensch und Maschine.

Der Handlungsfall „Objekt bearbeiten“ kann, wie Abbildung 3.8 zeigt bei diesen Rahmenbedingungen genauer spezifiziert werden, wenn wir uns wieder das Szenario vorstellen. Die handelnde Person muss, bevor sie das eigentliche Objekt individuell „manipulieren“ kann, dieses „auswählen“ und nur für sich selbst „greifbar“[16] machen. Nachdem die Handlung des Manipulierens abgeschlossen ist, wird das Objekt wieder für den allgemeinen Zugriff „freigegeben“ und damit anderen Personen wieder „greifbar“ gemacht.

Tritt der Fall auf, dass eine Person Teile eines Objektes in einem anderen Objekt verwenden will, ist dies in dieser Metapher nicht möglich, ohne den Handlungsraum des gemeinsamen Handelns zu nutzen. Um aber diese Handlung im individuellen Handlungsraum belassen zu können, wird die Handlung „Objektteil überführen“ eingeführt. Diese Handlung verbindet zwei individuelle Handlungsstränge miteinander.

Die gemeinsamen kommunikativen Anwendungsfälle beschreiben alle Fälle, in denen die Personen miteinander in Kommunikation treten bzw. sind. Sie treten entweder unmittelbar in eine gemeinsame Kommunikation ein, in dem sie über ihre Daten und Vorhaben miteinander diskutieren sowie wenn nötig, ihre Daten präsentieren, oder indirekt, in dem sie den anderen Personen Daten zur Verfügung stellen. Zur Verfügung gestellt sind die Daten, wenn sie für Andere „greifbar“ sind. Das bedeutet, dass sich die Daten im Wirkungsbereich der handelnden Personen befinden und gegebenenfalls in den „privaten Arbeitsbereich“ vgl.(Scott u. a., 2003) verschoben werden können. Dieser Fall kann dann einfach mit „Objekt verschieben“ bezeichnet werden, ohne damit eventuelle komplexere Vorgänge genauer bezeichnen zu wollen. Um sich

[15] Erklärung und Beispiele der Informationsanreicherung und Erklärung der verschiedenen Kommunikationskanäle (Lautsprache, Gesten usw.).

[16] Der Vorgang des nur sich selbst Greifbarmachens setzt sich aus Handlungen der gemeinsamen Art zusammen und ist damit ein Übergangszustand zwischen gemeinsamem und individuellem Handeln. Diese Handlungen werden unter „gemeinsames Handeln“ subsumiert, um sie nicht doppelt ausführen zu müssen.

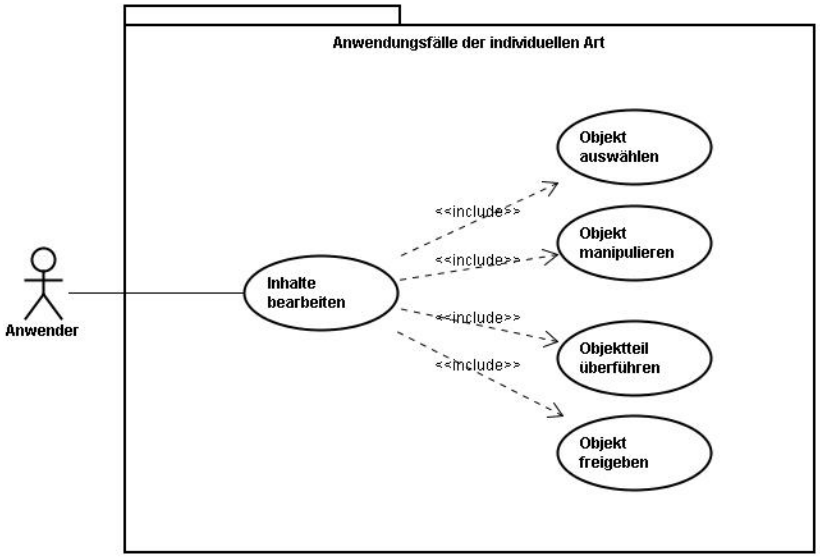

Abbildung 3.8: Use Cases in Bezug auf individuelles Arbeiten.

gegenseitig Daten zu präsentieren, sei es explizit oder peripher, müssen sie gegebenenfalls an verschiedenen Orten im Raum, in verschiedenen „Orientierungen“ und „Skalierungen“ sichtbar gemacht werden. Die Orientierung behebt den Fall, dass sich nicht alle Personen gleichzeitig in der gleichen Perspektive zum Objekt befinden. Die Skalierung ermöglicht das anpassen, wenn sich nicht alle Personen in der gleichen Entfernung befinden. Es soll also möglich sein, die Position des Objekts im Raum, die Größe des Objekts aus Sicht des Betrachters und die Ausrichtung des Objekts relativ zum Blickwinkel des Betrachters verändern zu können.

Betrachten wir die drei neu gefundenen Anwendungsfälle, die auf den aus dem Szenario extrahierten Anwendungsfällen aufbauen, lässt sich erkennen, dass sich alle Anwendungsfälle der Gruppe aus den drei Anwendungsfällen der Personen in dem Environment zusammensetzen lassen. Für die Interaktion mit dem Environment sollen künftig nur noch die drei Anwendungsfälle „Objekt verschieben“, „Objekt skalieren“ und „Objekt orientieren“, wie in Abbildung 3.9 zu sehen, genannt sein.

Die Frage: „Wer bin ich und wo komme ich her“ beschäftigt die Menschheit schon seit langer Zeit und soll auch hier gestellt werden. Denn Daten entstehen nicht aus dem Nichts und Benutzer sind nicht einfach da. Es soll also ein Ein- und Austreten von Personen und Objekten in den „Raum“ geben. Das Ergebnis dieses Ein- und Austretens beschreibt damit die handlungsfähige Masse innerhalb des Raumes selber. Befindet sich eine Person oder Ressource nicht im Raum, ist diese nicht handlungsfähig, weder im aktiven noch im passiven Sinn.

Bezogen auf das Environment wird das Betreten und Verlassen des Raumes einer Person und der zu ihr gehörenden Ressourcen in das Ein- und Austreten der Ressourcen und Rechner, welche die Objekte beinhaltet, aufgeteilt. Aus zwei Anwendungsfällen entstehen vier neue Anwendungsfälle, wie Abbildung 3.10 zeigt. Durch das Ersetzen der Personen durch die den Personen zugeteilten Rechnern und Objekte und das jeweilige Gleichsetzen der Ressourcen

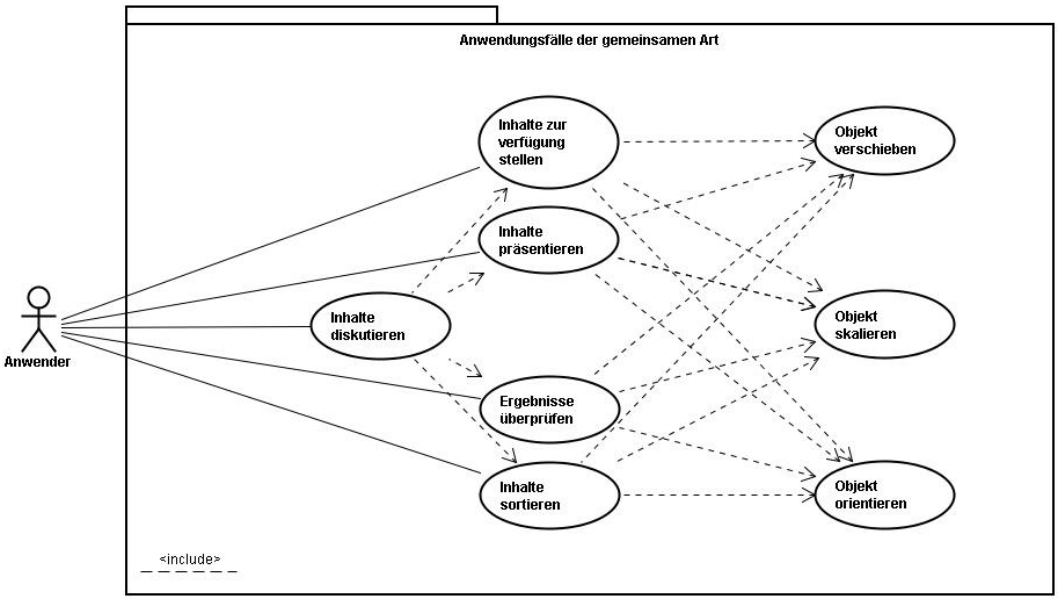

Abbildung 3.9: Use Cases in Bezug auf gemeinsames Arbeiten.

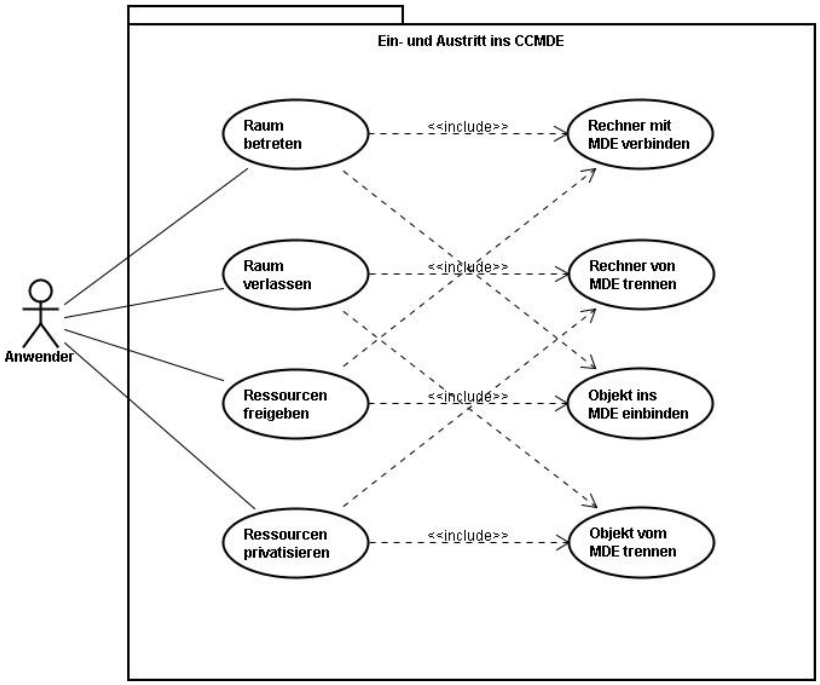

Abbildung 3.10: Use Cases in Bezug das Ein- und Austreten von Ressourcen in den gemeinsamen „Raum“.

mit diesen, überdecken sich die entstehenden Anwendungsfälle und es bleiben vier neue Anwendungsfälle, die sich auf den Ein- und Austritt in und aus dem Environment beziehen.

Damit ergeben sich für die weitere Arbeit elf Anwendungsfälle, die in drei Kategorien eingeteilt sind, wie Abbildung 3.11 zeigt. Die vier Anwendungsfälle des Ein- und Austretens sind nicht oder nur am Rande an der produktiven Interaktion der Personen beteiligt. Vier Anwendungsfälle für die Berarbeitung von Objekten beziehen sich hauptsächlich auf die Interaktion einer Person und subsumieren sehr komplexe Handlungen. Die zwei Anwendungsfälle „Objekt auswählen“ und „Objekt freigeben“ in der Gruppe der Bearbeitung beschreiben einen Übergang zwischen dem gemeinsamen und individuellen Arbeiten. Die übrigen drei Anwendungsfälle beschreiben die „Sprache“, mit der die Inhalte der Auseinandersetzung für die Personen am besten zugänglich gemacht werden können. Mit diesen drei Anwendungsfällen lassen sich die Objekte im Raum und damit in Präsenz und Sichtbarkeit navigieren. Die Reihenfolge der Unterteilung in die drei Gruppen Navigation, Bearbeitung und Übergang bezeichnet die Frequentierung und Gruppeninteraktionsdichte der Anwendungsfälle innerhalb des Produktionsprozesses.

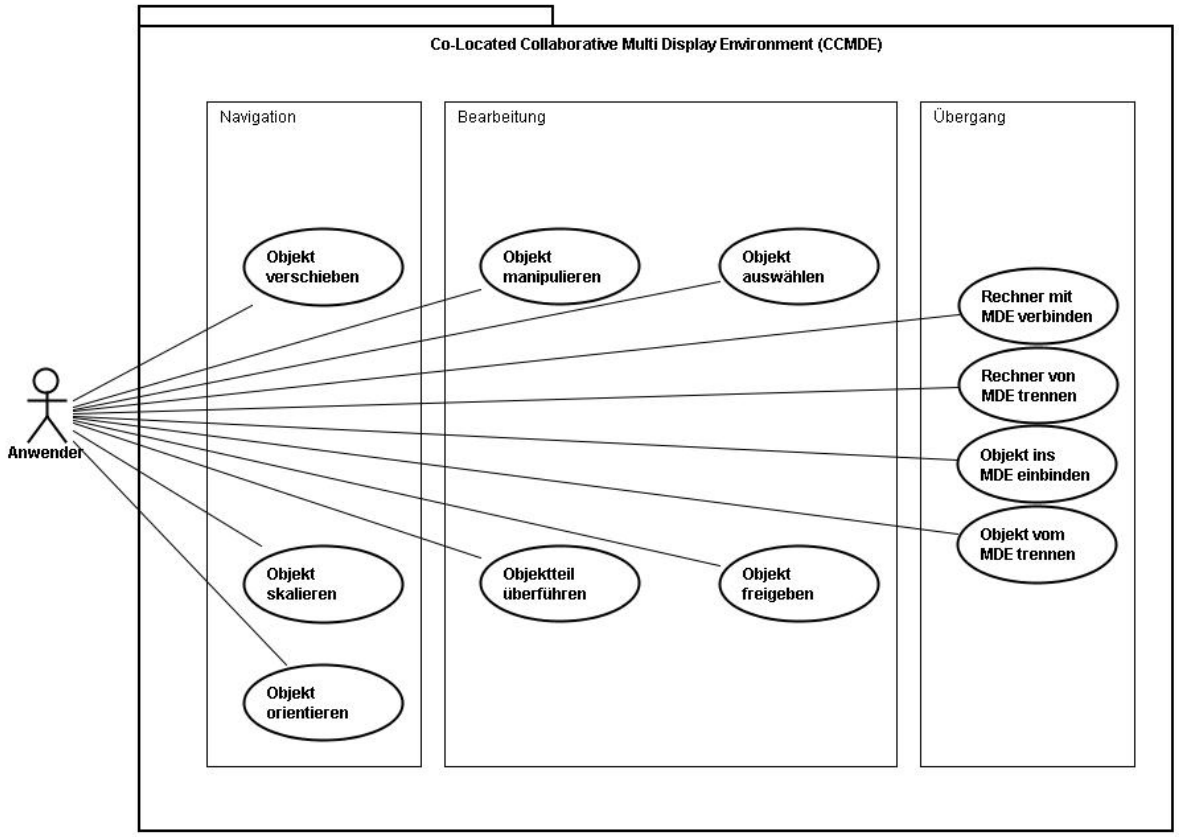

Abbildung 3.11: Use Cases in Bezug auf das Szenario mit Zuordnung der jeweiligen Kategorie.

3.3 Ausgewählte Anforderungen aus dem Forschungsbereich um MDEs

Im Folgenden werden einige ausgewählte Anforderungen aus verwandten Forschungsarbeiten zusammengetragen, die eine inhaltliche Verbindung mit der Interaktionssituation im vorher dargestellten Szenario der gemeinsamen Präsentationserstellung in einem MDE haben. Um den vorangegangenen Szenariobedingungen gerecht zu werden, wird angenommen, dass es sich bei der gesamten Arbeitsoberfläche um eine nicht durch Monitorgrenzen beschränkte und in der Ausrichtung festgelegte Benutzeroberfläche mit Desktop-Metapher handelt.

Ausschließende Bearbeitung

Bei einer Mehrbenutzerumgebung kann es vorkommen, dass mehr als eine Person ein Objekt bearbeitet oder bearbeiten will. (Greenberg und Marwood, 1994) betrachten verschiedene Techniken, die die Konflikte bei solchen Unternehmungen kontrollieren sollen, kommen aber zu keiner allgemeingültigen Empfehlung. (Dourish, 1998) entscheiden sich bei ihrer Arbeit für den einfachsten und generellsten Ansatz das „Locking“. Dieser Ansatz mit „lock-act-release“ Strategie kann auf alle Arten von Objekten angewendet werden. Die Verhandlungsmechanismen können in unserem Szenario hervorragend sozial ausgehandelt werden.

Authentifikation und Autorisierung

Da von einer gleichberechtigten Umgebung und einer Arbeitsweise zur Unterstützung kollaborativer Denkweisen ausgegangen wird, sollen alle beteiligten Personen in der Arbeitsumgebung gleiche Rechte erhalten. Dies bedeutet nicht zwangsläufig gleiche Zugangsrechte auf der Systemebene, wohl aber auf der Benutzungsebene. Das Ausschließen von ungebetenen Gästen soll hier nicht weiter betrachtet werden, ist aber natürlich bei Nutzung von drahtlosen Techniken erforderlich, da hier nicht die Anwesenheit im physikalischen Raum erforderlich ist. Ein „Sichtbarmachen“ von im „Raum“ eingebundenen Rechnern soll hier genügen, so dass ungebetene Gäste erkannt werden können. Die Anmeldung an das System kann durch das Starten der WeFace Unterstützungsanwendung erfolgen. Damit ist es nicht nötig, sich als Person mit Accountnamen und Passwort anzumelden. Es werden damit nicht Personen, sondern Rechnerressourcen in WeFace autorisiert und authentifiziert.

Einbinden von Objekten

Es muss eine Möglichkeit geben, Objekte in das System zu „importieren“ oder wieder zu „exportieren“ bzw. zu entfernen. Wird ein Objekt aus dem privaten Bereich freigegeben, so erscheint dieses im öffentlichen Bereich. Dies soll die einzige Schnittstelle sein, die nicht ohne eine explizite Handlung auskommt, weil hier die einzige Trennung zwischen „sicheren“ privaten

Objekten und „unsicheren“ öffentlichen Objekten stattfinden soll. Da der Transfer der Objekte hauptsächlich zum ersten Prozessschritt der Gruppenarbeit gehört, scheint die „explizite Handlung“ nicht als störend, da der Interaktionsfluss zu diesem Zeitpunkt durch die Auswahl der Objekte sowieso verzögert wird.

Eingabeumleitung in verteilten Systemen

Weil es sich um verteilte Arbeitsflächen handelt und diese nicht immer von der eigenen Position „erreichbar“ ist, ist es nötig die lokalen Eingabemöglichkeiten auf entfernte Arbeitsflächen umzuleiten, um damit eine Erreichbarkeit zu gewährleisten. „Pointright“ von (Johanson u. a., 2002b) ermöglicht z.B. mit einem „redirect“ der Maus und Tastatur das Arbeiten am Wandbildschirm, ohne den eigenen Arbeitsplatz zu verlassen.

Framework zum Einbinden von bekannten oder unbekannten Funktionen

Um den Zugang von vorhandenen Anwendungen oder Daten in ein Multi-Display-Environment zu unterstützen, stellen (Biehl u. a., 2008) das Framework „IMPROMPTU“ vor. Ein Framework für Eingaben in MDEs mit dem Namen „Universal Interaction Controller“ (UIC)bieten (Slay und Thomas, 2006). Das Entwickeln von nativen Anwendungen in ein MDE unterstützt das Framework „WeSpace“, das von (Wigdor u. a., 2009) vorgestellt wird.

Inhaltliche und visuelle Perspektive

Wird ein Objekt auf einen anderen Desktop verschoben, kann es sinnvoll sein, das Erscheinungsbild und die Manipulierbarkeit des Objektes zu verändern. Wenn zum Beispiel ein 3D-Objekt, das zur Manipulation als Tabelle dargestellt wird, zur Ansicht und Erfahrbarkeit auf einen anderen dafür installierten Bildschirm verschoben wird, müsste sich die Darstellung von der Tabellenform in eine gerenderte Oberflächenform ändern.

Die inhaltliche Perspektive kann neben der qualitativen Sicht auch als quantitative interpretiert werden. Ist dies der Fall, wird die Perspektive der möglichen Darstellung aus technisch machbarer Sicht betrachtet. Nutzen wir zum Beispiel einen Handheld, der nur über begrenzte Rechenkraft und Bildschirmauflösung verfügt, müsste sich nicht wegen des Kontextes das Erscheinungsbild des Objekts ändern, sondern wegen der technischen Voraussetzungen des darstellenden Gerätes. Dieser Umstand der kontextsensitiven Darstellungsform wird für mobile Geräte als „Multi-Channel-Presentation“ oder als „multikanalfähig“ bezeichnet. Eine Arbeit hierzu findet sich unter (Scherp und Boll, 2005).

Mehrbenutzerfähigkeit

In „A new collaborative software package: TeamSpace at Stanford University“ von (Hebert und Chen, 2005) wird die Erfahrung beschrieben, dass zwei öffentliche Displays mit jeweils dazugehörigem Rechner einen Vorteil gegenüber einem einzigen öffentlichem Display bieten, wie es (Shih und Winograd, 2004) als Grundkonstruktion vom Teamspace beschreiben. Der Grund für diesen Vorteil liegt darin, dass bei dem hier benutzten Tool „Pointright“ von (Johanson u. a., 2002b) zum Umleiten der Eingaben immer nur eine Person auf den öffentlichen Bildschirm zugreifen kann. Bei zwei separaten öffentlichen Bildschirmen können zwei Personen parallel im öffentlichen Bereich arbeiten. Hieraus ergibt sich die Annahme, dass eine allgemeinere Mehrbenutzerfähigkeit zur gleichen Zeit am gleichen Ort einen Vorteil bei der Zusammenarbeit im Team mit öffentlichem Bildschirm bietet. Auch Johanson spricht sich in (Johanson u. a., 2002b) für eine sinnvolle Erweiterung des Systems in Richtung einer Mehrbenutzerfähigkeit auf einem Rechner bzw. Display. Die Mehrbenutzerfähigkeit wird außerdem durch die Nutzung eines Tabletops von mehreren Personen gleichzeitig strukturell vorgegeben.

Objektinhalte überführen

Um Teile von Objekten in andere Objekte bei gleichzeitigem Zugriff mehrerer Personen im gleichen System zu integrieren, ist eine personengebundene Clipboard-Funktion mit angebundenem „Cut & Paste“ sowie „Drag & Drop“[17] nötig. Einen möglichen Lösungsansatz für Ubiquitous Computing Environments mit Hilfe eines „Clipboard Models“ präsentieren (Slay u. a., 2005). Einen anderen Ansatz mit Hilfe von physikalischen Objekten, die zu „Trägern“ von Informationen werden, um diese von einem Rechner zum anderen zu transferieren, präsentieren (Rekimoto und Saitoh, 1999) mit „hyperdragging“.

Orientierung von Objekten

Objekte, die von verschiedenen Benutzern aus unterschiedlichen Perspektiven betrachtet werden (siehe Abbildung 3.12), müssen manuell oder bei vorhersagbaren Verhältnissen automatisch neu orientiert werden können. Eine zusammenfassende Arbeit zu diesem Thema liefern (Ringel u. a., 2004).

Passive Wahrnehmung

Um eine größere Unterbrechung des Arbeitens zu vermeiden und um jeder beteiligten Person ein ähnliches Verständnis des Fortschreitens des gemeinsamen Projektes sowie der individuellen Arbeiten zu ermöglichen, sollen alle Arbeiten für alle Personen passiv wahrnehmbar sein. (Dourish und Bellotti, 1992) bringen diesen Zusammenhang mit einem Satz auf den Punkt:

[17] Eine sehr gelungene Arbeit zum Thema „Drag & Drop“ in MDE's liefert (Rekimoto, 1997). Hierbei handelt es sich allerdings nicht um den Transfer zwischen Objekten in MDE's sondern zwischen Rechnern.

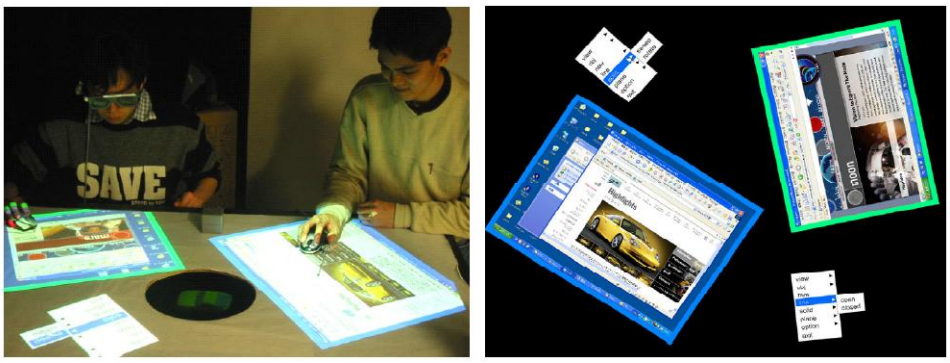

Abbildung 3.12: Zur Bearbeitung oder Betrachtung können bei (Nakashima u. a., 2005) Objekte neu orientiert werden.

„Awareness is an understanding of activities of others, which provides a context for your own activity.“

Räumliche Perspektive

Es kann nötig sein, dass sich die räumliche Perspektive der Objekte bzw. des Desktops auf verschieden ausgerichteten Bildschirmen ändern muss, um dem Betrachter die intendierte Sicht und damit einen anderen Einblick auf die Szene zu gewähren (Forlines u. a., 2006). Diese Anforderung bezieht sich nur auf die räumliche Ansicht zur Unterstützung der intuitiven Betrachtung des Benutzers. Eine umfangreiche Arbeit zur räumlichen Darstellung von Objekten in MDEs liefert (Nacenta u. a., 2007).

Reproduzierbarer Systemstatus

Sitzungen müssen unterbrechbar sein, da sie mehrere Tage andauern können. Bei längeren Unterbrechungen sollte das Equipment zwischenzeitlich anderweitig nutzbar sein, um ein Blockieren von Ressourcen zu vermeiden. Um eine Unterbrechung der Gruppenarbeit und die Wiederkehr zu genau dem selben Status zu ermöglichen, muss die Reproduzierbarkeit gewährleistet sein. Damit kann der Arbeitsprozess dort fortgeführt werden, wo er unterbrochen wurde.

Self-Configuring

Benutzerintegration, Raum-Architekturintegration, Gestaltung und Zustand der aktuellen Softwareumgebung werden in den meisten Fällen durch einen Administrator gewährleistet. Ziel muss sein, das System so zu gestalten, dass eine Modifikation des Raumes oder der Benutzerzusammensetzung intuitiv von den Benutzern selbst oder durch Sensoren automatisiert angepasst werden kann. Damit wird der administrative Aufwand und damit die Schwelle zur

Benutzung gering gehalten. In der Gruppe der Akteure braucht es dann keine Person mit speziellem Wissen. Wir können dann von „Zero-Administration“ aus (Szybalski, 2004) oder „self-configuring“ sprechen. Ein ähnliches Forschungsgebiet wird unter dem von (Horn, 2001) geprägten Namen „Autonomic-Computing“[18] bearbeitet. Es ähnelt den oben aufgeführten Forderungen sehr stark, weil auch hier beim Selbstkonfigurieren, Sensoren und Topologien angesprochen werden. Es handelt sich aber keineswegs um die Anordnung und Übergänge bei den Präsentationsbildschirmen, sondern um die Konfigurtation der Rechner. Trotzdem können gegebenfalls hier Methoden zum Erreichen des Ziels übernommen werden, soweit sie auf die hier geforderte Präsentationsanforderung abbildbar ist.

Sicherheit

Es muss sicher gestellt sein, dass die Daten nur dem für die Aufgabe bestimmten Zweck zur Verfügung stehen. Das bedeutet eine klare Trennung zwischen privatem und öffentlichem Bereich. Es kann davon ausgegangen werden, dass es kein destruktives Handeln der Gruppenmitglieder innerhalb des öffentlichen Bereichs gibt, da die beteiligten Personen ein gemeinsames Ziel haben. Somit sind keine Sicherheitsgrenzen erforderlich. Ohne ein gemeinsames Ziel wäre kollaboratives Handeln nicht möglich. Die nicht am kollaborativen Prozess beteiligten Daten auf den „Privatrechnern“ sollen aber geschützt werden, ebenso wie die Originaldateien, die als Grundlage der gemeinsamen Arbeit dienen. Öffentlich zur Verfügung gestellte Objekte sollen nur mit Zustimmung des Rechners[19], der das Original zur Verfügung stellt, von einem anderen Rechner privatisiert werden können.

Skalierung der Objekte

Verschiebt man ein Objekt von einem Display mit geringer Auflösung und niedriger Pixeldichte auf einen mit hoher Auflösung und hoher Pixeldichte bei gleichbleibender Entfernung zum Betrachter, so sollte gegebenenfalls die Skalierung des Objektes geändert werden können, um die Sichtbarkeit bzw. Erkennbarkeit zu waren. Dies kann bei fest eingestellten Skalierungseigenschaften des Displays automatisiert geschehen und soll bei nicht vorhersagbaren Verhältnissen manuell möglich sein. Die Umsetzung dieser Eigenschaften werden in (Forlines u. a., 2006) und (Nacenta u. a., 2007) besprochen und präsentiert.

[18] Bei der Firma IBM ist Anfang des zwanzigsten Jahrhunderts ein neuer Forschungszweig mit Namen „Autonomic Computing“ entstanden. Dieser geht von der Annahme aus, dass das vegetative Nervensystem von Lebewesen ohne gezielte Entscheidungen des Verstandes wichtige Funktionen des Körpers steuert. „Autonomic Computing“ soll ebenfalls ohne Zutun des Verstandes (in diesem Fall der Benutzer des Computersystems) wichtige Funktionen für ein sinnvolles Fortbestehen eines Systems (in diesem Fall des Computersystems) sichern. Eine Bestandsaufnahme zu „Autonomic Computing“ liefern (Huebscher und McCann, 2008) zum Ende der Dekade.

[19] Da sich keine Personen sondern die Rechner der Personen in das System integrieren, wird hier von Rechnern gesprochen. Gemeint sind indirekt die Personen, die die Rechner bedienen. Welche Person dies ist, wird real durch Anwesenheit der Personen und die dadurch real entstehenden Territorien der Personen ausgehandelt.

Synchronisation

Gibt es mehrere Kopien einer Datei oder einer laufenden Anwendung, so muss das System sicher stellen, dass diese ohne Zutun des Anwenders synchronisiert werden, damit es keine Inkonsistenzen im Datenbestand geben kann. Es wird explizit auch auf die Synchronisation der Anwendung hingewiesen, da im Prozess der Bearbeitung einer Datei die komplexesten Problematiken der Synchronisation auftreten. Der störende Schritt der Synchronisation, sei es manuell oder automatisch, kann umgangen werden, wenn vermieden wird, Kopien von Objekten anzufertigen. Kopien zu vermeiden soll damit eine Anforderung an das System sein, weil manuelles Synchronisieren den Arbeitsfluss unterbricht und automatisches Synchronisieren fehlerhaft oder nur in speziellen Fällen angewendet werden kann.

Überbrückung von Displayzwischenräumen

Bei einem Multi-Display-Environment muss der Raum zwischen den Monitoren mit in die Betrachtung der Darstellung einbezogen werden. Es muss also eine Lösung für die Überbrückung dieses Raums von Objekten formuliert und umgesetzt werden, so dass eine intuitive Bedienbarkeit erhalten bleibt. Bei einer Untersuchung von verschiedenen Ansätzen zur Zwischenraumüberbrückung von Zeigern kommen (Nacenta u. a., 2008) zu folgendem Schluss:

> „Stitching is still the safest choice. ... Stitching was the clear winner in our performance tests, but its performance was still negatively affected by the size of the gap.“

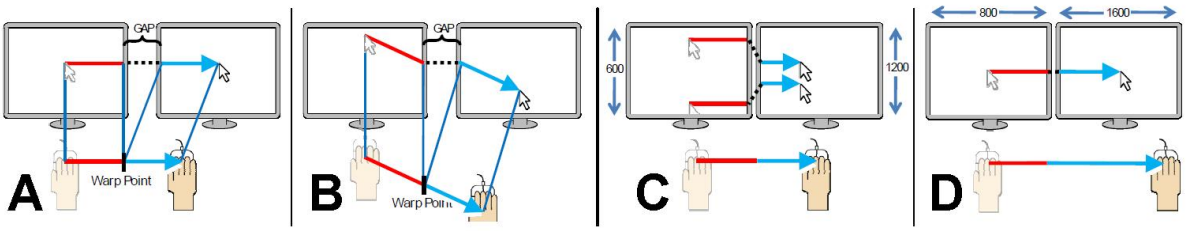

Abbildung 3.13: Stitching überbrückt den Zwischenraum von verschiedenen Monitoren. A.) Die Lücke wird ignoriert, indem die Lücke im motorischen Raum in den „Warp Point“ komprimiert wird. B.) Eine diagonale Bewegung wird in einen multilinearen Bahnverlauf umgewandelt. C.) Das Ausfluchten aller Bahnverläufe bei verschiedener Auflösung der Monitore ist nicht möglich. D.) Das Verhältnis zwischen den Bewegungen hängt vom Display ab. (Aus (Nacenta u. a., 2008))

Versionierung

In der Arbeit von (Burfeindt, 2006) findet sich die funktionale Anforderung der Versionierung, um einen Prozessschritt rückgängig machen zu können, der sich als Irrweg herausstellt. Dafür

sollte eine Kontrolle der genutzten Dateien im Hintergrund diese Anforderung erfüllen. Dies kann auch in den laufenden Anwendungen geschehen, auf die das MDE kein Einfluss hat, oder bei Speicherung eines Arbeitsstandes der Ressource. Bei Speicherung der Ressource kann dies auf das ressourcengebende System ausgelagert werden. Ist dies der Fall, betrifft die Versionierung nicht das MDE, sondern die darunter liegenden Strukturen.

Verteilung von Anwendungen

Anwendungen sollen jeder Person auf jedem Rechner im Raum zur Verfügung stehen. Deshalb muss es möglich sein, von jedem Rechner im Raum auf jede Anwendung zugreifen zu können und es ebenso auf den eigenen Arbeitsplatz holen und bearbeiten zu können. Verschiedene Anwendungsmanagementlösungen werden von (Biehl und Bailey, 2006) untersucht. Beim Vergleich von drei verschiedenen Ansätzen, die in Abbildung 3.14 zu sehen sind, kommen (Biehl und Bailey, 2006) zum Schluss, dass das Iconic Interface wegen seiner höheren räumlichen und visuellen Repräsentation des gesamten Arbeitsbereichs am besten geeignet ist. Deshalb sollte eine Anwendungsmanagementlösung eine hohe räumliche und visuelle Repräsentation bereitstellen.

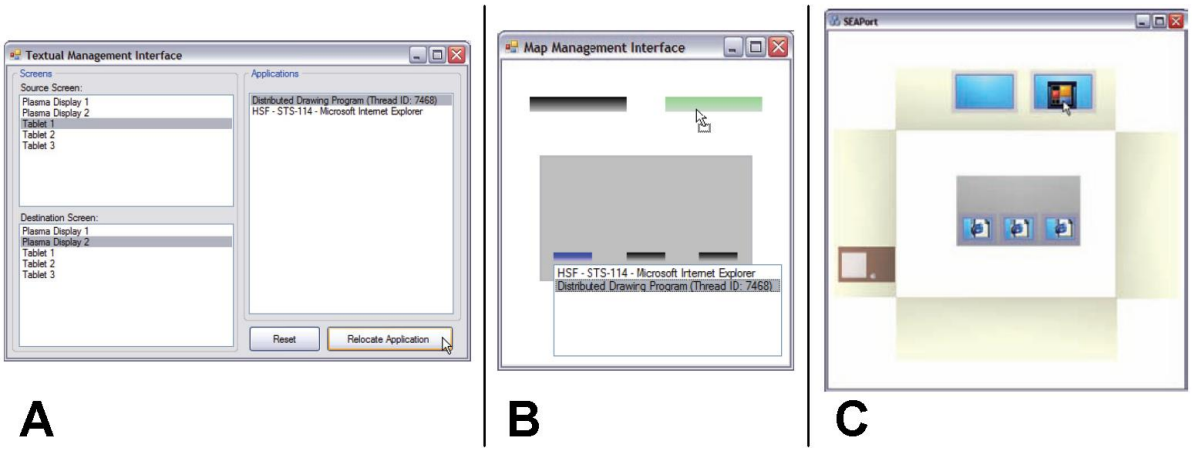

Abbildung 3.14: In (Biehl und Bailey, 2006) werden drei Verwaltungsinterfaces A.) „Textual Interface“ B.) „Map Interface“ und C.) „Iconic Interface“ für Anwendungen untersucht. Die Schnittstellen werden aus Perspektive eines Benutzers gezeigt, der am Tisch sitzt. A, B und C zeigen wie eine Anwendung vom Tablet (links) auf das große Display (rechts) verlagert wird.

3.4 Zusammenfassung

In Abschnitt 3.1 wurde der Handlungsraum beim Arbeiten in Gruppen in individuelle und kollektive Handlung unterteilt. Wenden wir diese Unterteilung auf die aggregierten Use Cases an,

so subsumieren sich die Use Cases, die Objekte im Raum navigieren zu den kollektiven Handlungen und die Use Cases, die Objekte innerhalb verschiedener Status „navigieren“ zu den individuellen. Bei den individuellen kann zusätzlich noch zwischen den *System verändernden* und *Objekt verändernden* unterschieden werden.

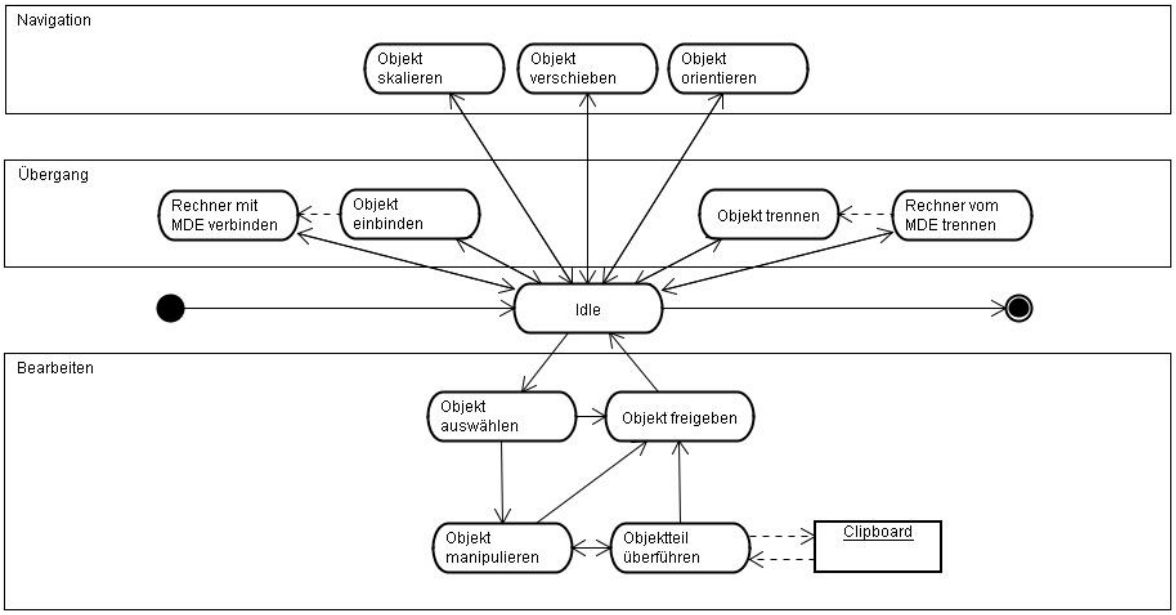

Abbildung 3.15: Aktivitätsdiagramm in Bezug auf das Gesamtsystem.

Bilden wir erst einmal die Uses Cases, die aus dem Szenario aus Abschnitt 3.2 resultieren in einem Aktivitätsdiagramm ab, wie in Abbildung 3.15 zu sehen. Weil es sich um aggregierte Use Cases handelt, können diese direkt übernommen werden. In den ausgewählten Anforderungen aus Abschnitt 3.3 gibt es die Forderung nach einem sich selbst konfigurierenden System. Gehen wir davon aus, dass das System sich selbst konfiguriert, können wir die zwei Aktivitäten zur Integration von Rechnern in das System ausschließen und verringern damit den Umfang der zu betrachtenden Aktivitäten. Die Aktivitäten zum Einbinden, Trennen, Auswählen und Freigeben eines Objektes bilden eine Verbindung zwischen der kollektiven Handlung und anderen Handlungen. Sie müssen deshalb weiter mit betrachtet werden. Die Aktivitäten zum Manipulieren und Überführen werden aus kollektiver Sicht der Handlungen von den Aktivitäten des Freigebens und Auswählens abgedeckt und können deshalb komplett unter dem individuellen Handlungsraum subsumiert werden.

Übrig bleibt ein minimiertes Aktivitätsdiagramm mit Aktivitäten in den Kategorien „Kollektiv“, „Individuell“ und „Individuell & Kollektiv“, wie in Abbildung 3.16 gezeigt. Weil die ausgewählten Anforderungen aus Abschnitt 3.3 entweder ausschließend oder übergreifend sind, integrieren diese sich nicht zusätzlich in diese Kategorien und ins Aktivitätsdiagramm. Die übergreifenden Anforderungen sollen nicht an der Interaktion beteiligt sein und damit unmerklich im Hintergrund ablaufend realisiert werden.

Bei der iterativen Entwicklung eines Systems zur Unterstützung gemeinsamer gleichzeitige

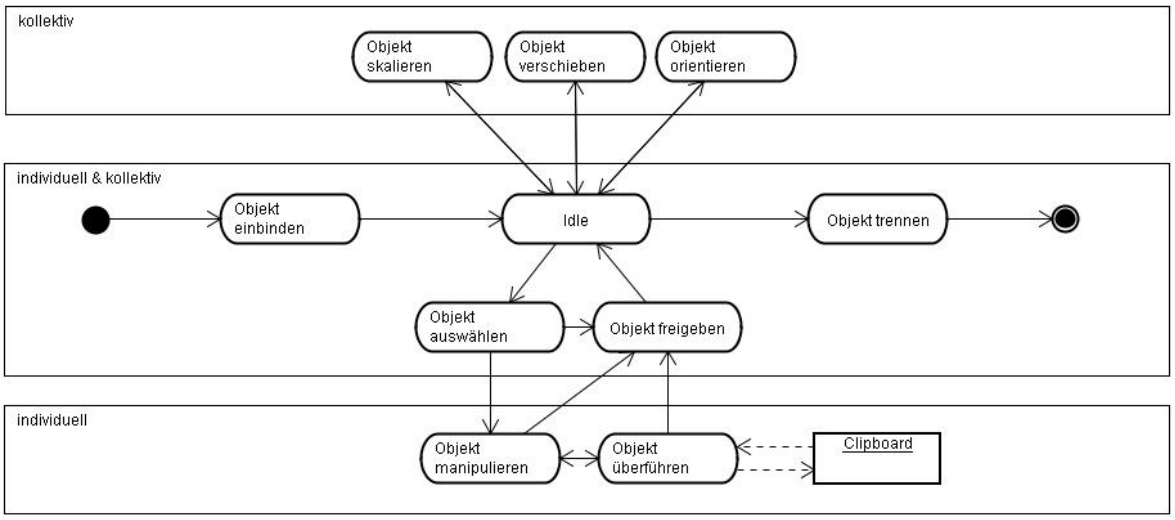

Abbildung 3.16: Aktivitätsdiagramm in Bezug auf ein Objekt.

Interaktion am selben Ort sollen folgende priorisierte funktionale Anforderungen gelten:

Höchste Priorität für kollektive Handlungen:

- Objekt verschieben
- Objekt skalieren
- Objekt orientieren

Normale Priorität für kollektive und individuelle Handlungen:

- Objekt auswählen
- Objekt freigeben
- Objekt einbinden
- Objekt trennen

Niedrige Priorität für individuelle Handlungen:

- Objekt manipulieren
- Objektteil überführen

Auch wenn die Anforderungen aus Abschnitt 3.3 nicht direkt an der Interaktion beteiligt sind, so sind diese bei der Entwicklung eines Systems aber stets mit zu betrachten, da dieses ohne diese nicht anwendbar wäre. Folgende notwendige Anforderungen aus anderen Forschungsarbeiten sind von Anfang an zu integrieren:

- Configuring
- Überbrückung von Displayzwischenräumen durch „Stitching“

- Assoziation von Eingabegeräten

Anforderungen, die nicht zwingend sofort integriert werden jedoch mit betrachtet müssen, sind:

- Self-Configuring
- Reproduzierbarer Systemstatus
- Application Interface für native Anwendungen
- Räumliche Perspektive
- Inhaltliche und visuelle Perspektive
- Überbrückung von Displayzwischenräumen
- Passive Wahrnehmung

Die Anforderungen *Verteilung von Anwendungen*, *Cut & Paste*, *Orientierung von Objekten*, *Skalierung von Objekten*, *Ausschließende Bearbeitung* und *Mehrbenutzerfähigkeit* werden bereits durch die funktionalen Anforderungen abgedeckt.

4 Design

Eine komplett neue Arbeitsumgebung mit eigenen Regeln und Anwendungen zu entwickeln, bedarf großer Anstrengungen, ist aber in der Konzeption freier, weil die Einschränkungen durch vorhandene Systeme ausgeblendet und damit übergangen werden können. Die vorhandenen Ressourcen weiter nutzen zu können, stellt von Anfang an einen größeren Kontent bereit, bedarf aber auch einer Anpassung des neuen an das alte. Da davon auszugehen ist, dass die Desktopmetapher und deren nutzbare Anwendungen den Benutzern bekannt sind und damit ein mentales Modell zur Funktionsweise eines „Desktops“ in der Vorstellung des Benutzers weitestgehend konstruiert ist, können sehr einfach alte Vorstellungen und Verhaltensweisen übertragen bzw. genutzt werden. Lässt sich auch der vorhandene Kontent technisch nutzbar machen, so erweitert und beschleunigt auch dies den Vorgang der Nutzbarmachung einer neuen Umgebung.

Gegen eine Übertragung der Desktopmetapher oder einiger Teile spricht, dass wenn eine neue Metapher eingeführt werden soll, diese aber in der Vorgehensweise und Struktur der alten so weit widerspricht, dass es zur Konfusion[1] des Benutzers und der Entwickler führt. Deshalb müssen entweder alle alten Metaphern beiseite gelegt werden oder wenn nötig eine klare Trennung zwischen den Metaphern gemacht werden. Ein gutes Beispiel hierfür ist der Übergang der Bedienoberflächen von der reinen Texteingabe im „Command-Prompt“ zu einer WIMP-Oberfläche[2]. Innerhalb des Graphical User Interfaces (GUI) des WIMP gibt es Enklaven der reinen Texteingabe, sowohl in Systemen der hybriden als auch der ausschließlichen grafischen Nutzung der Systemverwaltung.

Im Kapitel 3 wird zwischen individuellem und kollektivem Raum bzw. Handeln unterschieden. Diese Unterscheidung kann dazu genutzt werden, in der Konzeption an gleicher Stelle eine Unterscheidung zwischen Altem und Neuem vorzunehmen. Damit werden im Design Teile[3] des Desktops zum individuellem Arbeitsbereich und der gemeinsame Arbeitsbereich zum kollektiven digitalen Raum innerhalb der zusammen arbeitenden Menschen und Computer. Im

[1] Diese Konfusion kann sich darin äußern, dass es schwer ist, den Umgang mit der neuen Umgebung zu erlernen, weil die alten Verhaltensweisen die neuen überlagern oder sogar dazu führen, dass sich im Zwischenspiel zwischen Anwender und Entwickler immer wieder die alten Denkweisen durchsetzen und die eigentliche Bedeutung der Innovation sich nie wirklich durchsetzen kann. Ein Beispiel hierfür findet sich in einer Email von Alan Kay, in der er bedauert, das Wort „objects“ ins Leben gerufen zu haben, weil dies seiner Ansicht nach dazu geführt hat, dass die größere Idee des „messaging“ nicht gesehen wurde (Kay, 1998) (Kay, 2003).

[2] WIMP steht für „Windows Icons Menus und Pointing device“. Die reale Interaktion mit der WIMP-Oberfläche wurde in Xerox PARC 1973 eingeführt und wurde mit der Einführung des Macintosh Computer 1984 populär.

[3] Wichtig ist hier die Betonung auf Teile, denn nicht der gesamte Desktop soll hier als individueller Raum gelten. Der Raum des Desktops soll aufgelöst und in den kollektiven überführt werden. Die einzelnen Teile, die den Desktop füllen, wie z.B. Anwendungen und Daten sollen auf eine neue Interaktionsebene überführt werden.

Zusammenhang des gemeinsamen Raums sollen wie in Kapitel 3 gefordert, die Objekte für alle greifbar sein. Das heißt, dass alle Nutzer mit jedem Objekt in diesem gemeinsamen Bereich gleichberechtigt überall agieren und sich Personen und Objekte ohne Grenzen im physikalischen wie auch im virtuellen Raum bewegen können sollen. Grenzen werden lediglich durch das Besetzen eines Ortes gesetzt und können bei Bedarf sozial verhandelt werden. Die strukturellen Informationen über die Positionen der realen Personen im physikalischen Raum sowie die Informationen über die Positionen der Objekte im virtuellen Raum sind ein wichtiger Bestandteil des gemeinsamen Produktionsprozesses. Deshalb ist die ungehinderte Beweglichkeit ein wichtiger und zu berücksichtigender Faktor. Der Aspekt der technischen Reproduzierbarkeit dieser strukturellen Information, wie auch in 3.3 gefordert, soll durch die Konzeption so gesichert sein, dass eine Erweiterung des Systems mit einer Reproduzierbarkeit des Status keine Änderung des Grundsystems erfordert.

Der erste Abschnitt 4.1 setzt sich mit einem Vorschlag zur allgemeinen Systemstruktur für die Software und deren Einbettung in die vorhandene Systemstruktur auseinander. Die nächsten zwei Abschnitte 4.2 und 4.3 sollen die Voraussetzungen[4] und damit Leitlinien beschreiben, die für die Entwicklung der Systemarchitektur in Abschnitt 4.4 gelten sollen. Der Abschnitt 4.2 beschreibt somit die Voraussetzungen für ein funktionales Modell. Abschnitt 4.3 beschreibt die strukturellen Voraussetzungen, die nicht nur als technische Vorgabe für die Umsetzung, sondern auch als inhaltlich strukturgebende Komponente gedeutet werden sollen. In Abschnitt 4.4 wird eine Systemarchitektur entwickelt, die aus den zwei Voraussetzungen und den Anforderungen aus der Analyse in Kapitel 3 folgt. Die einzelnen Komponenten und Schnittstellen des Systems werden in 4.4.2 charakterisiert. Im vorletzten Abschnitt 4.5 werden Komponenten, die allerdings nicht zwingend für die gemeinsame Interaktion sind, in das zuvor vorgeschlagene integriert und besprochen. Zum Schluss in Abschnitt 4.6 wird die letztendlich vorgeschlagene Architektur mit drei Systemarchtikturen aus ähnlichen Bereichen abgeglichen.

4.1 Allgemeine Diskussion der Systemstruktur

In diesem Abschnitt wird argumentativ das Einfügen einer Interaktionschicht in die allgemeine Architektur der gemeinsam genutzten Einzelsysteme beschrieben. Abbildung 4.1 Bereich A zeigt, die Einzelsysteme[5], die sich in das Gesamtsystem[6] integrieren sollen. Diese bestehen vereinfacht gesehen aus einer Eingabe, einem Verarbeitungssystem und einer Ausgabe. Weil

[4] „In the early 1980s two main types of mental models that users employ when interacting with devices were identified: these are categorized as structural and functional models. (...) A simple distinction is to consider structural models as models of 'how-it-works' and functional models as models of 'how-to-use it'.“(Preece u. a., 1995)

[5] Das Wort Einzelsystem bezeichnet hier einen Ein-Personen-Arbeitsplatz, also ein Benutzer und ein Rechner mit Null bis X Eingaben und Null bis Y Ausgaben. Null Ein- oder Ausgaben machen bei einem Arbeitsplatz keinen Sinn, vereinigt aber auch Serversysteme in die Menge der Einzelsysteme.

[6] Das Wort Gesamtsystem subsumiert alle genutzten Rechner, Eingaben und Ausgaben der agierenden Personen im Interaktionsprozess.

Einzelsysteme mehrere Ausgaben und verschiedene Eingaben vereinen können, kann die Anzahl der Ausgabe- und Eingabeinstrumente sowie der Verarbeitungssysteme im Gesamtsystem differieren. Um ein gemeinsames Arbeiten zu vereinfachen, werden die Einzelsysteme miteinander gekoppelt. Dies geschieht durch das Erweitern von Bereich A, um die Möglichkeit des entfernten Zugriffs, z.B. dem „input redirect“ aus Abschnitt 3.3, und um den maschinengestützten bzw. automatisierten Austausch von Daten bzw. deren Manipulation. Das Einzelsystem in Bereich B beinhaltet nun zusätzlich die zwei Komponenten „Remotecontrol“ und „Collaborative Application“, wie in der Abbildung 4.1 zu sehen. Dieser Bereich bezeichnet weiterhin ein Einzelsystem, nun aber mit maschinengestützter Verbindung zu den anderen involvierten Einzelsytemen. Zusammen betrachtet erhalten wir eine einfache Architektur für kollaboratives Arbeiten mit mehreren Menschen und Systemen. Diese gängige Architektur der verbundenen Einzelsysteme verbindet nicht nur über die Kollaborationsanwendung und den entfernten Zugriff die Systeme und Menschen miteinander, sondern verharrt auch in der Struktur und Denkweise der separierten Wirkungsbereiche. Dies wird dadurch bestätigt, dass diese Architektur sowohl in Co-Located Collaborative Workplaces und in voneinander entfernten Collaborative Workplaces (Artman u. a., 2005) eingesetzt wird. Eine Übersicht verschiedener Aspekte von „Computer Supported Co-operative Work“ von zeigt (Dewan u. a., 1999).

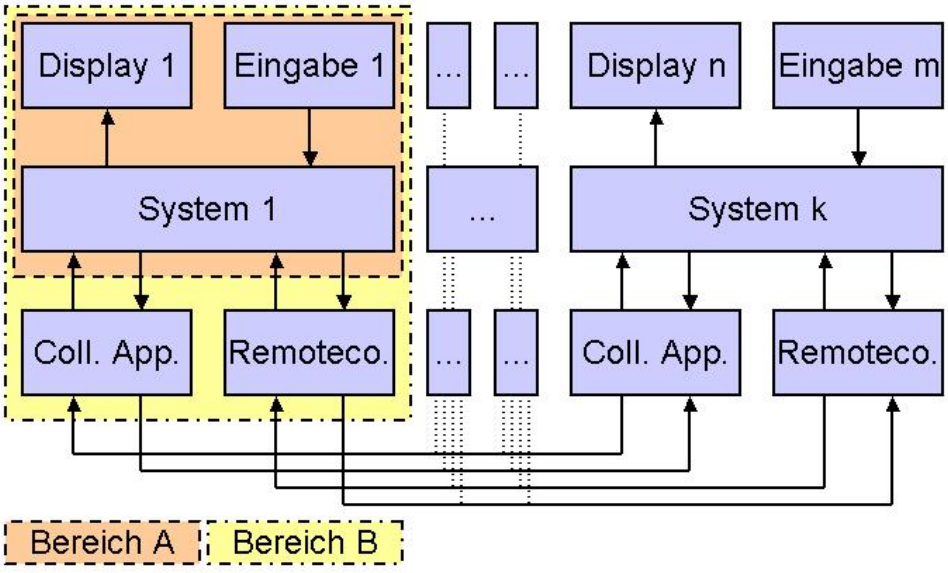

Abbildung 4.1: Verallgmeinerte Architektur für Kollaborationssysteme.

In der Praxis verbindet dieses Herangehen also weiterhin jeweils einen Benutzer mit „einer“ Eingabe und Ausgabe. Das bedeutet, dass weiterhin die Interaktion zwischen Mensch und Computer im Vordergrund steht. Das Gesamtsystem präsentiert sich damit weiterhin als eine Ansammlung von Einzelsystemen mit denen interagiert wird. Hier gibt die technisch zu Grunde liegende Struktur das Denken der Entwickler und Benutzer des Gesamtsystems vor, wie zum Beispiel der Prototyp „ProjectRoom“ von (Gottwald, 2007) zeigt.

Soll die Struktur vorgeben, dass die Benutzer und Entwickler des Gesamtsystems, dieses als eins begreifen, müssen die Einzelsysteme von einer gemeinsamen Struktur überdeckt werden.

Die Benutzer interagieren mit dieser Struktur mit Hilfe der physikalischen Schnittstellen und die Entwickler nutzen diese Struktur als Grundlage der Interaktion mit den Bestandteilen ihrer Anwendungen. Diese gemeinsame Struktur als Interaktionsschicht, wie Abblildung 4.2 zeigt, verbindet die Systeme miteinander und parallel dazu die Interaktion der Benutzer. Damit unterstützt und sichert diese Grobstruktur der Architektur den Fortbestand der zugrunde liegenden Idee durch die Entwurfs- und Entwicklungsschichten bzw. Benutzungs- und Denkschichten hindurch.

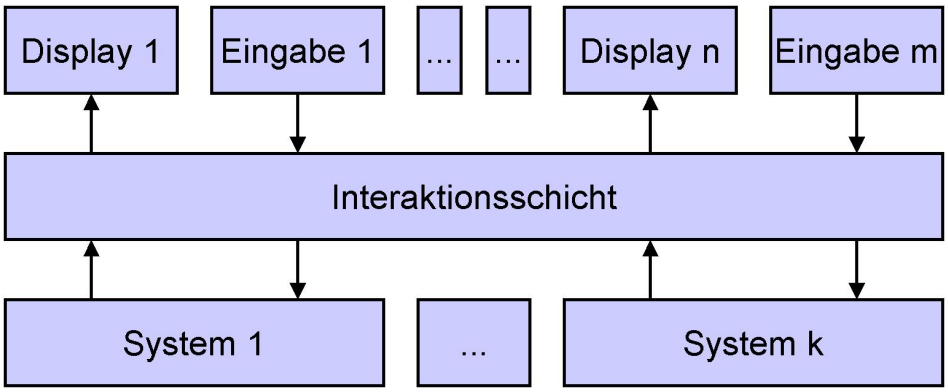

Abbildung 4.2: Die Interaktionsschicht verbindet alle Eingaben, Ausgaben und Systeme miteinander.

Mit Einfügen der gemeinsamen Interaktionsschicht in die Einzelsysteme aus Abbildung 4.1 Bereich A erweitern wir das vorgegebene und nicht veränderbare System um eine frei gestaltbare Komponente, wie in der Einleitung zu diesem Kapitel gefordert. Diese Komponente bedarf Schnittstellen zu den Einzelsystemen, welche die jeweilige Präsentation der fließenden Daten in ein für die Interaktionsschicht nutzbares Format übersetzen. Wie Abbildung 4.3 zeigt, werden die Präsentationen der Anwendungen in die Interaktionsschicht geleitet und an die jeweiligen Displays weitergeleitet. Ebenso werden die Benutzereingaben von der Interaktionsschicht verarbeitet und an die betreffende Anwendung weitergeleitet. Durch die Übersetzung in ein generalisiertes Format der Interaktionsschicht können systemspezifische Unterschiede ausgeglichen werden und damit Systemgrenzen überwunden werden.

Weil gegebenenfalls die Systeme auch unabhängig von der Interaktion der Benutzer Daten austauschen müssen, um z.B. die Konsitenz der Daten oder die Performance im Gesamtsystem zu sichern, muss die Möglichkeit der versteckten Interaktion zwischen den Einzelsystemen gegeben sein. Diese können dann entweder durch die Systeme selbst ausgelöst werden oder auch indirekt durch die Interaktion der Benutzer. Das Auslagern der „Datashare“-Komponente soll die Interaktonsschicht von technisch notwendigen Implementierungen freihalten, damit diese weitgehend frei von technisch bedingten Kompromissen bleibt. Weiterhin ist die reale Implementierung der „Datashare“-Komponente gekapselt und kann damit auch durch Standardlösungen unabhängig von der Interaktionsschicht realisiert werden. Aus diesem Grund werden im weiteren Verlauf der Designentwicklung die Aspekte der „Datashare“-Komponente nur peripher betrachtet. Die Dienste der „Datashare“-Komponente sollen nicht direkt mit der Interaktion der Benutzer mit dem Gesamtsystem zu tun haben.

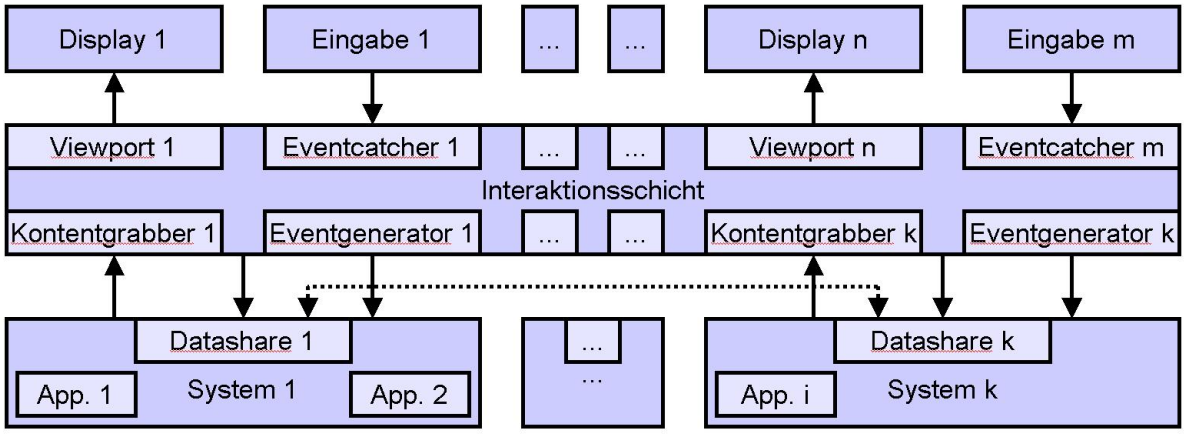

Abbildung 4.3: In der Interaktionsschicht werden die Schnittstellen zu den herkömmlichen Komponenten realisiert und anhand des Status der Interaktionsschicht projiziert bzw. extrahiert.

Die integrierte Interaktionsschicht auf den Einzelsystemen besteht genau betrachtet erst einmal aus vielen parallelen Interaktionsschichten, weil die vorgefundenen Architekturen keine Mechanismen zur transparenten Implementierung anbietet.[7] Die Komponente „Distribution“, wie in Abbildung 4.4 gezeigt, soll diese Transparenz über die Einzelsystemgrenzen gewährleisten. Die Entwickler der Interaktionsschicht müssen nur die Voraussetzungen für die Interaktion zwischen Distributionskomponente und Interaktionsschicht gewährleisten. Sind diese gewährleistet und damit die Transparenz der Interaktionsschicht über die Einzelsysteme hinaus, kann mit den lokalen Interaktionsschichten umgegangen werden als wäre es die gesamte Interaktionsschicht.

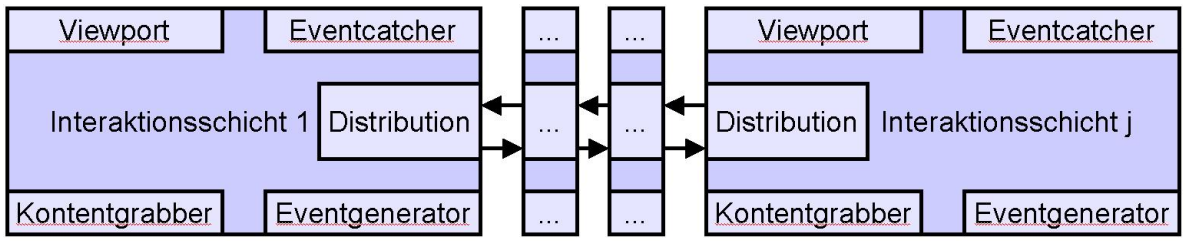

Abbildung 4.4: Der Status der Interaktionsschicht wird durch die Distributionskomponente über das Gesamtsystem verteilt.

Betrachten wir die Umsetzung der Distributionskomponente und die zugrunde liegende Architektur dieser, so müssen wir beim Gesamtsystem von einem dynamischen System ausgehen.

[7] Der findige Leser wird hier natürlich einwenden, dass genau aus diesem Grund viele Entwicklungen zu verteilten Anwendungen wie z.B. Corba und Java-RMI eine Lösung liefern. Weil wir uns aber auf der Metaebene zur Diskussion um verteilte Einzelsysteme befinden und nicht in der Diskussion um die Implementierung einer verteilten Anwendung, werden die wohl geformten Lösungen für die Bereitstellung der Transparenz hier nicht betrachtet.

Die Diskussion muss mit den beiden Polen „Client-Server“ und „Peer-to-Peer“ wegen der Dynamik der involvierten physikalischen Komponenten und damit der Unvorhersehbarkeit des Systems auf Seiten der Client-Server-Architektur als Lösung eingeschränkt werden. Bei Verwendung einer Client-Server-Architektur muss es möglich sein, dass sich zur Laufzeit ein Client zum Server und ein Server zum Client wandeln kann ohne die Benutzung des Systems zu beeinflussen.[8] Diese Vorgabe gewährleistet bei Austreten des Servers aus dem Gesamtsystem oder bei veränderter Lastverteilung, dass ein Client und damit ein anderes Einzelsystem die Aufgaben des Servers übernehmen kann.

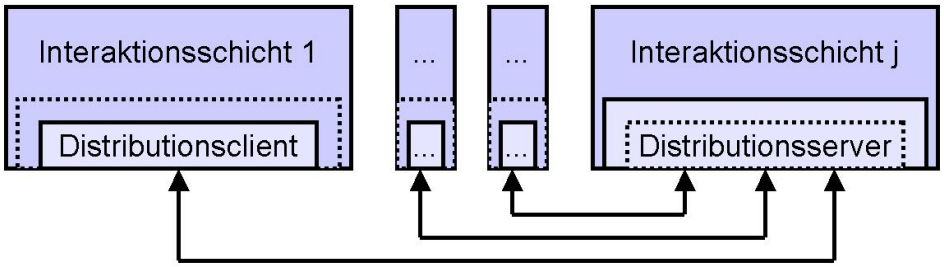

Abbildung 4.5: Den Nachteilen einer zentral organisierten Client-Server-Architektur wird mit einem dynamischen und situationsbedingten Serverwechsel begegnet.

Die Forderung nach Reproduzierbarkeit des Status des Gesamtsystems wird auch durch die Möglichkeit des dynamischen Serverwechsels unterstützt, da der Status im Gesamtsystem so gehalten werden muss, dass dieser innerhalb des Gesamtsystems transformierbar ist und damit innerhalb der Transformation auch konserviert werden kann. Wie Abbildung 4.5 zeigt, soll es möglich sein, dass die Komponente Distribution innerhalb der Interaktionsschicht auf dem Einzelsystem, entweder die Form des Servers oder des Clients annehmen kann. Diese Client-Server Transformation gilt dann auch für den nicht distributierten Teil der Interaktionsschicht und damit der Interaktionsschicht selber. Die lokale Interaktionsschicht muss also entweder auf Client und Server gleich sein oder transformierbar in den jeweiligen geforderten Status: Client oder Server.

Bei einer reinen Peer-to-Peer Architektur muss der durchdringende Gesamtzustand ebenfalls konservierbar und die Zusammensetzung des Gesamtsystems während der Laufzeit veränderbar sein. Zur Konservierung eignet sich ein ähnlicher Mechanismus wie bei der Transformierung der Einzelsysteme im Falle einer dynamischen Client-Server Architektur. Weil die Konservierung sich auf das Gesamtsystem bezieht, ist es nicht, diese Aufgabe nur den einzelnen Peers zu überlassen. Die jeweilige Konserve muss sich auf den Status des Gesamtsystems beziehen und wieder auf diesen reproduzierbar sein. Ein dynamisches Eintreten oder Austreten eines Peers in das Gesamtsystem erfordert gegebenenfalls eine Umstrucktuierung des Peergefüges. Diese Umstrukturierung muss ebenfalls dynamisch möglich sein.

[8] Die Möglichkeit, dass jeder Knoten im Netz Server als auch Client sein kann, wird auch in einem Bluetooth-Personal-Area-Net verwirklicht. Bei ZigBee können Router so konfiguriert werden, dass sie die „Koordinator“-Rolle übernehmen können. Dieser „Koordinator“ entspricht dem „Super Peer“ in einem unstrukturierten Peer-to-Peer Netz. Der in (Dustdar u. a., 2003) vorgestellte Dispatcher kann bei Veränderung der Client-Serverstruktur die Koordination übernehmen.

4.2 Eins, zwei, drei ... Desktops

Die in der Analyse in Abschnitt 3.4 dargestellten Anforderungen, die Grundlage für das zu entwickelnde Design sind, unterliegen bei der Übernahme von Teilen herkömmlicher Systeme und der vorhandenen Interaktionsgeräte weiterer Voraussetzungen und Einschränkungen der vorhandenen materiellen und logischen Systeme. In den nächsten Abschnitten sollen diese unter Annahme der Nutzung mehrerer Desktopsysteme beschrieben werden. Neben den theoretisch analytischen Betrachtungen kann somit die Betrachtung konkreter Umgebungsfunktionalitäten Eingang in das Design des Systems finden.

Wir gehen davon aus, dass auf jedem an der Interaktion beteiligten Geräte bisher ein System auf Basis der Desktopmetapher genutzt wurde. Wollen wir diese Geräte und dessen Inhalt, nach den Vorgaben der Analyse, während einer Gruppenarbeit nutzbar machen, müssen wir verschiedene Punkte beachten.

4.2.1 Der Desktop als Arbeitsbereich

Die Grundstruktur eines herkömmlichen Desktopsystems basiert darauf, dass es nur einen Nutzer, einen Rechner und eine Interaktionsinstanz[9] gibt. Der Desktop hat die Funktionalität, Daten visuell zugreifbar und Anwendungen visuell aufrufbar zu machen. Die Metapher ist angelehnt an die Vorstellung eines Schreibtischs, auf dem Zettel liegen und Stifte, mit denen der Inhalt der Zettel verändert werden kann. Vergleichen wir diese Metapher mit der des gemeinsamen Arbeitens, so besteht der gemeinsame Arbeitsbereich aus vielen Tischen oder einem großen Tisch, an dem viele arbeiten können. Der Unterschied zwischen einem großen Tisch und vielen kleinen Tischen bei funktional ähnlicher Anordnung besteht hauptsächlich darin, dass sich Lücken zwischen den einzelnen Tischen bilden und es generell kein Zentrum für die Blickrichtung der Beteiligten geben muss. Mit vielen Tischen sind funktional andere Anordnungen möglich, welche hier aber nicht betrachtet werden sollen. Nach (Scott, 2005) bilden sich bei der Gruppenarbeit an einem großen Tisch private Arbeitsbereiche zum individuellen Produzieren von Informationen und gemeinsame Arbeitsbereiche zum Lagern und Austauschen von Informationen zwischen den Gruppenmitgliedern. Der Zugriff auf die individuellen Bereiche wird dadurch beschränkt, dass dieser physikalisch durch die Reichweite der Arme beschränkt wird oder durch soziale Grenzen, die von den Personen eingehalten werden.

Wollen wir aus den vielen kleinen Tischen mit verschiedener Höhe, Ausrichtungen und Größen einen großen machen, so passen wir die Höhe an und legen eine große durchsichtige Platte über diese. Damit können alle ungehindert Daten austauschen ohne Gefahr zu laufen, dass Daten vom Tisch fallen könnten. Wollen wir dies in einem elektronischen System realisieren, so müssen auch hier ein minimaler Nenner der heterogenen Systeme gesucht werden und eine

[9] Mit Interaktionsinstanz ist das Tripel Mensch, Tastatur/Maus und Monitor gemeint. Eventuell kann der einzelne Monitor auch aus mehreren Monitoren bestehen, die als erweiterter Desktop betrieben werden. Aus funktionaler und struktureller Sicht kann der erweiterte Desktop aber als Monitor interpretiert werden.

neue gemeinsame Interaktionsebene eingeführt werden, die die gemeinsamen sowie individuellen Arbeitsbereiche gleichsam nutzbar macht und die Lücken zwischen den Tischen schließt. Der in Kapitel 3.2.2 geforderte gemeinsame Datenpool entsteht dann, wenn man einen „Tisch“ in die Mitte zwischen die anderen Tische stellt. Eine Pinwand entsteht, wenn man einen freien „Tisch“ an die Seite stellt.

Nicht zu vergessen ist, dass wenn man in der heutigen Zeit eine Person, die einen Rechner besitzt, einen Rechner bedienen lässt, sie keine Einweisung in das System benötigt. Die Bedienung eines PCs ist mittlerweile bei den meinsten Menschen in das Allgemeinwissen übergegangen. Veränderungen am Verhalten des Systems müssen entweder auf anderen schon vorhandenen Grundannahmen der Benutzer basieren oder eben auf Grundannahmen der Desktopmetapher zurückgreifen und vom Anwender transformierbar sein, damit es kein Spezialwissen oder eine explizite Einarbeitung zur Nutzung des Systems geben muss.

4.2.2 Daten

Ebenfalls wie das grafische Environment, welches keine Grenzen innerhalb des zusammengesetzten Gesamtsystems aufzeigen soll, sollen sich die Daten und damit die Dateien aus Sicht des Benutzers frei im System bewegen können. Gehen wir von der Desktopmetapher als Vorbild aus, werden hier beim visuellen Verschieben der Daten in Form von Icons keine Kopien angelegt und die eigentlichen Daten nicht verändert. Lediglich die Repräsentation, ausgedrückt durch die Position des Verweises und damit eines Icons wird verändert. Die eigentlichen Daten bleiben physikalisch an gleicher Stelle. So soll auch an dieser Stelle konzeptionell vorgegangen werden. Lediglich die Repräsentation des Zugriffkanals der Daten soll im System dem Benutzer dargestellt werden, frei beweglich sein und es sollen bei dieser Bewegung innerhalb des Gesamtsystems keine neuen Instanzen systembedingt erzeugt werden. Dieses Vorgehen hat den Vorteil, dass sich die eigentlichen Daten weiterhin im Besitz des ursprünglichen Eigentümers befinden, auch wenn alle am Kollaborationsprozess beteiligten Personen diese verändern können. Es müssen keine Synchronisationsmechanismen eingeführt werden, die die Datenkonsistenz gewährleisten. Beim Verlassen einer „datenbesitzenden“ Person kann gewährleistet werden, dass ein Verbleiben der Daten in der Gruppe durch die Übertragung der Daten auf den Rechner einer anderen Person, die in der Gruppe bleibt, mit Zustimmung der besitzenden Person erfolgt.

4.2.3 Anwendungen

Anwendungen verhalten sich in einem herkömmlichen Desktopsystem ähnlich wie Icons, nur mit dem Unterschied, dass ein Ausschnitt aus der Datenmenge repräsentiert wird und dass der Inhalt und damit die Daten selber durch Eingaben des Benutzers verändert werden können. Diese Repräsentation mit Option zur Manipulation ist ebenso frei auf dem Desktop beweglich und kann zusätzlich verschiedene Formen annehmen, ohne die Daten während des

Änderns der Repräsentation zu verändern, geschweige denn aus Interaktionsgründen neue Instanzen der Daten zu erstellen. Die technische Herausforderung an eine Umsetzung innerhalb eines verteilten Systems ist weit größer als bei der freien Beweglichkeit der Repräsentation der Datenexistenz. Dieser größere und nicht unwesentliche technische Aspekt soll im Konzept allgemein mit betrachtet werden, um einem späteren Scheitern der Umsetzung des Designs vorzubeugen.

4.2.4 Benutzereingaben

Benutzereingaben in einem herkömmlichen System laufen im allgemeinen so ab, dass ein Benutzer einen logischen Mauszeiger für verschieden gelagerte Eingaben und/oder einen Cursor für die Texteingabe per Tastatur hat. Bei einem System mit mehreren Benutzern und mehreren Mäusen und/oder Tastaturen muss bedacht werden, dass die einzelne Darstellung der einzelnen Mäuse unterscheidbar sein muss. Dies kann durch verschiedene Farben sowie verschiedene Ausrichtung bzw. Aussehen geschehen. Die gleichzeitige Benutzbarkeit und Sichtbarkeit muss wegen der synchronen Bearbeitung ebenfalls gewährleistet sein. Einem abstrakten Benutzer werden Eingabeinstrumente zugeordnet, welche dann die Handlungen dieses Benutzers im virtuellen Raum repräsentieren. Dies wird im Normalfall eine Maus und eine Tastatur sein. Ein Benutzer kann aber auch mehrere gleiche Eingabeinstrumente vereinen und es muss möglich sein noch andere Arten der Eingabe mit in die Menge der zugeordneten Eingabeinstrumente einführen zu können.

Stellen wir uns ein Multi-Display-Environment mit mehr als drei Bildschirmen vor, auf denen wir unseren Mauszeiger „aus den Augen verloren“[10] haben. Hier muss ein Mechanismus eingeführt werden, der dem Benutzer hilft den Mauszeiger wiederzufinden. Die gleiche Problematik stellte sich (Baudisch u. a., 2004) und wurde „by wiggling the mouse“ gelöst. Beim „wiggling“ wird die Maus schnell hin und her bewegt. Diese Bewegung wird als Geste erkannt und der Cursor bläht sich z.B. auf, wird also größer und damit besser erkennbar. Eine andere Methode kann eine ortsunabhängige Tastatureingabe sein, die dazu führt, dass die Position des Mauszeigers auf einen bekannten Ort gesetzt wird und damit ohne Suchen wiedergefunden werden kann. Dieser Mechnismus könnte auch dafür genutzt werden, Objekte, die mit dem Zeiger verbunden sind, leicht in die Nähe des individuellen Arbeitsbereiches zu bringen. Hierbei muss aber bedacht werden, dass die übrigen Teilnehmer diesen Vorgang ohne sehr genaue Beobachtung nicht wahrnehmen können. Dies würde damit der Anforderung 3.3 aus der Analyse widersprechen und sollte vermieden werden.

Um die Eingaben der Benutzer von den lokalen Systemen in das Gesamtsystem zu übertragen, muss beachtet werden, dass die lokalen Eingaben verschiedenen Orientierungen, Skalierungen und Formaten unterliegen. Diese Übersetzung der Eingaben vom lokalen ins globale und gegebenenfalls vom globalen ins lokale muss ein konfigurierbares oder selbst konfigurierendes

[10] Hier ist der direkte Sichtkontakt zwischen dem Auge und dem Zeiger gemeint. Um diesen wiederherzustellen, muss der Zeiger optisch gefunden werden. Auf einer großen unübersichtlichen Fläche bei relativ kleinem Zeiger kann dies zu einer mühseligen Arbeit werden.

Modul, welches gegebenenfalls auch kontextabhängig die Eingaben übersetzt, übernehmen und gewährleisten.

4.2.5 Fazit

Die vier vorgestellten Teilaspekte Desktop, Daten, Anwendungen und Benutzereingaben eines transformierten Desktopsystems lassen sich in drei Ebenen einteilen. Die gemeinsamen Desktops bilden den Träger für die Daten und Anwendungen. Die Daten und Anwendungen bilden die visuelle Repräsentation für den Zugriff auf die Daten. Die Benutzereingaben bilden die dritte Ebene, welche den Inhalt der zweiten Ebene entweder direkt oder indirekt über die zweite Ebene manipuliert.

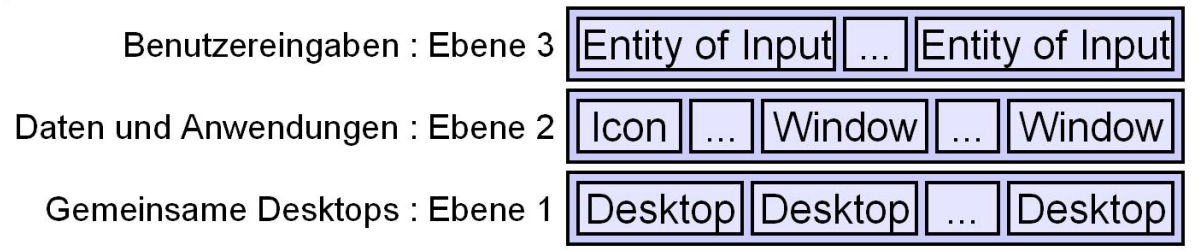

Abbildung 4.6: Die Desktopmetapher teilen wir in 3 Ebenen auf.

Für die drei Ebenen müssen noch verschiedene Aspekte beachtet werden. Die Ebene 1 ist eine Ebene, die nicht oder eingeschränkt während der Interaktion mit dem Gesamtsystem verändert werden sollte. Der Arbeitsbereich kann sich zwar während einer Gruppenproduktion ändern, sollte dies aber nicht fortwährend tun, um den Benutzern Struktur zu geben und den technischen Aufwand nicht ausufern zu lassen.

Die Ebene 2 der Daten und Anwendungen teilt sich in weitere zwei funktionale Ebenen auf. Zum einen wird die Interaktionsrepräsentation verändert, sprich die Ausrichtung, Größe und Position, zum anderen muss auch der Inhalt der Daten veränderbar sein. Die beiden Unterebenen 2a und 2b können mit den in der Analyse (Abschnitt 3.1) unterschiedenen Gesichtspunkten „Reflexion“ und „Produktion“ und damit mit der „kollektiven“ und „individuellen“ Handlung in Verbindung gebracht werden. Da die individuelle Handlung mit dem privaten Arbeitsbereich aus (Scott, 2005) assoziiert ist und in dieser ausschließlich eine Person handelt, sollte in Ebene 2b nur eine Eingabeeinheit einem Objekt der Ebene 2b zugeordnet sein. Um Objekten weiterhin bestimmte Freiheitsgrade zu gewähren, sollten die Objekte innerhalb der zwei Unterebenen wechseln können.

Ebene 3 bindet die Benutzereingaben in das Gesamtsystem ein. Da es sich um verschiedene Benutzereingaben handelt, müssen diese unterscheidbar und nachvollziehbar sein. Deshalb ist eine visuelle Rückkopplung für die Benutzer nötig. Weil in der Vergangenheit häufig verschiedene Eingabeinstrumente zusammen als Einheit benutzt wurden, ist es angeraten, Cluster von verschiedenen Eingabeinstrumenten zu einer Einheit zusammenzufassen. Diese Einheit

macht es auch möglich, den Eingabekontext, einen Status und Inhalte zu halten, um Konfigurationen anzupassen oder Inhalte zwischen den Objekten über den Benutzer auszutauschen. Eine Eingabeeinheit bildet sozusagen eine Identität für assoziierte Eingabeinstrumente.

4.3 Strukturelle Leitlinien

Dieser Abschnitt gibt die strukturellen Leitlinien zur Entwicklung des Architekturdesigns vor. Die Beschreibung der Leitlinien, die auf die Verteilung des Systems eingehen, ist in zwei verschiedene Abschnitte unterteilt. Diese zwei Abschnitte bewerten die Trennungslinie innerhalb der Architekturschichten und die Anwendbarkeit klassischer Verteilungsarchitekturen. Die darauf folgenden drei Abschnitte betrachten das Architekturpattern, das für das Architekturdesign zugrunde gelegt wird. Der erste Abschnitt führt uns zum MVC-Pattern, das schließlich im zweiten Abschnitt für unseren Problemkomplex betrachtet wird. Der dritte rundet mit einem Hinweis auf Evaluierbarkeit des Systems die Besprechung des Architekturpattern ab. Der letzte Abschnitt für die strukturellen Leitlinen betrachtet das strukturelle Modell eines Prototypen zur nahtlosen gemeinsamen Interaktion, aus welchem dann Leitlinien für das zu entwickelnde Architekturdesign abgeleitet werden.

4.3.1 Der goldene Schnitt durch die Architekturschichten

Eine eindeutige Zuordnung der Trennlinie innerhalb der Architekturschichten nach (Tanenbaum und Steen, 2002) fällt angesichts der vielschichtigen Anforderungen aus der Analyse in Kapitel 3 schwer. Zum einen soll es eine gemeinsame Interaktionsmöglichkeit für alle beteiligten Personen auf allen benutzten Rechnern geben und zum anderen soll die Nutzung aller vorhandenen Ressourcen in Form von Anwendungen und Daten möglich sein. Erstere Forderung wird durch die Verteilungsanforderung der Eingaben und Ausgaben zur eigenständigen Anwendung. Durch die Bedeutung des Zustands der Interaktion als Arbeitsergebnis geht diese über ein reines User Interface hinaus. Der zweiten Forderung kann mit einer Kapselung der vorhandenen Ressourcen und dem Ersetzen des vorhandenen User Interface durch eine Übersetzungschicht zwischen neuem und altem User Interface begegnet werden. Durch diese zwei Forderungen entstehen verschiedene Anforderungen an die Trennungslinie innerhalb der Architekturschichten ein.

Beim Ersetzen der User Interface Schicht durch eine Übersetzungsschicht würde, wie in Abbildung 4.7 gezeigt, nur Variante (a) oder (b) in Frage kommen, da die Daten und die Anwendung auf dem jeweiligen Rechner verbleiben. Der jeweilige Rechner wird damit zum Server und alle anderen Rechner gegebenenfalls Clients. Die auf den Clients stattfindende Interaktion mit dem Gesamtsystem muss innerhalb des Gesamtsystems interpretiert und nach Ort und Zeit extrahiert werden und universell zum Client gesandt werden. Universell deshalb, weil der Client nach den Anforderungen in Kapitel 3 im allgemeinen eine hybride Ausprägung hat. Der Client wiederum muss diese universelle Interpretation an sein jeweiliges System anpassen

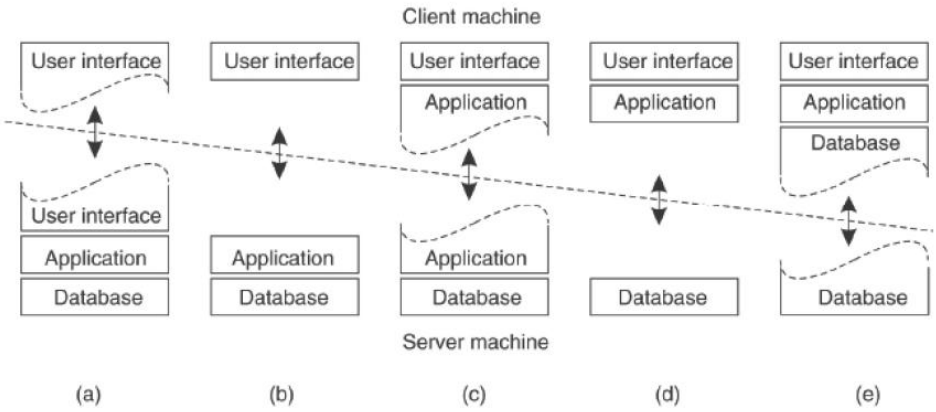

Abbildung 4.7: Einordnung der Trennung des verteilten Systems innerhalb der Architekturschichten. (Tanenbaum und Steen, 2002)

und an die eigentliche Anwendung senden. Nach dieser Interpretation würde es sich damit um Fall (a) handeln, in dem sich der Interaktions-Übersetzer als Ersatz für das User Interface auf dem Server teils auf dem Client befindet. Bei Einbringung serialisierbarer Anwendungen würde sich dieses Schema nicht ändern, da sich bei schnellem Wechsel verschiedener Clients der Aufwand für einen ständigen Transfer der Anwendung von einem Rechner zum anderen nicht lohnen würde. Lohnt sich dieser Übertragungsaufwand und wird die Anwendung auf den anwendungsnutzenden Client übertragen, befinden sich beide Teile der Client-Server Architektur auf dem gleichen Rechner und der Verteilungsaufwand sinkt gegen Null. Diese Variante ist damit auch im Fall einer Entwicklung spezieller Software für die angestrebte Lösung sinnvoll.

Die Trennlinie der Verteilung beim Thema der gemeinsamen Interaktionsanwendung lässt weit mehr Spielraum zur Diskussion. Strukturell widersprechen die Fälle (d) und (e) der Forderung nach einem gemeinsamen Arbeitsbereich, weil User-Interface und Anwendung auf jedem Client unabhängig von einander nur gekoppelt über die zugrunde liegenden Daten miteinander Arbeiten. Um mit diesen Schnitten einen gemeinsamen Arbeitsbereich aufzubauen, der der Forderung der nahtlosen gemeinsamen Interaktion nachkommt, bedarf es bei der Entwicklung des Gesamtsystems einer Kontrolle darüber, dieser Forderung auch nachzukommen, weil die aktiven Teile des Gesamtsystems nicht miteinander verbunden sind. Diese Forderung erfüllt sich deshalb nicht zwingend in der Architektur, was in vielen vorangegangenen Arbeiten gewollt oder ungewollt zu Resultaten von Einzelplatzlösungen mit Kollaborationssubstitution führte. Fall (a) legt dem Entwickler nahe, alles als Eins zu begreifen, weil das Gesamtsystem zum größten Teil auf dem Server läuft und damit alle Systeme auf einem bündelt. Nachteil dieser Variante ist, dass es einen leistungsstarken Server geben muss und dass bei dessen Ausfall oder Ausscheiden ad hock ein adäquater Ersatz gefunden werden muss und das gesamte System und dessen Status transferiert werden müssen. Ebenfalls widerspricht Fall (a) damit dem Streben nach struktureller Gleichberechtigung, um ein harmonisches unhierarchisches Handeln der arbeitenden Personen zu gewährleisten. Letzteres Argument spricht auch gegen Fall (b). Die größere Latenz in der Reihe Eingabe-Verarbeitung-Ausgabe spricht ebenfalls gegen Fall (a) und (b). Fall (c) stellt einen Ausweg aus dem Dilemma dar, weil dieser möglichst große

strukturelle Gleichberechtigung in der Ressourcenzuweisung sowie eine adäquate Latenz repräsentiert. Außerdem gewährleistet er eine strukturelle Verbindung der Interaktion sowie mehr Flexibilität bei Änderung der Struktur.

4.3.2 Peers, Clients und Server

Stellen wir uns vor der Konkretisierung der Verteilung die Frage nach einer eventuell geeigneten Verteilungsstruktur. Die Client-Server Struktur bezeichnet wohl die einfachste und verbreiteteste. Peer to Peer bezeichnet eine immer stärker vertretene anpassungsfähige aber komplizierte Struktur. Der hier nicht eindeutig zuzuweisende serviceorientierte Ansatz SOA soll hier mit betrachtet werden, weil dieser hinsichtlich verteilter Funktionalität und unterschiedlicher Besitzer Lösungsansätze für unsere Problematik bieten könnte.

Bei der Betrachtung von Client-Server Architekturen nennt (Bengel, 2008) auf Seite 118 folgende fünf Punkte als Nachteil für die Verteilung:

1. unbalanced load distribution
2. perfomance bottleneck
3. single point of failure
4. single point of attack
5. channel bottleneck

In unserem Szenario lässt sich Punkt 4 wegen des vorausgesetzten Vertrauens vernachlässigen. Punkt 5 muss nicht beachtet werden, weil sich alle an der Interaktion beteiligten Rechner tendenziell in ein und dem selben Netz befinden und der ungehinderte Zufluss von Datenströmen aus verschieden belasteten Netzen hiermit nicht stattfinden kann. Punkt 1 bis 3 stehen für eine relevante Kritik für eine Client-Server-Lösung in unserem Szenario. Entspricht die Ressourcenverteilung nicht der Ressourcennutzung, sollte es möglich sein, die Nutzung so umzustellen, dass jeder Teil des verteilten Systems den möglichst besten Beitrag für einen „balanced distribution load“ liefert. Das „performance bottleneck“ lässt sich minimieren, indem die Segregation der Gesamtverarbeitung möglichst fein granular einstellbar ist. Dies ist bei einem gut parallelisierbaren System möglich. Diese Aufteilung lässt genügend Spielraum für eine Aufteilung der Ressourcenanforderungen. Damit wird nie ein theoretisches und damit stetiges Optimum zu finden sein, wohl aber eine Annährung an ein unstetiges Optimum. Das Problem des „single point of failure“ besteht in unserem Szenario schon gewollt im Normalbetrieb und muss nicht zum Serverausfall führen. Die geforderte Replizierbarkeit des Systemstatus aus Abschnitt 3.3 löst dieses Problem nicht nur im Fall eines Ausfalls.

Die in (Bengel, 2008) vorgeschlagene Lösung mit einer serviceorientierte Architektur (SOA) ist für uns unbrauchbar. Die Anforderungen und Funktionsteile bei unserer Problematik lassen sich schlecht in Services abbilden. Zum Beispiel wäre die Funktionalität des Orientierens eines Objektes komplex und würde eine längere Kaskade von Funktionsaufrufen nach sich ziehen.

Dabei muss auch die Latenz betrachtet werden, die in diesem Fall nicht akzeptabel wäre. Eine Anreicherung für neue verteilte Anwendungen zur Bereitstellung von Funktionalität, soll hier aber nicht ausgeschlossen werden.

Die Autonomie und Selbstorganisierung mit dynamischer Anpassung des Systems an veränderte Voraus- und Zusammensetzungen spricht für eine Lösung des Problems durch eine Peer-to-Peer-Architektur. Allerdings erhöht eine reine Peer-to-Peer Lösung zur Verteilung der Interaktionsdaten den ohnehin schon komplexen und zeitrelevanten Datenstrom. (Smith u. a., 2004) verwenden den Peer-to-Peer Ansatz zur Verteilung von Interaktionen in einem gemeinsam genutzten Arbeitsraum in der von ihnen vorgestellten Architektur „Croquet“. Wegen des hohen Komunnikationsaufwandes und der damit erhöhten Latenz stellt Peer-to-Peer keine solide und den Aufwand rechtfertigende Lösung dar.

Alle drei hier vorgestellten Ansätze bringen verschiedene Vorteile und Nachteile mit sich. Da es sich um ein komplexes System handelt, dessen Teilsysteme verschiedene Anforderungen an die Verteilung von Daten haben, können auch verschiedene Lösungen für die jeweiligen Teile des Systems gefunden werden. Es ist somit angebracht, in einem System mit hybriden Verteilungsaufgaben auch eine hybride Architektur zu nutzen.

4.3.3 „Back to the Roots“

Da ich mich in der Vergangenheit mit Robotertechnik auseinander gesetzt habe, ist es naheliegend, im Diskussionsprozess über eine adäquate Systemarchitektur für einen Interaktionslayer für Collaborative Workspaces das Wissen dieser Disziplin mit einzubringen. In diesem Abschnitt versuche ich eine Brücke zu schlagen zwischen der Denkweise in der Robotertechnik und des Softwaredesigns im Bereich Ambient Intelligence, auch wenn die Welten der Robotertechnik und Ambient Intelligence sich auf den ersten Blick gegenüber stehen.[11] So sehen wir in der Robotertechnik zumindest in der autonomen Roboterwelt den Roboter als handelndes „Subjekt“ und damit die Programmierung als weltverändernde Instanz. Im Bereich der Ambient Intelligence stellt zwar die Hard- und Software vieles bereit, aber der Mensch ist hier der handelnde und wird damit wieder zum Subjekt und das Gerät zum Objekt.[12]

Neben den Sensoren zum Erkennen der realen Welt und den Aktoren zum Manipulieren der realen Welt gibt es in der autonomen Roboterwelt zwei größere Unterscheidungen von Systemarchitekturen: die reaktive Architektur mit vielen Teilmodellen und die deliberative Architektur

[11] Es sei unbestritten, dass in der Robotik und auch im Bereich Ambient Intelligence Sensoren und Aktoren eingesetzt werden. Das Auslesen, Verteilen, Bewerten und Ansteuern bedarf sicherlich der gleichen oder ähnlicher Techniken.

[12] In wie weit das Objekt handelt und damit dem Subjekt nahe kommt muss an anderer Stelle geklärt werden. Diese Frage stellt sich natürlich auch beim Roboter. Weil der Roboter aber in Literatur und Film als Subjekt präsentiert wird und dieser in der Realität einen klar differenzierbaren Körper hat, kann die Assoziation zum Subjekt leichter geknüpft werden.

mit einem Weltmodell. Beide Ansätze arbeiten mit der subjektiven[13] Erkennung der realen Umwelt. Anhand dieser Daten wird dann die reale Welt[14] manipuliert. Der wesentliche Unterschied besteht in der internen Repräsentation der Wahrnehmung. Steht im reaktiven System im Vordergrund, schnell anhand von Submengen der Daten zu handeln, so steht im deliberativen System im Vordergrund, die erfassten Daten zu einen Gesamtbild zu formieren und anhand dieser das Verhalten zu planen. In beiden Systemen ergeben allerdings die erfassten Daten das Bild des Roboters von seiner Umwelt, wenn auch im reaktiven Fall fragmentiert und flüchtiger. Vereinfacht bleiben die drei Teile: a) Sensordaten erfassen, b) Modell oder Modelle dieser Daten erfassen und c) das daraus resultierende Verhalten mit der entsprechenden Ausgabe in der realen Welt.

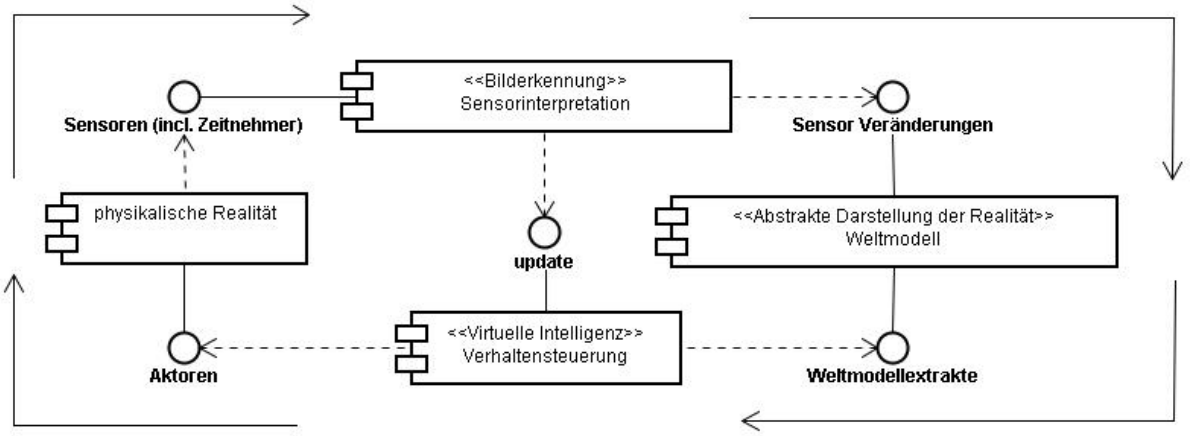

Abbildung 4.8: Verallgemeinerte Darstellung einer Architektur für Roboterprogrammierung.

Betrachten wir die verallgemeinerte Darstellung in Abbildung 4.8 für die Robotik und tauschen wir Realität und Virtualität in der Bedeutung und im Diagramm gegeneinander aus, so erhalten wir, wie Abbildung 4.9 zeigt, ein klassisches MVC-Pattern nach (Burbeck, 1992) mit dem Benutzer als Realität. Das virtuelle Weltmodell aus der Robotertechnik wird zur Realität und damit zur eigentlichen Welt. Die physikalische Realität aus der Robotertechnik wird zum Model des MVC-Pattern. Daraus folgt, dass das MVC-Pattern eine gute Struktur für einen interaktiven Zyklus zwischen Realität und Virtualität liefert.

4.3.4 Das MVC-Pattern als verteiltes MVC-Pattern

Folgen wir der Argumentation für ein MVC-Pattern aus dem vorangegangenen Abschnitt 4.3.3 und gehen in erster Linie von einem „Eingabe-Verarbeitung-Ausgabe“-System aus, liegt nahe

[13] Subjektiv deshalb, weil die Sensoren aus technischen Gründen nur eine bestimmte Auflösung, Ausrichtung und Art der Wahrnehmung haben. Damit liefern sie nur eine Untermenge der möglichen Wahrnehmungsbereiche und Dichte. Die Herkunft der Subjektivität bezieht sich beim Menschen eigentlich auf die Selektion der physischen Wahrnehmung hin zum bewussten Wahrnehmen. Da aber auch dieser Prozess zum „Ich“ gehört und auch die physischen Fähigkeiten des Menschen beschränkt sind, soll das Wort „subjektiv“ in diesem Zusammenhang in der Bedeutung gedehnt werden.

[14] Der Roboter wird wegen seiner materiellen Repräsentation unter „reale Welt“ subsumiert.

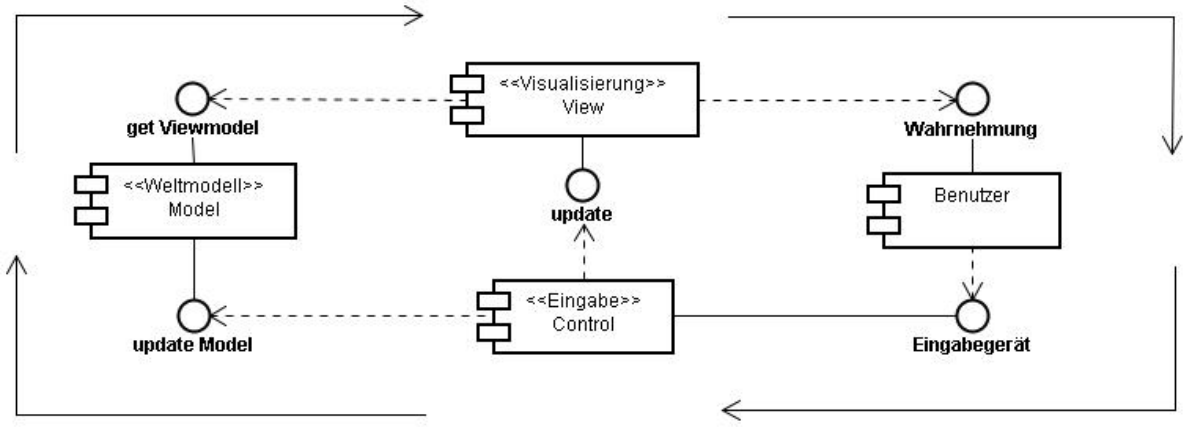

Abbildung 4.9: Allgemeine Darstellung eines Eingabe-Verarbeitung-Ausgabe Systems basierend auf dem MVC-Pattern.

bei der Entwicklung einer multiplen „grafischen-Benutzer-Schnittstelle“ das in diesem Zusammenhang häufig verwendete MVC-Pattern zu verwenden.

Das im Rahmen von GUI und verteiltem System verwendete MVP-Pattern[15] als „Weiterentwicklung“ des MVC-Pattern bringt keine besseren Lösungsansätze mit sich. Es ist zwar für verteilte Systeme konzipiert, aber die Notwendigkeit der Entkopplung von Model und View trifft in unserem Fall nicht zu. Client und Server sind in unserem Fall nicht so stark entkoppelt, dass eine reibungslose Interaktion mit GUI-Elementen möglich wäre und damit die kompensierende Kopplung über den Presenter nicht nötig ist. Viel mehr würde die Entkopplung über den Presenter zu einem höheren Kommunikationsaufwand führen.

Bei einem zwar eng gekoppelten aber verteiltem View-, Controller- und Modelsystems stellt sich die Frage nach der Kopplung der Systeme. Da aus der Analyse hervorgeht, dass es keine Grenzen zwischen den verschiedenen Views und auch keine Eingrenzung durch den jeweiligen View bei den Eingabemöglichkeiten geben soll, ist zu überlegen, ob ein MVC-Pattern überhaupt in Frage kommt. Das liegt daran, dass im ursprünglichen Fall der View und der Controller laut Definition ein uniques Gebinde bilden. Führen wir das MVC-Observer-Pattern aus (Gamma u. a., 1995) ein, ist die Kopplung von View und Control zur Aktualisierung des Views nicht mehr zwingend notwendig. Durch das Entkoppeln der View und Controllerpaare und durch eine verallgemeinerte Verbindung über den Observer und das Model ist es auch möglich, verschiedene Views mit verschiedenen Controllern zu verbinden und damit eine Variablilität einzuführen, die strukturell zu den in der Analyse geforderten Ergebnissen führen kann.

Im ursprünglichen Fall in Smalltalk sind aus technischen Gründen View und Control auch bei

[15] MVC wird häufig im Zusammenhang mit Netzanwendungen genannt. Für Webanwendungen wird es in eine Dreischichtenarchitektur mit dem Namen „Model-View-Presenter“ umgedeutet, um die Entkopplung zwischen Server und Model bzw. zwischen Client und View bzw. Control zu vollziehen. In diesen neu gedeuteten Mustern hat der View keine Verbindung mehr zum Model und der Controller oder hier Presenter leitet die darzustellenden Informationen vom Model zum View weiter.

Einführung des Observers weiter gebunden, weil die visuelle Ausgabe der Eingabeanzeige kein Bestandteil des Models ist, sondern des Views bzw. Controllers. Da es sich bei dem zu entwickelnden System um ein Interaktionsystem handelt und damit die Eingabeanzeige zum inhaltlichen Zusammenhang des Systems gehört, ist es naheliegend, die Eingabeanzeige auch im Model zu repräsentieren und ein stellvertretendes Element für die Eingabe in ihm zu verankern. Durch die Eingliederung der Eingabevisualisierung und über den Observer lässt sich der View unabhängig vom Controller steuern. Mit dieser Unabhängigkeit lassen sich View und Controller entkoppeln und trotzdem indirekt über das Model und den Observer verbinden, ohne das Model in Abhängigkeit von Controller oder View zu bringen.

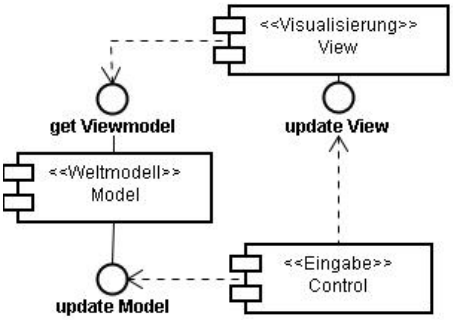

Abbildung 4.10: Das MVC-Pattern.

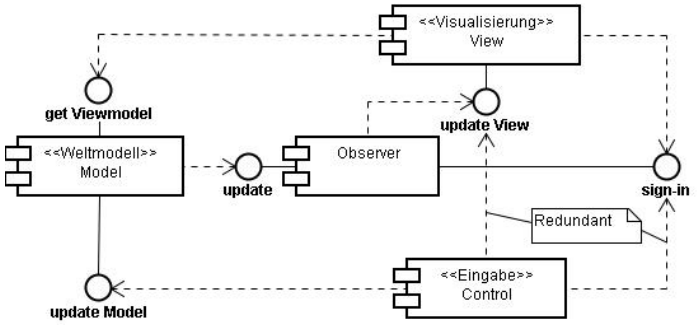

Abbildung 4.11: Das MVC-Observer-Pattern.

Die Entkopplung der Viewkomponte von der Controllerkomponente macht es möglich, bei starker Heterogenität innerhalb des Gesamtsystems, die Anzeige- und Eingabeelemente der Einzelsysteme unabhängig anzupassen. Die Skalierung und andere Eigenschaften werden dadurch unabhängig, weil die Manipulation erst innerhalb des Modells geschieht, in dem der Systemkontext neutralisiert wurde.

Die Viewkomponente visualisiert abhängig von der Perspektive das Modell. Der Controller steuert den Ablauf und generiert eine generalisierte Eingabe, die damit Bestandteil des Modells werden soll. Das Modell soll sich bei Veränderungen selbst konsistent halten.

4.3.5 Vorgabe zur Evaluierbarkeit

Für die Evaluierbarkeit von „Nützlichkeit und Verwendbarkeit der Softwarewerkzeuge im Arbeitsprozess“ gibt Jörg Raasch in (Raasch, 2006) eine Architekturnotwendigkeit an. Diese besteht aus einer Funktionskomponente, die die funktionale Ebene implementiert und einer Interaktionskomponente, die die Gestaltebene implementiert. Diese Trennung in Interaktion und Funktionalität hat sich in der Softwarearchitektur (Züllighoven u. a., 2004) bereits etabliert.

Eine Aufgabenstellung zur Evaluation innerhalb einer computergestützten Gruppenarbeitsumgebung stellen (Tan u. a., 2005) mit dem „collaborative jobshop scheduling“ vor. Ersteres gibt die Struktur vor und letzteres die Testfälle.

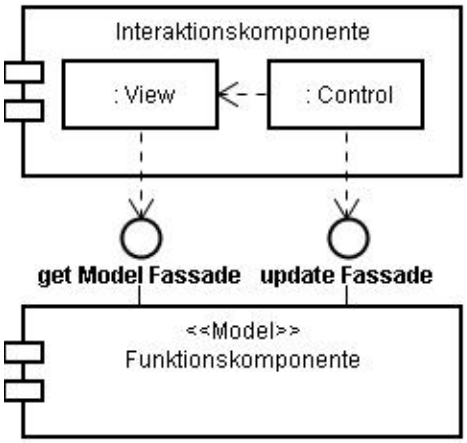

Abbildung 4.12: Einbettung des MVC-Pattern in die Vorgabe zur Evaluierbarkeit.

4.3.6 Vorbildmetapher als Strukturgeber

Das Clearboard aus (Ishii und Kobayashi, 1992) ist einer der ersten Prototypen zur Demonstration von nahtlosem gemeinsamem Zeichnen auf einer gemeinsamen Arbeitsfläche in Echtzeit mit Augenkontakt. Der Aufbau soll in diesem Abschnitt kurz betrachtet werden und als Vorbildmetapher für ein strukturelles Modell dienen. Abbildung 4.13 zeigt den Aufbau des Prototypen I des Clearboards.

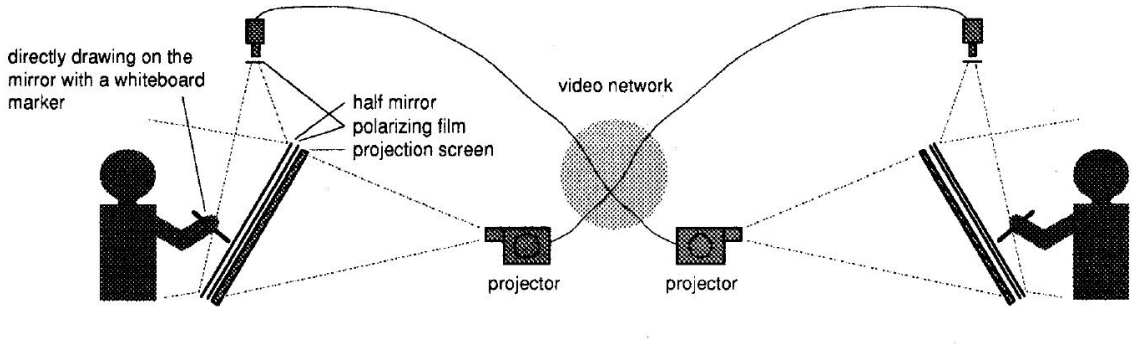

Abbildung 4.13: Das gezeichnete der linken Person repräsentiert sich zusammen mit dem gezeichneten der rechten Person durch Überlagerung von Projektion und Zeichnung im Display und umgekehrt. Clearboard aus (Ishii und Kobayashi, 1992)

Das Clearboard besteht aus zwei Glasscheiben, auf denen jeweils das Bild der anderen Glasscheibe projiziert wird, so dass beide Arbeitsergebnisse sich überlagern und als eins erscheinen. Jede Person kann auf die gesamte gemeinsame Arbeitsfläche schauen und ebenfalls auf der gesamten gemeinsamen Arbeitsfläche alles hinzufügen was sie möchte. Gleichzeitig können beide Personen Face to Face miteinander kommunizieren.

Übersetzen wir diese Anordnung in ein strukturelles Modell, so folgt daraus, dass es eine ge-

meinsame Repräsentation der Arbeitsflächen geben soll. Diese beinhaltet gestreamt oder orginal die Inhalte aller beteiligten Systeme. Trotzdem kann jeder Inhalt auf der jeweiligen Seite verändert und eingesehen werden. Der Sicht- und Wirkungsbereich erzeugt keine privaten Bereiche. Es gibt Arbeitsgeräte, die Personen zugeordnet sind. Diese Vorgaben sollen als Hinweis für die Struktur einer Architektur dienen.

4.4 Architektur der Interaktionsschicht

In diesem Abschnitt wird nun ein konkreter Architekturvorschlag in Form eines Komponentendiagrammes entwickelt, in dem ein einfacher Vorschlag kritisch betrachtet und dann in weiteren Schritten verändert, erweitert und aggregiert wird. Die Komponenten des aggregierten Ergebnisses werden daraufhin beschrieben und durch Aktivitätsdiagramme dargestellt. Die Schnittstellen der minimalen Architektur werden im Anschluss beschrieben. Um zu überprüfen, ob das Zusammenspiel der Komponenten aus der minimalen Architektur mit den vorgestellten Definitionen und Verhaltensweisen funktioniert, wird dieses mit logischen Testfällen über ein Sequenzdiagramm verifiziert. Zum Ende wird die vorgeschlagene Minimalarchitektur für die restlichen Anforderungen erweitert, so dass eine Gesamtsystemarchitektur entsteht. Diese wird dann bezüglich drei verschiedener kontextbezogenen Architekturen diskutiert.

4.4.1 Konkrete Architekturdiskussion zu Interaktion und Verteilung

In Abschnitt 4.3 wird nahe gelegt, dass die Grundstruktur des Systems eine dynamische Client-Server-Struktur bildet. Diese Struktur beinhaltet im beginnenden Verhandlungsmodus Peers und im produktiven Interaktionsmodus Clients und Server. Für die Ebene des Kontenttransfers handelt es sich wiederum um eine Client-Server Beziehung, die durch den Server aus der Interaktionsschicht vermittelt wird.

Einfache Struktur

Formen wir zunächst, wie in Abbildung 4.14 zu sehen, eine einfache Struktur nach den vorangegangenen Forderungen bezogen auf das Gesamtsystem. Der Forderung aus Abschnitt 4.3.4, MVC-Muster für die Interaktion zu verwenden, gehen wir auf den lokalen Client-Systemen nach, weil sich hier konkret die Ein- und Ausgaben befinden.

Um der Forderung gerecht zu werden, dass Clients auch Server sein können, versuchen wir, die Struktur in den Clients und im Server ähnlich zu halten, um mögliche Überschneidungspunkte zu ermöglichen. Diese können später beim Wechsel zwischen Client und Server genutzt werden. Aus diesem Vorgehen folgt, dass der Server ebenfalls mit einem MVC-Muster realisiert wird. Das angewendete MVC-Muster bildet beim Server nicht die Verbindung zum realen Benutzer über die Control- und View-Komponente, sondern eine separierte View- und Control-Schnittstelle für die Clients. So kann über die Schnittstellen der jeweilige View auf das

Model an den anfragenden Client und der Eingabefluss für das Welt-Modell im Server angepasst werden.

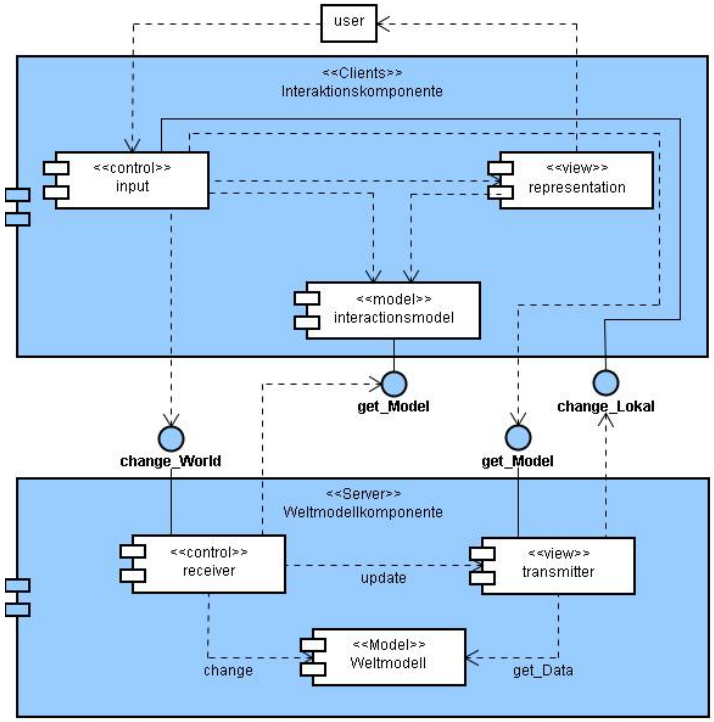

Abbildung 4.14: Einfache Architektur nach MVC-Pattern im Server und in den Clients.

Ein Modell ebenfalls im Client zu halten bedeutet zwar eine gewisse Redundanz, verringert aber den Kommunikationsaufwand und beschleunigt den lokalen unbearbeiteten einfließenden Eingabestrom sowie den lokalen zu bearbeitenden Ausgabestrom. Der Client-Controller triggert bei Aktivität nicht nur den Client-View sondern auch den Server-Controller, der wiederum die lokalen Veränderungen aus dem Clientmodell anfordert. Nachdem die Veränderungen in das Servermodell eingepflegt wurden, wird die Veränderung im Weltmodell an alle betroffenen Clients und deren lokale Modelle über den Server-Transmitter und Client-Controller weitergeleitet. Dieses Kaskadieren garantiert eine minimale Latenz zwischen Ein- und Ausgabe mit Bereitstellung eines minimalen Kommunikationsaufwandes.

Einfügen einer Distributionskomponente

Dieser erste Ansatz verbindet die beiden Komponenten über vier Schnittstellen, die zusammen den Ablauf der Komponenten und des Gesamtsystems bestimmen. Drei Verbindungen bestehen zum Controller des Clients, der damit die Koordination clientseitig übernimmt. Der View des Servers muss die Koordination des Gesamtsystems übernehmen. Um die direkte Verbindung zwischen Server und Client zu trennen und um die Komplexitiät der Verteilung aus den Komponenten innerhalb der Interaktionskomponente und der Weltmodellkomponente zu nehmen, fügen wir, wie in Abbildung 4.15 gezeigt, die Verbindungskomponente „Distribution und

Synchronisation“ zwischen Client und Server ein. Um die Schnittstellenanzahl zu verringern und der Client/Controllerkomponente Unabhängigkeit in Bezug auf die Verteilung zu gewähren, fügen wir clientseitig einen Observer ein.

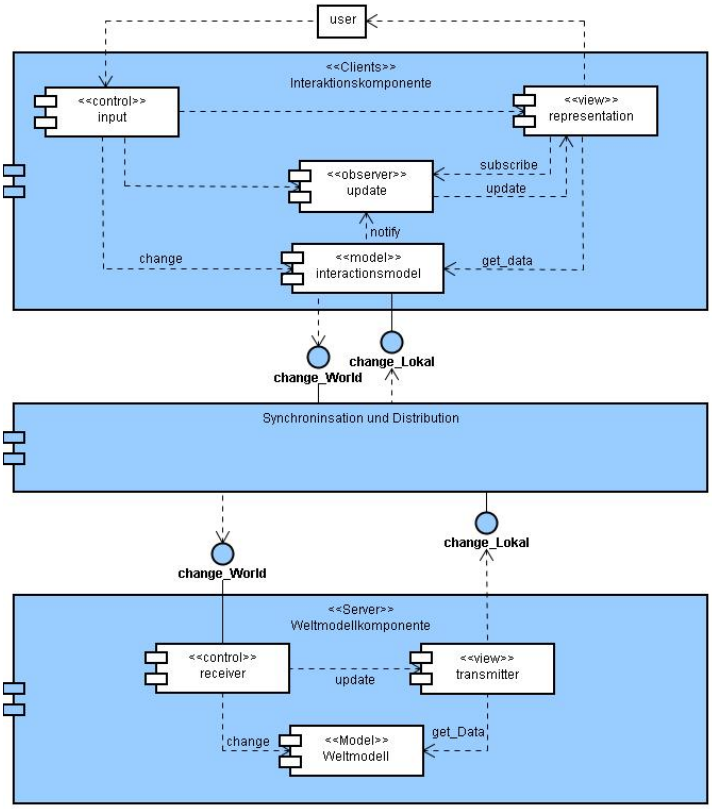

Abbildung 4.15: Systemarchitektur mit Ausgliederung der Distribution und notwendiger Erweiterung durch das Observer-Pattern.

Weil der Eingabe-Controller in der Interaktionskomponente aus Abbildung 4.14 die externen Änderungen der Weltmodellkomponente koordiniert hat, war es nicht nötig bei externen Modellveränderungen eine Notifikation des Views durch das Modell einzubringen. Dieses Vorgehen erforderte eine Schnittstelle zum Eingabe-Controller und das Nutzen zweier Schnittstellen vom Eingabe-Controller selbst. Durch Einführung eines Observers kann indirekt der Repräsentations-View angstoßen werden. Damit ist ein Management der externen Modellveränderungen durch den Eingabe-Controller nicht mehr notwendig. Das Interaktionsmodell kann nun die erforderlichen Maßnahmen zur Erneuerung der Repräsentation oder des Weltmodells einleiten. Dadurch verringern sich die Schnittstellen der Interaktionskomponente auf eine ausgehende und eine eingehende, wenn die entsprechenden Daten über die change-Schnittstellen gesendet werden.

Die neue Verbindungskomponente „Distribution und Synchronisation“ lokalisiert sich physikalisch auf Server und Client und macht das Gesamtsystem für die Interaktions- und Weltmodellkomponente bezogen auf die Verteilung transparent. Diese Komponente enthält neben den Verteilungselementen zusätzlich einen Verhandlungsteil, der das gegenseitige finden und die

Rollenverteilung der beteiligten Einzelsysteme implementieren soll. Auch das Ausscheiden eines Einzelsystems aus dem Gesamtsystem innerhalb der Verbindungskomonente ist mit zu beachten. Serverseitig kann die Schnittstelle „change World“ wegen der 1 zu N Beziehung einfach durch die Verbindungskomponente durchgeschleift werden, wenn die Transfersprache von Client und Server gleich ist. Ist sie nicht gleich, muss der Datenstrom clientseitig in ein für den Server verständliches Format konvertiert werden. Dadurch erweitert sich die Verbindungskomponente serverseitig nur um den Verhandlungsteil.

Angleichen und Formen einer Submenge

Um den Server zur Submenge vom Client zu machen, verschieben wir die Teilkomponenten aus der „Distribution und Synchronisation“-Komponente wieder in die Interaktions- und Weltmodellkomponente. Damit erweitert sich die Interaktionskomponente um einen Transmitter, Receiver und den Distributionsscout. Die Weltmodellkomponente erweitert sich um den Distributionsscout. Ebenfalls muss nun der serverseitige Transmitter die Übertragung an die Clients komplett übernehmen. Es gibt nun einen distributionsunabhängigen Teil und und einen distributionsabhängigen Teil innerhalb der Interaktionskomponente und der Weltmodellkomponente.

In Abbildung 4.16 zeigt sich eine gespiegelte Submenge der Komponenten innerhalb des Servers und des Clients. Die scheinbare Disharmonie der „update“ Abhängigkeit zwischen Receiver und Transmitter innerhalb der beiden Mengen bestimmt erstmal eine Ungleiche, lässt sich aber über Einfügen eines Observers serverseitig auflösen.

Wir haben nun ein klassisches Client-Server-System, dass Clients aufnehmen und entlassen kann und die Synchronisation sicher stellt. Der zu transportierende Kontent der Objekte sollte nicht, kann aber über den Synchronisationsmechanismus „change World“ abgebildet werden. Eine parallele Schnittstelle zur „change World“ Schnittstelle sollte aus Performance- und Managementgründen implementiert werden. Dies wird hier nicht explizit im Design eingeführt, um das Design so einfach wie möglich zu halten.

Falten der Architektur

Die Forderung nach einer dynamischen Client-Server-Struktur aus Abschnitt 4.3 ist bis hier noch nicht integriert. Weil die Grundstruktur des Servers, wie in Abbildung 4.16 gezeigt, eine Untermenge des Clients ist und die Schnittstellen symmetrisch angeordnet sind, können wir nun den Server dem Client strukurell anpassen und die zweigeteilte Architektur zusammenfalten, so dass diese nur noch aus einer übergeordneten Komponente für Server und Client besteht. (Abbildung 4.17)

Aufgrund der Symmetrie der Schnittstelle „change World“ wird diese zu einer Schnittstelle zwischen jeweils einem potentiellen Client und einem potentiellen Server. Die unterschiedliche

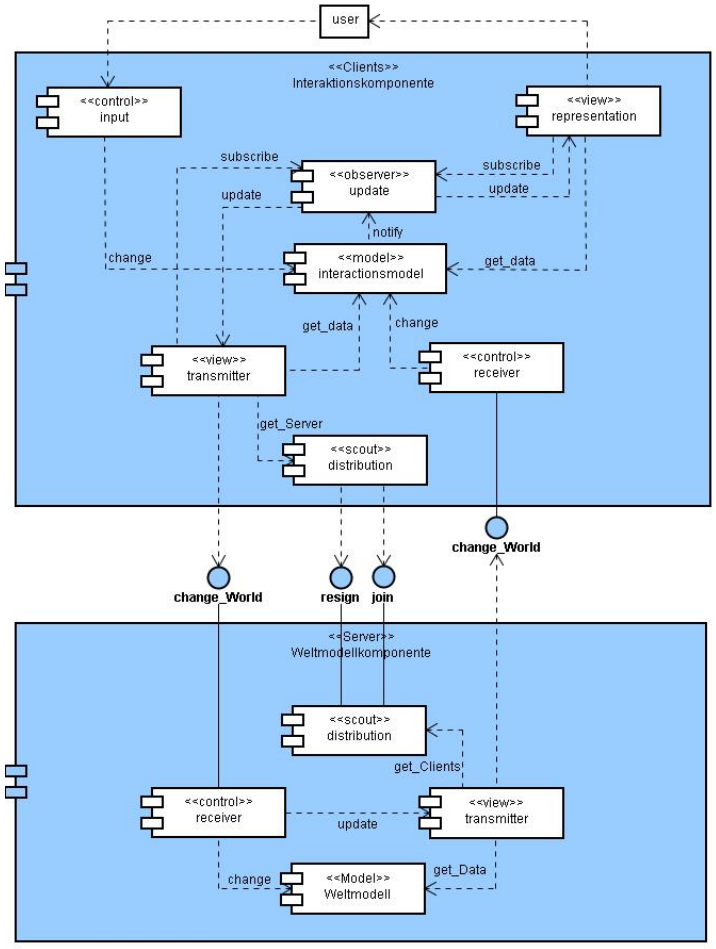

Abbildung 4.16: Entwicklungsschritt der Architektur mit unabhängigem Modell und angegliche-
ner Client-Server- Distribution.

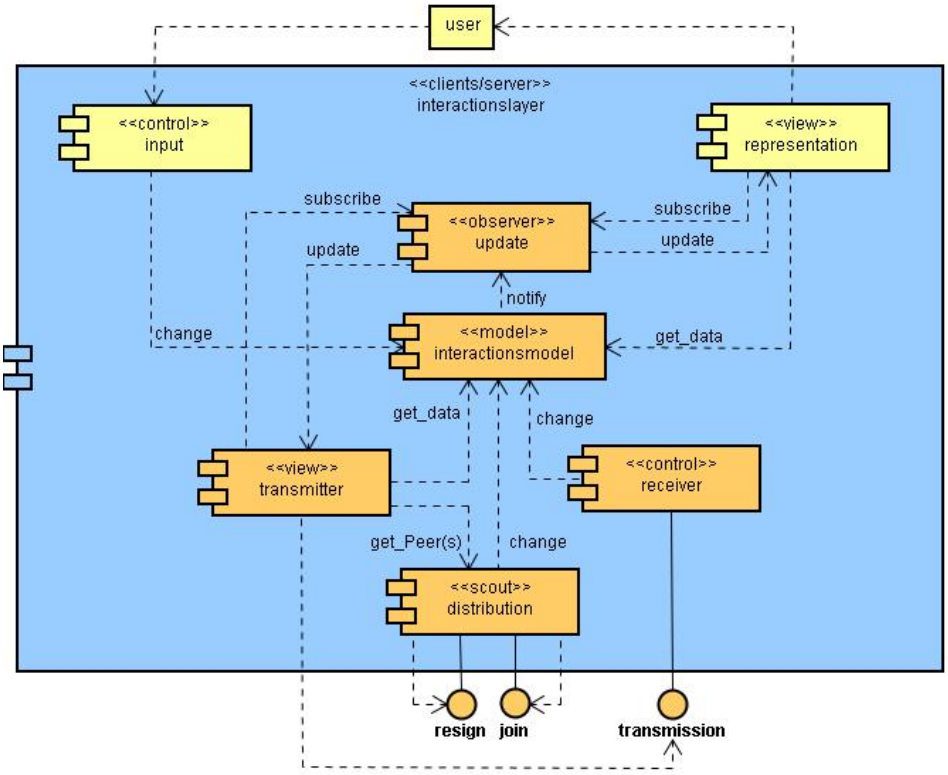

Abbildung 4.17: Einheitliche Architektur, in der Server und Client gleich gesetzt sind mit Unterscheidung durch die variablen Komponenten Eingaben, Repräsentation und Interaktionsmodell.

Ausrichtung ergibt sich aus den unterschiedlichen Inhalten der übertragenen Daten. Der Transmitter des Servers übermittelt Daten über die „change World“ Schnittstelle, die dem jeweiligen Client zuzuordnen sind und die Clients übermitteln über diese Schnittstelle die Daten, die dem Server zugeordnet werden.

Bevor sich eine Instanz dazu entscheidet, eine Server oder Client Erscheinung anzunehmen, kann über die „join“ Schnittstelle im undefinierten Modus über die Rollenverteilung verhandelt werden. Dies kann explizit zwischen allen gleichzeitig geschehen. Es wäre eine dynamische Variante der Verhandlung, wenn die Verhandlung nur zwischen Zweien geschieht. Diese schließen sich zu einer Gruppe zusammen. Diese Gruppe tritt dann über den nun entstandenen Server in neue Verhandlungen mit den restlichen zu integrierenden Einzelsystemen. Hierbei wird das Wissen über das Gesamtsystem immer beim Server gehalten, so dass dieser als delegierter für seine Clients mit entscheidet, wie bei Eintritt eines Einzelsystems in das Gesamtsystem die Rollen verteilt werden. Dieses Prinzip kann auch für den Austritt gelten, so dass der Server bei Austritt den nachfolgenden Server bestimmt. Fällt der Server aber unerwartet aus, müssen sich die Clients von Anfang an neu orientieren.

Um in der Startphase ein ständiges Server-Client-Wechselspiel zu vermeiden, sollte sich ein entstandener Server vor der Metamorphose vergewissern, ob noch andere Einzelsysteme integriert werden sollen. Erst nach Absichern der Vollständigkeit des Gesamtsystems sollen alle Systeme in den vorübergehend festen Status Server oder Client übergehen.

4.4.2 Komponenten

Um neben der Existenz der Komponenten und deren Zusammenspiel auch die Arbeitsweise dieser zu erläutern und zu definieren, werden im folgenden alle sieben Komponenten anhand einer Beschreibung und eines Diagrams veranschaulicht.

Eingabekomponente

Die Komponente für Benutzereingaben kann sowohl im Server-Mode als auch im Client-Mode aktiv sein. Soll der Server nicht direkt an der Interaktion beteiligt sein und stellt er nur die Rechenressourcen zur Verfügung, so kann diese Komponente abgeschaltet werden.

Die Eingabekomponente fasst jeweils alle an einen Rechner angeschlossenen Eingabeinstrumente zusammen und verbindet diese im einfachsten Fall zu einer Einheit, damit verschiedene Formen der Eingabe einem einzigen User zugeordnet werden können. Bei komplexen Eingabeeinstrumenten wie z.B. Multitouch-Eingaben oder Gestenerkennung[16], bei der verschiedene Anwender einem Eingabeinstrument zugeordnet werden können, müssen über eine Spezialbehandlung verschiedene Eingabeeinheiten generiert werden, damit diese dann den einzelnen Benutzern zugeordnet werden können. Um die anfängliche Komplexität gering zu halten,

[16] Ein Bericht und eine Analyse von „Gesteninteraktionsforschung“ findet sich unter (Karam, 2006)

gehen wir in dem in Abbildung 4.18 gezeigten Diagramm davon aus, dass alle aktiven Eingabeinstrumente beim Start des System diesem bekannt bzw. für dieses erkennbar sind.

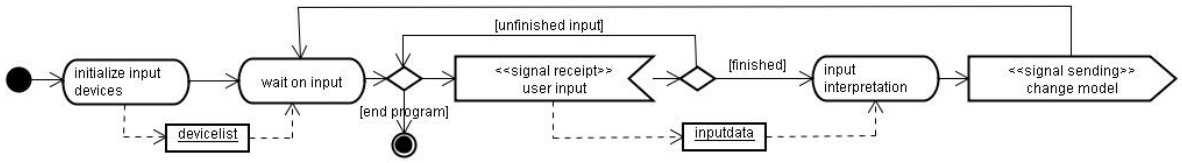

Abbildung 4.18: Eingabekomponente.

Wird eine Eingabe von den definierten Eingabeinstrumenten gesendet, so wird diese systemspezifisch interpretiert und an das Interaktionsmodell gesendet. Handelt es sich um einfache Eingaben wie z.B. eine Tastatureingabe, so fällt die Aufnahme und Interpretation der Eingabe einfach aus. Eine Folge von Eingaben wie z.B. ein Doppelklick der Maus bedarf einer komplexeren Aufnahme der Eingabe. Durch den in Abbildung 4.18 gezeigten Ablauf ist es ebenfalls möglich, innerhalb der Eingabekomponente auch sehr komplexe Eingaben wie Gesten oder Spracheingaben abzubilden. Werden komplexe Eingaben von einer externen Anwendung geliefert, so wird die Eingabe als fertige Eingabe entgegengenommen und nicht interpretiert, sondern in ein systemkonformes Format übersetzt.

Um eine spontanes Programmende während einer komplexen Eingabe zuzulassen, wird das Programmende in dieser Komponente als Eingabe interpretiert. Bei Programmende wird die begonnene Eingabe verworfen und die Komponente nach Schließung aller offenen Ressourcen beendet.

Repäsentationskomponente

Nach der Initialisierung, die unter anderem die Möglichkeiten des Einzelsystems zur Repräsentation festlegt, schreibt sich die Repräsentationskomponente beim Observer ein, um bei Änderung des Modells benachrichtigt zu werden. Danach fordert die Komponente im Umfang seiner Darstellungsmöglichkeiten die Daten erstmals von der Interaktionmodellkomponente an. Diese Daten werden nun von der Komponente dem Benutzer angezeigt. Die Komponente kann ab hier, wie in Abbildung 4.19 gezeigt, normal abgebrochen werden.

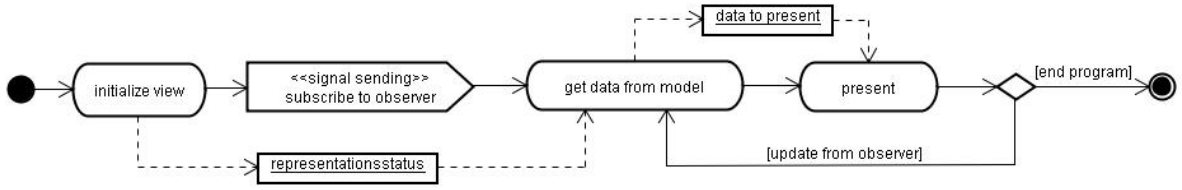

Abbildung 4.19: Repäsentationskomponente.

Wird die Repräsentationskomponente von der Observerkomponente bei Änderung des Modells getriggert, so fordert die Komponente erneut die aktuellen Daten zur Repräsentation an.

Diese Daten werden dann genau wie beim ersten Durchlauf dem Benutzer angezeigt. Der letzte Durchlauf sollte bei visuellen Bewegungen innerhalb der Anzeige mindestens 30 mal pro Sekunde durchlaufen werden, um visuell flüssig zu wirken.

Updatekomponente

Die Updatekomponente besteht aus zwei Teilen. Der erste Teil behandelt das Eintragen und der zweite Teil das Benachrichtigen der Subscriber. Die Komponente braucht mindestens eine Subskription, wie in Abbildung 4.20 gezeigt, um überhaupt in einen sinnvollen Zustand zu kommen. Die Existenz eines ersten Subscribers ist dadurch gesichert, dass jedes Einzelsystem mindestens einen Transmitter beinhaltet. Selbst im unwahrscheinlichen Fall, dass nur ein Einzelsystem gestartet wird, ohne es mit anderen verbinden zu wollen, braucht es mindestens eine Anzeige und damit einen Subscriber, um eine sinnvolle Arbeitsumgebung zu schaffen.

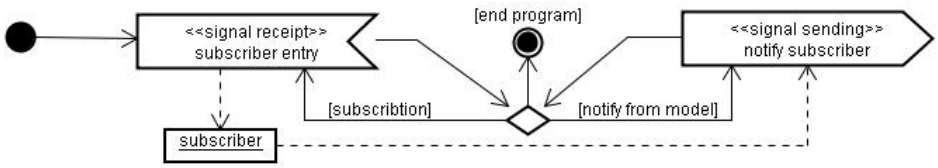

Abbildung 4.20: Updatekomponente.

Interaktionsmodellkomponente

Jedes Einzelsystem benötigt wenn es Server oder Client wird ein rudimentäres Modell von sich selbst. Im Falle der Serverwerdung werden die Modelle der Clients mit dem eigenen Modell verbunden, so dass ein Gesamtmodell entsteht. Dieses entstandene Modell inklusive der Kalkulationsregeln ergeben das Interaktionsmodell des Gesamtsystems.

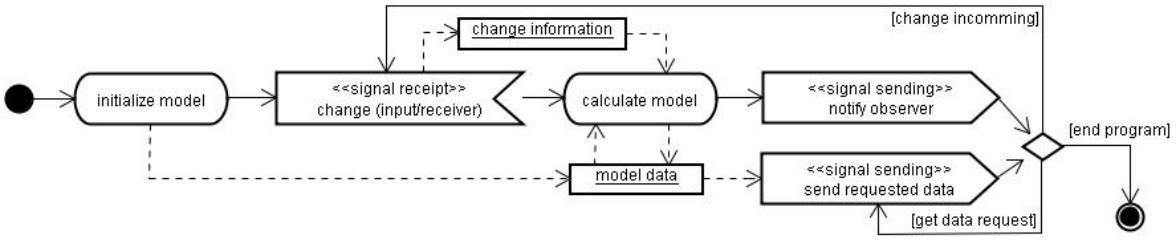

Abbildung 4.21: Interaktionsmodellkomponente.

Wenden wir den Ablauf aus Abbildung 4.21 für Client und Server gleich an, entsteht nach der Initialisierung eine Deadlock-Situation, da sowohl Client als auch Server auf eine externe „change“-Nachricht warten. Aus diesem Grund muss im Servermode nach der Initialisierung

direkt zur Kalkulation gesprungen werden. Weil der Server durch die Aushandlungen der Distributionskomponente die rudimentären Modelle der Clients mit in sein Modell integriert hat, kann nun der Server das Gesamtmodell an alle Clients verteilen.

Die Clients warten nach der Initialisierung auf die initiale Verteilung des Gesamtsystems durch den Server. Zu beachten ist hierbei die Reihenfolge der zu startenden Komponenten. Die Eingabekomponente darf aus diesem Grund erst nach der Interaktionsmodellkomponente gestartet werden, um das Zusammenspiel der Komponenten in einem definierten Zustand zu halten. Andere Abläufe können ebenfalls unproblematisch sein, aber sind hier nicht explizit definiert.

Transmitterkomponente

Die Transmitterkomponente muss sich zu Beginn wie die Repräsentationskomponente beim Observer einschreiben, um bei Änderungen am Modell durch eine Benutzereingabe oder eine externe Modelländerung vom Observer angestoßen werden zu können.

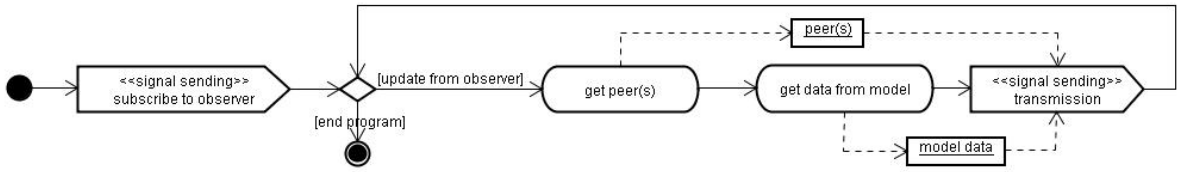

Abbildung 4.22: Transmitterkomponente.

Wird die Transmitterkomponente vom Observer angestoßen, fordert sie von der Distributionskomponente die Kommunikationspartner und die zu kommunizierenden Daten von der Interaktionskomponente an. Die Daten werden am Ende des Zyklus an die Partner verteilt.

Der Ablauf ist für Server und Client gleich. Unterschiedlich sind nur die Daten und die Anzahl der Partner. Dies muß bei der Transmission gegebenfalls mit betrachtet werden. Den Unterschied der Daten entscheidet hier nicht der Transmitter, sondern das Daten generierende Modell. Soll der Server z.B. aus Gründen der Datenreduktion unterschiedliche Daten an die jeweiligen Clients liefern, so muss der in Abbildung 4.22 gezeigte Ablauf in sofern geändert werden, dass die Datenerhebung aus dem Modell mit folgender Verteilung über alle Peers einzeln abgearbeitet wird.

Receiverkomponente

Die Receiverkomponente in Abbildung 4.23 hat die einfachste Ablaufstruktur. Die eigentliche Aufgabe, eine eingehende Modelländerung an das Modell weiterzuleiten, wird hier nur dadurch angereichert, dass die Nachricht gegebenenfalls noch aufbereitet werden muss. Die Aufbereitung ist nicht zwingend, sondern beschreibt nur die Möglichkeit, einen abstrakten Datenfluss zwischen den Einzelsystemen zu gewährleisten.

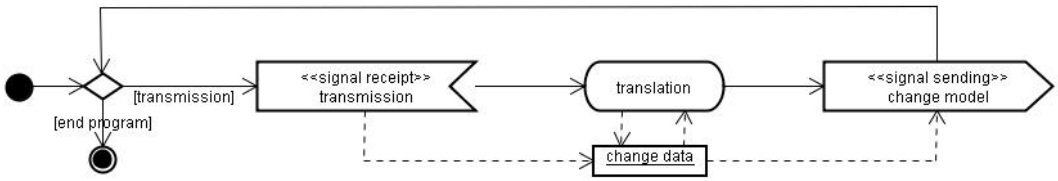

Abbildung 4.23: Receiverkomponente.

Die Übersetzung darf nicht dazu genutzt werden, eine Anpassung von clientseitig verschiedenen Formaten zu kompensieren. Weil jeder Client auch Server sein kann, würde es dazu führen, dass jedes Einzelsystem alle Darstellungsformate unterstützen muss. Beim Abweichen vom generellen Darstellungsformat muss der Client die Anpassung im Transmitter und Receiver bzw. im Interaktionsmodell gewährleisten. So müssen Sonderfälle nur auf dem jeweils abweichenden System realisiert werden.

Distributionskomponente

Die Distributionskomponente ist die erste Komponente, die nach der Interaktionskomponente angestoßen werden muss, weil sie die Grundlage der Interaktionsebene für das Interaktionsmodell generiert. Die Distributionskomponente beginnt mit dem Finden eines zweiten Einzelsystems. Es wird zuerst versucht, einen Server zu finden, weil dies bei einem Gesamtsystem ab drei Rechnern der häufigste Fall ist. Findet sich kein Server, muss versucht werden, ein anderes undefiniertes Einzelsystem zu finden, um zu verhindern, dass zwei Server gleichzeitig gestartet werden. Findet sich kein weiteres Einzelsystem, muss mangels Partner der Serverstatus ohne Verhandlung übernommen werden. Nicht in Abbildung 4.24 eingezeichnet ist ein Überprüfen der Konsistenz des Gesamtsystems für den Fehlerfall, dass zwei Server initialisiert worden sind. Da davon ausgegangen werden muss, dass in der Anordnung innerhalb des Gesamtsystems unvorteilhafte Hierarchien oder Lücken während der Benutzung entstehen, muss eine immer wieder kehrende übergeordnete Überprüfung, die gegebenenfalls eine Neuordnung veranlasst, zusätzlich noch eingefügt werden.

Wird das Gesamtsystem gestartet, startet das erste Einzelsystem meist ohne vorhandene Peers und übernimmt den Serverstatus, weil die Einzelsysteme nicht gleichzeitig unter gleichen Konditionen in den Raum kommen oder gestartet werden. Im Fall eines speziellen Raumes, wie das Szenario aus Abschnitt 3.2 ihn beschreibt, ist eine Powerwall, ein Tabletop oder ein explizit unterstützender separater Server in diesem Raum vorhanden auf denen der Interaktionslayer ausgeführt wird. Damit wäre für den Anfang die Rollenverteilung zum Startpunkt schon geklärt. Im schlechtesten Fall müsste die Interaktionskomponente als Server für n Geräte n mal initialisiert werden. Dies geschieht jedes mal auf einem anderen Rechner. Im schlechtesten Fall würde Client und Server 2n - 1 mal initialisiert werden. Im besten Fall n mal.

Finden sich zwei Peers, berechnen diese anhand der übergebenen Einzelsysteminformationen welcher von beiden die Rolle des Servers und welcher die Rolle des Clients annehmen soll.

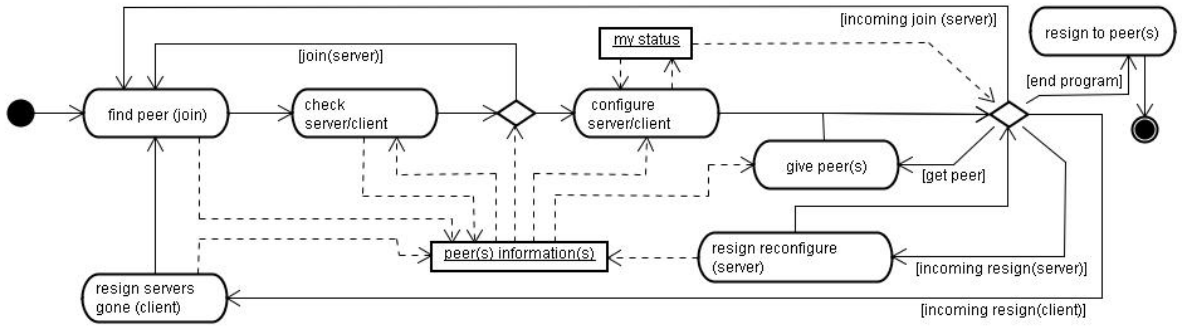

Abbildung 4.24: Distributionskomponente.

Weil diese Berechnung zu einem deterministischen Ergebnis führt, bedarf es keiner weiteren Kommunikation in dieser Phase. Das Ergebnis wird neben den anderen Informationen über die beteiligten Systeme abgelegt. Finden sich noch andere Einzelsysteme, so übernimmt der resultierende Server die Beitrittskommunikation.

Nach dem Schritt der Aushandlung folgt die Konfiguration des eigenen lokalen Systems. Diese wird anhand des eigenen vorhandenen Status und des errechneten Zielstatus ausgeführt. Bei unterschiedlichen Status wird der errechnete Status konfiguriert. Bei gleichen Status wird clientseitig nicht neu konfiguriert. Serverseitig wird die Information des neuen Client in die vorhandene Konfiguration eingefügt.

Die Distributionskomponente befindet sich nach der Konfiguration im Use-Mode. Hier werden die für die Verteilung notwendigen Peerinfomationen von den anderen Komponenten abgefragt. Tritt das jeweilige Einzelsystem aus dem Gesamtsystem aus, reicht ein Resign-Signal zu allen bei sich eingetragenen Peers. Tritt ein System aus, unterscheiden wir zwei Fälle. Serverseitig wird das austretende System aus der Peerliste entfernt. Clientseitig verliert der Client seinen Server sowie alle anderen Clients auch und ein neuer Server muss unter den Clients ausgehandelt werden. In diesem Fall macht das Aushandeln vor der Konfiguration einen signifikanten Unterschied, weil alle Systeme fast gleichzeitig in die Verhandlung eintreten.

4.4.3 Schnittstellen

In diesem Abschnitt werden die „externen“ Schnittstellen, „resign“, „join“ und „transmission“ des Interaktionslayers beschrieben. Es handelt sich zwar um externe Schnittstellen, weil sie aber nur vom Interaktionslayer selbst genutzt werden, sind sie eher als Distributionsschnittstellen zu bezeichnen. Eine umfassende Darstellung der Distribution des Interaktionslayers in dieser Stufe des Entwicklungsprozesses aufzuzeigen, würde eine zu starre Kommunikationsvorgabe für die ersten Entwicklungsschritte zwischen den Teilen des Interaktionslayers bedeuten. Deshalb werden hier nur die Rahmenbedingungen kurz beschrieben. Eine erste Ausführung und Leitlinie für spätere Entwicklungsschritte findet sich im Kapitel 5 „Realisierung“.

Die Schnittstellen zu externen Komponenten werden hier nicht betrachtet, weil diese erst in späteren Entwicklungsschritten eingeführt werden. Diese späteren Schritte beruhen dann auf dem erweiterten Design des Interaktionslayer, welches in Abschnitt 4.5 dargestellt wird.

„Join“-Schnittstelle

Die „join“-Schnittstelle dient in erster Linie dem Finden von Kommunikationspartnern. Da es zum Zeitpunkt des „join“ noch keine gefundene Verbindung zwischen den Kommunikationspartnern gibt, muss die „join“-Schnittstelle offen und überwacht sein. Diese Schnittstelle erlaubt es anderen Teilnehmern auch ohne das Wissen über deren Existenz, eine mögliche Verbindung durch Ausprobieren zu etablieren. Der Mechanismus ähnelt dem „anpingen“[17] mit entsprechendem Echo.

Nachdem Etablieren einer Verbindung überträgt sie neben den Kommunikationsdaten auch die Initialdaten und damit die statischen Daten des Clientsystems zum Serversystem. Aus den Initialdaten wird die unterste Interaktionsebene, siehe Abbildung 4.6 in Abschnitt 4.2.5, konstruiert.

„Resign“-Schnittstelle

Die „resign“-Schnittstelle dient nur dem Austreten aus dem System. Dafür muss nur die eigene Kommunikationsverbindung über diese Schnittstelle gesendet werden. Bei Austritt eines Clients stehen dem Server intern die zur Kommuniktationverbindung assoziierten Daten zur Verfügung. Bei Austritt des Servers muss, wie in Abschnitt 4.4.2 beschrieben, das Gesamtsystem neu aufgebaut werden. Bei dieser einfachen Nutzung der „resign“-Schnittstelle könnte diese auch von der „join“-Schnittstelle übernommen werden. Dies könnte mit einem negativen „Ping“ realisiert werden.

Eine Alternative zum Neuaufbau wäre, wenn der ausscheidende Server anhand der deterministischen Berechnungsgrundlage aus Abschnitt 4.4.2 einen neuen Server aus seiner Clientliste bestimmt und diesem dann die gesamten Interaktionsmodelldaten über die „resign“- oder „transmission“-Schnittstelle sendet. Hierfür müsste im Komponenten-Diagramm in Abbildung 4.17 aus Abschnitt 4.4.1 gegebenenfalls eine neue Verbindung zwischen Distribution und Transmitter eingeführt werden oder die Verbindung zwischen Distribution und Interaktionsmodell ebenfalls dazu benutzt werden um das gesamte Modell anzufordern.

[17] In IP-Netzwerken kann mit dem Diagnose-Werkzeug „Ping“ überprüft werden, ob ein bestimmter Host erreichbar ist. Zu diesem Zweck sendet „Ping“ ein ICMP-“Echo-Request“-Paket an den zu überprüfenden Host. Der vermeintliche Host sendet „Pong“, ein ICMP „Echo-Reply“ zurück. Ist der vermeintliche Host nicht erreichbar, wird vom zuständigen Router mit „Network unreachable“ oder „Host unreachable“ geantwortet.

„Transmission“-Schnittstelle

Über diese Schnittstelle synchronisieren sich die einzelnen lokalen Interaktionsmodelle. Ein gesamter Ablauf von der Veränderung des lokalen Modells bis zum Einpflegen auf allen lokalen Ebenen wird in zwei Sequenzdiagrammen in den Abbildungen 4.25 und 4.26 im anschließenden Abschnitt 4.4.4 gezeigt.

Die Daten, die über diese Schnittstelle gesendet bzw. empfangen werden, werden von der Transmitterkomponente in ein versendefähiges Format verpackt, über einen technisch üblichen verbindungslosen Datenstrom an die Receiverkomponente gesendet und dort nach gleichem Vorgehen, wie zuvor verpackt, wieder entpackt. Durch n Clients und einen Server, bildet sich hier eine 1 zu n Beziehung zwischen Server und Client aus. Es entstehen n Datenströme zum Server und 1 bis n Datenströme vom Server zu den Clients. Ein Datenstrom im Fall, dass ein Broadcast genutzt werden kann und n falls der Server jeden einzelnen Client gesondert bedienen muss.

Es bietet sich hier an, eine unsichere verbindungslose Übertragung zu verwenden, da es sich bei den zu übertragenden Daten um Interaktionsdaten handelt, die fortlaufend während der Interaktion generiert werden. Bei der Übertragung ist es wichtig, ähnlich wie bei einer Videoübertragung, möglichst zeitnah die Information zu erhalten. Falls ein Paket korrupt ist, ist zeitnah das nachfolgende schon eingetroffen. Eine Fehlerbehandlung würde nur verworfen werden, weil die korrigierte Information schon obsolet ist. Bei den vorgenannten „join“- und „resign“-Schnittstellen bietet sich eine sichere verbindungsorientierte Übertragung von Daten an, weil hier die Kommunikation abschließend und damit für den weiteren Ablauf des Interaktionslayer entscheidend ist.

4.4.4 Evaluation der Architektur mit logischen Testfällen

Um die Vollständigkeit für die Verteilungs- und Interaktionskomponenten der Architektur zu überprüfen, betrachten wir einen abstrakten Eingabefall eines Users am Client. Abbildung 4.25 zeigt die clientseitige Sequenz und Abbildung 4.26 die serverseitige. Die Komponenten der jeweiligen Kommunikationspartner vom Client bzw. Server werden in beiden Sequenzdiagrammen abgebildet.

Bei der Abarbeitung der Eingabe auf dem Client veranlasst die Updatekomponente die Transmitter- und Repräsentationskomponente parallel die Eingabedaten abzuarbeiten. Bei einer technisch möglichen Parallelverarbeitung kann diese eventuell Performancevorteile bringen. Endgültige globale Modellveränderungen treten im lokalen Clientmodell erst nach dem Update durch den Server ein. Die nicht validierte lokale Modellvariante kann zwar schneller angezeigt werden, kann aber Fehler enthalten. Weil ein Zyklus nur 1/25 Sek. oder besser 1/60 Sek.[18] dauern darf

[18] Der Wert von 60 Frames pro Sekunde hat sich bei Tests mit dem später beschriebenen Prototypen als angenehm und uneinschränkend heraus gestellt. Bei Werten um die 30 Frames pro Sekunde fühlte sich das System etwas zäh an. Bei Werten unter 20 Frames pro Sekunde wurde eine visuelle Verzögerung sichtbar. (Napitupulu, 2008) empfiehlt ebenfalls Werte von 60 Bildern pro Sekunde.

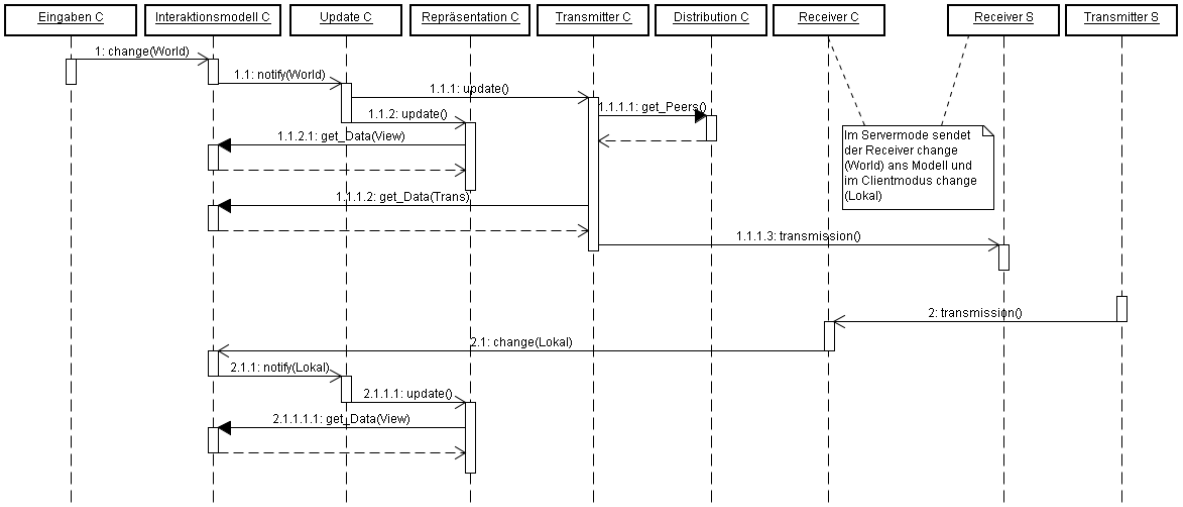

Abbildung 4.25: Clientseitiges Sequenzdiagramm.

und damit die anzuzeigenden Veränderungen kaum unterscheidbare Unterschiede aufweisen, kann bei der Implementierung gegebenfalls das update 1.1.2 weggelassen werden und nur das endgültige Update 2.1.1.1 ausgeführt werden.

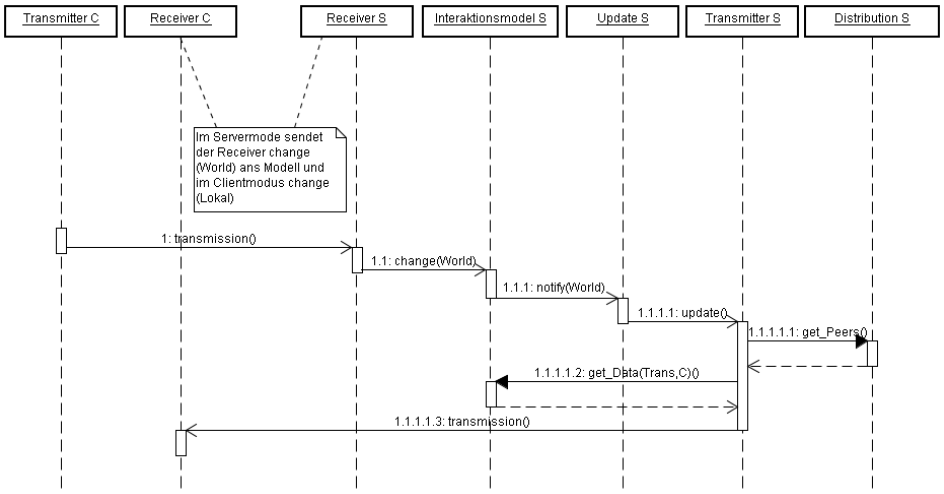

Abbildung 4.26: Serverseitiges Sequenzdiagramm.

Nachdem der Server sein Modell gemäß der übertragenen Eingabe geändert hat, überträgt dieser das neue Modell an alle Clients, die diese Veränderung betrifft. Der Transmitter des Servers entscheidet bei der Übertragung der Modellveränderungen, welche Teile in welcher Form an die Client Receiver übertragen werden. Jeder Client entscheidet bei Eintritt in das System, welches Profil für die Form der Übertragung er benötigt. Der Bereich der relevanten Daten wird vom verteilenden Server bei Eintritt in das System festgelegt.

4.5 Integration weiterer Komponenten zu einem Gesamtkonzept

Die Konzeption einer verteilten Interaktionsschicht mit frei beweglichen Objekten dient zwar dem Zweck der Erforschung sinnvoller und machbarer Interaktionstechniken, erzeugt aber keinen produktiven Mehrwert. Um einen Mehrwert zu skizzieren, erweitern wir das Design der Architektur des Interaktionslayers um Komponenten, die im Rahmen gemeinsamer verteilter Computerarbeitsplätze aus der Forschung vorgeschlagen oder schon angewendet werden.

Vier Fragen stellen sich beim Betrachten der Interaktionslayerarchitektur.

1. Wie erkennt das System die Anordnung der einzelnen Arbeitsflächen?
2. Wie werden vorhandene Applikationen in das System integriert?
3. Kann man gegebenenfalls neue Applikationen entwickeln, die die Möglichkeiten der Interaktionsschicht nativ nutzen können?
4. Wer, wie und was speichert eigentlich Daten ab?

4.5.1 Anordnung der Arbeitsflächen

Um eine gemeinsame Arbeitsfläche der Systeme zu bilden, werden die Bereiche der Einzelsysteme miteínander verbunden. Abbildung 4.27 zeigt ein Beispiel, in dem die räumliche Konstellation der Einzelsysteme in der realen Welt topologisch auf die virtuelle Welt abgebildet werden. Diese Anordnung der Arbeitsflächen kann z.B. wie in der Arbeit von (Johanson u. a., 2002b) über eine Konfigurationsdatei festgelegt werden. Dies ist nur für statische Anordnungen sinnvoll.

Abbildung 4.27: Ein Beispiel der Abbildung von der realen in die virtuelle Welt.

Bei täglichem Wechsel der Anordnung kann dies auch über eine interaktive grafische Konfigurationsanwendung geschehen. Dies macht die Konfiguration anschaulich und einfach zu bedienen. Einfach kann hier auch intuitiv heißen, weil das Anordnen von mehreren Monitoren bei Einzelplatzsystemen mittlerweile im täglichen Gebrauch der Benutzer stattfindet. Kommt eine weitere Person mit System zur arbeitenden Gruppe dazu, könnte dies auch durch die dazu kommende Person organisiert werden, ohne die Sitzung der vorhandenen Personen und Systeme zu unterbrechen. Denkbar ist hier auch eine Eingabe durch die Benutzer, die durch Berechnungen des Systems so aufgewertet werden, dass die Eingaben mit einem „Fingerstreich“ vollzogen werden können. Der Vorgang ist vergleichbar mit einer Situation, in der eine Person zu einer an einem großen Tisch sitzenden Arbeitsgruppe stößt, sich einen freien Platz auswählt und sich setzt. Ist kein Platz vorhanden, so reicht es, wenn ein bis zwei Personen etwas rücken und der neuen Person einen nun freien Platz anbieten. An der Rekonfiguration wären in diesen Fällen dann nur ein bis drei Personen beteiligt. Die anderen würden nicht durch die Konfiguration gestört.

Wenn sich aber auch während der Arbeit die Konfiguration ändert, um durch eine dynamische Anordnung den Arbeitsfluss zu unterstützen, wäre es gut, wenn sich die Anordnung der Arbeitsflächen automatisch den Gegebenheiten anpasst. Gegebenheiten können hier eine physikalische Anordnung der Geräte sein, oder eine Anordnung, die das Zusammmenspiel der beteiligten Personen unterstützt. Im ersten Fall würden wir von einem strukturellen Bezug der Anordnung und im zweiten von einem funktionalen Bezug sprechen.

Ein funktionaler Bezug zur Anordnung könnte über eine Analyse der Benutzerinteraktionen geschehen. Ein Bewertungssystem würde dann die Vorgaben für eine kalkulierbare Anordnung ergeben. Ein struktureller Bezug würde auf Positions- und Ausrichtungsdaten zurückgreifen. Diese müssten, wie in Abbildung 4.28 zu sehen, über ein „Motion-Capturing“ System oder andere „Indoor-Positioning“ Systeme geschehen.

Für die Repräsentation des Sachverhaltes der Arbeitsflächenanordnung haben wir in unserem erweiterten Modell eine Komponente eingefügt, die reale Objekte in das System einfügen kann. Dies entspricht einem Lösungsansatz mit strukturellem Bezug. Ein „System Positioning Grabber“ erhebt die Daten für die Positionen und Anordnungen von Objekten im Raum. Diese Daten können durch eine Konfigurationsdatei, ein grafisches Eingabeinstrument oder ein „Indoor Positioning“ System bereitgestellt werden. Die neue Komponente „real object input“ des Interaktionslayers verarbeitet diese Daten zusammen mit den Daten der beteiligten Geräte und bestimmt die realen und virtuellen Paare. Mit diesem Ergebnis wird dann das Interaktionsmodell manipuliert. Weil der Mechanismus der Manipulation, dem Mechanismus aus der Inputkomponente gleicht, könnte die „real object input“ Komponente auch in die Inputkomponente integriert werden. Dieses Vorgehen differenziert nicht zwischen Interaktionsinstrumenten und Arbeitsflächeninstrumenten und verbindet verarbeitungstechnisch die erste und dritte Ebene aus Abbildung 4.6 in Abschnitt 4.2.5 miteinander.

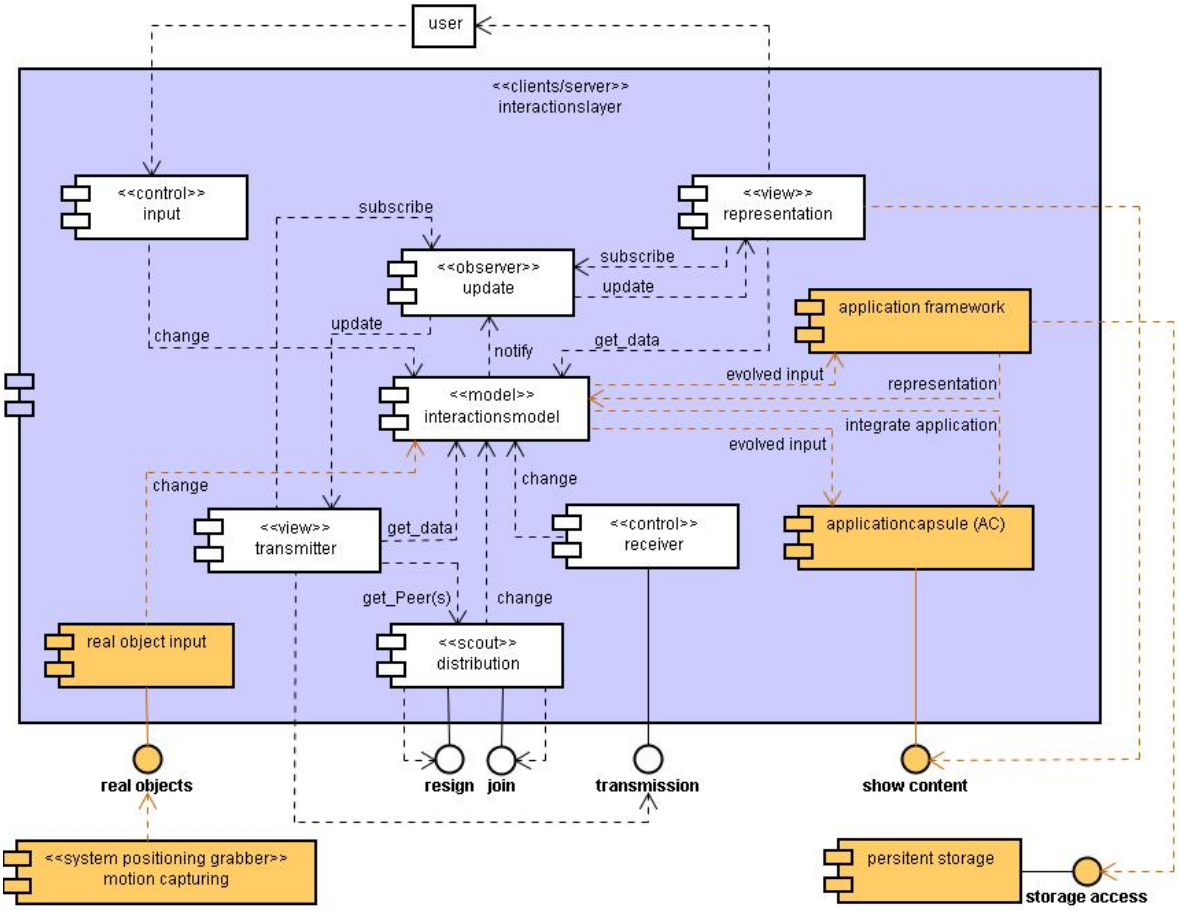

Abbildung 4.28: Die Architektur des Interaktionslayer erweitert durch drei Komponenten.

4.5.2 Entwicklung neuer Applikationen

Um den vorgeschlagenen Interaktionslayer produktiv nutzen zu können, müssen Anwendungen in diesen integrierbar sein. Eine „application framework“-Komponente bietet hier Entwicklern die entsprechenden Rahmenbedingungen und Unterstützungsmechanismen zur Neuentwicklung von Anwendungen, die über den Interaktionslayer den Benutzern zugänglich gemacht werden.

Die Erscheinung der Eingaben von Benutzern innerhalb des Interaktionsmodells werden aus diesem herausgebildet und über die Frameworkfunktionen an die Anwendungen weitergeleitet. Die Repräsentation der Anwendung wird umgekehrt über die Frameworkfunktionalität an das Interaktionsmodell gesendet und durch dieses an die jeweiligen Repräsentationskomponenten der betroffenen Einzelsysteme weitergeleitet.

Die in die erweiterte Architektur in Abbildung 4.28 eingezeichneten Verbindungen „evolved input“ und „representation“ zwischen der „application framework“ Komponente und der „interactionmodel“-Komponente sind hier stellvertretend für architektonisch korrekte Verbindungen im Sinne des MVC-Musters. Die „evolved input“ Verbindung entspricht dem View auf das Modell und die „representation“ Verbindung dem Control. Das „application framework“ verbindet sich damit über ein MVC-Muster mit dem Interaktionsmodell. Innerhalb des Frameworks sollte ein gespiegeltes[19] MVC-Muster eine gute Anpassung möglich machen.

Das „application framework“ und die damit entstandenen Anwendungen entsprechen den Kollaborationswerkzeugen, die in der Vergangenheit für existierende Desktopsysteme entstanden sind. Diese müssten bei Bedarf neu entwickelt werden oder an das Framework angepasst werden.

4.5.3 Integration vorhandener Applikationen

Das „application framework“ bietet dem Interaktionslayer erst in der Zukunft einen produktiven Status. Heutige Problemlösungen und Anwendungen müssen erst langsam den Weg in das neue System finden. Die „applicationcapsule“ Komponente bietet deshalb einen fließenden Übergang. Es bindet vorhandene Desktopanwendungen in den Interaktionslayer ein. Diese können, wie in Abschnitt 3.1 gefordert, auf der individuellen Ebene als Desktopanwendung und auf der kollektiven Ebene als Interaktionslayeranwendung genutzt werden.

Die Ebene 2 aus den funktionalen Voraussetzungen in Abbildung 4.6 wird deshalb in die zwei Unterebenen „collective“ und „individual“ aufgeteilt. Abbildung 4.29 zeigt die Erweiterung der Abbildung 4.6 um die zwei Unterebenen.

Die individuelle Ebene verbindet den Benutzer über die ihm zugeordneten Schnittstellen zur Interaktion, dominierend mit der gekapselten Anwendung, die sich auf der individuellen Ebene

[19] Gespiegelt heißt hier, dass der Controller des Frameworks dem View auf das Interaktionsmodell entspricht und der View des Frameworks einem Controller für das Interaktionsmodell.

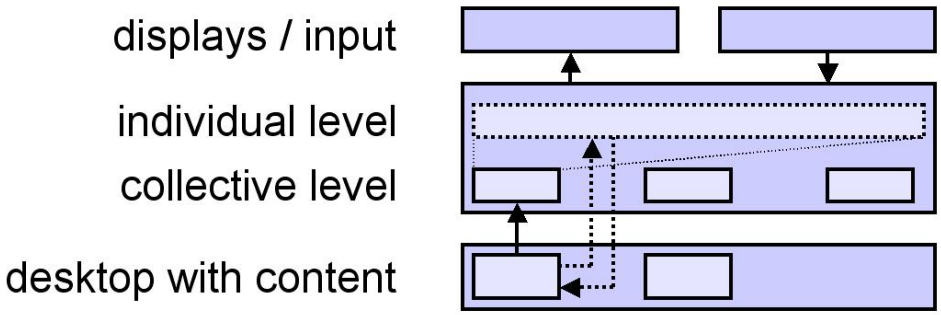

Abbildung 4.29: Aufteilung der Anwendungsebene in zwei Unterebenen.

befindet. Die Interaktion wird an die gekapselte Anwendung entsprechend weitergeleitet. Befindet sich eine Anwendung nicht im produktiven sondern im reflexiven Status und damit auf der kollektiven Ebene, so werden alle Interaktionen an den Interaktionslayer und dessen Objekte und Verhalten geleitet. Die gekapselte Anwendung bleibt auf dieser Ebene unberührt.

Für die Integration der Komponente in die Interaktionslayerarchitektur gilt für die Verbindungen „evolved input“ das gleiche Schema wie für die Einbindung des „application framework“ und die Einhaltung des MVC-Musters. Die „representation“ wird hier aber durch eine „show content“ Schnittstelle realisiert, da der Inhalt der Anzeige der jeweiligen Anwendungen nicht durch das Gesamtsystem geschleift werden soll. Lediglich die jeweilige Schnittstellenverbindung wird dem zugehörigen „applicationcapsule“ Objekt zugeordnet und an die „representation“ Komponente weitergeleitet.

Das Interaktionsmodell und dessen Funktionalität muss für die Produktion von „applicationcapsule“ Objekten erweitert werden. Bei einem definierten Modellstatus mit entsprechender Benutzereingabe muss ein neues Objekt generiert werden, welches die jeweilige Anwendung kapselt und repräsentiert.

4.5.4 Speichern

Wie viele der existierenden Kollaborationswerkzeuge einen transparenten Speicher anbieten oder abbilden, bietet eine Verbindung des „application framework“ zu einem verteilten transparenten nichtflüchtigen Speicher die gleiche Unterstützung an. Das Design der „persistent storage“ Komponente wird hier offen gelassen und auf zahlreiche schon existierende Lösungen in diesem Bereich verwiesen. Die Anbindung an diese Lösungen wird in der „application framework“ Komponente angesiedelt. Gekapselte Anwendungen greifen auf die ihnen schon vorher zur Verfügung stehenden Speicherlösungen zu.

4.6 Vergleich der vorgeschlagenen Architektur mit anderen Architekturen

Um das hier entwickelte Architekturdesign für eine nahtlose Komposition verteilter Arbeitsflächen zu reflektieren und gegebenenfalls in andere Architekturen zu integrieren bzw. sie mit ihnen zu verbinden werden nachfolgend drei Architekturen unter verschiedenen Aspekten vorgestellt und entsprechende Aspekte diskutiert.

Zwei wichtige Aspekte des Designs sind die Verteilung und die Integration des MVC-Pattern. In (Napitupulu, 2008) werden zwei Verteilungsmöglichkeiten von „MVC-Komponenten“[20] vorgestellt und besprochen. Abbildung 4.30 zeigt die Variante A, bei der die Eingaben über den Controller verteilt werden und der Client-View die darzustellenden Daten von dem Server-Model anfordert. Variante B zeigt drei vollständige MVC-Realisierungen, bei denen alle Modelle über den Controller synchronisiert werden.

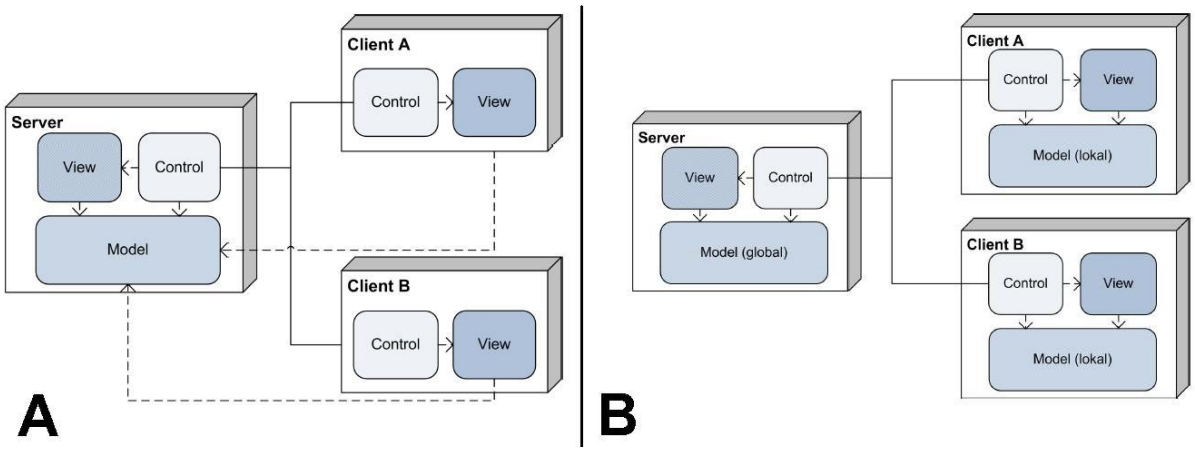

Abbildung 4.30: Ebenfalls MVC in Spielearchitektur aus (Napitupulu, 2008).

Die in dieser Arbeit vorgeschlagene Architektur entspricht annährend der Variante B, weil auch diese in den Clients ein Modell realisiert. Der Unterschied liegt in der differenzierteren Ausformulierung des Controllers. Der gesamte Synchronisationsmechanismus ist in dieser Architektur aus dem Controller ausgelagert. Mit dieser Auslagerung ist es nicht möglich, lokale Veränderungen über den Controller an dem lokalen Modell vorbei zu führen, was einen weiteren Strukturierungsvorteil mit sich bringt. Aus lokaler Sicht ist das lokale Modell immer vollständig, weil es immer die aktuelle lokale Version in sich trägt. Dies kann aber muss nicht in Variante B der Fall sein. Der Controller wird durch dieses Vorgehen leichtgewichtiger, weil er keine globale Eingabeverwaltung und Modellveränderung realisieren muss. Der Controller muss lediglich die lokalen Eingaben umsetzen. Die Auslagerungen aus dem Controller finden sich in den drei Verteilungskomponenten: Distribution, Transmitter und Receiver.

[20] Der ursprüngliche Begriff aus (Burbeck, 1992) des „MVC paradigm“ bezieht sich auf Objekte. Der Begriff „Komponente“ wird sehr vielschichtig benutzt wie Kapitel 3 aus (Töpfer, 2006) darstellt.

Eine weitere Architektur, die einen Mechanismus der Selbstorganisation realisieren soll, wird in Abbildung 4.31 dargestellt. Die Autonomic Management Engine (AME) baut auf das Konzept des „Autonomic computing control loop“ auf. Sie überwacht das System und deren Ereignisse. Spricht ie Interpretation der Ereignisse dafür, das System umzukonfigurieren, werden entsprechende Aktionen ausgeführt. Zwei Perspektiven bei der Betrachtung der AME sind für uns interessant. Erstens die funktionale und zweitens die strukturelle Perspektive. Die funktionale Perspektive beschreibt das in Abschnitt 3.3 geforderte Self-Configuring. Die strukturelle Perspektive wird durch den Aufbau der AME beschrieben.

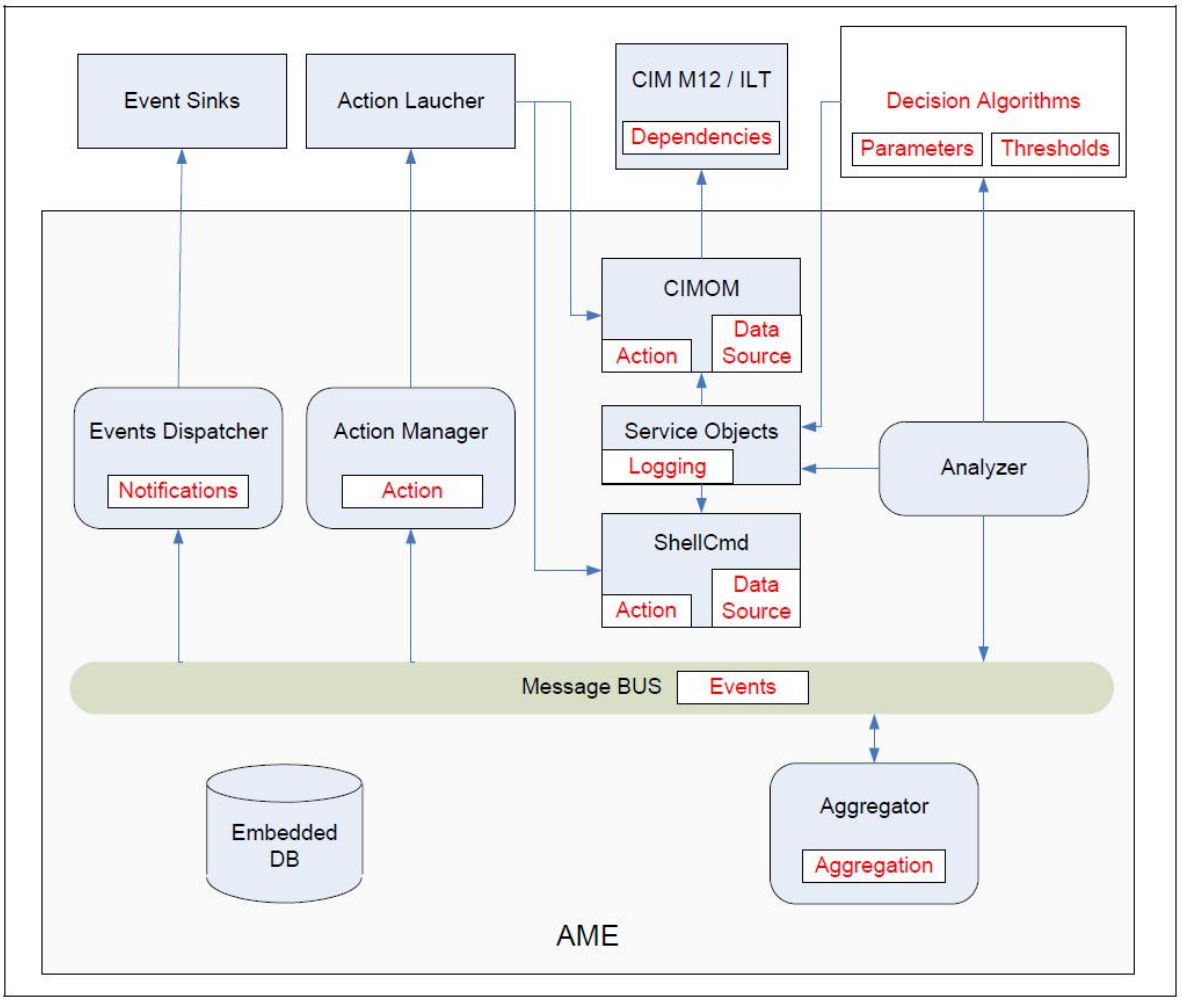

Abbildung 4.31: Die Architektur der Autonomic Management Engine (AME). (Manoel u. a., 2005)

Das Self-Configuring findet sich in unserer Architektur in der Komponente „real object input“ wieder. Diese bekommt Daten von der „motion capturing“-Komponente und löst gegebenenfalls die systemverändernden Aktionen im Interaktionsmodell aus. Ein solcher Vorgang wird in der AME durch ca. acht Module verwaltet. Diese höhere Komplexität ist Hinweis auf einen größeren Differenzierungsgrad bei der zugrunde liegende „control loop“ Lösung. In unserem Fall ist die Auseinandersetzung mit Self-Configuring nicht so weit fortgeschritten. Die Architektur kann

aber als Ideengeber für eine Realisierung eines Self-Configuring-Mechanismus dienen.

Bei der Betrachtung der Struktur fällt auf, dass der Message BUS ein zentrales Verteilungselement zwischen den Modulen ist und es eine Event- und Actionverwaltung gibt. In unserer Architektur werden Events von den Controllern (entspricht Analyzer) an das Interaktionsmodell weiter gegeben. Diese Events manipulieren entweder das Interaktionsmodell (entspricht Aggregation) oder werden an die Anwendungen weitergeleitet (entspricht Action Manager). Dieser Vergleich ergibt in der Interpretation unseres Design keinen Mehrwert. Interessant ist nur die Überlegung alle Komponenten über einem Message Bus miteinander zu verbinden. Die Verteilung würde lokal sowie global über diesen Message Bus laufen. Die Komponenten könnten somit überall lokalisiert werden. Ein anderer Verteilungsmechanismus wäre nicht nötig. Verglichen mit Abbildung 4.30 würde die Verbindung zwischen dem Server Modell und allen Views und Controllern über diesen Bus laufen. Diese Lösung würde eine sehr einfache Struktur ergeben. Fraglich hierbei ist, ob die Übertragung über den Message Bus immer performant genug ist.

In der von (Roßberger, 2008) vorgeschlagenen übergeordneten Architektur zur Erweiterung der DynAmbient Anwendung (Abbildung 4.32) geht es darum, wie und woher die von DynAmbient dargestellten Bilder kommen und wie bei speziellen Ereignissen innerhalb von DynAmbient Ereignisse im Gesamtsystem ausgelöst werden können. Ich schlage vor, beides über Services abzuwickeln. Die Bilder können z.B. extern von dem Verwaltungsdienst für Datenobjekte angefordert werden. Bei Ereignissen innerhalb von DynAmbient, die extern wirken sollen, wie zum Beispiel „zeige das Video“ kann ein Dienst genutzt werden, der dieses adäquat dezentral löst.

Mit dieser Servicelösung können in unserem Beispiel verschiedene Views angesprochen werden. Für eine Echtzeitinteraktion wird diese Lösung nicht anpassbar sein. Es ist aber denkbar, dieses als übergeordnete Struktur zur Verwaltung von externen Ereignissen zu nutzen, so wie für DynAmbient.

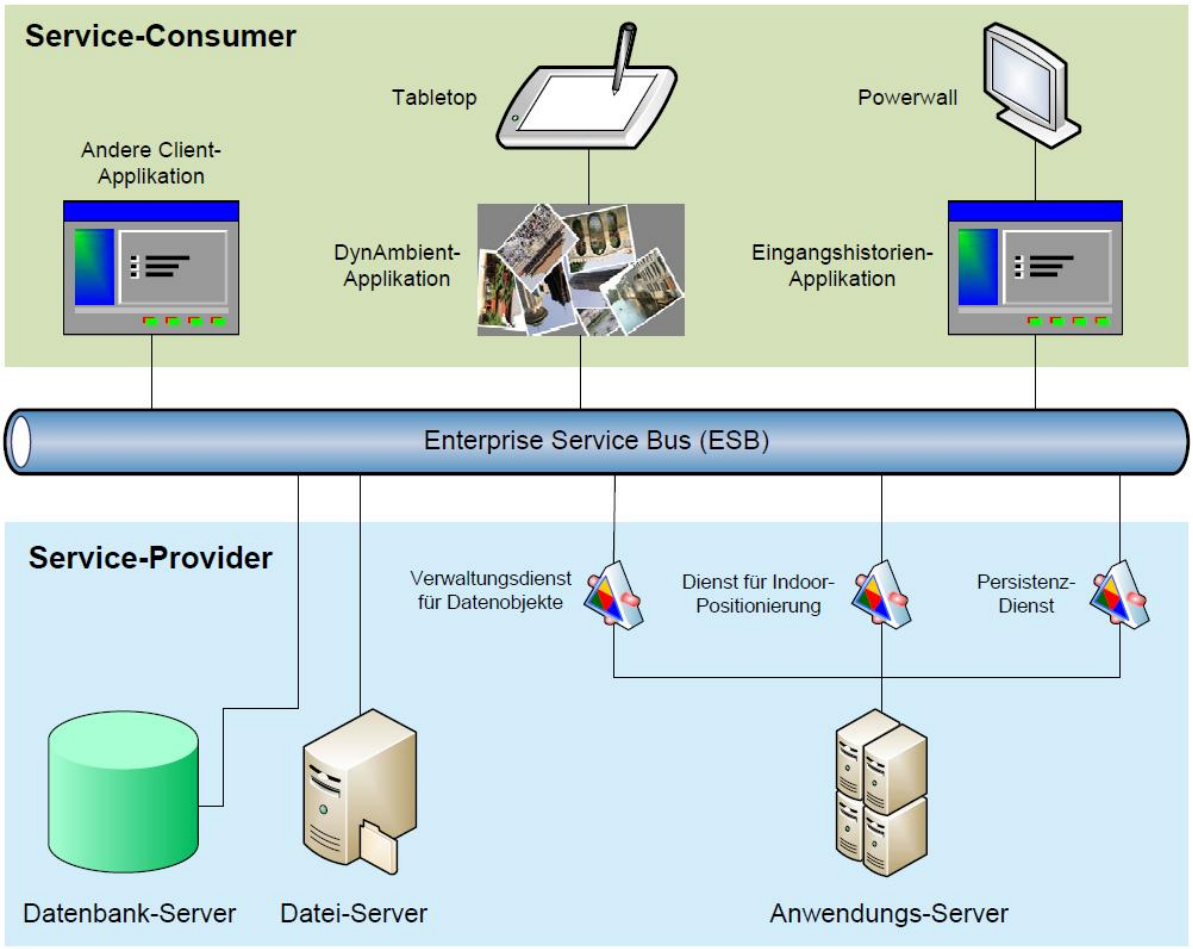

Abbildung 4.32: Vorgeschlagene übergeordnete Architektur für DynAmbient von (Roßberger, 2008).

5 Realisierung

In diesem Kapitel werden verschiedene Aspekte der Umsetzung eines ersten Prototypen auf Grundlage des Designs aus Kapitel 4 und der Analyse aus Kapitel 3 dargestellt. Weil das zugrunde liegende Konzept zu umfangreich für eine Komplettimplementation innerhalb dieser Arbeit ist, wird im ersten Abschnitt 5.1 der Funktionsbereich für den Prototypen eingeschränkt. Im zweiten Abschnitt 5.2 wird gezeigt und begründet welche technischen Systeme für die Realisierung des Prototypen und seiner Komponenten genutzt werden sollten und wurden. Der dritte Abschnitt 5.3 beschreibt die Umsetzung und deren Ergebnis von den einzelnen Komponenten des Prototypen. Neben der Realisierung der Komponenten wird in diesem Abschnitt auch auf die sich ergebenen Schwierigkeiten bei der Entwicklung des Prototypen eingegangen. Im letzten Abschnitt 5.4 werden die Erfahrungen mit und Einschätzungen über den entstandenen Prototypen referiert.

5.1 Eingrenzung der Realisierung auf die Essenz der verteilten gemeinsamen Interaktion

Für die Realisierung eines Prototypen sind die funktionalen Anforderungen aus dem Fazit von Kapitel 3 und dem daraus folgenden Komponentendiagramm aus Abschnitt 4.4.1 maßgeblich. Sie werden in die zwei Kategorien *gemeinsame* und *Übergang zur individuellen Interaktion* eingeordnet. Beide Kategorien reichen in die Menge der gemeinsamen Interaktion und sind deshalb für den Prototypen maßgeblich. Um einen erfahrbaren Prototypen zu realisieren, wurden die drei funktionalen Anforderungen der gemeinsamen Interaktion in vollem Umfang realisiert. Für die vier funktionalen Anforderungen des Übergangs reicht der Funktionsumfang, der die Schnittmenge zur gemeinsamen Interaktion bildet. Durch dieses Vorgehen wird eine gemeinsame Interaktion mit dem Prototypen möglich und es können erste Erfahrungen und Eindrücke gesammelt werden.

Für die Umsetzung des Designs gilt, dass nur der Kern aus Abschnitt 4.4.1 Abbildung 4.17 der verteilten Interaktionschicht realisiert werden muss. Dieser stellt die drei Forderungen *skalieren*, *verschieben* und *orientieren* aus der gemeinsamen Interaktion dar. Die restlichen sechs Anforderungen aus der oben genannten Schnittmenge sind in der erweiterten Architektur in Abschnitt 4.5 Abbildung 4.28 abgebildet. Diese wurden nicht in Gänze sondern nur zu Veranschaulichungszwecken realisiert.

Konkret bedeutet dies für die Komponenten „real object input“ und „distribution“, dass hier der einfachste Lösungsvorschlag mithilfe einer Konfigurationsdatei ausreichend ist. Weil sich Clients und Server anhand einer Datei konfigurieren, in der ein statisches Gesamtsystem und dessen Parameter bekannt gegeben werden, kann auf das Finden von Peers und auf die Eingabe von Systemparametern verzichtet werden.

Für das Veranschaulichen des Ein- und Austritts der Anwendungen genügt für beide Anwendungskomponenten, die Funktionen des Erstellens und Austretens von Anwendungobjekten zu realisieren. Damit können Anwendungsobjekte an der gemeinsamenen Interaktion beteiligt werden. Weiterhin muss für beteiligte Anwendungsobjekte der Übergang zwischen den beiden Unterebenen wie in Abbildung 4.29 gezeigt möglich sein. Die Implementierung der beiden Unterebenen soll der Interpretation der Handlungsräume aus Abschnitt 3.1 gerecht werden.

Eingabeinstrumente können abhängig, an welchem Rechner sie angeschlossen sind, in einer Instanz, wie in Abschnitt 4.2.5 vorgegeben, zusammengefasst werden. Dies macht die Implementierung einfacher, weil die Eingabemechanik nur umgelenkt werden muss, aber nicht auf Betriebsystemebene neu interpretiert werden muss. Singletouch kann durch Zufügen einer Historie als Maus implementiert werden. Soll Multitouch sinnvoll mit eingebunden werden, müssen Ansammlungen von gleichen Eingabeeinheiten und deren Historie möglich sein. Eine Interpretationsstufe auf diese Ansammlung liefert dem System die entsprechenden Eingaben. Sehr deutlich wird dies anhand von Eingaben durch Gesten, bei denen nur noch Interpretationen weitergereicht werden.

Die Anforderungen aus anderen Forschungsarbeiten aus Abschnitt 3.3 fließen bei der Realisierung eines ersten Prototypen nur über die sieben zu implementierenden Komponenten aus Abbildung 4.17 mit ein. Diese Anforderungen sollten bei der Entwicklung mit betrachtet werden, um spätere Erweiterungen besser zufügen zu können.

5.2 Auswahl der eingesetzten Tools und Techniken

Um „echte“ Interaktionen zu verwalten braucht es eine elektronische Echtzeitumgebung. Das heißt, dass alle Interaktionen der Benutzer möglichst ohne Verzögerung oder zumindest mit vorhersagbarer geringer Verzögerung von dem System verarbeitet werden können. Unvorhersagbare Verzögerungen oder unregelmäßiges Verhalten fördert die Konfusion bei der Benutzung und führt zum Scheitern. Ebenso wenig kann ein System mit unregelmäßigem Verhalten evaluiert werden, weil nicht das eigentliche Verhalten des Systems evaluiert werden würde, sondern hauptsächlich die Unregelmäßigkeiten.

Klassische Programmiersprachen für Echtzeitsysteme sind C, C++ und anwendungsspezifische Programmiersprachen. Eine jüngere ist RT-Java. Letztere garantiert im Gegensatz zu Java den Ablauf des Systems nicht zu unterbrechen und bietet die Vorzüge von Java, wie zum Beispiel Garbage-Collecting und Programmieren mit Hilfe umfangreicher Bibliotheken. C++ bietet eine gute Performance sowie betriebssystemnahe Programmierung und ebenfalls einen großen Funktionsumfang sowie objektorientiertes Programmieren.

Eine ähnliche wie die im Design vorgesschlagene Architektur finden sich in der Spieleentwicklung. Spielearchitekturen beinhalten ebenfalls eine Eingabeverarbeitung verschiedener Eingabeinstrumente, ein Spielmodell und eine Visualisierung des Spielmodells. Sollten mehrere Spieler mit verschiedenen Rechnern gleichzeitig in Interaktion treten, kommt ebenfalls eine Distribution der Eingaben und Spielemodelle dazu. Zur Umsetzung und Unterstützung eines solchen Vorhabens gibt es Game-Engines[1], Render-Engines[2] und Physik-Engines. Die meistens dieser Unterstützungsmittel setzen auf C++ auf oder unterstützen diese Programmiersprache. Aus diesem Grund und wegen der guten Perfomance von C++, wurde sich für C++ als Programmiersprache entschieden.

Die drei Hauptgesichtspunkte Interaktionsmodell, Verteilung und Visualisierung müssen nun noch betrachtet werden. Für das Interaktionsmodell wurde sich für eine physikbasierte Variante entschieden. Neben vielen Spieleumsetzungen, in denen physikbasiertes Objektverhalten den Kern der Modellberechung stellt, zeigen (Agarawala und Balakrishnan, 2006) eine vielversprechende Desktopvariante mit einem auf physikalischen Eigenschaften von Objekten auf dem Desktop basierenden Verhaltensmodell.

Um die dreidimensionalen physikbasierten Objekte entsprechend zu visualisieren, liegt eine dreidimensionale visuelle Darstellung der Objekte und der Interaktionsschicht nahe. Diese ist nicht zwingend, denn Abstraktionsebene und Visualisierung können auch verschieden dargestellt eine sinnvolle Verbindung ergeben. Eine dreidimensionale Sicht auf das Interaktionsmodell benötigt wenig bis keine mentale Transferleistung, weder auf Seiten der Benutzer als auch auf Seiten der Entwickler. Um die dritte Dimension im Desktopbereich ranken verschiedene Ansätze unter verschiedenen Aspekten. Der Nutzen als funktionstragenes Element hat sich bis heute nicht bestätigt. Genutzt wird es hauptsächlich als dekoratives[3] Element um die Aufmerksamkeit auf sich zu lenken oder als Veranschaulichungselement[4], um Vorgänge dem Benutzer nachvollziehbar zu machen. Die aufgeführten Argumente und vor allem die gute Verfügbarkeit von 3D-Render-Engines haben uns dazu veranlasst, für die Visualisierung des Modells eine 3D-Render-Engine zu wählen.

Die Physikengine „Open Dynamics Engine“ (ODE) und OpenSG (Szenengraph) als Renderengine wurden beide als „open-Source“-Projekt unter C++ ausgewählt und die Tauglichkeit in den Projekten (Roßberger, 2007) zum Zusammenspiel von Physikengine und Grafikengine und (Köckritz, 2007) zur Verteilung von Eingaben und Visualisierung überprüft. Im Rahmen

[1] Game-Engines können skriptgesteuerte Spielmodelle, Soundsysteme, Zustandspeicherung, Netzwerksupport, Grafik-Engines und Physik-Engines enthalten.

[2] Hierbei handelt es sich um Grafik-Rendering-Engines die dreidimensionale Grafiken rendern. Nicht gemeint sind Render-Engines wie z.B. die Gecko-Rendering-Engine, die in Browsern verwendet werden, um Seiten aus dem World Wide Web darzustellen.

[3] In aktuellen Betriebssystemen, wie Windows 7 ,MacOS X oder verschiedenen Linux Varianten werden Desktopelemente dreidimensional dargestellt, um einen zielgerichteten Blickfang beim Benutzer zu erreichen.

[4] Zum Beispiel erreichen die Projekte „Looking Glass“ oder „Beryl“ mit einer dreidimensionalen Bewegung der Objekte, dass der Benutzer sich besser den Verbleib oder die Herkunft von Objekten vorstellen kann. Damit wird eine sichtbare Verbindung zwischen verschiedenen Zuständen von Objekten dargestellt und der Benutzer kann die Vorgänge optisch nachvollziehen und damit später auch besser verorten. Ein Bericht und eine Bewertung von existierenden „3D Desktops“ findet sich unter (Weiss, 2007)

dieser Evaluation wurde ein verteiltes Pong Spiel mit Hilfe von OpenSG, wie in Abbildung 5.1 zu sehen, entwickelt.

Abbildung 5.1: Verteiltes Pong als Testszenario für Hardwareumgebungen und Softwaredevelopment.

Beide Projekte wurden mit einem negativ behafteten Fazit beendet. Das Projekt von Roßberger wurde mit einem negativen behafteten Fazit beendet, weil die angebotenen externen Manipulationsfähigkeiten von physikbasierten Objekten zu eingeschränkt waren und die Erweiterung der ODE zu aufwendig erschienen. Das Projekt von Köckritz wurde mit einem negativ behafteten Fazit beendet, weil sich die in OpenSG integrierte Verteilung des Szenengraphen zur verteilten Darstellung unter den gegebenen technischen Umständen als nicht echtzeitfähig herausstellte. Der Versuch die OpenSG Engine zu verbessern und zu erweitern, stellte sich als sehr aufwendig dar, weil anscheinend bei der Entwicklung der Engine hauptsächlich auf Performance geachtet wurde und nicht auf einen modularen Aufbau mit echten Schnittstellen.

Nach Recherchearbeiten und kurzen Evaluationsphasen wurde sich für eine Paarung von der „Game-Engine“[5] „Irrlicht“ und der Physik-Engine „PhysX“ entschieden.

„PhysX“ bietet ein professionelles PhysX-SDK an und ist in das CUDA-System integriert, so dass Berechnungen zum physikalischen Verhalten von gängigen Grafikchips übernommen werden können. PhysX bietet zwar unzureichende Formen von Physikobjekten an, weshalb Aussehen und Verhalten von Objekten differieren können oder komplexe Objektstrukturen generiert werden müssten, aber die angebotene externe Manipulation der Objekte ist ausreichend.

[5] Es handelt sich bei der Game-Engine „Irrlicht“ lediglich um eine Engine zum reinen Rendering. Erst durch Erweiterungen, die mit der Irrlicht-Engine verbunden werden können, wird diese neue Verbindung von verschiedenen Engines dann dem Begriff „Game-Engine“ gerecht. Ein Beispiel für eine echte Game-Engine bietet zum Beispiel die Torque-Engine.

Das „Irrlicht“-Projekt befand sich in einem prototypischen aber guten Zustand. Eine Grundstruktur von Irrlicht ist die Nachbildung einer Interface-Mechanik wie sie aus Java bekannt ist. Der Sourcecode schien übersichtlich und lesbar zu sein, so dass Erweiterungen sich an die gegebenen Schnittstellen andocken können und nicht dokumentierte Funktionsweisen durch eine Codeinspektion eruiert werden können. Mittlerweile wurde Irrlicht auch für C# portiert und unterstützt genauso wie PhysX neben Linux und Windows auch MacOS X, so dass bei einer Fortführung des Projektes auf eine angemessene Programmiersprache zurück gegriffen werden kann und alle drei gängigen Betriebssyteme unterstützt werden können.

Nach dem Ausscheiden von OpenSG, muss noch eine Technik zur Verteilung gewählt werden. Die auf der offizellen Irrlicht-Website vorgeschlagene Erweiterung zur Verteilung für die Irrlicht-Engine „RakNet“ erwiesen sich als nicht nutzbar, da die aktuelle Version auf Rollenspiele und nicht auf Echtzeitinteraktion ausgelegt war. Das Einbinden der „RakNet“ Erweiterung gestaltete sich ebenfalls als sehr aufwendig. Auch die im Labor genutzte IROS-Eventheap (Ponnekanti u. a., 2003) Infrastruktur war ungeeignet wegen der unzureichenden Echtzeitfähigkeit. Resultierend aus den schlechten Erfahrungen mit der Echtzeitfähigkeit von angebotenen Verteilungspaketen wurde sich entschieden, die Verteilung selbst über UDP oder TCP zu implementieren. Die größere Kontrolle und Anpassungsfähigkeit der selbst implementierten Lösung schien das Risiko in Sackgassen zu geraten zu minimieren.

5.3 Realisierung der Komponenten

Die in Abschnitt 4.4 entwickelte Softwarearchitektur zur Unterstützung der gemeinsamen Interaktion teilt sich in verschiedene Komponenten. Die Realisierung der Komponenten Interaktionsmodell, Input, Repräsentation, Update, Transmitter und Receiver werden in den folgenden Abschnitten beschrieben. Die Entwicklung einer Software trifft häufig auf unvorhergesehene Probleme, die den Entwicklungsprozess verlangsamen oder erschweren. Zur Reflexion und Erfahrungserweiterung wird am Ende dieses Abschnittes noch auf die aufgetretenen Probleme eingegangen.

5.3.1 Umsetzung des Interaktionsmodells

Das Interaktionsmodell aus Abschnitt 4.4.2 wird, wie in Abschnitt 5.2 vorgeschlagen, physikbasiert umgesetzt. PhysX bildet damit die Kalkulationsregeln und die interne Darstellung des Modells. Verändert sich ein Objekt in Größe, Form oder Position, so wird das Modell neu berechnet.

Das Modell wird in die zwei verschiedenen Kategorien statische und dynamische Objekte aufgeteilt. Statische Objekte sind unbeeinflusst von den Kalkulationregeln und damit nicht in Wechselwirkung mit anderen Objekten. Statische Objekte wirken aber auf dynamische Objekte. Das Verhalten von dynamischen Objekten wird von den Kalkulationsregeln festgelegt und sie treten in Wechselwirkung mit anderen dynamischen Objekten.

Verorten wir die verschiedenen Objekte in die Ebenen von Abbildung 4.6, so handelt es sich bei Objekten der Ebene 1 und Ebene 3 um statische. Die Arbeitsoberfläche aus Ebene 1 bildet eine physikalisch wirkende Grundfläche, die sich nicht durch die Interaktion mit Anwendungsobjekten verändert. Sie wird durch die „real object input“ Komponente verändert und folgt damit nicht dem Veränderungsmodell der PhysX-Engine. In unserem Prototypen ist die „real object input“ Komponente durch Einlesen einer Konfigurationsdatei gelöst und bietet damit kein dynamisches Verhalten auf Ebene 1.

In Ebene 3 sind die Eingabeeinheiten verortet. Sie sind ebenfalls statisch, wirken aber über die PhysX-Engine auf dynamische Objekte der Ebene 2. Die Parameter und damit das Verhalten der Objekte auf Ebene 3 sind an die Eingaben der „input“ Komponente gekoppelt. Sie folgen damit ebenfalls nicht dem Veränderungsmodell, sondern einzig der Interaktionsintention der Anwender.

Ebene 2 teilt sich in die beiden Unterebenen „collective“ und „individual“. Auf der gemeinsamen Ebene folgen die Objekte den Veränderungsregeln der PhysX-Engine. Dies gilt auch für den Einfluss von Ebene 1 und 3 auf Ebene 2. Wird ein Objekt von Ebene 2 durch ein Interaktionsobjekt auf die individuelle Ebene gebracht, so wird dieses Objekt von den Veränderungsregeln der PhysX-Engie ausgenommen und temporär zu einem statischen Objekt. Jetzt werden alle Interaktionen des Interaktionsobjektes der Ebene 3 mit dem temporär statischen Objekt der Ebene 2 verbunden.

Im Design Abschnitt 4.4.2 wird beschrieben, dass sowohl Clients als auch Server ein Interaktionsmodell besitzen sollen. Diese Forderung wird im Prototypen nicht in Gänze umgesetzt, weil dies einen zusätzlichen Faktor an Komplexität für die Implementierung der Synchronisation bedeuten würde und der Vorteil an Performance für einen Prototypen nicht relevant ist. Die lokale Interaktion wirkt lokal nur statisch und erst auf dem Server dynamisch. Der Server nutzt als einziger das vollständige Interaktionsmodell. Die Clients enthalten lediglich ein lebloses Interaktionsmodell.

5.3.2 Eingabeobjekte als „Avatar of Interaction“

Bei der Transformation von einem Paradigma zu einem anderen müssen die wiederverwendeten Teile ebenfalls transformiert werden, damit sie sich adäquat in das andere Paradigma einpassen können. In diesem Abschnitt handelt es sich bei den zu transformierenden Teilen um Eingabeinstrumente des Einzelsystems in das Gesamtsystem. Die meisten Einzelsysteme aus dem in Abschnitt 3.2 geschilderten Szenario unterstützen eine Maus und eine Tastatur als Eingabeinstrument. Diese sind über die Instanz des Einzelrechners gekoppelt. Die Maus ist fest an das Betriebssystem gekoppelt und die Tastatur mit stellvertretendem Cursor ebenfalls. Maus und Tastatur haben durch das visuelle Feedback und die Verortung auf dem Bildschirm, scheinbar eine physikalische Präsenz auf dem Bildschirm und werden dadurch „konkret“. Eingabeeinheiten wie Spracheingabe oder Gestenerkennung bleiben bei Benutzung „abstrakt“[6].

[6] Sie sind vergleichbar mit einer Off-Stimme, wie wir sie aus Filmen kennen. Bei einer visuellen Produktion ist bei einer Off-Stimme die sprechende Person nicht im Bild zu sehen im Gegensatz zu sprechenden Personen,

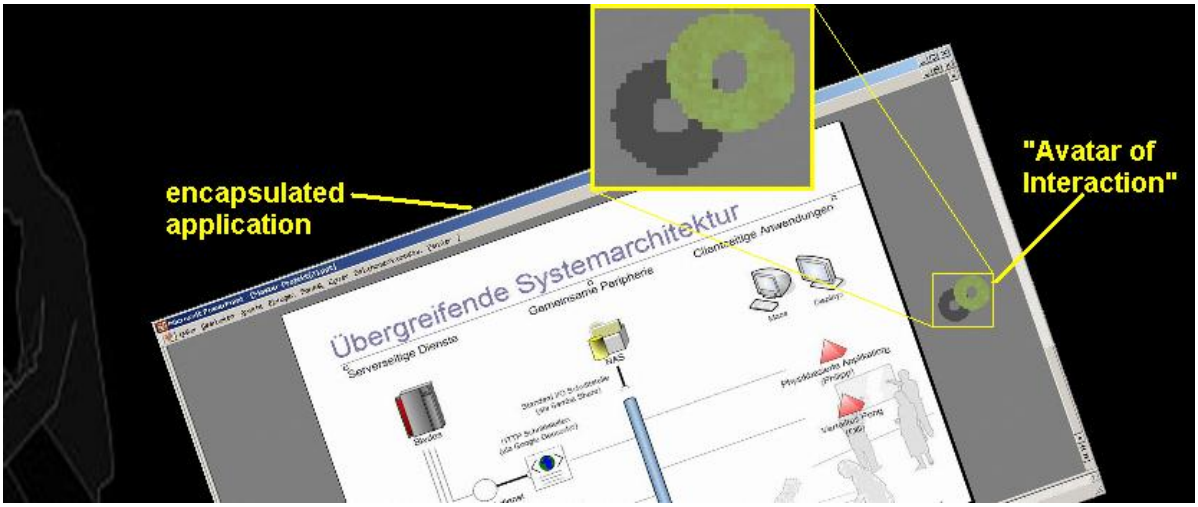

Abbildung 5.2: Der Avatar of Interaction zeigt sich in Form eines Donuts über den Anwendungsobjekten.

Um die „konkreten“ Eingabeinstrumente vom Einzelsystem zu entkoppeln, verbinden wir sie mit einer abstrakten Instanz. Diese Instanz bildet die Verbindung des Einzelsystems zu einer Eingabeeinheit, damit der Eingabekontext bewahrt wird. Der Eingabekontext ist zum Beispiel beim „cut & paste“ grundlegend für die Funktion. Weil diese Abstraktion in der systemübergreifenden Interaktionsschicht geschieht, werden die Eingabeeinheiten systemunabhängig. Durch die Instanziierung der Eingabeeinheiten in dem Interaktionsmodell ist es ebenfalls möglich, den Eingabeeinheiten bei der Repräsentation ein unterscheidbares Aussehen zu verleihen.

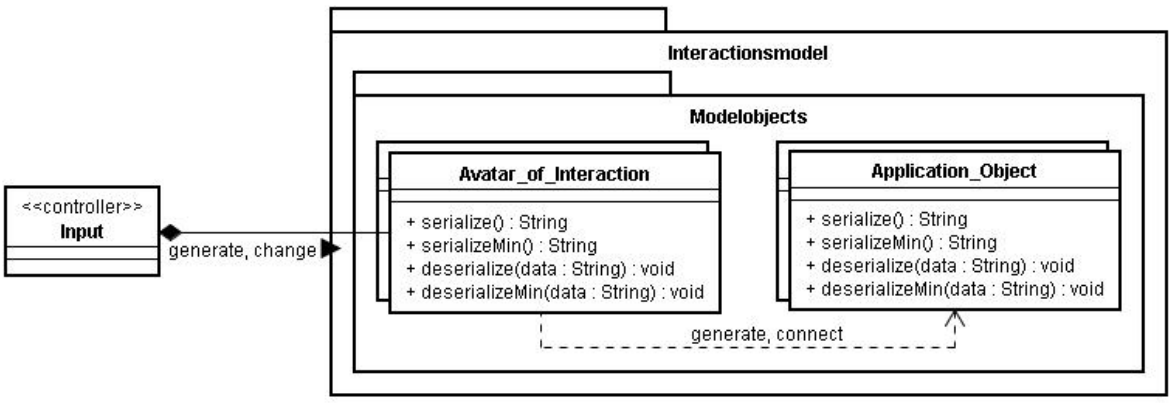

Abbildung 5.3: Das Eingabecontrollerobjekt generiert und verändert sein Alter Ego im Interaktionsmodell.

Bei der Implementierung wurde ein Mausobjekt instantiiert und diesem jeweils die Tastatur und

die im Bild zu sehen sind. Bei der Inszenierung der Stimme aus dem Off handelt es sich normalerweise um die Stimme des Erzählers, die Gedanken des Betrachters oder eine Stimme, die nicht nur dem Zuschauer aus dem Off erscheint, sondern auch den Personen im Film selbst.

die Maus eines Rechners zugeordnet. Dieses Eingabeobjekt ist Bestandteil des Interaktionsmodells und wird bei Statusänderung über die Distribution synchronisiert. Alle Eingaben des Benutzers durch Tastatur oder Maus verändern den Status des Objektes. Das Eingabeobjekt beinhaltet ebenfalls Form, Position und Größe innerhalb der abstrakten internen Darstellung mit der PhysX-Engine. Diese abstrakte Darstellung korrespondiert mit der visuellen Darstellung, die dem Benutzer zum Feedback sichtbar gemacht wird.

Die interne Darstellung und damit das Verhalten des Eingabeobjektes gleicht dem Verhalten eines Puks. Die Repräsentation des Eingabeobjektes ist, wie in Abbildung 5.2 zu sehen, einem Donut nach empfunden. Die Unterscheidbarkeit der Eingabeobjekte wird durch die Farbe realisiert. Die Form des Donuts passt zu den Eigenschaften eines Puks und lässt die Sicht auf die darunter liegenden Objekte zu. Eine optische Ausrichtung des Eingabeobjektes wurde für den ersten Prototypen explizit durch die Form des Donuts herausgenommen, weil sich die Orientierung beim Bewegen über die Interaktionslandschaft ändern kann. Die Implementierung dieses Parameters ist wissentlich auf spätere Prototypen verschoben worden.

Die Interaktion des Eingabeobjektes mit anderen Objekten im „collective“-Modus wird über die Funktionalität der PhysX-Engine abgebildet. Dieses geschieht über „joints“ zwischen den interagierenden Objekten. Im ersten Prototypen wird, wie in Abbildung 5.8 zu sehen, nur ein „joint“ gesetzt. Es ist aber ratsam, mehrere „joints“ zu setzen, um die Realität besser und damit intuitiver nachbilden zu können.

Die Verbindung und Subsumierung der Eingabeinstrumente zu einer Eingabeeinheit gibt diesen eine Autonomie und damit eine Art Existenz innerhalb des Gesamtsystems. Ebenfalls symbolisiert und implementiert das Eingabeobjekt selbst die Intentionen des Benutzers, die handelnde stellvertretene Instanz des Benutzers und einen Status für die Benutzung selbst. Diese Merkmale ähneln stark einem „Avatar“ aus dem Computerspielbereich (vgl. (Boberg u. a., 2008)). Wir nennen das Eingabeobjekt deshalb „Avatar of Interaction“ (AoI). Diese Namensgebung geschieht nicht nur aus dem Grund einen ansprechenden Namen für den Stellvertreter zu finden, sondern in erster Linie, um dem Eingabeobjekt Entwickler und Benutzer gegenüber eine umfassende Bedeutung[7] mit zu geben.

5.3.3 Die visuelle Repräsentation des Interaktionsmodells

Für den Prototypen wurde erst einmal nur ein Typ von Visualisierung implementiert. Diese Umsetzung basiert auf der 3D-Grafikengine Irrlicht, welche dreidimensionale Objekte anzeigt und mit Texturen belegen kann. Diese Objekte repräsentieren die zusammengesetzte Arbeitsoberfläche, die Anwendungsobjekte und die Objekte für die Eingabeeinheiten.

[7] Im Gegensatz zu Computerspielen soll hier nicht erreicht werden, dass sich der Spieler mit dem Avatar identifiziert, sondern dass der Avatar die Intentionen und Handlungen des Benutzer repräsentiert, deshalb „Avatar of Interaction“. Es findet damit keine emotionale Bindung statt. Deswegen und wegen der nicht individuellen Gestaltung des AoI können Benutzer ungehindert zwischen den AoIs wechseln.

Mit Irrlicht und der Software-Bibliothek libavcodec ist es möglich, Videostreams anstelle von Texturen auf die Oberflächen von Objekten zu legen. Damit kann der visuelle Inhalt einer gekapselten Anwendung über die Schnittstelle „show-content“ auf das stellvertretende Objekt projiziert werden, um dem Gesamtkonzept aus Abbildung 4.28 gerecht zu werden. Die Komponente „applicationcapsule“ muss lediglich einen Videostream generieren, der von dem visuellen Objekt und damit von der Repräsentationskomponente abgerufen werden kann.

Für den Testaufbau ist es nötig, dass jeder Rechner einen 3D-Grafik-Beschleunigungchip[8] bereitstellt, damit die Berechnung der Anzeige nur einen Teil der Berechnungsressourcen beansprucht. Bei einer Professionalisierung der Interaktionsschicht und dem Wissen über die Beanspruchung und Aufteilung der Ressourcen der Systeme, können die Belastungen eingeschätzt werden und eine entsprechende Anzeigeform gewählt werden. Wegen der einheitlichen und anspruchsvollen Umsetzung sollte der Berechnungsaufwand der Präsentation keine Rolle spielen.

Die funktionale Repräsentation des Interaktionsmodells geschieht durch die PhysX-Engine. Die visuelle Repräsentation geschieht durch die Irrlicht-Engine. Beide Engines haben eine interne Repräsentation, die durch eine gegenseitige Umrechnungsmatrix vollzogen wird. Diese Umrechnungsfunktion inklusive der Initialisierung der beiden Engines wurde aus dem Prototypen „dynAmbient“, der im Rahmen der Masterarbeit (Roßberger, 2008) entstanden ist, entnommen.

Abbildung 5.4: Jede Destopkachel wird einem Viewport (Kameraobjekt) zugeordnet.

[8] Mit 3D-Beschleunigung ist nicht die State-of-the-Art-Beschleunigung gemeint. Die einfachste 3D-Beschleunigungstechnik wie zum Beispiel die Onboard-Grafikprozessoren „NVIDIA GeForce 9300“ oder „ATI Radeon HD 3200“ performen auch mobil ausreichend.

Die einzelnen Monitore zeigen die jeweiligen Ausschnitte aus der Interaktionslandschaft, die dem korrespondierenden Desktopobjekt des Einzelsystems entsprechen. Festgelegt wird dieser Ausschnitt durch das Setzen eines Kameraobjektes, wie in Abbildung 5.4 zu sehen, innerhalb des Interaktionsmodells bzw. der Irrlichtengine. Desktopobjekt und Kameraobjekt bilden ein Paar. Verschiebt sich das Desktopobjekt im Raum, so tut dies auch das Kameraobjekt.

5.3.4 Umsetzung der Verteilung

Beim ersten Prototypen wurde erst einmal davon ausgegangen, dass es sich beim Gesamtsystem um ein homogenes handelt. Deshalb ist es nicht nötig, einen großen Funktionsumfang zur Übersetzung und Anpassung der Sichtweisen auf das Interaktionsmodell in der Verteilung zu realisieren. Der einfache Transfer des Modells soll hier ausreichend sein. Die Komponenten Transmitter und Receiver werden deshalb zu einer Komponente Distribution zusammengefasst. Die eigentliche Komponente Distribution aus Abbildung 4.17 entfällt erst einmal, weil diese Funktionalität durch das Einlesen einer Konfigurationsdatei gelöst wird.

Um die Kommunikation zwischen den Komponenten Distribution und Interaktionsmodell zu verringern wurde eine Vererbungsstruktur wie in Abbildung 5.5 zu sehen eingeführt. Dabei muss jede Klasse eines Objektes, das verteilt werden soll, von der Klasse Distributed_Objekt abgeleitet werden. Die dazu gehörigen abstrakten Funktionen zur Serialisierung müssen jeweils implementiert werden. Der Status des Objektes zur Distribution wird erst einmal auf default Werte gesetzt. Dieser kann aber genutzt werden um die Distribution zu beeinflussen. Beinhaltet ein Objekt andere Objekte und sollen diese auch mit verteilt werden, so müssen sie sich auch serialisieren können und ebenfalls die abstrakten Funktionen zur Serialisierung implementieren. Weil der Verteilungsmechanismus des besitzenden Objektes benutzt wird, wurde die Kaskadierung innerhalb der Serialisierung implementiert. Verweist das abgeleitete Objekt auf sich selbst oder auf ein anderes abgeleitetes Objekt, dann muss sicher gestellt sein, dass der Status „active“ nur bei dem ersten besitzenden Objekt einer Reihe auf den Wert „true“ und bei allen eingebetteten Objekten auf „false“ gesetzt wird, damit jedes Objekt nur einmal übertragen wird und die Struktur[9] der Objekte mit verteilt wird.

Bei der Deserialisierung werden dann aus den Abschnitten der Zeichenfolgen jeweils auf Instanzebene die Objekte des Baumes produziert bzw. bei ihrem Vorhandensein rekonfiguriert. Die Deserialisierung wie auch die Serialisierung kann hierbei auch rekursiv angewendet werden.

Da sich die Objekte im Prototypen zu Überprüfungszwecken zu einer komplexen xml-Zeichenkette serialisieren, trat schnell der Fall auf, dass große zu übertragene Datenmengen entstanden sind, die den Ablauf signifikant störten. Um die Gesamtmenge der zu übertragenden Datenmenge zu verringern, wurde auch ein minimales Serialisieren der Objekte eingeführt. Die minimale Serialisierung beinhaltet nur die Veränderungen des Objektes, nicht aber die des gesam-

[9] Die meisten Lösungen für diese Problematik arbeiten mit eindeutigen Objekt IDs, über die die Struktur wieder aufgebaut wird. Hier wurde die folgende Idee verfolgt: Serialisiere ein Objekt und damit alle assoziierten. Die Verwaltung löst sich rekursiv. Die Performance sinkt aber mit steigender Komplexität der Struktur.

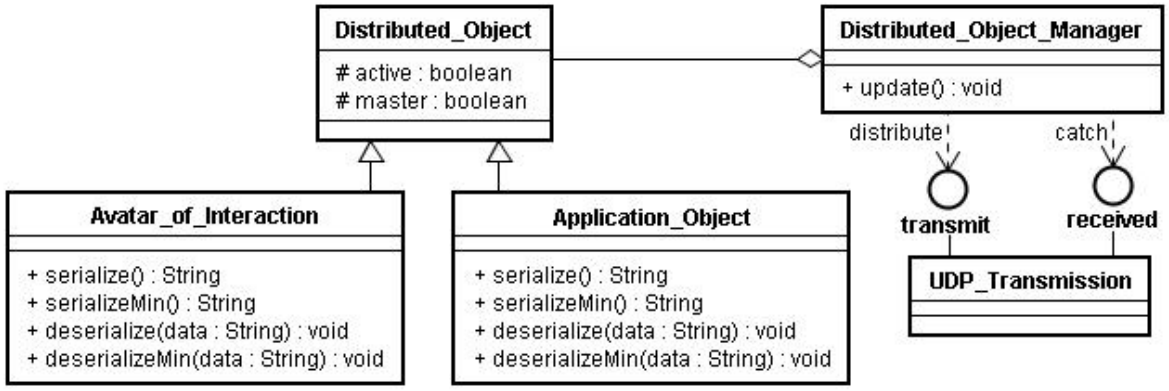

Abbildung 5.5: Die Verbindung zwischen Interaktionsmodell und Distribution wird über Vererbung gelöst.

ten Objekts. Damit ist eine komprimierte Übertragung über einen Stream, ähnlich der komprimierten Videoübertragung mit „key-frames“ als vollständige Serialisierung und „inter-frames“ als minimale Serialisierung wie sie beim MPEG-Video-Streaming (Le Gall, 1991) angewendet werden, möglich.

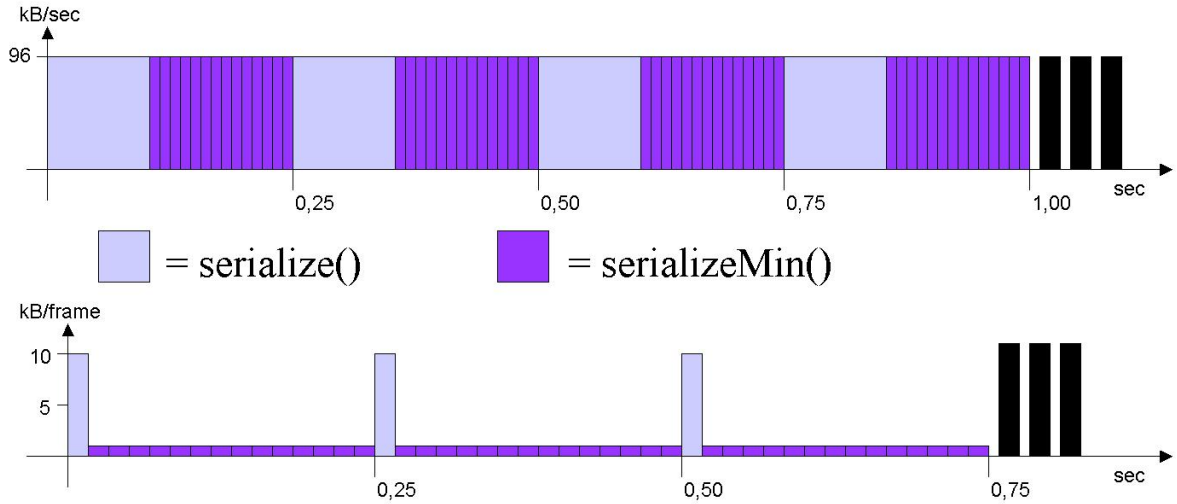

Abbildung 5.6: Die Datenübertragungsrate würde ohne die Komprimierung in diesem Beispiel das sechsfache betragen und die mögliche Framerate bei beschränkter Kanalkapazität ein sechstel.

Um einen fortlaufenden Objektstrom zu erreichen, werden zyklisch alle Distributionsobjekte übertragen. Dies passiert je nach Auslastung der Übertragungswege in verschiedenen Mischungsverhältnissen von minimalen und vollständigen Serialisierungen. Dieses Vorgehen garantiert im Fall eines Übertragungsfehlers das Korrigieren des Gesamtstatus innerhalb eines Zyklus. Gleichzeitig minimiert es die Übertragungsmenge auf das Wesentliche.

Die eigentliche Datenübertragung wird durch die Klasse UDP_Transmission realisiert. Mit die-

ser Trennungslinie wird die Übertragungsmethode von dem Rest der Verteilung gelöst. Damit ist es möglich, mit dem ersten Prototypen verschiedene Übertragungsmodule wie in Abschnitt 4.6 besprochen zu nutzen und zu evaluieren.

5.3.5 Schwierigkeiten

Während der Implementierung sind wir auf drei Hauptprobleme gestoßen. Erstens: Um ein fertiges funktionierendes Interaktionsmodell und deren Visualisierung nutzen zu können, wurde sich entschieden, die Umrechnungsfunktionalität aus (Roßberger, 2008) zu nutzen. Diese Lösung arbeitet mit der Irrlicht Version 1.3.1. und der PhysX-Engine Version 2.7.2.. Bei der Irrlicht Version 1.3.1. wurde für die Basisklasse der Name IUnknown verwendet. Dieser Name wird ebenfalls in windows.h benutzt, was unweigerlich zum Konflikt führt. windows.h wird von der winsock.h eingebunden und diese ist Grundlage für die Distribution über UDP oder TCP. In den neueren Versionen von Irrlicht wurde die Klasse IUnknown in IReferenceCounted umbenannt, um diesen und andere Konflikte zu lösen. Es wurden zwischen der Version 1.3.1 und 1.4 aber auch in der Klasse irrlichtEventReceiver zur Verwendung von Events eine neue Datenstruktur für Events eingeführt. Diese Datenstruktur zieht sich durch das gesamte Projekt und hätte eine Neuimplementierung erfordert. Aus diesem Grund wurde die Irrlicht Version 1.3.1 projektspezifisch verändert. Im gesamten Irrlichtprojekt wurde die Klasse „IUnknown“ in IReferenceCounted umbenannt und eine personalisierte Version 1.3.1.IL erstellt. Diese Problematik und die darauf folgende Lösung bezeichnet den prototypischen Charakter von Irrlicht, weil das Irrlichtprojekt noch nicht mit anderen Standardlösungen harmoniert.

Das zweite größere Problem zeigte sich bei der Nutzung der Serialisierung und Deserialisierung von Irrlichtobjekten. Ein scheinbar positiver Aspekt, der für die Nutzung von Irrlicht sprach, war der vorhandene Serialisierungsmechanismus von Irrlichtobjekten. Das Entwicklerteam von Irrlicht entwickelte aber keine allgemeingültige Serialisierungfunktionalität, sondern eine für das Einlesen von Objektkonfigurationen in die Irrlicht-Engine in Form von verschiedenen Formaten. Sie legten dabei keinen Wert auf nutzbare Zwischenergebnisse, die über Schnittstellen weitergereicht werden, sondern implementierten die Deserialisierung so problemspezifisch, dass eine Wiederverwendung des Codes sehr umständlich war und sich die Änderungen durch die gesamte Kaskadierung der Klassen zog. Eine Serialisierung war zwar angedacht aber nicht implementiert. Zusätzlich wurde die entwickelte Deserialisierung über die verschiedenen Versionen von Irrlicht verändert, so dass Serialisierung und Deserialisierung ein unsymetrisches Grundmuster innerhalb der Klassen aufweisen.

Das Problem wurde ebenfalls durch Personalisierung der Irrlicht-Engine gelöst. Bei dieser Personalisierung handelt es sich nicht wie beim ersten Problem um das gesamte Projekt, sondern nur um die veränderten Irrlichtklassen: „ISceneNode“, „CAttributes“ und „CFileSystem“. Wobei die gesamte Serialisierungsfunktionalität mit Hilfe von Fragmenten der vorhandenen Deserialisierung neu implementiert wurde. Abbildung 5.7 zeigt die Abhängigkeiten der Klassen und die Erweiterungen der Irrlicht-Engine.

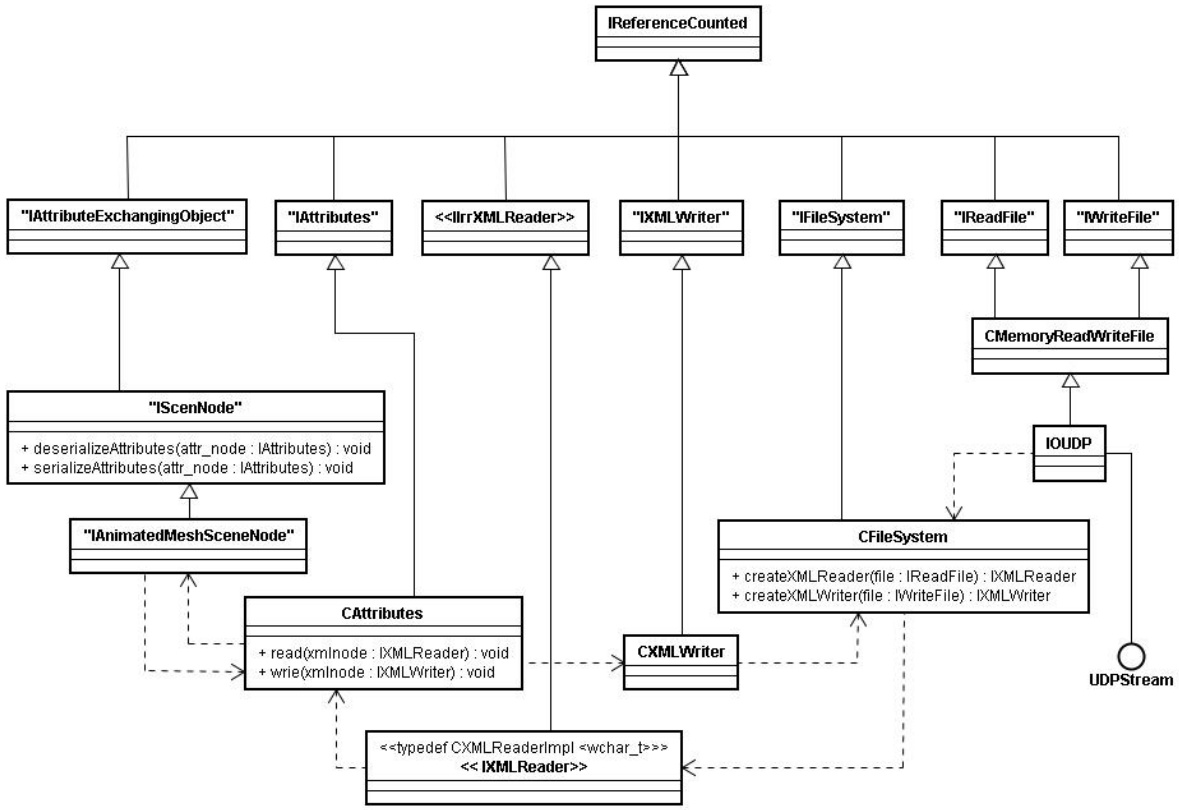

Abbildung 5.7: Die erweiterte Serialisierungsmechanik in Irrlicht.

Die letzte größere Problematik betrifft die sinnvolle Parametrisierung und Nutzung der PhysX-Engine. Eigentlich soll das Konzept der Zuhilfenahme eines physikbasierten Interaktionsmodells dazu führen, dass sich der Entwickler nicht um das Verhaltensmodell kümmern muss. Diese Lösung gestaltete sich zwar in erster Linie einfach, aber im Nachhinein nicht immer als sinnvoll.

Damit sich Objekte mit Hilfe einer Physikengine verbinden können, um miteinander zu Interagieren, werden diese durch so genannte „joints“, wie in Abbildung 5.8 zu sehen, verbunden. Dabei werden beide Objekte durch einen Joint, ähnlich einem Gummiband, verbunden. Niemand würde auf die Idee kommen, physikalische Objekte mit Hilfe von einem Gummiband durch die Gegend zu ziehen. Erstens dreht sich das Objekt ständig hin und her und zweitens tritt beim Ziehen immer eine Verzögerung der wirkenden Kräfte ein. Um ein gute funktionierendes Verhaltensmodell zu erhalten, müsste man entweder Glück beim Ausprobieren der Parameter haben, die Parameter über eine empirische Studie herausfinden oder komplizierte Differentialgleichungen lösen.

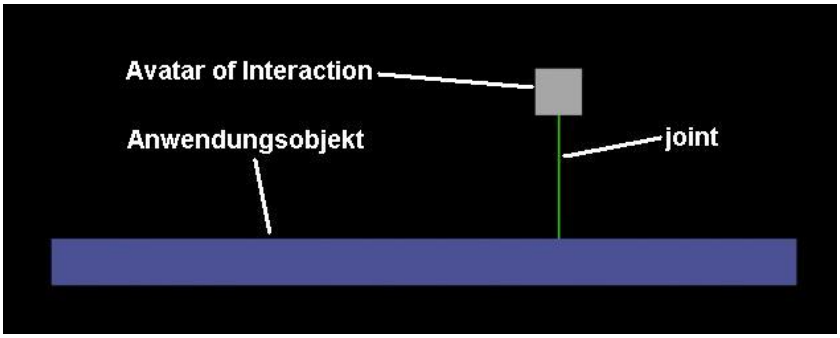

Abbildung 5.8: Bei einer Physikengine werden zwei Objekte durch einen „joint“ verbunden.

Auch die Metapher von dem Finger auf einem Blattpapier passt nicht, denn der Finger geht zahllose recht feste Verbindungen mit dem Papier ein. Den Parameter der Festigkeit des Joints zu erhöhen, erhöht leider auch das Aufschwingen. Das kann nur durch eine erhöhte Reibung eingedämmt werden. Eine Verbesserung könnte man erreichen, wenn die Verbindung von zwei Objekten durch eine Vielzahl von verschieden positionierten Joints vollzogen wird. Dieser multiple Joint müsste innerhalb des Verhaltensmodells (PhysX) implementiert werden oder die AoI-Objekte müssten aus mehreren Objekten zusammengesetzt werden, die dann verschiedene Joints bilden können.

Die restlichen Parameter der Physikengine müssen durch aufwendige Benutzerstudien herausgefunden werden. Aufwendig deshalb, weil so viele Unbekannte in Wechselwirkung sind und damit eine große Menge an Parametereinstellung überprüft werden müssten.

Weitere Probleme sind z.B. durch die Distribution über Streams entstanden. Weil Übertragungsfehler nicht im Übertragungsprotokol erkannt und korrigiert werden, musste die Modellstruktur so angepasst werden, dass sie Fehler erkennt und korrigiert. Hier wurde zum Beispiel die Funktion „deleteZombieObjects“ eingeführt. Diese Funktion überprüft, ob Löschungen von

Objekten eventuell im Stream verloren gegangen sind und löscht dann die nicht gelöschten Objekte explizit.

5.4 Fazit und Ausblick zum Prototypen

Der Prototyp wurde nach den eingegrenzenden Vorgaben aus Abschnitt 5.1 implementiert. Er ist lauffähig und funktioniert auf handelsüblichen Rechnern unter Windows XP recht gut. Der während der Entwicklungsphase genutzte Testaufbau[10] hat 60 Frames pro Sekunde wie in Abschnitt 4.4.4 gefordert generiert. Es gibt zwei Faktoren, die die Framerate einschränken. Einerseits spielt die grafische Berechnungsleistung eine Rolle und andererseits die Übertragungsrate bzw. -menge. Beide Faktoren treten dabei in Wechselwirkung und beeinflussen somit die Framerate. Werden grafische Berechnung und Verteilung auf der CPU abgewickelt, gehen mehr UDP Pakete verloren. Ohne eingehende Veränderungen ist keine Neuberechnung der Ansicht nötig. Dies verringert indirekt die mögliche Framerate. Wird die grafische Berechnung auf einem seperaten Grafikchip erledigt, gehen weniger UDP-Pakete verloren und ermöglicht eine höhere Framerate seitens der Verteilung. Bezogen auf diese Abhängigkeit von Modell und Anzeige und der damit verbundenen Leistung des Systems kann feststellend gesagt werden, dass es bei dem hier entwickelten Prototypen sowie bei dem Design zwei interne Frameraten gibt, die nicht ohne weiteres entkoppelt werden können. Einen neuen Anzeigeframe mit gleicher Information zu generieren ist überflüssig und ständig neue Zustandsframes zu generieren, ohne diese Anzeigen zu können ebenfalls. Das Load Balancing hängt bei diesem Prototypen von den jeweiligen Hardwarekomponenten ab und passt sich nicht an veränderte Voraussetzungen an.

Während der Entwicklung haben sich, wie in Abschnitt 5.3.5 geschildert, verschiedene individuelle Implementierungslösungen innerhalb des Prototypen etabliert. Diese bilden eine Grenze für die Weiterentwicklung. Ein Beispiel: Es wurde eine personalisierte Irrlicht Version 1.3.1L erstellt. Durch die ständige Weiterentwicklung der Irrlicht-Engine werden Fehler bereinigt und Funktionen und Konzept weiterentwickelt. Wie geschildert verändert sich die API der Irrlicht-Engine so stark, dass eine Anpassung immer einen Kompromiss und eine große Fehlerquelle beinhaltet. Zusätzlich scheint der Aufwand der Anpassung auf die neueren Versionen größer als eine komplette Neuentwicklung zu sein. Aus diesem Grund ist es ratsam bei weiterem Vorgehen noch einmal von Anfang an mit den aktuellsten Versionen zu beginnen und die schon implementierten Funktionalitäten und Objekte neu überarbeitet einzufügen. Diese Revision garantiert einen fehlerfreieren und ausgereifteren Prototypen II, der dann gut erweitert werden kann.

Es zeigte sich, dass die gemeinsame Interaktion mit dem Prototypen gut möglich ist. Deshalb sollte der vorgeschlagene Prototyp II durch folgende Erweiterungen ergänzt werden. Erstens sollte eine globale Beobachtungs- und Protokolliereinheit eingefügt werden. Diese würde eine

[10] Der minimale Testaufbau bestand aus zwei Rechnern mit jeweils einem 2.0 GHz Dual Core Prozessor, einem 3D-Grafikchip der Klasse „NVIDIA GeForce 9300“ oder „ATI Radeon HD 3200“, einem Monitor mit einer WXGA Auflösung, einer Tastatur, einer Maus bzw. Trackpoint und einer drahtlosen IEEE 802.11g Netzwerkverbindung.

fundierte Evaluation ermöglichen. Zweitens sollte eine sich selbst konfigurierende Einheit für die Konfiguration der gemeinsamen Arbeitsfläche eingefügt werden, damit ein dynamisches Konfigurieren des Systems möglich ist. Drittens sollte die Anwendungskapsel implementiert werden, damit ein produktives Arbeiten mit dem System möglich ist. Viertens sollte das Interaktionskonzept und damit die Nutzung der PhysX-Engine überdacht und erweitert werden. Fünftens sollte die Kopplung von Multi-Touch und Avatar of Interaction implementiert werden, um ein komplexeres Interagieren mit dem Tabletop zu ermöglichen. Sechstens sollte das eben vorgeschlagene Load-Balancing integriert werden, damit die Komponenten eines Versuchsaufbaus variieren können, ohne die Software oder deren Konfiguration zu ändern.

Sollte das Projekt sich etablieren, könnte durch die Erweiterung von nativen Anwendungen und einer dann möglichen Anbindung von Services ein Paradigmenshift in der Interaktion möglich werden. „Guerilla-Interaction“ könnte ein passendes Wort für diese Art der Interaktion sein, um eine möglich Benennung vorweg zu nehmen. Die Einbindung eines „Indoor-Positioning-Systems“ (IPS) könnte eine benötigte Dynamik innerhalb der Arbeitsflächenkonstruktion möglich machen. Eine Erweiterung des „Avatar of Interaction“ um eine abstrakte positionsgebundene Verbindungsmöglichkeit zu Personen, die jeweils Verbindungen zu den Standard AoIs aufnehmen können, würde eine übergreifende personengebundene Nutzung von Objekten und AoI Eigenschaften möglich machen. Dafür müsste eine personengebundene Identifikation und Position möglich sein, damit der unexistente AoI dem Besitzer räumlich über die Systeme folgen kann.

6 Fazit

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde der erste Prototyp WeFace v0.1 entwickelt. Dieser baut auf ein in dieser Arbeit entwickeltes Software-Design auf. Gleichzeitig wird dadurch das Software-Design evaluiert. Grundlage für das Software-Design war die anfängliche Analyse der Themenschwerpunkte „Co-located Collaborative Workspaces“ und „Handlungsräume bei Gruppenarbeit“.

Die Einführung in das Themengebiet in Kapitel 2 beschreibt den Anwendungs- und Analysekontext für das zu entwickelnde Software-Design. Zuerst wurde der Betrachtungsbereich vom allgemeinen Computer-Supported-Collaborative-Work hin zu der gemeinsamen gleichzeitigen Arbeit am selben Ort mit Hilfe von Computern konkretisiert. Anschließend wurden die zwei grundlegende Vertreter von Co-located Collaborative Workspaces „iRoom“ und „Roomware - The Second Generation“ in Design und Erscheinung dargestellt.

Die Analyse des zuvor dargestellten Forschungsbereichs in Kapitel 3 besteht aus vier Teilen und beginnt mit der Analyse von drei verschiedenen Sichtweisen auf Handlungsräume von Personen während einer Gruppenarbeit. Aus der detaillierten Betrachtung der verschiedenen Herangehensweisen folgt die Unterscheidung von „kollektiven Handlungen“ und „individuellen Handlungen“. Diese Unterscheidung wurde in der Analyse angewendet, um die funktionalen Anforderungen zu klassifizieren. Im Design und später auch in der Realisierung wurde die Ebene für Anwendungsobjekte auf Grundlage dieser Unterscheidung in die zwei Ebenen „kollektiv“ und „individuell“ unterteilt.

Im zweiten Teil der Analyse wurde ein Szenario entworfen und betrachtet. Hier wurden ersteinmal Use-Cases aus dem Szenario extrahiert, um sie im Anschluss zu funktionalen Anforderungen zusammen zu fassen. Im dritten Teil der Analyse wurden ausgewählte Anforderungen aus anderen Forschungsarbeiten beleuchtet und bewertet. Diese wurden entweder als gewollt oder nicht gewollt deklariert und gehen entsprechend in das Design ein. Am Ende der Analyse wurden die funktionalen Anforderungen aus dem zweiten Teil mit Hilfe des Ergebnisses aus dem ersten Teil klassifiziert. Zur zwingend zu integrierenden Klasse gehören die drei funktionalen Anforderungen „Objekt verschieben“, „Objekt skalieren“ und „Objekt orientieren“. Ebenfalls klassifiziert wurden die Anforderungen aus den anderen Forschungsarbeiten. Zu den zuerst zu implementierenden Anforderungen gehören „Configuring“, „Überbrückung von Displayzwischenräumen durch Stitching“ und „Assoziation von Eingabegeräten“. Die Klassfizierungen

wurden später im Design und Realisierung dazu genutzt, eine Unterscheidung zwischen den zu integrierenden Anforderungen zu machen.

Im Hauptteil der Arbeit in Kapitel 4 wurde das Software-Design für „WeFace“ in sechs Schritten entwickelt. Im ersten Schritt wurde festgelegt, dass die Interaktionsschicht sich zwischen System und Ein- und Ausgabe einnisten soll. In den folgenden zwei Schritten wurden Leitlinien für ein funktionales und ein strukturelles Modell aufgestellt. Diese Leitlinien wurden später in die Realisierung mit eingebracht. Im vierten Schritt wurde wie in Abbildung 4.17 gezeigt eine minimale Architektur mit Komponenten und Schnittstellen entwickelt, beschrieben und evaluiert, die den wichtigsten Anforderungen aus der Analyse gerecht wird. Im fünften Schritt wurde wie in Abbildung 4.28 zu sehen das minimale Design so erweitert, dass es auch den übrigen Anforderungen gerecht wird. Im sechsten Schritt wurde das vorgeschlagene Software-Design mit anderen Architekturen abgeglichen. Das Kapitel 4 liefert uns eine minimale und eine komplette Software-Architektur mit zu betrachtenden Leitlinien und einer Aussicht auf die Integration einer übergreifenden Architektur wie Abbildung 4.32 zeigt.

Im Kapitel 5 wurden verschiedene Aspekte der Entwicklung des Prototypen dargestellt. Es wurde evaluiert, welche Technik eingesetzt werden soll und welche nicht. Die Umsetzung der realen Komponenten mit ihren individuellen Lösungen wurde beschrieben. Hierzu gehören z.B. die Umsetzung des Verteilungmechanimus durch eine Vererbungsstruktur und die Serialisierung oder das Senken der Übertragungsrate durch Einführen eines minimalen Serialisierens und Übertragens ähnlich dem MPEG-Video-Streaming mit k-frames und i-frames. Durch die Beschreibung der aufgetretenen Schwierigkeiten wurde auf Hindernisse hingewiesen, die bei einer Realisierung auftreten können. Am Ende wurde mit der Erfahrung der Implementierung des ersten Prototypen erläutert, wie der Prototyp verbessert und erweitert werden kann. Zu den Erweiterungen gehören z.B. die Implementierung eines dynamischen Self-Configuring der Arbeitsflächen mit Hilfe eines Indoor-Positioning-Systems und die Erweiterung des „Avatar of Interaction“ um eine abstrakte positionsgebundene Verbindungsmöglichkeit zu Personen mit variabler Verbindung zu verschiedenen Standard Avatars of Interaction. Damit wird ermöglicht, dass Objekte einem Besitzer räumlich folgen können.

6.2 Beurteilung

Es ist kein Problem mit einem Rechner und vier Grafikkarten ein System mit acht Displays aufzubauen, bei dem sich eine einzige Arbeitsoberfläche bildet. Mit modernen virtuellen Maschinen wie VMware Fusion oder Parallels können auch Anwendungsfenster zwischen Windows und Mac Os ausgetauscht werden, so dass scheinbar ein hybrides System entsteht. Diese Techniken sind aber weiterhin auf nur einen aktiven Benutzer ausgelegt. Aus diesem Grund ist auch nur eine Perspektive möglich und die Interaktion mit vielen Benutzern auf der gleichen Arbeitsoberfläche wird nicht explizit unterstützt, da noch keine etablierten Konzepte für die Perspektivenangleichung der verschiedenen Benutzer und keine allgemeingültige Regelung für die Konflikte bei gleichzeitiger Manipulation von Daten existieren. Ebenfalls ungeklärt

ist die Frage nach der Verbindung von Arbeitsflächen. Diese Fragen wurden zwar schon teilweise beantwortet, lieferten aber bis heute nur problemspezifische Antworten. Alle Versuche, allgemeingültige Lösungen für die Problematiken zu finden, sind bisher gescheitert.

Das Konzept der WeFace-Software bietet eine Zusammenfassung und Neuinterpretation der geeignetesten Lösungen für die soziale Interaktion, für verschiedene Perspektiven, für eine Ressourcenwiederverwendung, für Bearbeitungskonflikte, Arbeitsflächenorganisation und sogar für die Lastverteilung von Arbeitsgeräten. Ein großer Faktor für den Erfolg des Konzeptes ist die Integration von sozialen Strukturen für die Systemorganisation und die gute Modularisierung des Systems für Anpassungen und Erweiterungen bei gleichzeitiger Strukturvorgabe.

6.3 Ausblick

Verschiedene Aspekte von bereits implementierten oder noch zu erweiternden Funktionsweisen von WeFace führen zu den folgenden Überlegungen und Möglichkeiten. Nach deren Beschreibung wage ich einen Blick in die Zukunft.

Durch Sichtweise aus verschiedenen Perspektiven auf die Benutzung von Handlungsräumen im ersten Teil der Analyse ist mit WeFace möglich, verschieden gelagerte Forderungen zu erfüllen. So wird die Forderung aus (Winograd und Flores, 1987), die Kommunikationsmöglichkeiten und damit die soziale Interaktion zu stärken, erfüllt. Das wiederum kann dazu führen, dass bei einer Untersuchung der produktiven Nutzung von WeFace neue Erkenntnisse gewonnen werden können und die Form und Art der Kommunikation differenzierter betrachtet werden kann.

Auch die Forderung aus (Wallace und Scott, 2008) nach einer technologieverbindenden und soziologischen Auseinandersetzung wird in dieser Arbeit erfüllt. Dies kann und soll nur der Anfang einer fächerübergreifenden Auseinandersetzung mit dem Thema der technologischen Unterstützung von gemeinsamer Interaktion sein. Um mehr Nachhaltigkeit zu gewinnen, müssten bei weiteren Arbeiten soziologische, neurowissenschaftliche, kommunikationswissenschaftliche, psychologische und informationstechnische Werkzeuge und Vorgehensweisen konsequent angewendet werden.

In (Nacenta, 2008) wird gefordert, die herkömmliche Sicht auf Displays zu überdenken. Die hier entworfene Interaktionsschicht führt eine neue Abstraktionsebene mit Eventmapping und Arbeitsflächenkomposition oberhalb der Pixelebene ein. Das Geschehen auf dem Display entspricht nicht mehr dem im Computer. Wo ein Display aufhört, hört die Interaktionsfläche noch lange nicht auf. Und trotzdem gibt es keine optische Unterbrechung und keinen Interaktionsshift wie z.B. bei der virtuellen Realität. Ob und wie sich die Sichtweise verändert, wird mit Einführung von WeFace noch nicht deutlich und wird sich erst bei intensiver Nutzung und Veränderung von WeFace heraus kristallisieren.

Die durch einen erweiterten Prototypen mögliche neue Interaktion ohne Grenzen in einem MDE gemeinsam zu interagieren erzeugt neue Erfahrungswerte. Diese Erfahrung erweitert

den Vorstellungsraum und damit die Fragestellungen und deren Antworten. So wurde z.B. die PhysX-Engine als Interaktionsmodell genutzt, was wegen der zu einfachen Simulation kritisiert wurde. Dies wird unweigerlich zu neuen Lösungen führen. Die Interaktionschicht mit ihren gut entkoppelten Komponenten bietet viel Spielraum um neue Ideen ausprobieren zu können.

Durch das Einbinden von vorhandenen Ressourcen und Fragmenten der Desktopmetapher in die Interaktionsschicht ist die sofortige produktive Nutzung möglich. Mit WeFace wird der Benutzer dort abgeholt, wo er sich aufhält. Gleichzeitig wird er durch die klare Trennung von individueller und kollektiver Ebene nicht dazu verleitet dort zu verharren. Er kann sein altes mentales Modell auf der individuellen Ebene weiter nutzen, wird aber auf der kollektiven Ebene ein neues aufbauen. Mit der Neubildung eines mentalen Modells zur gemeinsamen Interaktion und der Möglichkeit zur Integration von nativen Anwendungen ist es in Zukunft möglich, neue Ideen und Lösungen situationsabhängig zu realisieren und damit auch das neue mentale Modell weiter zu entwickeln.

Der erste Prototyp von WeFace ermöglicht gemeinsame Interaktionen und macht die Software evaluierbar. Nach Implementierung der Anwendungskapsel bietet das System eine gute Lösung, um mit vorhandenen Mitteln einen produktiven gemeinsamen Arbeitsplatz zu schaffen. Nach weiteren Beobachtungen der realen Benutzung von WeFace wird ein Bedarf nach angepassten Anwendungen entstehen. Mit der Implementierung des Anwendungsframeworks wurde eine umfangreiche und gute Lösung gefunden, um das Arbeiten und die soziale Interkation in Multi-Display-Environments oder auch Co-located Collaborative Workspaces zu unterstützen.

Es ist auch möglich, WeFace in anderen Kontexten als dem des Co-located Collaborative Workspace einzusetzen bzw. der Workspace kann beliebig in den Alltag und dessen Ausprägungen ausgedehnt werden. WeFace macht das Handling von Objekten in einer Umgebung mit vielen Rechnern und Displays allgemein möglich. Stellen wir uns vor, dass zahlreiche physikalische Objekte unseres Lebens als interaktionsfähige Anzeige funktionieren und diese eine relative Position zueinander erkennen und sich damit räumlich bedingt vernetzen können. Damit wäre es möglich, auf allen angereicherten physikalischen Objekten seine virtuellen Objekte zu nutzen und zu lagern. Die Rechner im Hintergrund organisieren sich selbst und stellen eine einzige Ebene auf den überall präsenten Interaktionsflächen zur Verfügung. Nicht die Geräte werden zu den Benutzern assoziiert, sondern die virtuellen Objekte. Damit wären nicht mehr Computer im Focus, sondern die virtuellen Objekte, die sich vorher auf den Rechnern befanden. Nicht der Rechner gehört mir, sondern das Objekt auf diesem. Die elektronische Infrastruktur würde damit zum öffentlichen Raum, in dem sich die Objekte frei bewegen können.

Abbildungsverzeichnis

Literaturverzeichnis

[Agarawala und Balakrishnan 2006] AGARAWALA, Anand ; BALAKRISHNAN, Ravin: Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In: *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. New York, NY, USA : ACM Press, 2006, S. 1283–1292. – ISBN 1-59593-372-7

[Artman u. a. 2005] ARTMAN, Henrik ; RAMBERG, Robert ; SUNDHOLM, Hillevi ; CERRATTO-PARGMAN, Teresa: Action context and target context representations: a case study on collaborative design learning. In: *CSCL '05: Proceedings of th 2005 conference on Computer support for collaborative learning*, International Society of the Learning Sciences, 2005, S. 1–7. – ISBN 0-8058-5782-6

[Ballagas u. a. 2004] BALLAGAS, Rafael ; SZYBALSKI, Andy ; FOX, Armando: Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. In: *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*. Washington, DC, USA : IEEE Computer Society, 2004, S. 241. – ISBN 0-7695-2090-1

[Baudisch u. a. 2004] BAUDISCH, Patrick ; CUTRELL, Edward ; HINCKLEY, Ken ; GRUEN, Robert: Mouse ether: accelerating the acquisition of targets across multi-monitor displays. In: *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2004, S. 1379–1382. – ISBN 1-58113-703-6

[Bengel 2008] BENGEL, Günther: *Masterkurs Parallele und Verteilte Systeme - Grundlagen und Programmierung von Multicoreprozessoren, Multiprozessoren, Cluster und Grid*. Wiesbaden, Germany : Vieweg + Teubner, 2008. – 503 S. – ISBN 978-3-8348-0394-8

[Biehl und Bailey 2006] BIEHL, Jacob T. ; BAILEY, Brian P.: Improving interfaces for managing applications in multiple-device environments. (2006), S. 35–42. ISBN 1-59593-353-0

[Biehl u. a. 2008] BIEHL, Jacob T. ; BAKER, William T. ; BAILEY, Brian P. ; TAN, Desney S. ; INKPEN, Kori M. ; CZERWINSKI, Mary: Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In: *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 2008, S. 939–948. – ISBN 978-1-60558-011-1

[Boberg u. a. 2008] BOBERG, Marion ; PIIPPO, Petri ; OLLILA, Elina: Designing avatars. In: *DIMEA '08: Proceedings of the 3rd international conference on Digital Interactive Media in*

Entertainment and Arts. New York, NY, USA : ACM, 2008, S. 232–239. – ISBN 978-1-60558-248-1

[Bowman 1999] BOWMAN, Douglas A.: *Interaction techniques for common tasks in immersive virtual environments: design, evaluation, and application*. Atlanta, GA, USA, Georgia Institute of Technology, Dissertation, Juni 1999. – URL http://people.cs.vt.edu/~bowman/thesis/thesis.pdf. – Zugriff: am 27.08.2010

[Burbeck 1992] BURBECK, Steve: *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. Webseite. 1992. – URL http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html. – Zugriff: 13.7.2010

[Burfeindt 2006] BURFEINDT, Lars: *Konstruktion einer Middleware für computergestützte Gruppenarbeit in ubiquitärer Systemumgebung*, Fakultät Technik und Informatikder Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, August 2006. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/burfeindt.pdf. – Zugriff: am 27.08.2010

[Dewan u. a. 1999] DEWAN, Prasun ; EHRLICH, Kate ; GREENBERG, Saul ; JOHNSON, Chris ; PRAKASH, Atul ; DOURISH, Paul ; ELLIS, Clarence ; ISHII, Hiroshi ; MACKAY, Wendy E. ; ROSEMAN, Mark ; BEAUDOUIN-LAFON, Michel (Hrsg.): *Computer Supported Co-operative Work*. John Wiley and Sons, 1999. – ISBN: 0-471-96736-X

[Dix 2001] DIX, Alan: *Mensch, Maschine, Methodik*. Upper Saddle River, NJ, USA : Prentice Hall PTR, April 2001. – ISBN 0131428985

[Dourish 1998] DOURISH, Paul: Using metalevel techniques in a flexible toolkit for CSCW applications. In: *ACM Trans. Comput.-Hum. Interact.* 5 (1998), Nr. 2, S. 109–155. – ISSN 1073-0516

[Dourish und Bellotti 1992] DOURISH, Paul ; BELLOTTI, Victoria: Awareness and coordination in shared workspaces. In: *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. New York, NY, USA : ACM, 1992, S. 107–114. – ISBN 0-89791-542-9

[Dustdar u. a. 2003] DUSTDAR, Schahram ; GALL, Harald ; HAUSWIRTH, Manfred: *Software-Architekturen für verteilte Systeme : Prinzipien, Bausteine und Standardarchitekturen für moderne Software*. Berlin Heidelberg, Germany : Springer-Verlag, 2003. – 264 S. – ISBN 3-540-43088-1

[Everitt u. a. 2006] EVERITT, Katherine ; SHEN, Chia ; RYALL, Kathy ; FORLINES, Clifton: MultiSpace: Enabling Electronic Document Micro-mobility in Table-Centric, Multi-Device Environments. In: *TABLETOP '06: Proceedings of the First IEEE International Workshop on Horizontal Interactive Human-Computer Systems*. Washington, DC, USA : IEEE Computer Society, 2006, S. 27–34. – ISBN 0-7695-2494-X

[Forlines u. a. 2006] FORLINES, Clifton ; ESENTHER, Alan ; SHEN, Chia ; WIGDOR, Daniel ; RYALL, Kathy: Multi-user, multi-display interaction with a single-user, single-display geospatial application. (2006), S. 273–276. ISBN 1-59593-313-1

[Gamma u. a. 1995] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0-201-63361-2

[Gottwald 2007] GOTTWALD, Michael: *Untersuchung von Interaktionsmöglichkeiten bei synchroner computergestützter Gruppenarbeit in heterogener Systemumgebung*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, November 2007

[Greenberg und Marwood 1994] GREENBERG, Saul ; MARWOOD, David: Real time groupware as a distributed system: concurrency control and its effect on the interface. In: *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*. New York, NY, USA : ACM, 1994, S. 207–217. – ISBN 0-89791-689-1

[Hebert und Chen 2005] HEBERT, Alan ; CHEN, Allan: A new collaborative software package: TeamSpace at Stanford University. In: *SIGUCCS '05: Proceedings of the 33rd annual ACM SIGUCCS conference on User services*. New York, NY, USA : ACM, 2005, S. 109–112. – ISBN 1-59593-200-3

[Horn 2001] HORN, Paul: AUTONOMIC COMPUTING: IBM's Perspective on the State of Information Technology / International Business Machines Corporation (IBM). New Orchard Road, Armonk, NY 10504, Oktober 2001. – Forschungsbericht. – URL http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf. Zugriff: am 27.08.2010

[Huebscher und McCann 2008] HUEBSCHER, Markus C. ; MCCANN, Julie A.: A survey of autonomic computing—degrees, models, and applications. In: *ACM Comput. Surv.* 40 (2008), Nr. 3, S. 1–28. – ISSN 0360-0300

[Ishii und Kobayashi 1992] ISHII, Hiroshi ; KOBAYASHI, Minoru: ClearBoard: a seamless medium for shared drawing and conversation with eye contact. In: *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 1992, S. 525–532. – ISBN 0-89791-513-5

[Ishii u. a. 1994] ISHII, Hiroshi ; KOBAYASHI, Minoru ; ARITA, Kazuho: Iterative design of seamless collaboration media. In: *Commun. ACM* 37 (1994), Nr. 8, S. 83–97. – ISSN 0001-0782

[Johansen u. a. 1991] JOHANSEN, Robert ; SIBBET, David ; BENSON, Suzyan ; MARTIN, Alixia ; MITTMAN, Robert ; SAFFO, Paul: *Leading Business Teams: How Teams Can Use Technology and Group Process Tools to Enhance Performance*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1991. – ISBN 0201528290

[Johanson u. a. 2002a] JOHANSON, Brad ; FOX, Armando ; WINOGRAD, Terry: The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. In: *IEEE Pervasive Computing* 1 (2002), Nr. 2, S. 67–74. – ISSN 1536-1268

[Johanson u. a. 2001] JOHANSON, Brad ; HUTCHINS, Greg ; WINOGRAD, Terry: *PointRight: Pointer/Keyboard Redirection for Interactive Workspaces*. Internet. September 2001. – URL http://graphics.stanford.edu/papers/pointrightubicomp/pointright_ubicomp.pdf. – Zugriff: 1.10.2010

[Johanson u. a. 2002b] JOHANSON, Brad ; HUTCHINS, Greg ; WINOGRAD, Terry ; STONE, Maureen: PointRight: experience with flexible input redirection in interactive workspaces. In: *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 2002, S. 227–234. – ISBN 1-58113-488-6

[Johanson 2003] JOHANSON, Bradley E.: *Application coordination infrastructure for ubiquitous computing rooms*. Stanford, CA, USA, Dissertation, 2003. – URL http://graphics.stanford.edu/~bjohanso/dissertation/johanson-thesis.pdf. – Zugriff: am 27.08.2008

[Karam 2006] KARAM, Maria: *PhD Thesis: A framework for research and design of gesture-based human-computer interactions*. October 2006. – URL http://eprints.ecs.soton.ac.uk/13149/. – Zugriff: 1.10.2010

[Kay 1998] KAY, Alan: *prototypes vs classes was: Re: Sun's HotSpot (Alan Kay about the term 'objects' and the big idea is 'messaging').*. Email. Oktober 1998. – URL http://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html. – Zugriff: am 27.08.2009

[Kay 2003] KAY, Alan: *Re: Clarification of 'object-oriented' (Dr. Alan Kay on the Meaning of 'Object-Oriented Programming')*. Email. July 2003. – URL http://www.purl.org/stefan_ram/pub/doc_kay_oop_en. – Zugriff: am 27.08.2009

[Kay 1972] KAY, Alan C.: A Personal Computer for Children of All Ages. In: *In Proceedings of the ACM National Conference*. Boston : Press, August 1972

[Köckritz 2007] KÖCKRITZ, Oliver: *Kollaborationsspiel für Collaborative Workspaces*. Seminararbeit an der Hochschule für Angewandte Wissenschaften Hamburg. März 2007

[Le Gall 1991] LE GALL, Didier: MPEG: a video compression standard for multimedia applications. In: *Commun. ACM* 34 (1991), Nr. 4, S. 46–58. – ISSN 0001-0782

[Manoel u. a. 2005] MANOEL, Edson ; NIELSEN, Morten J. ; SALAHSHOUR, Abdi ; SAMPATH, Sai ; SUDARSHANAN, Sanjeev: *Problem Determination Using Self-Managing Autonomic Technology*. First Edition. IBM Redbooks, Juni 2005. – URL http://www.redbooks.ibm.com/redbooks/pdfs/sg246665.pdf. – Zugriff: 1.10.2010

[Nacenta 2008] NACENTA, Miguel A.: Inventing the Future of Multi-Display Environments. In: *In: Proceedings of the CSCW 2008 workshop: Beyond the Laboratory: Supporting Authentic Collaboration with Multiple Displays*, November 2008

[Nacenta u. a. 2008] NACENTA, Miguel A. ; MANDRYK, Regan L. ; GUTWIN, Carl: Targeting across displayless space. In: *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 2008, S. 777–786. – ISBN 978-1-60558-011-1

[Nacenta u. a. 2007] NACENTA, Miguel A. ; SAKURAI, Satoshi ; YAMAGUCHI, Tokuo ; MIKI, Yohei ; ITOH, Yuichi ; KITAMURA, Yoshifumi ; SUBRAMANIAN, Sriram ; GUTWIN, Carl: E-conic: a perspective-aware interface for multi-display environments. In: *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 2007, S. 279–288. – ISBN 978-1-59593-679-2

[Nakashima u. a. 2005] NAKASHIMA, Kousuke ; MACHIDA, Takashi ; KIYOKAWA, Kiyoshi ; TAKEMURA, Haruo: A 2D-3D integrated environment for cooperative work. In: *VRST '05: Proceedings of the ACM symposium on Virtual reality software and technology*. New York, NY, USA : ACM, 2005, S. 16–22. – ISBN 1-59593-098-1

[Napitupulu 2008] NAPITUPULU, Jan: *Ein System mit skalierbarer Visualisierung zur Entwicklung kollaborativer Serious Games*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, Juni 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/napitupulu.pdf. – Zugriff: 1.10.2010

[Nastansky u. a. 2002] NASTANSKY, Ludwig ; BRUSE, Thomas ; HABERSTOCK, Philipp ; HUTH, Carsten ; SMOLNIK, Stefan: *Büroinformations- und Kommunikationssysteme: Groupware, Workflow Management, Organisationsmodellierung und Messaging-Systeme*. S. 235–322. In: FISCHER, Joachim (Hrsg.) ; HEROLD, Werner (Hrsg.) ; DANGELMAIER, Wilhelm (Hrsg.) ; NASTANSKY, Ludwig (Hrsg.) ; SUHL, Leena (Hrsg.): *Bausteine der Wirtschaftsinformatik - Grundlagen, Anwendungen, PC-Praxis*. Berlin : Erich Schmidt Verlag, 2002

[Oesterreich und Resch 1985] OESTERREICH, Rainer ; RESCH, Marianne: Zur Analyse arbeitsbezogener Kommunikation. In: *Zeitschrift für Sozialisationsforschung und Erziehungssoziologie* 5 (1985), S. 271 – 290

[Ponnekanti u. a. 2003] PONNEKANTI, Shankar R. ; JOHANSON, Brad ; KICIMAN, Emre ; FOX, Armando: Portability, Extensibility and Robustness in iROS. In: *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA : IEEE Computer Society, 2003, S. 11. – ISBN 0-7695-1893-1

[Preece u. a. 1995] PREECE, J. ; ROGERS, Y. ; SHARP, H. ; BENYON, D. ; HOLLAND, S. ; CAREY, T.: *Human-Computer Interaction*. Addison-Wesley Publishing Company, 1995

[Raasch 2006] RAASCH, Jörg: Usability von Anwendungssystemen - didaktische Aspekte. In: *HDI*, 2006, S. 23–36

[Rekimoto 1997] REKIMOTO, Jun: Pick-and-drop: a direct manipulation technique for multiple computer environments. In: *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*. New York, NY, USA : ACM, 1997, S. 31–39. – ISBN 0-89791-881-9

[Rekimoto 2000] REKIMOTO, Jun: Multiple-computer user interfaces: "beyond the desktop"direct manipulation environments. In: *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2000, S. 6–7. – ISBN 1-58113-248-4

[Rekimoto und Saitoh 1999] REKIMOTO, Jun ; SAITOH, Masanori: Augmented surfaces: a spatially continuous work space for hybrid computing environments. In: *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1999, S. 378–385. – ISBN 0-201-48559-1

[Ringel u. a. 2004] RINGEL, Meredith ; RYALL, Kathy ; SHEN, Chia ; FORLINES, Clifton ; VERNIER, Frederic: Release, relocate, reorient, resize: fluid techniques for document sharing on multi-user interactive tables. In: *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2004, S. 1441–1444. – ISBN 1-58113-703-6

[Roßberger 2007] ROSSBERGER, Philipp: *Entwicklung einer Anwendung zur physikbasierten Manipulation von Objekten*. Seminararbeit an der Hochschule für Angewandte Wissenschaften Hamburg. Februar 2007

[Roßberger 2008] ROSSBERGER, Philipp: *Physikbasierte Interaktion in kollaborativen computergestützten Umgebungen*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, Januar 2008. – URL http://www.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/rossberger.pdf. – Zugriff: am 27.08.2010

[Román u. a. 2002] ROMÁN, Manuel ; HESS, Christopher ; CERQUEIRA, Renato ; RANGANATHAN, Anand ; CAMPBELL, Roy H. ; NAHRSTEDT, Klara: Gaia: a middleware platform for active spaces. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 6 (2002), Nr. 4, S. 65–67. – ISSN 1559-1662

[Scherp und Boll 2005] SCHERP, Ansgar ; BOLL, Susanne: Paving the Last Mile for Multi-Channel Multimedia Presentation Generation. In: *MMM '05: Proceedings of the 11th International Multimedia Modelling Conference*. Washington, DC, USA : IEEE Computer Society, 2005, S. 190–197. – ISBN 0-7695-2164-9

[Schümmer u. a. 2000] SCHÜMMER, Jan ; SCHÜMMER, Till ; SCHUCKMANN, Christian: COAST - Ein Anwendungsframework für synchrone Groupware. In: *Proceedings of the net.objectDays*, 2000

[Scott 2005] SCOTT, Stacey D.: *Territoriality in collaborative tabletop workspaces*. Calgary, Alta., Canada, Canada, Dissertation, 2005. – URL http://portal.acm.org/citation.cfm?id=1123527. – Zugriff: am 27.08.2010

[Scott u. a. 2003] SCOTT, Stacey D. ; GRANT, Karen D. ; MANDRYK, Regan L.: System guidelines for co-located, collaborative work on a tabletop display. In: *ECSCW'03: Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*. Norwell, MA, USA : Kluwer Academic Publishers, 2003, S. 159–178

[Shafer u. a. 1998] SHAFER, Steve ; KRUMM, John ; BRUMITT, Barry ; MEYERS, Brian ; CZERWINSKI, Mary ; ROBBINS, Daniel: The new EasyLiving Project at Microsoft Research. In: *Proc. Joint DARPA/NIST Smart Spaces Workshop*, 1998, S. 30–31

[Shih und Winograd 2004] SHIH, Crone M. Fox A. ; WINOGRAD, T.: *Teamspace: A Simple, Low-Cost and Self-Sufficient Workspace for Small-Group Collaborative Computing*. Internet, CSCW 2004 Interactive Poster. November 2004. – URL http://hci.stanford.edu/research/teamspace_CSCW.pdf. – Zugriff: am 27.08.2010

[Slay und Thomas 2006] SLAY, Hannah ; THOMAS, Bruce: Interaction and visualisation across multiple displays in ubiquitous computing environments. In: *Afrigaph '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. New York, NY, USA : ACM, 2006, S. 75–84. – ISBN 1-59593-288-7

[Slay u. a. 2005] SLAY, Hannah ; THOMAS, Bruce ; VERNIK, Rudi: A clipboard model for Ubiquitous Computing Environments. In: *WISICT '05: Proceedings of the 4th international symposium on Information and communication technologies*, Trinity College Dublin, 2005, S. 173–178. – ISBN 1-59593-169-4

[Smith u. a. 2004] SMITH, David A. ; RAAB, Andreas ; REED, David P. ; KAY, Alan: Croquet: A Menagerie of New User Interfaces. In: *C5 '04: Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing*. Washington, DC, USA : IEEE Computer Society, 2004, S. 4–11. – ISBN 0-7695-2166-5

[Streitz u. a. 2002] STREITZ, Norbert ; PRANTE, Thorsten ; MÜLLER-TOMFELDE, Christian ; TANDLER, Peter ; MAGERKURTH, Carsten: Roomware: the second generation. In: *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*. New York, NY, USA : ACM, 2002, S. 506–507. – ISBN 1-58113-454-1

[Streitz u. a. 1999] STREITZ, Norbert A. ; GEISSLER, Jörg ; HOLMER, Torsten ; KONOMI, Shin'ichi ; MÜLLER-TOMFELDE, Christian ; REISCHL, Wolfgang ; REXROTH, Petra ; SEITZ, Peter ; STEINMETZ, Ralf: i-LAND: an interactive landscape for creativity and innovation. In: *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1999, S. 120–127. – ISBN 0-201-48559-1

[Szybalski 2004] SZYBALSKI, Andy: *Balancing Zero-Admin and Incremental Integration in UbiComp Enviroments*. may 2004. – URL http://andy.bigwhitebox.org/thesis.pdf. – Zugriff: am 27.04.2010

[Tan u. a. 2005] TAN, D. S. ; GERGLE, D. ; CZERWINSKI, M.: A Job-Shop Scheduling Task for Evaluating Coordination during Computer Supported Collaboration. In: *vol. TR-2005-107*. Redmond, WA 98052, USA : Microsoft Research, 2005. – Zugriff: 1.10.2010

[Tandler 2004a] TANDLER, Peter: The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments. In: *J. Syst. Softw.* 69 (2004), Nr. 3, S. 267–296. – ISSN 0164-1212

[Tandler 2004b] TANDLER, Peter: *Synchronous Collaboration in Ubiquitous Computing Environments*, TU Darmstadt, Fachbereich Informatik, Dissertation, 2004. – URL http://elib.tu-darmstadt.de/diss/000506. – Zugriff: 12.2.2010

[Tanenbaum und Steen 2002] TANENBAUM, Andrew S. ; STEEN, Martin v.: *Distributed Systems - Principles and Paradigms*. Pearson Education (Prntice Hall), 2002

[Töpfer 2006] TÖPFER, Guido R.: *Endnutzerprogrammierung auf Basis nachrichtenbasierter Komponentenvernetzung*, Johannes Gutenberg-Universität Mainz, Dissertation, 2006

[Wallace und Scott 2008] WALLACE, James R. ; SCOTT, Stacey D.: Towards Contextual Design Requirements for MDEs. In: *In: Proceedings of the CSCW 2008 workshop: Beyond the Laboratory: Supporting Authentic Collaboration with Multiple Displays*, November 2008

[Weber 1997] WEBER, Wolfgang G.: *Analyse von Gruppenarbeit - Kollektive Handlungsregulation in soziotechnischen Systemen*. Bern, Germany : Hans Huber, 1997

[Weiser 1994] WEISER, Marc: The world is not a desktop. In: *interactions* 1 (1994), Nr. 1, S. 7–8. – ISSN 1072-5520

[Weiser 1991] WEISER, Mark: The computer for the 21st century. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 3 (1991), Nr. 3, S. 3–11. – ISSN 1559-1662

[Weiss 2007] WEISS, Aaron: Desktops in 3D. In: *netWorker* 11 (2007), Nr. 1, S. 26–33. – ISSN 1091-3556

[Wigdor u. a. 2009] WIGDOR, Daniel ; JIANG, Hao ; FORLINES, Clifton ; BORKIN, Michelle ; SHEN, Chia: WeSpace: the design development and deployment of a walk-up and share multi-surface visual collaboration system. In: *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*. New York, NY, USA : ACM, 2009, S. 1237–1246. – ISBN 978-1-60558-246-7

[Winograd 2003] WINOGRAD, Terry: *Stanford Interactive Workspaces Project Overview*. Webseite. 2003. – URL http://iwork.stanford.edu/. – Zugriff: 1.10.2010

[Winograd und Flores 1987] WINOGRAD, Terry ; FLORES, Fernando: *Understanding Computers and Cognition: A New Foundation for Design*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1987. – ISBN 0201112973

[Winograd und Flores 1989] WINOGRAD, Terry ; FLORES, Fernando: *Erkenntnis Maschinen Verstehen: Zur Neugestaltung von Computersystemen*. Berlin, Germany : Rotbuch Verlag, 1989. – ISBN 388022-750-0

[Züllighoven u. a. 2004] ZÜLLIGHOVEN, Heinz ; BEEGER, Robert F. ; BLEEK, Wolf-Gideon ; GRYCZAN, Guido ; LILIENTHAL, Carola ; LIPPERT, Martin ; ROOCK, Stefan ; SIBERSKI, Wolf ; SLOTOS, Thomas ; WESKE, Dirk ; WETZEL, Ingrid: *Object-Oriented Construction Handbook*. dpunkt.verlag; Morgan-Kaufmann, Oktober 2004 (3-89864-254-2). – 544 S. – URL http://www.dpunkt.de/buch/3-89864-254-2.html. – Zugriff:1.10.2010

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 18. November 2010
Ort, Datum

Unterschrift