

**Entwicklung einer interaktiven
Möbelbauanleitung auf Mixed Reality
Basis**

ALEXANDER BRANDL

DIPLOMARBEIT

eingereicht am
Fachhochschul-Diplomstudiengang
MEDIEN-TECHNIK UND -DESIGN
in Hagenberg

im Juli 2003

© Copyright 2003 Alexander Brandl

Alle Rechte vorbehalten

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 10. Juni 2003

Alexander Brandl

Inhaltsverzeichnis

Erklärung	iii
Danksagung	vii
Kurzfassung	viii
Abstract	ix
1 Einleitung	1
1.1 Motivation und Anwendungsbereich	1
1.2 Zielsetzung	2
1.3 Aufbau der Arbeit	2
2 Grundlagen	3
2.1 Mixed Reality	3
2.2 ARToolkit	4
2.2.1 Kurzbeschreibung	4
2.2.2 Positionsbestimmung durch Markertracking	4
2.2.3 Probleme	4
2.3 Das AMIRE Framework	5
2.3.1 Kurzbeschreibung	5
2.3.2 Komponenten	6
2.4 Montageanleitungen	7
3 Methodik	10
3.1 Das AMIRE Framework als Grundlage	10
3.2 Funktionsumfang	10
3.2.1 Authoring Modus	11
3.2.2 Application Modus	13
3.2.3 Abgrenzung	16
3.3 Anforderungen und Problemstellungen	17
3.3.1 Positionen der Bauteile	18
3.3.2 Struktur einer Montageanleitung	19
3.3.3 Definition der Datenstruktur	20

3.3.4	Persistenthaltung der Daten	21
3.3.5	Verdeckte Marker	21
3.3.6	Tracking von Kleinteilen	22
3.3.7	Darstellung eines 3D-Pfeilobjekts	23
3.3.8	Animation eines Bauteils	25
3.3.9	Prüfen, ob ein Bauteil korrekt eingebaut wurde	31
4	Implementierung	33
4.1	Datenstruktur	33
4.2	Persistenthaltung der Daten	34
4.3	Authoring Modus	35
4.3.1	MarkerDetectionComponent: Bauteiltracking über die Markererkennung	36
4.3.2	GeometryComponent: Darstellung der 3D-Objekte . .	38
4.3.3	SoundComponent: Audiosteuerung	42
4.3.4	ButtonComponent: Steuerelemente	43
4.3.5	SchematicComponent: Grafische Benutzeroberfläche und Bilder	44
4.3.6	AMListComponent: Speichern der Montageschritte . .	45
4.3.7	MatrixCalculateComponent: Berechnen der Zielpo- sition bereits gespeicherter Kleinteile	48
4.3.8	AUTMLogicComponent: Zentrale Logik	49
4.4	Application Modus	52
4.4.1	MarkerDetectionComponent: Markererkennung für das Bauteiltracking	54
4.4.2	GeometryComponent: Darstellung der 3D-Objekte . .	54
4.4.3	SchematicComponent: Anzeige von Grafiken	54
4.4.4	TextComponent: Anzeige von Erklärungstexten	54
4.4.5	MatrixCalculateComponent: Berechnen der Zielpo- sition von Bauteilen und Kleinteilen	56
4.4.6	ArrowComponent: Anzeige der Pfeilobjekte	56
4.4.7	QAnimationComponent: Bauteilanimation	57
4.4.8	CheckAssemblyComponent: Ermitteln, ob ein Bauteil korrekt montiert wurde	58
4.4.9	ButtonComponent: Steuerelemente	60
4.4.10	APPMLogicComponent: Zentrale Logikkomponente . .	61
5	Arbeiten mit dem $F AI^{MR}$	66
5.1	Speichern einer Anleitung im Authoring Modus	66
5.2	Nachvollziehen einer Anleitung im Application Modus	67
6	Vergleich mit anderen Anwendungen	69
6.1	Animated Vision – Animierte Anleitungen	69
6.2	Proaktive Unterstützung	70

<i>INHALTSVERZEICHNIS</i>	vi
6.3 AEKI	71
7 Schlußbemerkungen	73
7.1 Mögliche Weiterentwicklungen	73
7.1.1 Anleitungen für beliebige Möbelstücke	73
7.1.2 Komplexere Bauteile erfassen	73
7.1.3 Auf verschiedene Benutzergruppen eingehen	74
7.1.4 Mobilität	74
7.1.5 Resümee	74
A Das XML Schema	75
B Komponenten der beiden Programm-Modi	77
Literaturverzeichnis	81

Danksagung

Ich möchte mich bei all jenen Menschen bedanken, die mich in der Zeit des Studiums an der Fachhochschule Hagenberg, speziell jetzt im Diplomsemester, begleitet und unterstützt haben.

Zuallererst möchte ich mich bei meinen Eltern und Großeltern bedanken. Sie haben mir im letzten Semester Unterkunft, Veköstigung und jeglichen Rückhalt gegeben. Als Test- und Korrekturleser war mein Vater unverzichtbar.

Meinem Bruder Peter möchte ich dafür danken, dass er mich während des Studiums immer motiviert und mir auch über Durststrecken hinweg geholfen hat.

Meine Freundin Tanja Heimberger hat mir den nötigen Freiraum gegeben, um mein Studium so erfolgreich abzuschließen. Sie hat mir geholfen mein seelisches Gleichgewicht zu bewahren, dafür bin ich ihr sehr dankbar.

Ich möchte mich auch bei meinem Betreuer DI Dr. Michael Haller und DI Jürgen Zauner bedanken. Ihre Anregungen und das Feedback waren immer hilfreich, um die Diplomarbeit bis zu ihrem endgültigen Stand zu entwickeln.

Schließlich möchte ich es nicht versäumen, mich bei all meinen Kollegen des Jahrgangs 1999 zu bedanken. Mit ihnen waren die letzten Jahre in Hagenberg etwas Besonderes, es haben sich viele Freundschaften ergeben und ich werde mich immer gerne an diese Zeit erinnern.

Kurzfassung

Das große Problem herkömmlicher Anleitungen in gedruckter Form (sei es für die Bedienung des Videorekorders oder den Zusammenbau eines Möbelstücks) besteht darin, dass der Anwender die Informationen interpretieren und auf die realen Gegenstände umlegen muss. Die Vorgehensweise wird vielfach in einzelnen Bildern visualisiert, daher wird vom Anwender verlangt, 2-dimensionale Informationen in die 3-dimensionale Realität zu transformieren. Darüber hinaus fehlt die Zeitachse im linearen, schriftlichen Medium. Es obliegt der Vorstellungskraft des Anwenders, die tatsächliche Abfolge von Schritten zu extrahieren.

Aus dieser Problemstellung ergibt sich der Lösungsansatz dieser Arbeit, nämlich eine Anleitung in direkten Kontext mit den realen Elementen zu stellen. Zu diesem Zweck wird ein Prototyp auf Mixed Reality Basis entwickelt, der eine Möbelbauanleitung abbildet. Dieser Prototyp erlaubt dem Anwender sowohl das Erstellen einer Montageanleitung als auch deren Auslesen und lineares Abarbeiten. Aufbauend auf dem AMIRE Framework, das ARToolkit als kostengünstiges und flexibles Trackingsystem verwendet, wird die Anleitung eines Beispielmöbelstücks verarbeitet. Die einzelnen Bauteile dieses Möbelstücks werden mit Markern versehen und können über eine WebCam von der Anwendung erfasst und voneinander unterschieden werden.

Das Ziel dieser Arbeit ist es, die Konzepte und Grundlagen vorzustellen, auf denen der funktionierende und ausbaufähige Prototyp aufbaut. Darüber hinaus werden Ansätze anderer Projekte vorgestellt und mit der vorliegenden Arbeit verglichen.

Abstract

Printed instruction manuals for furniture assembly often have one disadvantage in common: it takes a lot of time to make sense of their meaning since they show several steps of assembly together in a few pictures. Furthermore, it is hard to find the connection between the instructions printed in 2D and the real parts.

The idea of this work is to connect the instruction directly to the parts of a piece of furniture. To do this, mixed reality is used, which combines reality (grabbed by a webcam) with additional information using common computer graphics in 2D and 3D which are overlaid. A prototype is being developed that allows the authoring and storage of an instruction as well as the use of a stored instruction. The prototype is based on the AMIRE framework, which uses ARToolkit as a cheap and flexible solution for tracking. So called *markers* are placed on the parts of a furniture. By tracking these markers via webcam, the application can distinguish between different parts.

The aim of this thesis is to present the concepts on which the working prototype is based. Finally, some other approaches to this topic are presented and compared to the concepts in this work.

Kapitel 1

Einleitung

Es genügt nicht, zum Fluss zu kommen
mit dem Wunsche, Fische zu fangen.
Man muss auch das Netz mitbringen.

Aus China

1.1 Motivation und Anwendungsbereich

Herkömmliche Anleitungen auf Papier haben meist einen großen Nachteil: Es besteht die Notwendigkeit des Umsetzens der Erklärungen auf die realen Gegenstände. Nicht umsonst machen viele Käufer unliebsame Erfahrungen mit Anleitungen zur Bedienung des neuen Videorekorders¹ oder zum Aufbau des Küchenkastens. Wenn beispielsweise in einer Montageanleitung eines Möbelstücks in einem einzigen Bild mehrere Schritte erklärt werden („Montieren Sie Schraube A an Bauteil B, das wiederum mit Bauteil C verbunden werden muss...“), muss der Monteur diese erst in reale Handlungen umsetzen. 2-dimensionale Abbildungen müssen in die reale (3-dimensionale) Umgebung übersetzt werden. Zusätzlich fehlt in einer gedruckten Anleitung (naturegebenermaßen) die Zeitachse, es obliegt der Vorstellungskraft des Monteurs, aus einem Bild eine Abfolge von Handlungen abzuleiten.

Daher ist der Ansatz dieser Arbeit, eine Montageanleitung durch den Einsatz von Mixed Reality Technologie in einen direkten Kontext mit dem Möbelstück zu stellen. Ein Beispiel soll dies verdeutlichen: Es ist die Aufgabe einer Möbelbauanleitung, verständlich zu erklären, wo und wie ein bestimmter Bauteil montiert werden muss. Der Mixed Reality Ansatz visualisiert genau diese Informationen. Dazu wird ein reales Möbelstück um virtuelle Inhalte bereichert, wodurch die oben erwähnten Probleme gelöst werden: Der Anwender muss die Informationen nicht mehr von einem Medium (Papier) auf ein anderes (Bauteile) übertragen. Zeitliche Abfolgen werden zusätzlich in ihrer tatsächlichen Reihenfolge abgearbeitet.

¹Ein vielstrapaziertes Beispiel, aber mit Sicherheit nicht ohne Grund.

Durch den Einsatz von Mixed Reality können daher die Fehlerquote und auch die Zeit bei einem Montageprozess positiv beeinflusst werden, wie [TANG et al. 2002] bzw. [TANG et al. 2003] in ihren Studien bewiesen haben.

1.2 Zielsetzung

Zur Umsetzung des oben genannten Ansatzes wird ein Prototyp entwickelt, der eine Möbelbauanleitung abbildet. Dieser Prototyp trägt den Arbeitstitel (*Mixed Reality*) *Furniture Assembly Instructor* (*FAI^{MR}*) und unterstützt zwei wesentliche Funktionen: Einerseits wird ein Werkzeug für einen Möbelkonstrukteur bereitgestellt, mit dem er eine Möbelbauanleitung in Mixed Reality entwirft. Andererseits führt die Anwendung jemand, der ein Möbelstück zusammenbauen will, durch den Montageprozess. In beiden Programmen steht die Interaktion mittels der realen Bauteile, die vom System getrackt werden, im Vordergrund. Darüber hinaus werden Möglichkeiten zur Visualisierung der Anleitung konzipiert.

Als Grundlage für die Programmierung dient das AMIRE Framework² bzw. die Programmiersprache C/C++. Das Framework verwendet für das Tracking ARToolkit [KATO et al. 2000].

1.3 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen für die Arbeit vorgestellt: es wird der Begriff *Mixed Reality* genauer definiert und die Arbeitsweise mit dem AMIRE Framework und ARToolkit erklärt.

Der Hauptteil der Arbeit beschäftigt sich mit den Anforderungen an die Anwendung, den daraus resultierenden Problemstellungen und Lösungsansätzen (Kapitel 3), sowie der programmiertechnischen Umsetzung (Kapitel 4).

In Kapitel 5 wird die Verwendung des fertiggestellten Prototyps erklärt. Danach wird der Vergleich des eigenen Ansatzes mit anderen Projekten angestellt (Kapitel 6). Schließlich wird die Zusammenfassung der erreichten Ziele und der möglichen Erweiterungen erörtert (Kapitel 7).

Anhang A enthält das vollständige XML-Schema, das der Persistenthaltung der Daten dient. In Anhang B ist der Komponentenaufbau der beiden Programmteile komplett abgebildet. Dieser Arbeit ist eine CD-Rom beigelegt, deren Inhalt in den Dateien *Inhalt.txt* bzw. *content.txt* aufgelistet ist.

²www.amire.net

Kapitel 2

Grundlagen

Im folgenden Abschnitt werden die Grundlagen für das Verständnis dieser Arbeit gelegt. Zunächst wird eine Definition des Begriffs *Mixed Reality* festgelegt, anschließend wird kurz auf das Trackingsystem ARToolkit und den Aufbau des AMIRE Frameworks eingegangen.

2.1 Mixed Reality

Es existieren zahlreiche Definitionen für den Begriff *Mixed Reality*. Eine einheitliche Taxonomie hat sich noch nicht durchgesetzt, daher soll das Verständnis dieses Begriffs in der vorliegenden Arbeit definiert werden. Ausführlichere Abhandlungen zu diesem Thema sind in den Quellen [MILGRAM und KISHINO 1994] und [TOMIC-KOLUDROVIC et al. 2002] zu finden.

Um eine Anwendung mit dem Begriff *Mixed Reality* zu bezeichnen, müssen zwei Voraussetzungen gegeben sein:

- **Einbeziehen der realen Umgebung:** Die reale Umgebung wird als Interaktionsmittel in der Anwendung verwendet. Eine Veränderung der relevanten Objekte (im Fall der vorliegenden Arbeit die Bestandteile eines Möbelstücks) wird registriert und löst entsprechende Funktionen im Programm aus (zum Beispiel grafisches oder akustisches Feedback). Für das Erkennen der Veränderungen ist das *Tracking* verantwortlich.
- **Erweitern der Realität:** Computergenerierte Elemente (Grafik, Text, Sound) werden zusätzlich zur realen Szene angezeigt/abgespielt. Dadurch wird der Informationsgehalt der realen Welt erweitert.

Diese Definition genügt dem Verständnis von *Mixed Reality* in der vorliegenden Arbeit. Um das oben erwähnte Tracking zu realisieren, wird im AMIRE Framework ARToolkit verwendet. Dieses Trackingsystem wird im Folgenden näher beschrieben.

2.2 ARToolkit

2.2.1 Kurzbeschreibung

ARToolkit ist eine C-Bibliothek, mit der relativ einfach Mixed Reality Anwendungen implementiert werden können. Sogenannte Marker werden verwendet, um die Positionen realer Objekte relativ zur Kamera (in der Regel eine WebCam) zu bestimmen (siehe Abb. 2.1). Daraus resultiert die große Stärke des Systems: es ist kostengünstig und funktioniert unter geeigneten Bedingungen (Lichtverhältnisse, Qualität der Kamera) ausreichend genau und zuverlässig. ARToolkit wurde bereits in diversen Projekten eingesetzt, eine tiefere Betrachtung würde den Rahmen dieser Arbeit sprengen und wäre daher wenig sinnvoll. Weitere Informationen zu diesem Thema sind in [KATO et al. 2000], [BILLINGHURST et al. 1999] und [BILLINGHURST und KATO 1999] zu finden. In den folgenden Absätzen wird auf die relevanten Themen und Probleme für die Prototypentwicklung eingegangen.

2.2.2 Positionsbestimmung durch Markertracking

Das Markertracking liefert grundsätzlich zwei essentielle Informationen:

1. Welcher Marker wurde erkannt? Dadurch wird die Identifikation des Objekts möglich, auf dem der Marker angebracht wurde.
2. An welcher Position befindet sich der Marker? Diese Position wird in Form einer Transformationsmatrix ermittelt. Diese Matrix transformiert einen Punkt im (Welt-)Koordinatensystem der Kamera in das Koordinatensystem des Markers und hat folgendes Aussehen:

$$m = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.1)$$

In dieser Matrix sind die Position (Translationsvektor t) und Ausrichtung (Rotationsmatrix r) eines Markers festgelegt (siehe Abb. 2.1(b)).

2.2.3 Probleme

Probleme treten dann auf, wenn das System den/die Marker nicht erkennt. Dafür kann es mehrere Ursachen geben:

- Die Marker sind zu klein/zu weit weg von der Kamera. In [KATO et al. 2000] wird die Größe der Marker in Relation zur maximalen Distanz von der Kamera gesetzt.
- Der Betrachtungswinkel ist zu flach.

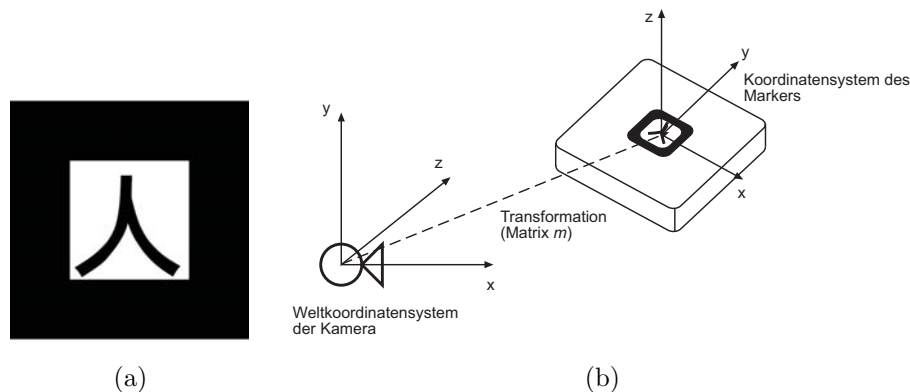


Abbildung 2.1: (a) Beispiel für einen Marker. (b) ARToolkit ermittelt die Position eines Markers relativ zur Kamera als Transformationsmatrix m .

- Das Muster eines Markers ist zu kompliziert.
- Die Muster der Marker unterscheiden sich kaum.
- Schlechte Lichtverhältnisse verursachen beispielsweise Reflexionen auf den Markern oder resultieren in einem zu geringen Kontrast (Unterscheidung zwischen schwarzen und weißen Flächen erschwert).

Außerdem ist die Qualität der verwendeten WebCam entscheidend für das Ergebnis. Die für die Entwicklung Prototyps verwendete WebCam hat eine maximale Auflösung von 640x480 Pixeln bei einer Framerate von 30 Bildern pro Sekunde.

2.3 Das AMIRE Framework

2.3.1 Kurzbeschreibung

AMIRE (**A**uthoring **MI**xed **RE**ality)¹ ist der Name eines EU-Projekts, dessen Ziel die Entwicklung eines Frameworks für Mixed Reality Anwendungen ist [HALLER et al. 2003]. Das Framework ermöglicht das einfache Entwickeln solcher Anwendungen auch für Anwender ohne Programmierkenntnisse. Zu diesem Zweck wird eine erweiterbare Bibliothek sogenannter „Gems“² angelegt. Diese Gems enthalten Lösungen für Mixed Reality Anwendungen in Form von Algorithmen und Programmier Techniken in den verschiedensten Bereichen, wie z. B. Computergrafik, Sound, User Interfaces, Tracking etc. Das Tracking wird derzeit beispielsweise über ARToolkit

¹www.amire.net

²engl.: Edelstein; Der Begriff wird in Anlehnung an die Buchreihe *Graphic Gems* von Academic Press (www.graphicsgems.org) verwendet.

realisiert, ist jedoch austauschbar mit anderen Methoden (z. B. Magnetisches Tracking) [DÖRNER et al. 2002].

Eine Mixed Reality Anwendung wird als ein Netzwerk von *Komponenten* realisiert. Komponenten stellen wiederverwendbare Bausteine dar, die auf den oben erwähnten Gems aufbauen. Diese Komponenten übernehmen verschiedene Aufgaben innerhalb der Anwendung. Einige Beispiele dafür sind:

- Verbindung mit ARTolkit herstellen für das Tracken von Markern,
- 3D-Modelle laden und anzeigen,
- 2D-Grafiken laden und anzeigen,
- Sounddateien laden und abspielen,
- Logische Verknüpfungen herstellen (UND-, ODER-Bausteine),
- Elemente für grafische Benutzeroberflächen (z. B. Buttons) darstellen und Benutzerinteraktionen verwerten.

Das Framework ermöglicht die Verwendung von Gems innerhalb der Komponenten.

Über ein eigenes *Authoring* Werkzeug (daher der Name des Projekts) verknüpft der Entwickler einer Mixed Reality Anwendung vorhandene Komponenten, um den gewünschten Funktionsumfang zu realisieren. Dafür benötigt er, wie bereits erwähnt, keine Programmierkenntnisse, der tatsächliche Programmieraufwand geschieht in der Entwicklung der Komponenten (bzw. des Frameworks und der Gems). Für die Umsetzung eines solchen Authoring Werkzeuges gibt es verschiedene Ansätze. Im Idealfall geschieht der Authoring-Prozess im direkten Kontext mit der realen Umgebung, die Bestandteil der Mixed Reality Anwendung ist. Bezogen auf das Beispiel einer Möbelbauanleitung heißt das, die Bauteile des Möbelstücks selbst dienen dem Autor als Werkzeuge für den Entwicklungsprozess.

2.3.2 Komponenten

Das AMIRE Framework ermöglicht die Verknüpfung der Komponenten. Zu diesem Zweck können diese *Eingänge (In Slots)* und *Ausgänge (Out Slots)* haben. Zusätzlich können gewisse Grundparameter von Komponenten über sogenannte *Eigenschaften (properties)* definiert werden. Über diese Eigenschaften werden z. B. Dateinamen von 3D-Modellen oder 2D-Grafiken festgelegt. Die Grafik in Abbildung 2.2 zeigt den Aufbau einer Komponente. Für die Verbindung zweier Komponenten wird ein Ausgang mit einem Eingang verknüpft. Das Framework stellt sicher, dass nur Kanäle mit gleichen Datentypen miteinander verbunden werden können.

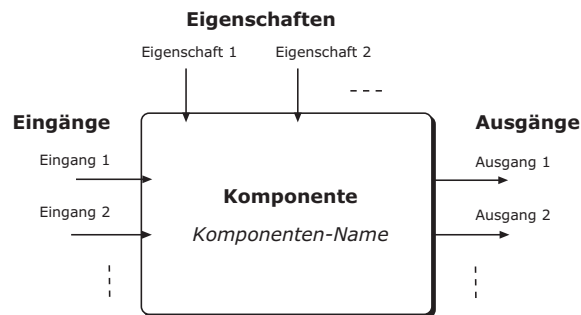


Abbildung 2.2: Aufbau einer Komponente: Eingänge, Ausgänge, Eigenschaften.

In komplexeren Anwendungen ist es sehr wahrscheinlich, dass gewisse Verknüpfungen von Komponenten mehrmals vorkommen. Aus diesem Grund sieht das Konzept des Frameworks sogenannte *zusammengesetzte Komponenten* (*Composed Components*) vor. Diese Komponenten sind eine Art Container, in denen mehrere „kleine“ Komponenten miteinander verknüpft werden, um insgesamt eine komplexere Aufgabe zu erfüllen (siehe Abb. 2.3). Die mehrfache Verwendung solcher zusammengesetzter Komponenten vereinfacht die Struktur einer Anwendung, da die Anzahl der verwendeten Komponenten geringer ist. Zum Zeitpunkt der Entwicklung des Prototyps standen diese Container noch nicht zur Verfügung. Daher wurden für diese Arbeit teilweise mehrere (und komplexere) Funktionen in einzelne, größere Komponenten verpackt, um die Verknüpfungen überschaubar zu halten.

2.4 Montageanleitungen

Das Ziel einer Montageanleitung besteht im Grunde darin, das nötige Wissen zu vermitteln, um einen Montageprozess erfolgreich nachvollziehen zu können. Diese Wissensvermittlung erfolgt bei herkömmlichen gedruckten Anleitungen meist nach einem gewissen Schema. In der Regel wird eine bestimmte Vorgehensweise vorgegeben und linear beschrieben. Dabei besteht die Annahme, dass der vorgeschlagene Weg am effizientesten ist. Es gibt jedoch auch andere Ansätze der Hilfe beim Montageprozess.

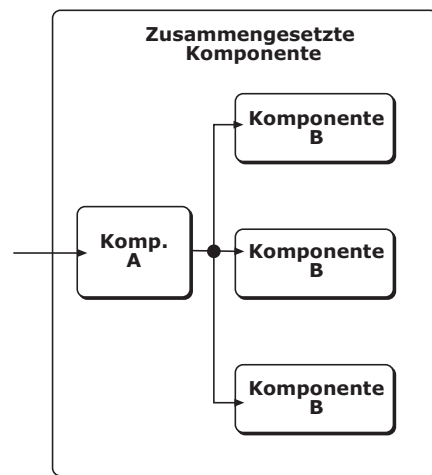


Abbildung 2.3: Zusammengesetzte Komponenten (Container) erfüllen komplexere Aufgaben und können wie normale Komponenten mehrmals eingesetzt werden.

In [ANTIFAKOS et al. 2002] werden drei Modelle vorgestellt, die sich am Wissensstand des Anwenders orientieren:

1. **Step-By-Step:** Dieser Ansatz richtet sich vor allem an Benutzer, die mit dem Montageprozess nicht vertraut sind. Die einzelnen Schritte werden sequentiell durchlaufen und mit Informationen unterstützt. Wie bereits oben erwähnt, ist diese Methode die Grundlage der meisten schriftlichen Montageanleitungen (schließlich ist ein geschriebener Text ein lineares Medium, vgl. die Ansätze von McLuhan, Flusser [KLOOCK und SPAHR 1996]).
2. **Help-On-Demand** ist für Anwender interessant, die weder Experten sind, noch den genauen Montagevorgang kennen. Diese Anwender wählen einen eigenen Weg, ziehen jedoch bei Bedarf die Anleitung zu Rate.
3. Der **Rescue-From-Trap**-Ansatz dient vor allem erfahrenen Anwendern, die im Grunde wissen, wie die Montage erfolgen muss. Der einzige Fall, wo das Hilfe-System aktiv wird ist, wenn der Anwender Gefahr läuft, einen Fehler in der Montage zu machen (indem er beispielsweise Sicherheitsregeln missachtet). Außerdem kann der Anwender das System zu Rate ziehen, wenn er allein nicht mehr weiter weiß.

Das große Problem der letzten beiden Ansätze besteht im Anlegen (*Authoring*) einer Anleitung, die nach dem beschriebenen Muster funktioniert.

Es ist die große Herausforderung an das System, herauszufinden, an welchem Punkt der Anwender gerade steht, wenn er Hilfe anfordert. Zu diesem Zweck muss das System zuvor im Idealfall alle möglichen Lösungswege eines Montageprozesses, aber auch eventuelle Sackgassen kennen. Nur dann kann im entsprechenden Fall Hilfe zur Verfügung gestellt werden. Ein solches System müsste also lernfähig sein und von einem Möbelkonstrukteur auf alle Möglichkeiten trainiert werden.

Kapitel 3

Methodik

Das Ziel dieser Arbeit ist es, eine interaktive Möbelbauanleitung auf Mixed Reality Basis zu entwickeln. Das zu implementierende Werkzeug mit dem Arbeitstitel (*Mixed Reality*) *Furniture Assembly Instructor* (FAI^{MR}) soll sowohl das Entwerfen einer solchen Anleitung als auch deren Auslesen ermöglichen. Im folgenden Kapitel werden die Funktionen des Programms definiert und entsprechende Anforderungen abgeleitet. Danach werden grundlegende Konzepte erarbeitet, die für die Umsetzung relevant sind.

3.1 Das AMIRE Framework als Grundlage

Der FAI^{MR} setzt auf dem AMIRE Framework auf, das eine komponentenbasierende Umsetzung von Mixed Reality Anwendungen ermöglicht. Um den Funktionsumfang zu realisieren, werden verschiedene Komponenten (siehe Abschnitt 2.3) implementiert und miteinander verknüpft.

Ein Möbelstück besteht aus mehreren Bauteilen, die für eine Bauanleitung eindeutig voneinander unterschieden werden müssen. Das Tracken dieser Bauteile erfolgt über die Markererkennung, die vom AMIRE Framework über ARToolkit unterstützt wird. Jeder Bauteil wird über einen Marker (oder mehrere) identifiziert. Dieses Prinzip ist die essentielle Voraussetzung für die gesamte Anwendung.

3.2 Funktionsumfang

Aufgrund der obigen Vorgaben ergibt sich die Teilung der Anwendung in zwei Modi, wobei jeder Teil ein eigenes Programm darstellt. Der erste Modus trägt die Bezeichnung *Authoring* Modus und dient der Eingabe einer Montageanleitung, wobei hier die Positionen und Reihenfolge der einzelnen Bauteile gespeichert werden. Dieser Programmteil stellt das Werkzeug des sogenannten *Autors* (z.B. ein Möbelbauer) dar, der die Bauanleitung zu einem Möbelstück entwirft.

Die gespeicherten Daten (die Anleitung) werden im zweiten Programmteil, dem *Application* Modus ausgewertet, um einem *Anwender* (z. B. ein Kunde oder Montagearbeiter) die erforderlichen Montageschritte anzuzeigen (siehe die Begriffsdefinitionen in Tabelle 3.1).

<i>Bezeichnung</i>	<i>Beschreibung</i>
Authoring Modus	Programm zum Eingeben einer Montageanleitung
Autor	Benutzer des Authoring Modus (z. B. Möbelbauer)
Application Modus	Programm, das die Anleitung ausliest und geeignet wiedergibt
Anwender	Benutzer des Application Modus

Tabelle 3.1: Begriffsdefinitionen für diese Arbeit.

Dem FAI^{MR} liegt ein Konzept zugrunde, das sich am Aufbau herkömmlicher Möbelbauanleitungen auf Papier orientiert. Diese umfassen hauptsächlich folgende Informationen:

- Abbildungen der verwendeten Bauteile, Kleinteile und Werkzeuge,
- Darstellung der Reihenfolge der Montageschritte,
- Darstellung der Position, an der ein Bauteil/Kleinteil montiert werden muss,
- Erklärungstexte zu den einzelnen Montageschritten.

Die Mixed Reality Anwendung hat im Gegensatz zu gedruckten Anleitungen den Vorteil, dass diese Informationen direkt im Zusammenhang mit den realen Bauteilen und dem Baufortschritt dynamisch angezeigt werden. Diese Anzeige wird sowohl in Form von 2D-Grafik und Text als auch durch 3D-Grafiken samt Animation direkt in der Szene realisiert. Eine akustische Unterstützung ist ebenfalls vorgesehen.

3.2.1 Authoring Modus

Im Authoring Modus erstellt der Autor eine Montageanleitung. Der Zusammenbau eines Möbelstücks besteht grundsätzlich aus zwei immer wiederkehrenden Handlungen:

- Kleinteile an Bauteilen anbringen (dafür wird meist ein Werkzeug benötigt) und
- Bauteile zusammensetzen (unter Umständen ebenfalls mit einem Werkzeug).

Der Autor speichert die Reihenfolge dieser Handlungen schrittweise. Für die Anwendung wird dabei eine bestimmte Abfolge definiert, nach der der Autor vorgehen muss:

1. Bauteil auswählen,
2. Kleinteil auswählen, das am aktuellen Bauteil montiert werden muss,
3. Kleinteil montieren (d. h. die Position relativ zum Bauteil speichern),
4. Bauteil montieren (d. h. die Position speichern),
5. Anleitung beenden/speichern.

In Abbildung 3.1 ist dieser Ablauf mit den möglichen Wiederholungsschleifen skizziert. In der Anwendung werden dafür Interaktionsmöglichkeiten in Form von grafischen Bedienelementen (Buttons) bereitgestellt.

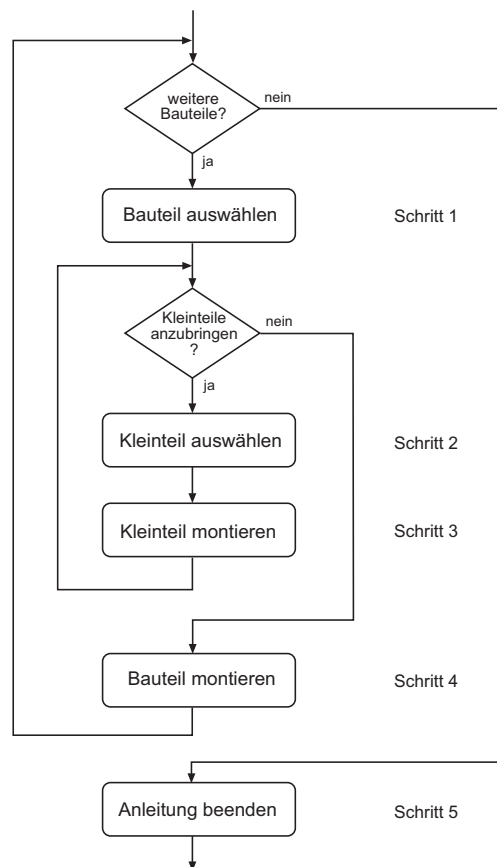


Abbildung 3.1: Ablaufdiagramm des *Authoring* Modus.

Die vom System getrackten Bauteile werden durch 3D-Rahmen markiert. Über die Farbe dieser Rahmen wird der Status der Bauteile angezeigt.

Die getrackten Kleinteile werden als 3D-Objekte direkt in der Szene eingeblendet. Außerdem werden Grafiken der einzelnen Bauteile und Kleinteile angezeigt, um die Orientierung zu erleichtern. Änderungen in der Szene oder den Grafiken dienen dem Benutzer als Rückmeldung zu seinen Aktionen. Im Weiteren werden die Schritte 3, 4 und 5 (s. Abb. 3.1) durch akustische Rückmeldungen bestätigt. Tabelle 3.2 zeigt die visuellen und akustischen Elemente in Zusammenhang mit den Benutzeraktionen. Abbildung 3.2 zeigt eine mögliche Konstellation im Authoring Modus.

<i>Benutzeraktion</i>	<i>Schritt</i>	<i>Visuelle Rückmeldung</i>	<i>Akustische Rückmeldung</i>
Bauteil auswählen	1	Ein erfasstes Bauteil mit grünem Rahmen markieren; Bild dieses Bauteils links unten anzeigen.	
Kleinteil auswählen	2	Kleinteil als 3D-Objekt in der Szene und als Bild rechts unten anzeigen.	
Kleinteil montieren	3	3D-Modell des montierten Kleinteils permanent in der Szene an seiner Position auf dem aktuellen Bauteil anzeigen.	„Kleinteil gesichert“
Bauteil montieren	4	Aktuellen Bauteil mit rotem Rahmen markieren.	„Bauteil gesichert“
Anleitung beenden	5		„Anleitung gespeichert“

Tabelle 3.2: Visuelle und akustische Rückmeldungen im Authoring Modus entsprechend den Benutzeraktionen.

3.2.2 Application Modus

Wie bereits erwähnt, dient dieser Modus dem Anwender zur Unterstützung beim Zusammenbau eines Möbelstückes. Eine Anleitung wird vom Programm entsprechend dem **Step-By-Step**-Ansatz (siehe Abschnitt 2.4) stets linear abgearbeitet: Das heißt, es müssen alle Schritte von Anfang an und der Reihe nach durchgegangen werden. Diese Festlegung geht von der Annahme aus, dass der Anwender keinerlei Vorwissen hat, weder im Umgang



Abbildung 3.2: Rahmen markieren die erkannten Bauteile A, B, C, D und zeigen an, dass Bauteil A (grün markiert) gerade selektiert wurde. Zusätzlich ist am linken unteren Rand ein Bild des Bauteils A zu sehen.

mit dem Programm, noch mit dem Zusammenbau des Möbelstückes. Daher gibt es auch nur zwei Möglichkeiten der Navigation, nämlich innerhalb der Anleitung einen Schritt vorwärts oder rückwärts zu gehen.

In diesem Modus steht vor allem die Visualisierung der gespeicherten Daten einer Anleitung im Vordergrund. Um den Anwender mit der Vielzahl an Informationen nicht zu überfordern, ist das Programm in verschiedene Abschnitte unterteilt. In jedem Abschnitt wird ein Teilschritt der Montage erklärt und der Benutzer zu gezielten Aktionen aufgefordert.

Die einzelnen Schritte sind in Abb. 3.3 dargestellt und enthalten folgende Visualisierungen:

- Positionsrahmen der getrackten Bauteile,
- 3D-Grafik eines Bauteils bzw. Kleinteils an der Zielposition direkt im Videobild (Die Farbe des Bauteils an der Zielposition ändert sich, sobald der reale Bauteil korrekt montiert wird),
- 3D-Pfeil von einem realen Bauteil zu seiner Zielposition,
- Anstelle des Pfeils kann der Anwender auch eine Animation einblenden, die zusätzlich auch die korrekte Ausrichtung eines Bauteils verdeutlicht,
- Erklärungstexte zu den Montageschritten,
- Bilder der einzelnen Bauteile und Kleinteile
- Werkzeuge, mit denen die Montage erfolgt.

Abbildung 3.4 zeigt einige Beispiele der Umsetzung in der Anwendung.

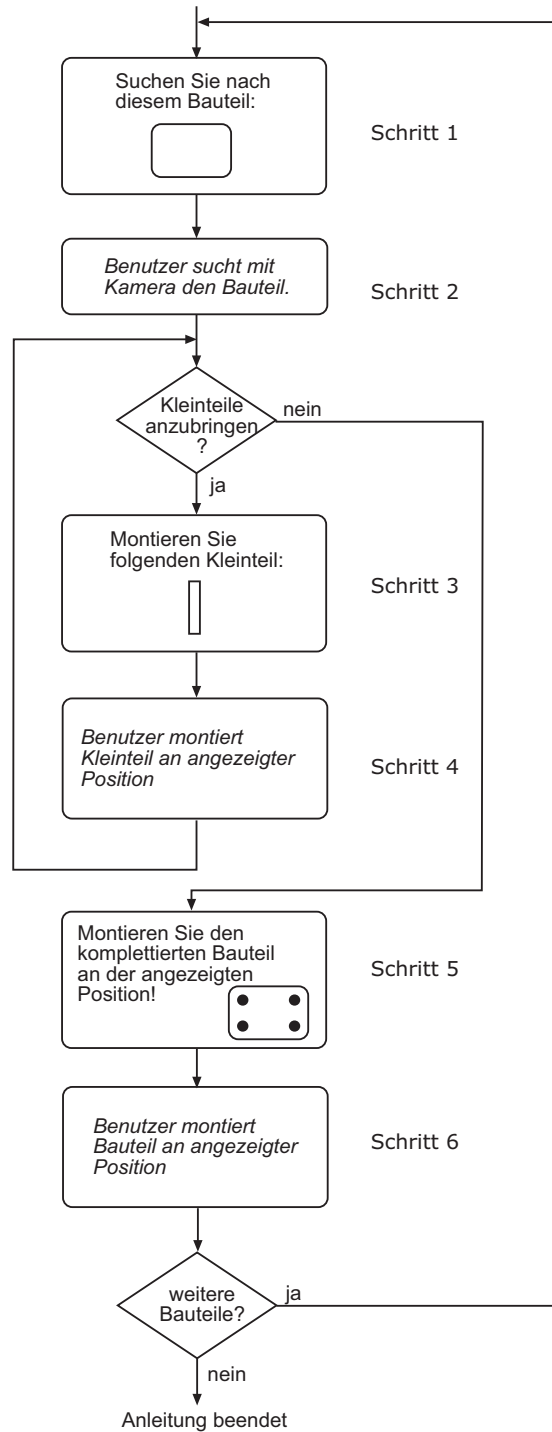


Abbildung 3.3: Ablaufdiagramm des *Application* Modus.

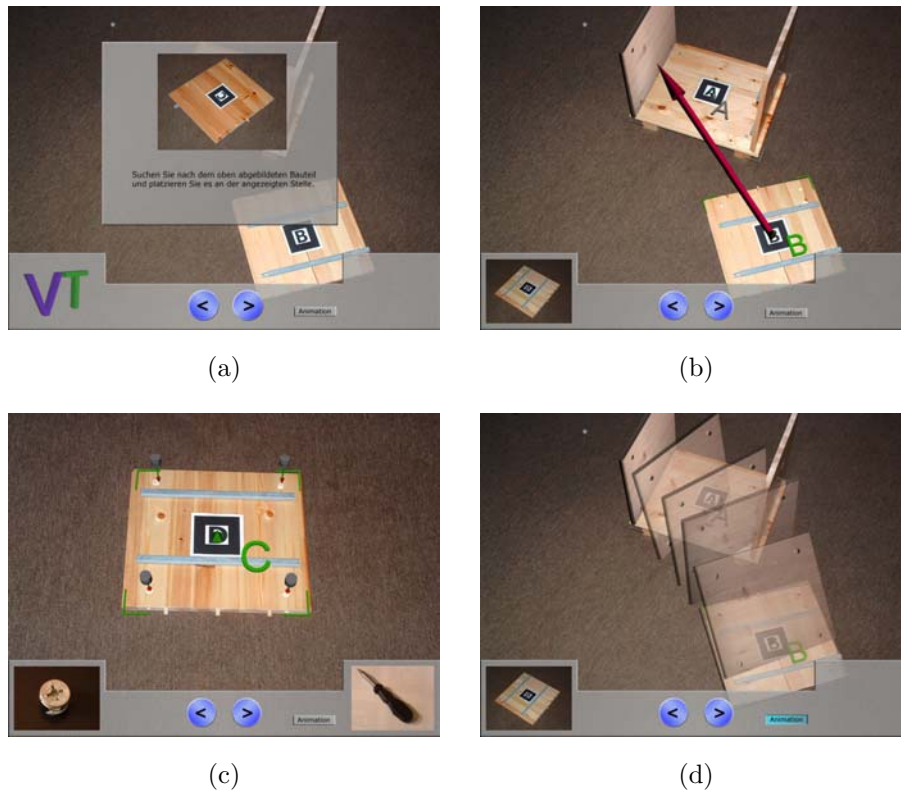


Abbildung 3.4: Screenshots der einzelnen Schritte aus dem Application Modus: (a) Bild des zu montierenden Bauteils (oder eines Kleinteils) mit Erklärungstext [Schritt 1]. (b) Links unten: Bild des Bauteils B, das in der Szene grün markiert ist; Ein Pfeil zeigt die Zielposition an, an der zusätzlich eine 3D-Grafik eingeblendet wird [Schritt 6]. (c) Anzeige der Position von Kleinteilen; Hier wird außer dem Bild des Kleinteils noch ein Werkzeug eingeblendet (rechts unten) [Schritt 4]. (d) Eine Animation zeigt die korrekte Ausrichtung eines Bauteils an [Schritt 6].

3.2.3 Abgrenzung

Der *FAI^{MR}* ist als Konzeptstudie ausgelegt. Er beschränkt sich auf die Verarbeitung der Anleitung für ein Beispiel-Möbelstück, die nicht ausgetauscht werden kann. Dieses Möbelstück, ein Nachtkästchen, erweist sich aufgrund seiner kompakten Größe als praktikables Testobjekt. Davon werden vier verschiedene Bauteile und drei Arten von Kleinteilen erfasst. Die Bauteile sind:

- Eine Bodenplatte,
- zwei Seitenteile,

- eine Deckplatte.

Die Kleinteile sind:

- Lange Schrauben,
- Schrauben-Gegenhalter,
- Holzdübel.

Zu jedem Bauteil können insgesamt maximal vier Stück dieser Kleinteile gespeichert werden. Der Umstand, dass das Möbelstück aus flachen und quaderförmigen Bauteilen besteht, erleichtert die Platzierung der Marker darauf (die Problematik diesbezüglich wird in Abschnitt 3.3.5 genauer erläutert).

3.3 Anforderungen und Problemstellungen

Aus dem zuvor definierten Funktionsumfang ergeben sich folgende Problemstellungen:

- Wie werden die Positionen der Bauteile ermittelt?
- Welche Struktur hat eine Montageanleitung?
- Welche Datenstruktur eignet sich zum Speichern der Montageanleitung?
- In welcher Form werden die Daten einer Anleitung persistent gehalten, um zwischen den beiden Programmteilen übertragen werden zu können?
- Wie wird das Problem verdeckter Marker gelöst?
- Wie werden Kleinteile vom System erfasst?
- Wie wird im Application Modus die Anzeige der Zielposition eines Bauteils, eines Pfeils und einer Animation realisiert?
- Wie wird ermittelt, ob ein Bauteil korrekt eingebaut wurde, um entsprechende Rückmeldungen zu geben?
- Welche Komponenten werden benötigt, um den gewünschten Funktionsumfang zu realisieren? Wie müssen diese verknüpft werden? (siehe Kapitel 4).

Diese Fragestellungen werden in den folgenden Abschnitten genauer erläutert und die Lösungsansätze dieser Arbeit vorgestellt. In der obigen Liste sind nur die wichtigsten Punkte aufgezählt. Hier nicht enthaltene Bereiche werden in Kapitel 4 bei der Beschreibung der Implementierung behandelt.

3.3.1 Positionen der Bauteile

In einer von der Anwendung erstellten Anleitung wird, neben anderen Informationen, festgehalten, an welcher Stelle ein bestimmter Bauteil zu montieren ist. Grundsätzlich wird die Position eines Bauteiles *relativ* zu einem anderen Bauteil ermittelt (siehe die Begriffsdefinitionen in Tabelle 3.3). Daher wird in dieser Arbeit von einer Relativposition gesprochen. Die mathematischen Zusammenhänge für die Berechnung dieser Position werden im Folgenden erklärt.

<i>Bezeichnung</i>	<i>Beschreibung</i>
Position eines Bauteils	Absolut: Die Position eines Bauteils im Welt-Koordinatensystem der Kamera (von ARToolkit vorgegeben). Definiert als Transformationsmatrix. Relativ: Die Position eines Bauteils <i>relativ</i> zu einem anderen Bauteil, ebenfalls als Transformationsmatrix.
Zielposition eines Bauteils	Jene <i>absolute</i> Position eines Bauteils, an dem es entsprechend der Anleitung montiert werden muss. Die Berechnung dieser Position erfolgt über die absolute Position eines getrackten Bauteils und die gespeicherte(n) Relativposition(en).

Tabelle 3.3: Begriffsdefinitionen zur Position eines Bauteils.

Die absoluten Positionsdaten eines Bauteils werden vom Markertracking in Form einer Transformationsmatrix m ermittelt:

$$m = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.1)$$

Diese Matrix enthält die Position und Ausrichtung eines Markers bzw. eines Bauteils. Der Berechnung der Relativposition eines Bauteils zu einem anderen Bauteil liegen folgende mathematische Überlegungen zugrunde:

Gesucht wird die Transformationsmatrix $m_{A \rightarrow B}$, welche das Koordinatensystem eines Bauteils A (gegeben durch m_A) auf das Koordinatensystem eines Bauteils B (gegeben durch m_B) transformiert. Diese Transformation

wird durch die Multiplikation

$$m_A \cdot m_{A \rightarrow B} = m_B \quad (3.2)$$

ausgedrückt. Durch die Multiplikation mit der Inversen m_A^{-1} von links ergibt sich

$$m_A^{-1} \cdot m_A \cdot m_{A \rightarrow B} = m_A^{-1} \cdot m_B. \quad (3.3)$$

Da

$$m_A^{-1} \cdot m_A = E \quad (3.4)$$

ist (E entspricht der Einheitsmatrix), wird weitergeführt:

$$E \cdot m_{A \rightarrow B} = m_A^{-1} \cdot m_B. \quad (3.5)$$

Das endgültige Ergebnis lautet:

$$m_{A \rightarrow B} = m_A^{-1} \cdot m_B. \quad (3.6)$$

Diese Formel wird im *Authoring* Modus herangezogen, um die Position eines Bauteils relativ zu einem anderen Bauteil zu berechnen.

Im *Application* Modus wird eine solcherart berechnete Matrix verwendet, um die Zielposition eines Bauteils zu ermitteln.

Gegeben sind die absolute Position von Bauteil A (m_A) und die Position von Bauteil B relativ zu A ($m_{A \rightarrow B}$). Gesucht wird jene Transformationsmatrix m_B , die Bauteil B an seine Zielposition laut Anleitung bewegt. Die Multiplikation der beiden gegebenen Matrizen liefert das entsprechende Ergebnis

$$m_B = m_A \cdot m_{A \rightarrow B}. \quad (3.7)$$

3.3.2 Struktur einer Montageanleitung

Es wird festgelegt, welche Daten in einer Anleitung gespeichert werden. Die Problematik besteht in der Abgrenzung der Informationen, die vom Programm verwendet werden.

Die Anwendung verwaltet eine Liste von Bauteilen in der Reihenfolge ihrer Montage. Dafür wird jedem Bauteil eine eindeutige ID gegeben. Für jeden Bauteil werden folgende Daten gespeichert:

- ID des Bauteils,
- Die Position des Bauteils *relativ* zum zuvor montierten Bauteil,
- Eine Liste von Kleinteilen und deren Positionen relativ zum Bauteil.

Abbildung 3.5 zeigt ein Beispiel für einen Listeneintrag. An der Seitenwand B müssen zwei Schrauben (S1, S2) und zwei Dübel (D1, D2) montiert

werden. Die Positionen dieser Kleinteile relativ zum Bauteil B sind als Matrizen $m_{B \rightarrow S1/S2/D1/D2}$ gespeichert. Die Position des Bauteils B auf dem Möbelstück ist relativ zum zuvor montierten Bauteil (Bodenplatte A) festgelegt (Transformationsmatrix $m_{A \rightarrow B}$). Abbildung 3.6 zeigt den Aufbau einer Liste mit den Relativpositionen zum jeweils vorhergehenden Bauteil. Aufgrund dieser Liste wird die Zielposition eines Bauteils im Application Modus berechnet. Gegeben ist die Liste der gespeicherten Bauteile A, B, C und D mit den dazugehörigen Relativpositionen $m_{A \rightarrow B}$, $m_{B \rightarrow C}$ und $m_{C \rightarrow D}$. Gesucht ist die Zielposition des Bauteils C (Matrix $m_{targetC}$), wobei nur die Position des Bauteils A (Matrix m_A) durch das Tracking bekannt ist. Die Berechnung wird in Abbildung 3.6 erklärt.

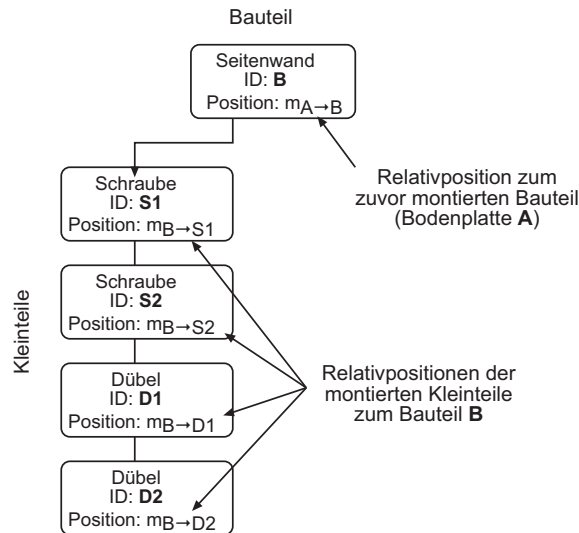


Abbildung 3.5: Eintrag eines Bauteils in die Liste der Montageanleitung.

3.3.3 Definition der Datenstruktur

Die oben festgelegten Daten werden in einer Adjazenzliste [KAISER 2001] verwaltet. Diese Datenstruktur wird in Form einer Liste implementiert, wobei jeder Listeneintrag wiederum eine Liste verwaltet. In jedem Eintrag werden, wie in Abbildung 3.5 gezeigt, die Kleinteile eines Bauteils in einer eigenen Liste gespeichert. Außerdem wird für jeden Bauteil die Relativposition zum zuvor montierten Bauteil gespeichert.

Die Zielposition jedes neuen Bauteils ist von *allen* bereits montierten bzw. gespeicherten Bauteilen abhängig. Das heißt, die Position jedes beliebigen, fertig montierten Bauteils, der vom Markertracking gerade erfasst wird, dient im Application Modus zur Berechnung der Zielposition des neuen Bauteils (siehe Abb. 3.6).

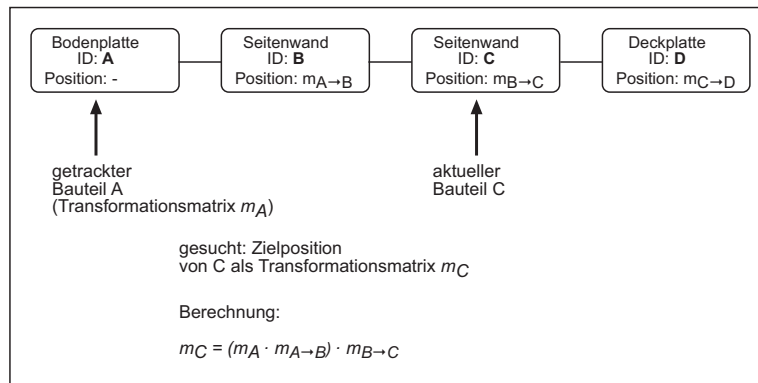


Abbildung 3.6: Beispiel für die Berechnung der Zielposition des Bauteils C (Matrix $m_{targetC}$) aus der getrackten absoluten Position des Bauteils A (Matrix m_A) und den gespeicherten Relativpositionen (Matrizen $m_{A \rightarrow B}$ und $m_{B \rightarrow C}$).

3.3.4 Persistenzhaltung der Daten

Die Daten einer Anleitung aus dem Authoring Modus werden in einer XML-Datei gespeichert und damit für den Application Modus persistent gehalten. Diesem XML-Format liegt ein eigens definiertes Schema [HAROLD 2002] zugrunde. Schemas besitzen gegenüber DTDs den Vorteil, dass auch Datentypen in Feldern definiert werden können. Das Schema bildet die Listenstruktur mit den vorkommenden Bauteilen, Kleinteilen und Abhängigkeiten ab (vgl. Quellcode in Anhang A).

Eine im XML Format gespeicherte Anleitung wird im Application Modus unter Verwendung des weitverbreiteten SAX Parsers von *Xerces*¹ eingelesen.

3.3.5 Verdeckte Marker

Durch das Anbringen eines Markers auf jedem Bauteil wird die eindeutige Identifizierung aller Bauteile eines Möbelstücks gewährleistet. Im Zuge des Montagefortschrittes ist es jedoch sehr wahrscheinlich, dass einige Marker permanent von anderen Bauteilen verdeckt werden und dadurch das System nicht mehr funktioniert (siehe Abb. 3.7).

Als logische Konsequenz dieser Überlegung wird auf einem Bauteil nicht nur ein einziger Marker angebracht. Der *FAI^{MR}* verwendet zwei verschiedene Marker, die genau gegenüberliegend an der Vorder- und Rückseite eines Bauteils angebracht werden. Diese Vorgehensweise wird den meist flachen Ausführungen von Bauteilen (z. B. Bodenplatten, Seitenwände oder Rückwände) gerecht. Die weiterführenden Überlegungen sind jedoch dahin-

¹<http://xml.apache.org/xerces-c/index.html>

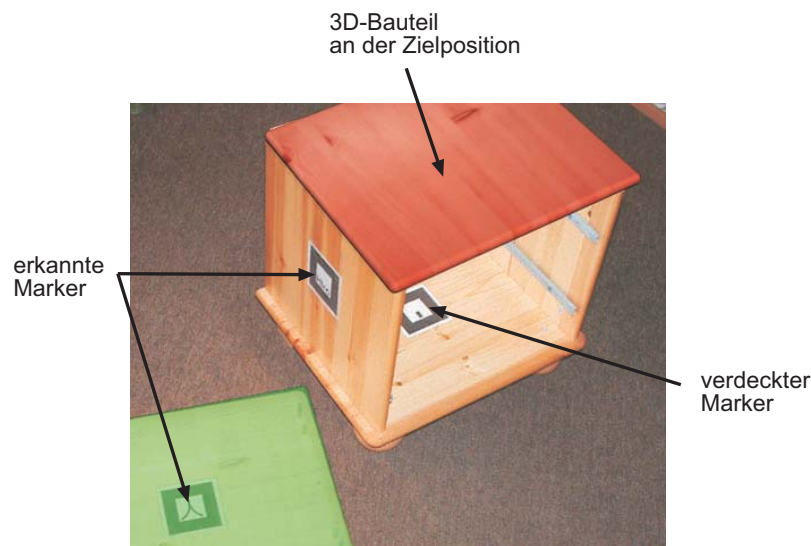


Abbildung 3.7: Teilweise verdeckter Marker.

gehend, jeden Bauteil mit einer beliebigen Anzahl an Markern zu bestücken. Damit ließen sich auch komplexere Bauteile (z. B. Sesselteile) erfassen.

3.3.6 Tracking von Kleinteilen

Es wird die Annahme getroffen, dass die Bauteile eines Möbelstückes groß genug sind, sodass darauf die Marker angebracht werden können. Damit wird das Tracken und Identifizieren der Bauteile gesichert. Es ist jedoch laut Konzept auch notwendig, Kleinteile zu erfassen, auf denen keine Marker angebracht werden können.

Das Problem des Kleinteiltrackings wird mit Hilfe eines eigenen Markers gelöst. Dieser Marker dient dem Autor zum Festlegen einer Kleinteilposition relativ zu jenem Bauteil, an dem ein Kleinteil montiert werden muss. Dabei sind zwei Schritte durchzuführen:

1. Den notwendigen Kleinteil aus einer Liste aller Kleinteile auswählen. Diese Liste wird schrittweise über eine Schaltfläche durchlaufen.
2. Danach platziert der Autor den Kleinteilmarker auf dem Bauteil an der vorgesehenen Stelle und betätigt eine Schaltfläche zum Speichern dieser Aktion.

Abbildung 3.8 zeigt einen Ausschnitt aus der Anwendung mit dem Kleinteilmarker und überblendetem 3D-Modell des Kleinteils.

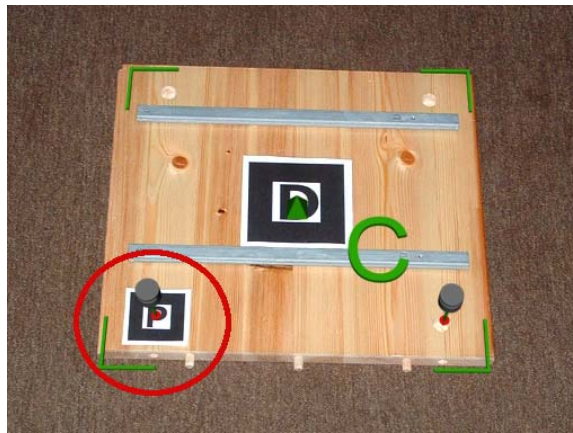


Abbildung 3.8: Der Autor speichert die Position eines Kleinteils mit Hilfe eines eigenen Markers (rot eingekreist). In diesem Beispiel wurde bereits ein Kleinteil hinzugefügt, der ebenfalls noch angezeigt wird.

3.3.7 Darstellung eines 3D-Pfeilobjekts

Entsprechend der Festlegung des Funktionsumfanges soll im Application Modus ein Pfeil vom aktuellen Bauteil zu dessen Zielposition angezeigt werden. Das AMIRE Framework stellt ein solches Pfeilobjekt in Form eines OpenGL-Modells zur Verfügung (siehe dazu Abb. 3.9(a)). Dieser Pfeil soll entsprechend der Abbildung 3.9(b) in die Szene transformiert werden.

Es sind drei Schritte notwendig, um die gewünschte Transformation durchzuführen:

1. Rotation des Pfeils in die korrekte Lage,
2. Translation des Pfeils an die absolute Position des Bauteils,
3. Anpassen der Länge des Pfeils.

Für die Berechnung der Rotation und Translation eines solchen Pfeilobjekts werden zwei Transformationsmatrizen benötigt: Einerseits die Position des realen Bauteils A (gegeben durch die Matrix m_A) und andererseits dessen Zielposition (gegeben durch die Matrix $m_{targetA}$). Gesucht ist eine Transformationsmatrix m_{arrow} , die ein Pfeilobjekt an die Stelle des aktuellen Bauteils A transliert und so rotiert, dass es in Richtung der Zielposition zeigt. Dabei ist zu beachten, dass das Pfeilobjekt grundsätzlich im Nullpunkt eines lokalen Koordinatensystems beginnt und in z -Richtung zeigt (siehe Abb. 3.9(a)). Mathematisch ausgedrückt heißt das, es wird eine Orthonormalbasis (ONB) gesucht, deren z -Achse vom aktuellen Bauteil zur Zielposition zeigt. Die Achsen dieser ONB werden mit *up*, *side* und *direction* bezeichnet (siehe Abb. 3.9(b)).

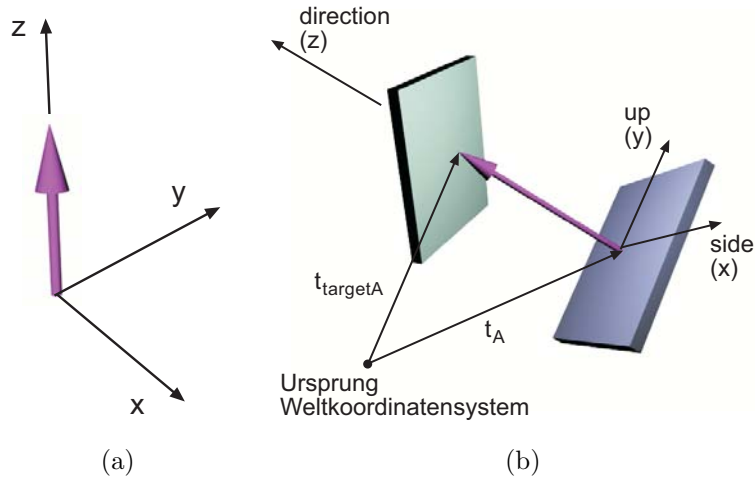


Abbildung 3.9: (a) Das Pfeilobjekt in seinem lokalen Koordinatensystem und (b) in das gesuchte Koordinatensystem (die Orthonormalbasis *side*, *direction* und *up*) transformiert.

Die Richtung dieser Achse ist als Richtungsvektor von t_A nach $t_{targetA}$ definiert, wobei t_A und $t_{targetA}$ den Translationsvektoren der beiden Matrizen m_A und $m_{targetA}$ entsprechen. Mit der Formel

$$direction = \frac{t_{targetA} - t_A}{|t_{targetA} - t_A|} \quad (3.8)$$

wird der normierte Richtungsvektor berechnet. Die Achse *side* wird mit einer der beiden Hauptachsenrichtungen X oder Y festgesetzt, die nicht parallel zu *direction* sein darf. Zuerst wird die X-Achse gewählt,

$$side = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \quad (3.9)$$

Wenn jedoch

$$direction \parallel side \quad (3.10)$$

ist, dann wird

$$side = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad (3.11)$$

gesetzt. Diese Achse *side* steht jedoch noch nicht normal zur bereits errechneten *direction*-Achse. Daher wird vorerst die normierte Achse *up*

$$up = \frac{side \times direction}{|side \times direction|} \quad (3.12)$$

berechnet. Schließlich erfolgt die Berechnung der endgültigen *side*-Achse, die auf beide bisherigen Achsen normal stehen muss:

$$side = direction \times up. \quad (3.13)$$

Durch diese Rechenschritte entsteht eine Orthonormalbasis, deren z -Achse in die gewünschte Richtung zeigt. Diese ONB entspricht einer Rotationsmatrix, die den Pfeil in die richtige Lage dreht. Der Pfeil befindet sich aber noch im Nullpunkt des globalen Koordinatensystems und muss daher an die richtige Stelle transliert werden. Da sein Startpunkt beim aktuellen Bauteil liegen soll, ist der entsprechende Vektor dafür t_A . Damit stehen alle erforderlichen Einträge der gesuchten Transformationsmatrix m_{arrow} zur Verfügung. Diese hat letztendlich folgendes Aussehen:

$$m_{arrow} = \begin{bmatrix} side_x & up_x & direction_x & t_{Ax} \\ side_y & up_y & direction_y & t_{Ay} \\ side_z & up_z & direction_z & t_{Az} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.14)$$

Die Länge des Pfeils l entspricht dem Absolutbetrag des Richtungsvektors von Bauteil A (t_A) zu seiner Zielposition ($t_{targetA}$),

$$l = \frac{t_{targetA} - t_A}{|t_{targetA} - t_A|}. \quad (3.15)$$

3.3.8 Animation eines Bauteils

Die Darstellung eines beispielsweise symmetrischen Bauteils an seiner Zielposition ist unter Umständen nicht ausreichend, um exakt festzulegen, wie dieser Bauteil montiert werden muss. Es geht aus der Darstellung nicht eindeutig hervor, wie der Bauteil auszurichten ist. Daher sieht das Konzept vor, eine Animation einzublenden, die einer Bewegung des Bauteils von seiner aktuellen an die Zielposition entspricht. Dadurch wird die Ausrichtung für den Anwender ersichtlich.

Es geht mathematisch betrachtet um die Interpolation zwischen zwei Transformationen im Raum. Diese Interpolation lässt sich jedoch mit Transformationsmatrizen nicht einfach und ohne Einschränkungen lösen. Es ist kein Problem, die Translationen zweier Matrizen zu interpolieren, aber für die Rotationen ist dies nicht möglich. Dieses Problem lässt sich auch mit Hilfe der sogenannten Euler Winkel nicht lösen (vgl. [SHOEMAKE 1985] und [DAM et al. 1998]). Ein Grund dafür liegt darin, dass eine Rotation durch verschiedene Rotationsmatrizen abgebildet werden kann. Daher ist es auch nicht möglich, Rückschlüsse auf die Ausrichtung eines Objekts vor der Rotation zu ziehen. Ein weiterer wichtiger Grund für das Interpolationsproblem liegt im sogenannten *Gimbal Lock*.

Gimbal Lock

Ursprünglich kommt der Begriff aus der Luft- und Raumfahrtindustrie, in der Gyroskope² verwendet werden. Ein Gyroskop besteht im Wesentlichen aus drei konzentrischen Ringen. In Abbildung 3.10 ist das Prinzip eines Gyroskops skizziert. Mit dem Begriff *Gimbal Lock* ist folgende Situation gemeint: Es ist unter bestimmten Umständen möglich, einen Freiheitsgrad in der Berechnung zu verlieren, wodurch die interpolierte Rotation nicht mehr eindeutig festgelegt ist. Ein Beispiel für eine solche Konstellation ist in den Abbildungen 3.10 und 3.11 skizziert. Die drei Rahmen repräsentieren die drei Achsenrichtungen X, Y und Z. Rotationen mit Hilfe von Euler Winkeln werden immer in einer festen Reihenfolge um die drei Achsen durchgeführt, d. h. zuerst um die X-Achse, dann um Y, dann um Z. Eine Rotation um die X-Achse hat das Aussehen, wie in Abbildung 3.10(b) dargestellt. Wenn man diese Konstellation um die Y-Achse rotiert, wie in Abbildung 3.11 zu sehen, tritt der Gimbal Lock auf. Rotationen um die X- und die Z-Achse liefern dann das gleiche Ergebnis, wodurch ein Freiheitsgrad verloren wird [DAM et al. 1998].

Abhilfe bei diesem Problem schafft das mathematische Modell der sogenannten Quaternionen³. Es ist nicht das Ziel dieser Arbeit, eine umfassende Einführung in das Themengebiet der Quaternionen zu geben. Einige gute Quellen zu diesem Thema sind [SHOEMAKE 1985] und [DAM et al. 1998]. Im Folgenden werden jene mathematische Modelle kurz erklärt, die für die vorliegende Arbeit relevant sind.

Interpolation zwischen Rotationen mittels Quaternionen

Das Modell der Quaternionen beschreibt die Rotation eines Punktes um eine beliebige Achse im Raum (siehe Abb. 3.12). Quaternionen werden als 4-dimensionale Vektoren angeschrieben und haben die Form

$$q = s + ix + jy + kz = (s, (x, y, z)). \quad (3.16)$$

Quaternionen können im Prinzip als Verallgemeinerung komplexer Zahlen betrachtet werden, wobei s den realen und x, y, z den imaginären Teil darstellen.

²**Gyroskop** Messgerät für den Nachweis der Achsendrehung der Erde (zu griech. *gyros* „rund“) [HERMANN 1976]

³Quaternionen wurden erstmals 1843 von W.R. Hamilton beschrieben und von Shoemake 1995 in das Feld der Computergrafik übertragen.

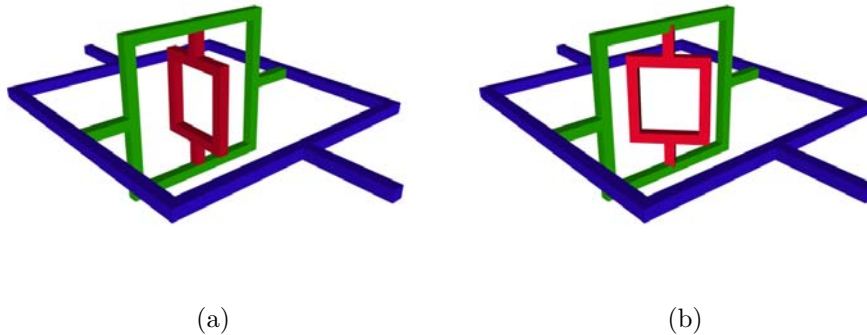


Abbildung 3.10: Die Skizze stellt ein Koordinatensystem dar, wobei der Z-Achse blau ist, die Y-Achse grün und X-Achse rot. Von der Ausgangsstellung (a) wird die X-Achse (rot) um 45° gedreht (b). Aus [DAM et al. 1998].

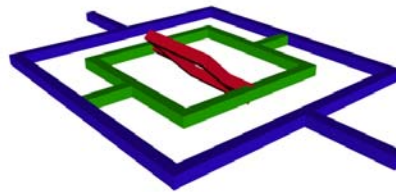


Abbildung 3.11: Gimbal Lock: In dieser Situation sind die Rotationen um die X- und die Z-Achse äquivalent, man hat einen Freiheitsgrad verloren. Aus [DAM et al. 1998].

Jedes Quaternion kann in eine Rotationsmatrix übergeführt werden, die das Aussehen

$$R = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2xy - 2sz & 2sy + 2xz & 0 \\ 2xy + 2sz & 1 - 2(x^2 + z^2) & -2sx + 2yz & 0 \\ -2sy + 2xz & 2sx - 2yz & 1 - 2(x^2 + y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.17)$$

hat [DAM et al. 1998].

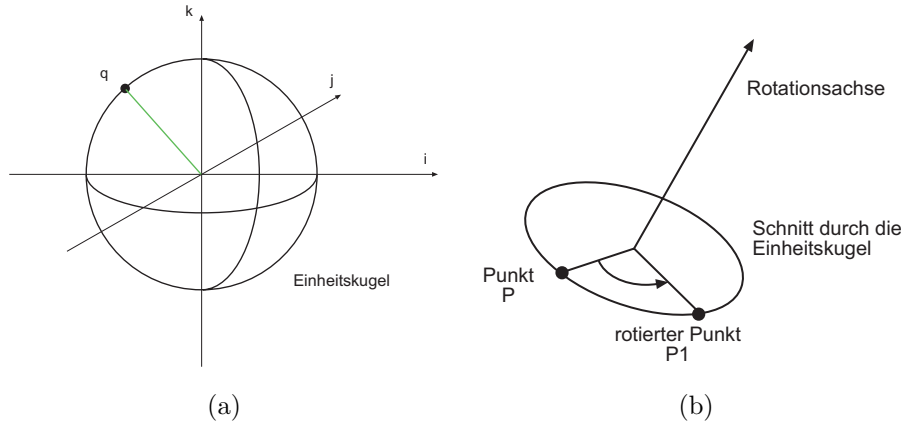


Abbildung 3.12: Quaternionen: Rotation eines Punktes um eine beliebige Achse im Raum auf der Einheitskugel (a) als 3D-Skizze und (b) als Kugelschnitt.

Umgekehrt wird eine Rotationsmatrix über folgende Zusammenhänge in ein Quaternion umgerechnet:

$$R_{11} + R_{22} + R_{33} + R_{44} = 4 - 4(x^2 + y^2 + z^2). \quad (3.18)$$

Es gilt

$$s^2 + x^2 + y^2 + z^2 = 1, \quad (3.19)$$

die Umformung ergibt

$$R_{11} + R_{22} + R_{33} + R_{44} = 4 - 4(1 - s^2). \quad (3.20)$$

Aus diesem Ausdruck werden s und in weiterer Folge die anderen Werte x , y und z berechnet:

$$\begin{aligned} s &= \pm \frac{1}{2} \sqrt{R_{11} + R_{22} + R_{33} + R_{44}}, \\ x &= \frac{R_{32} - R_{23}}{4s}, \\ y &= \frac{R_{13} - R_{31}}{4s}, \\ z &= \frac{R_{21} - R_{12}}{4s}. \end{aligned} \quad (3.21)$$

Abhängig vom Vorzeichen von s ändern sich auch die Vorzeichen der anderen Parameter. Grundsätzlich wird dadurch dieselbe Rotation abgebildet, aber bei einer Interpolation kann sich das Verhalten der Rotation ändern [DAM et al. 1998].

Für die Interpolation zwischen zwei Quaternionen gibt es mehrere mathematische Ansätze. Die drei wichtigsten sind:

1. **L**inear Quaternion **I**nterpolation (Lerp),
2. **S**pherical **L**inear Quaternion **I**nterpolation (Slerp),
3. Spherical Spline Quaternion interpolation (Squad: **s**pherical and **q**uad-rangle).

Diese Ansätze unterscheiden sich hauptsächlich im Beschleunigungsverhalten der Animation eines Objekts, d. h. wie sich die Geschwindigkeit eines Objekts entlang eines Animationspfades ändert. In dieser Arbeit wird auf die zweite Methode zurückgegriffen, mit der zwei Rotationen auf der Einheitskugel interpoliert werden (siehe Abb. 3.13(a)). Es wird für einen Zeitraum $t = [0, 1]$ die kürzeste Verbindung auf der Einheitskugel errechnet (siehe Abb. 3.13(b)). Gegeben seien zwei Quaternionen q_1 und q_2 , gesucht ist das Quaternion q_t zum Zeitpunkt t . Folgende Formel wird zur Berechnung herangezogen:

$$q_t = \text{slerp}(q_1, q_2, t) = q_1 \frac{\sin(\Omega(1-t))}{\sin(\Omega)} + q_2 \frac{\sin(\Omega t)}{\sin(\Omega)}. \quad (3.22)$$

Dabei gilt:

$$\cos(\Omega) = q_1 \cdot q_2 = s_{q_1} s_{q_2} + x_{q_1} x_{q_2} + y_{q_1} y_{q_2} + z_{q_1} z_{q_2}. \quad (3.23)$$

Mit dieser Formel (slerp) wird für jeden Zeitpunkt t eine interpolierte Rotation eines Bauteils berechnet.

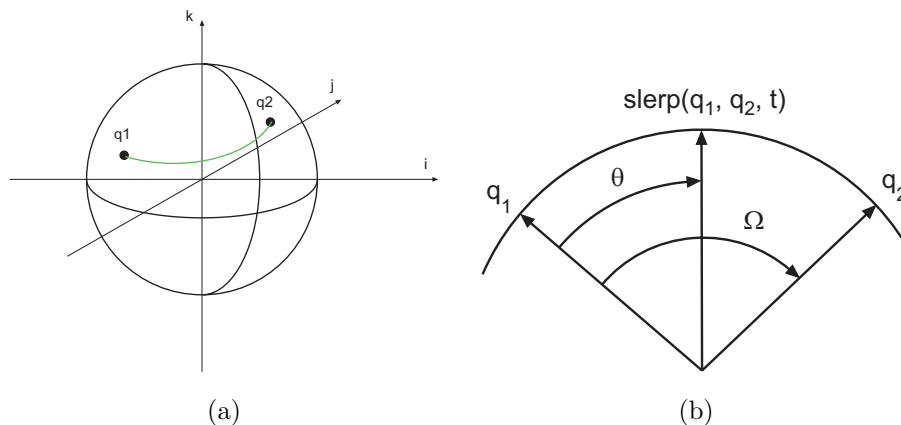


Abbildung 3.13: (a) Interpolation zweier Rotationen auf der Einheitskugel. (b) Auffinden der kürzesten Verbindung (auf der Einheitskugel) zwischen zwei Quaternionen mit der Slerp-Formel.

Interpolation zwischen Translationen

Die Interpolation zwischen zwei Translationsmatrizen lässt sich, im Gegensatz zu den Rotationen, direkt in der Matrixform lösen. Es werden die beiden Translationsvektoren in demselben Zeitraum $t = [0, 1]$ interpoliert.

Gegeben sind die beiden Vektoren t_A und $t_{targetA}$, die man aus den Transformationsmatrizen (m_A und $m_{targetA}$) erhält. Gesucht wird die Position P zum Zeitpunkt t , die zwischen den beiden gegebenen Positionen liegt. Dazu wird die Gerade vom Bauteil A zu dessen Zielposition in Parameterform aufgestellt. Der dafür benötigte Richtungsvektor $t_{direction}$ wird genauso berechnet, wie bereits in Abschnitt 3.3.7 (Darstellung eines Pfeilobjekts) erklärt:

$$t_{direction} = \frac{t_{targetA} - t_A}{|t_{targetA} - t_A|}. \quad (3.24)$$

Eingesetzt in die Parameterform der Geraden, ergibt sich

$$P(t) = t_A + t \cdot t_{direction}. \quad (3.25)$$

Über den Parameter t lassen sich die einzelnen Positionen entlang dieser Geraden ermitteln.

Umsetzung in der Anwendung

Aufbauend auf diesem Konzept laufen in der Anwendung folgende Schritte ab:

- Die Transformationsmatrizen eines Bauteils und dessen Zielposition werden in Quaternionen umgerechnet.
- Zwischen diesen Quaternionen werden mit Hilfe der slerp-Formel alle interpolierten Rotationen innerhalb eines Zeitraums von $t = [0, 1]$ berechnet.
- Diese Rotationen werden jeweils wieder in Rotationsmatrizen zurückgerechnet.
- Auch die Translationen werden wie oben beschrieben interpoliert.
- Die interpolierte Rotation und Translation werden in eine Matrix übertragen und zur Transformation eines 3D-Objekts verwendet.

Die letztendlich berechnete Matrix hat das in Abschnitt 3.3.1 beschriebene Aussehen

$$m = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.26)$$

3.3.9 Prüfen, ob ein Bauteil korrekt eingebaut wurde

Im Application Modus wird dem Anwender entsprechend den Vorgaben aus Abschnitt 3.2 angezeigt, ob er ein Bauteil korrekt montiert hat. Diese Anzeige erfolgt durch eine Änderung der Farbe des 3D-Objekts, das an der Zielposition eingeblendet wird. In diesem Absatz wird erklärt, auf welchen Grundlagen das Programm ermittelt, ob ein Bauteil richtig platziert wurde oder nicht.

Es wird davon ausgegangen, dass zwei Parameter für die Ermittlung der korrekten Montage ausreichen:

1. Die Position (Translation) eines Bauteils.
2. Die Ausrichtung (Rotation) eines Bauteils.

Diese beiden Parameter werden für den Vergleich eines Bauteils mit dessen Zielposition und -ausrichtung herangezogen.

Vergleich der Translationen

Die beiden Translationsvektoren des Bauteils t_A und dessen Zielposition $t_{targetA}$ dienen als Grundlage für die Berechnung. Es wird ein empirisch ermittelter Schwellwert $threshT$ definiert. Sobald der Absolutbetrag der Differenz beider Translationen (bzw. der drei Koordinaten der Vektoren) kleiner als dieser Schwellwert ist, wird ein Bauteil als korrekt positioniert eingestuft. Eine Boole'sche Variable b_{trans} wird entsprechend auf 0 oder 1 gesetzt. Mathematisch ausgedrückt, heißt das

$$\begin{aligned} threshT &= 50, \\ b_{trans} &= (|t_A.x - t_{targetA}.x| < threshT) \wedge \\ &\quad (|t_A.y - t_{targetA}.y| < threshT) \wedge \\ &\quad (|t_A.z - t_{targetA}.z| < threshT). \end{aligned} \tag{3.27}$$

Vergleich der Rotationen

Aus einer Rotationsmatrix geht nicht eindeutig hervor, um welche Achse und mit welchem Winkel ein Punkt rotiert wird. Mit den in Abschnitt 3.3.8 beschriebenen Berechnungsmöglichkeiten der Quaternionen wird dieses Problem gelöst. Quaternionen haben im Gegensatz zu Transformationsmatrizen den Vorteil, dass sie eine Rotation um eine beliebige Achse im Raum mathematisch abbilden. Der Rotationswinkel Θ um diese Achse wird aus einem Quaternion q errechnet und für den Vergleich herangezogen. Folgende Formel dient dabei als Grundlage:

$$\begin{aligned} q &= (s, (x, y, z)), \\ \Theta &= 2 \arccos(s). \end{aligned} \tag{3.28}$$

Für den Vergleich der Rotation eines Bauteils mit dessen Zielrotation werden die beiden Transformationsmatrizen (gegeben durch r_A und $r_{targetA}$) in Quaternionen (q_A und $q_{targetA}$) umgewandelt und die Winkel (Θ_A und $\Theta_{targetA}$) miteinander verglichen. Auch hier wird ein Schwellwert ($threshR$) empirisch ermittelt, unterhalb dessen eine Bauteilrotation als korrekt gilt. Die Boole'sche Variable b_{rot} wird dementsprechend 0 oder 1 gesetzt.

Das Quaternion

$$q_A = (s_A(x_A, y_A, z_A)) \quad (3.29)$$

dreht einen Punkt um den Winkel

$$\Theta_A = 2 \arccos(s_A). \quad (3.30)$$

Das Quaternion

$$q_{targetA} = (s_{targetA}(x_{targetA}, y_{targetA}, z_{targetA})) \quad (3.31)$$

dreht einen Punkt um den Winkel

$$\Theta_{targetA} = 2 \arccos(s_{targetA}). \quad (3.32)$$

Der Schwellwert wird als

$$threshR = 30^\circ = \frac{30PI}{180} rad \approx 0.5 \quad (3.33)$$

festgelegt und der Vergleich

$$b_{rot} = |\Theta_A - \Theta_{targetA}| < threshR \quad (3.34)$$

angestellt.

Schließlich werden die errechneten Variablen b_{trans} und b_{rot} verknüpft. Das endgültige Ergebnis ist die Boole'sche Variable

$$b = b_{trans} \wedge b_{rot}. \quad (3.35)$$

Diese Überprüfung funktioniert in den meisten Fällen mit ausreichender Genauigkeit. Probleme können dann auftreten, wenn die vom Markertracking berechnete Transformationsmatrix starken Schwankungen unterliegt. Dieses Phänomen ist vor allem dann festzustellen, wenn der Betrachtungswinkel zu flach ist und ARToolkit die exakte Ausrichtung des Markers nicht detektieren kann.

Kapitel 4

Implementierung

Der in Abschnitt 3.2 beschriebene Funktionsumfang beider Programmteile wird, unter Berücksichtigung der vorgestellten Konzepte, in entsprechende Komponenten umgesetzt. Außerdem wird die Datenstruktur in geeigneten Klassen implementiert. Der Aufbau beider Modi (Authoring und Application Modus), die Funktionsweise der einzelnen Komponenten und deren Implementierung werden im Folgenden genauer beschrieben. Wie bereits in Abschnitt 2.3 erklärt wurde, unterstützte das AMIRE Framework zum Zeitpunkt der Prototypentwicklung noch keine *Composed Components* (Zusammengesetzte Komponenten). Daher sind teilweise mehrere Funktionen in erweiterten Komponenten zusammengefasst, um die Programmstruktur überschaubar zu halten.

4.1 Datenstruktur

Die Datenstruktur ist entsprechend den Überlegungen aus Abschnitt 3.3.3 als Adjazenzliste festgelegt. Die Implementierung dieser Struktur erfolgt in drei Klassen:

1. **VTInstructionList**: In dieser Klasse wird die Liste der Bauteile in der Reihenfolge ihrer Montage verwaltet.
2. **VTPart**: Diese Klasse verwaltet die Daten eines Bauteils: eine eindeutige ID, die Relativposition zum zuvor montierten Bauteil und eine Liste der Kleinteile.
3. **VTSmallPart**: Diese Klasse ist von **VTPart** abgeleitet. Hier werden die Daten eines Kleinteils verwaltet: die ID, der Anzeigemodus und die Position relativ zu dem Bauteil, auf dem der Kleinteil zu montieren ist.

Das ER-Diagramm in Abbildung 4.1 zeigt das Zusammenspiel dieser Klassen, die Anzahl der auftretenden Entitäten und deren Attribute. Die oben

erwähnten Listen der einzelnen Klassen sind als **vector**-Objekte implementiert. Entsprechende Zugriffsmethoden erlauben jeweils das Hinzufügen von Daten in diese Listen bzw. das gezielte Auslesen von gespeicherten Daten.

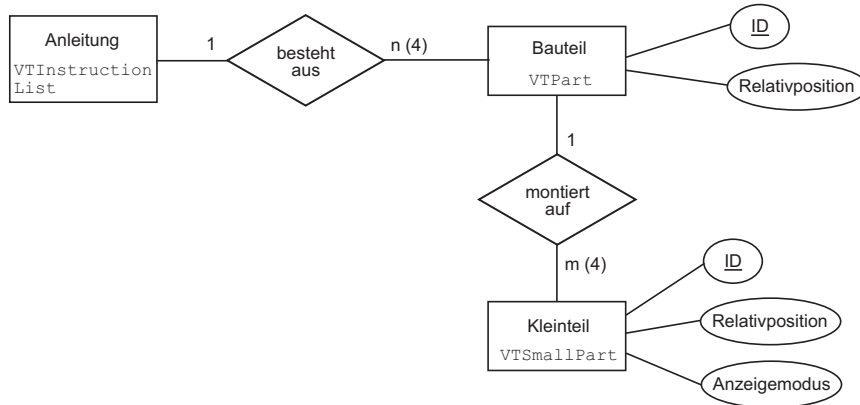


Abbildung 4.1: ER-Diagramm der Klassen für die Datenhaltung.

4.2 Persistenzhaltung der Daten

Die Klasse `VTInstructionList` stellt für das Speichern einer Anleitung im spezifizierten XML-Format (s. Abschnitt 3.3.4) die Methode `save` zur Verfügung. Darin werden alle Daten der Anleitung direkt in einer XML-Datei serialisiert.

Für das Auslesen einer solchen XML-Datei ist die Klasse `XMLLoader` zuständig. Diese Klasse erstellt ein Objekt vom Typ `VTInstructionList` und ein Parserobjekt (`SAXParser`). Diesem Parserobjekt wird ein `Handler` vom Typ `SaxHandler` übergeben. Der Handler verwertet die eingelesenen Daten und füllt die Listen des Objekts `VTInstructionList` (siehe Abb. 4.2).

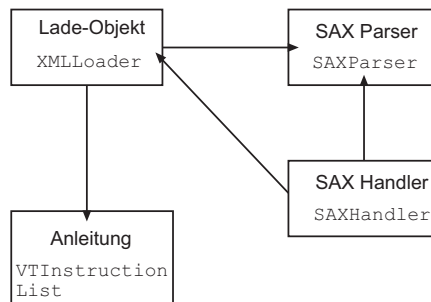


Abbildung 4.2: Die Klassen für das Laden einer Montageanleitung.

4.3 Authoring Modus

Die Aufgaben der Komponenten dieses Programmteiles sind:

- *Markererkennung* für das Bauteil- und Kleinteiltracking,
- Darstellung der *3D-Objekte* (Bauteil-Markierungsrahmen und Kleinteile),
- Anzeige von *Grafiken* der Bauteile, Kleinteile und der Benutzeroberfläche,
- *Abspielen von Soundfiles* für die akustische Rückmeldung,
- *Steuerelemente* (Buttons) für die Benutzerinteraktion,
- *Speichern der Montageschritte*: Schnittstelle zur Datenstruktur und Speichern der Daten im spezifizierten XML-Format,
- *Berechnen der Zielpositionen* bereits gespeicherter Kleinteile eines Bauteils,
- Eine *zentrale Logikkomponente* ist zuständig für: Steuerung der Bildschirmanzeige und Reaktionen auf Benutzeraktionen.

Diese Komponenten werden verknüpft, wie in Abbildung 4.3 skizziert (genauer Verknüpfungsplan siehe Anhang A).

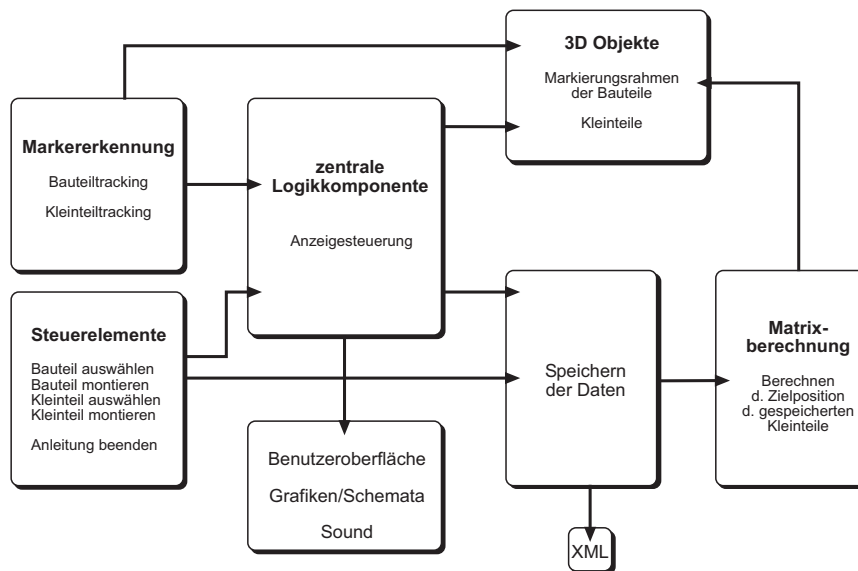


Abbildung 4.3: Komponenten des *Authoring* Modus.

Da das AMIRE Framework bereits diverse Grundfunktionen durch Komponenten bereitstellt, werden diese in der folgenden Beschreibung zuerst betrachtet. Danach werden jeweils eventuelle Erweiterungen für den FAI^{MR} erklärt.

4.3.1 MarkerDetectionComponent: Bauteiltracking über die Markererkennung

Komponente des AMIRE Frameworks: In dieser Komponente wird aufbauend auf ARToolkit das Tracking eines beliebigen Markers realisiert. Die Größe dieses Markers kann frei definiert werden. Sobald der Marker erkannt wird, berechnet die Komponente dessen absolute Position und gibt diese in Form einer Transformationsmatrix aus. Außerdem wird der Trackingstatus eines Markers als Boole'scher Wert *true/false* ausgegeben (siehe Tabelle 4.1 und Abbildung 4.4(a)).

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
markerFile	String	Dateiname des Markers.
markerWidth	Double	Seitenlänge des Markers in cm.
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
detected	Boolean	Trackingstatus: Liefert true, wenn der Marker erkannt wurde.
transformation	Double Vector	Liefert die Transformationsmatrix der absoluten Position des getrackten Markers.

Tabelle 4.1: Bestandteile der MarkerDetectionComponent des AMIRE Frameworks.

Erweiterte Komponente: Entsprechend den Überlegungen aus Abschnitt 3.3.5 wird ein Bauteil durch zwei Marker an der Vorder- und Rückseite identifiziert. Daher wird die vorhandene Komponente so erweitert, dass der zweite Marker ebenfalls getrackt wird. Die vom Markertracking berechnete Transformationsmatrix dieses Markers wird 180° um die Y-Achse gedreht, damit die Ausrichtung des Bauteils korrekt erfasst wird. Die Umsetzung der

Berechnung sieht folgendermaßen aus (Auszug aus dem Programmcode):

```
/* aus dem Markertracking gewonnene Transformationsmatrix */
Matrix transformation;

/* Rotationsmatrix */
Matrix rotM180;

/* Setzen des Drehwinkels (180° entspricht PI in Radiant) */
rotM180.setRotateYAxis (PI);

/* Multiplikation von rechts */
transformation.dotRight(rotM);
```

Die ID eines Bauteils (siehe Abschnitt 3.3.2) wird als eigene Eigenschaft eingegeben und über einen Ausgang an andere Komponenten weitergeleitet (Tabelle 4.2 und Abbildung 4.4(b)). Durch diese Erweiterungen wird aus der ursprünglichen Komponente zum Tracken von Markern eine Komponente zum Tracken von Bauteilen.

Verwendung im Programm: Der FAI^{MR} verwaltet vier verschiedene Bauteile. Aus diesem Grund gibt es im Authoring Modus für das Tracken dieser Teile vier `MarkerDetectionComponents`. Der Marker für das Tracking der Kleinteile wird über eine fünfte Markererkennungskomponente erfasst.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
markerFile_front	String	Dateiname des Markers an der Vorderseite eines Bauteils.
markerFile_back	String	Dateiname des Markers an der Rückseite eines Bauteils.
markerWidth	Double	Seitenlänge beider Marker in cm.
ID	String	ID des Bauteils zur eindeutigen Identifikation.
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
detected	Boolean	Trackingstatus: Liefert true, wenn der Marker erkannt wurde.
transformation	Double Vector	Liefert die Transformationsmatrix der absoluten Position des getrackten Markers.
ID	String	Liefert die ID des Bauteils.

Tabelle 4.2: Für das Programm erweiterte `MarkerDetectionComponent` zum Tracken von Bauteilen und Kleinteilen.

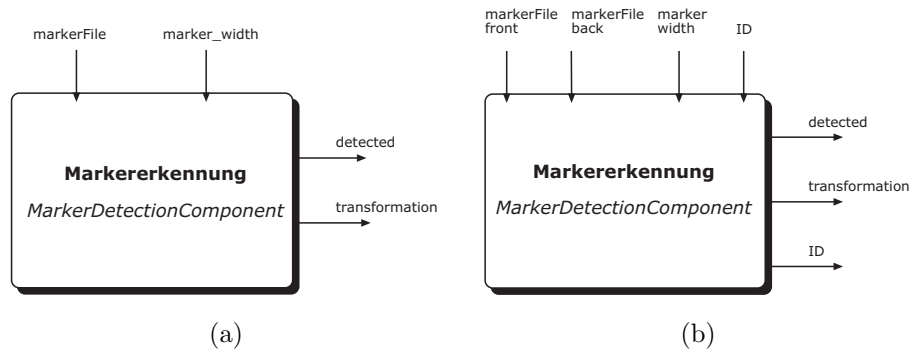


Abbildung 4.4: (a) Die ursprüngliche und (b) die erweiterte `MarkerDetectionComponent`.

4.3.2 GeometryComponent: Darstellung der 3D-Objekte

Komponente des AMIRE Frameworks: Diese Komponente lädt ein 3D-Modell (im 3ds Format) und stellt dieses in der Szene dar. Über eine Eigenschaft (`occluder`) lässt sich das Modell als *Occluder*¹ definieren. Die Position des Modells wird zur Programm-Laufzeit über einen Eingangskanal durch eine Transformationsmatrix definiert/verändert. Außerdem wird dessen Sichtbarkeit, abhängig vom Trackingstatus, gesteuert (siehe Tabelle 4.3 und Abbildung 4.5(a)).

Erweiterte Komponente: Diese Komponente wird so erweitert, dass drei verschiedene 3D-Modelle geladen werden können. Über einen zusätzlichen Eingang (`displayMode`) wird angesteuert, welches dieser drei Modelle angezeigt wird. Damit kann beispielsweise eine Geometrie in drei verschiedenen Farben mit *einer* `GeometryComponent` verwaltet werden. In Tabelle 4.4 und Abbildung 4.5(b) sind die Erweiterungen zusammengefasst.

¹engl. occlude: absorbieren; hier im Sinne von verdecken/Verdeckung gebraucht. Das Objekt verdeckt andere 3D-Objekte in der Szene, die dahinter liegen, ist aber selbst transparent.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
geometryFileName	String	Dateiname des 3D-Objekts (im 3ds Format).
occluder	Boolean	<i>true</i> : Objekt ist ein Occluder, verdeckt daher dahinterliegende Objekte und ist selbst transparent. <i>false</i> : 3D-Objekt wird normal dargestellt.
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
detected	Boolean	<i>true</i> : 3D-Objekt wird angezeigt an der Position von transformation . <i>false</i> : Objekt wird nicht angezeigt.
transformation	Double Vector	Transformationsmatrix der absoluten Position des 3D-Objekts in der Szene.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-

Tabelle 4.3: Bestandteile der GeometryComponent des AMIRE Frameworks.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
geometryFile1	String	Dateiname des ersten 3D-Objekts (im 3ds Format).
geometryFile2	String	Dateiname des zweiten 3D-Objekts.
geometryFile3	String	Dateiname des dritten 3D-Objekts.
occluder	Boolean	<i>true</i> : Objekt ist ein Occluder, verdeckt daher dahinterliegende Objekte und ist selbst transparent. <i>false</i> : 3D-Objekt wird normal dargestellt.
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
detected	Boolean	<i>true</i> : 3D-Objekt wird angezeigt an der Position von transformation . <i>false</i> : Objekt wird nicht angezeigt.
transformation	Double Vector	Transformationsmatrix der absoluten Position des 3D-Objekts in der Szene.
displayMode	Integer	Dieser Eingang steuert, welches der drei Objekte angezeigt wird; er kann die Werte 0, 1 oder 2 haben.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-

Tabelle 4.4: Parameter der erweiterten GeometryComponent.

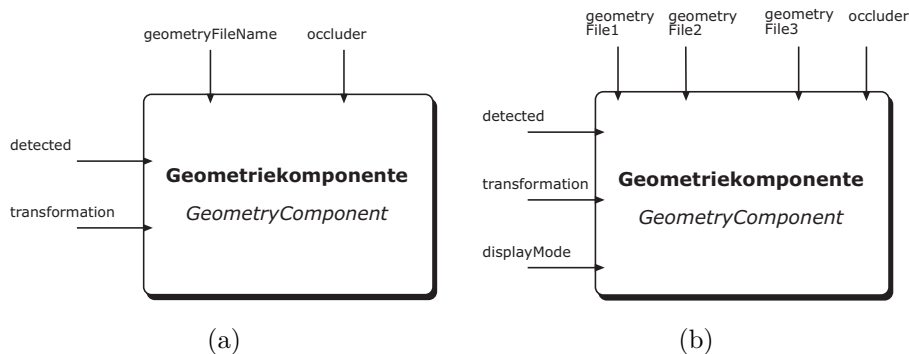


Abbildung 4.5: (a) Die ursprüngliche und (b) die erweiterte `GeometryComponent`.

Verwendung im Programm: Der Authoring Modus besteht aus folgenden Geometrie-Komponenten:

- Vier `GeometryComponents` für die Markierungs-Rahmen der Bauteile,
- eine Geometrie-Komponente enthält die drei Kleinteile (Schraube, Dübel, Schrauben-Gegenhalter),
- vier weitere Geometrie-Komponenten enthalten nochmals die Kleinteile. Diese werden angezeigt, wenn der Autor die Kleinteile eines Bauteils speichert (maximal vier Stück pro Bauteil möglich).

Die Farbänderungen der 3D-Rahmen zum Markieren der Bauteile werden realisiert, indem ein solcher Rahmen in den drei Farben grau, grün und rot geladen wird (siehe Abb. 4.6). Die entsprechende Zuordnung ist:

- **Anzeigemodus 0:** Grauer Rahmen (Standardfarbe).
- **Anzeigemodus 1:** Grüner Rahmen (Markierter Bauteil).
- **Anzeigemodus 2:** Roter Rahmen (Montierter/gesicherter Bauteil).

Die Zuordnung der drei Kleinteile des Testmöbelstückes ist (siehe Abb. 4.7):

- **Anzeigemodus 0:** Lange Schraube.
- **Anzeigemodus 1:** Schrauben-Gegenhalter.
- **Anzeigemodus 2:** Dübel.

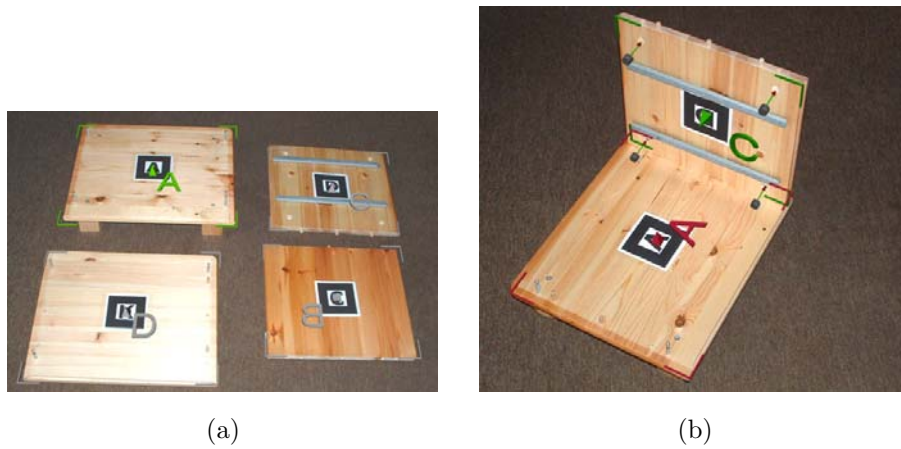


Abbildung 4.6: Markierungs-Rahmen der Bauteile: (a) Nicht-markierte Bauteile sind grau umrahmt, der selektierte Bauteil A grün. (b) Der gespeicherte Bauteil A ist rot umrahmt, der neu selektierte Bauteil C grün.

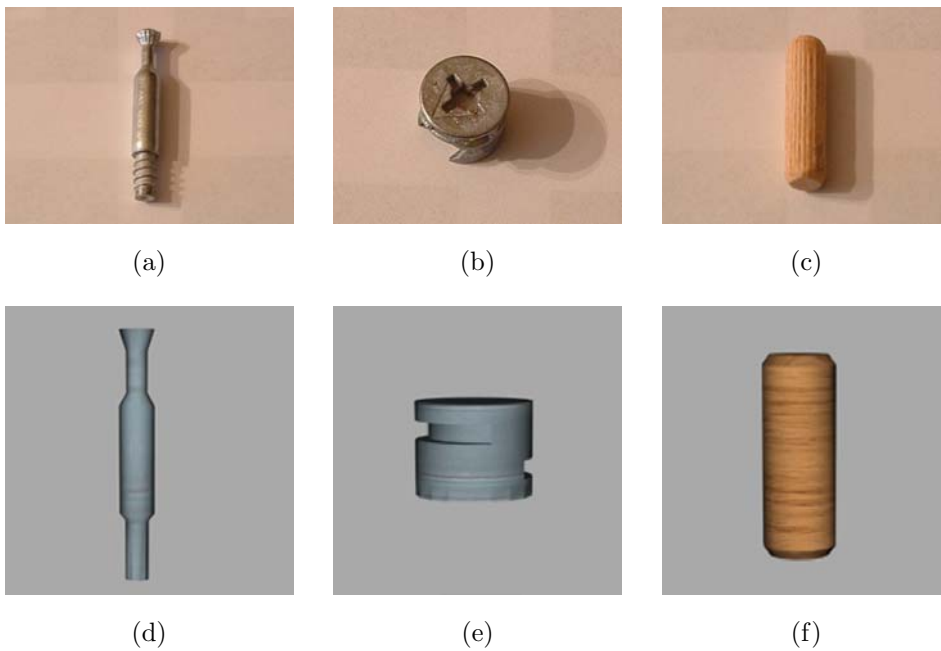


Abbildung 4.7: Die drei Kleinteile des Testmöbelstücks. Reale Bauteile: (a) Schraube, (b) Schrauben-Gegenhalter und (c) Dübel. 3D-Modelle dieser Teile : (d), (e) und (f).

4.3.3 SoundComponent: Audiosteuerung

Für das Abspielen von Sounddateien gab es zum Zeitpunkt der Prototypentwicklung noch keine eigene Komponente vom AMIRE Framework. Die für das Programm entworfene Komponente ermöglicht das Abspielen von maximal zehn verschiedenen Audiodateien. Diese werden über entsprechende Eigenschaften geladen. Das Abspielen der einzelnen Dateien wird über Komponenten-Eingänge aktiviert (siehe Tabelle 4.5 und Abbildung 4.8). Als Grundlage dafür dient die Bibliothek *OpenAL*².

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
soundFileNames	String Vector	Dateiname(n) der maximal zehn Audiodateien.
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
play_sound1-10	Boolean	Abspielen der Sounddatei 1..10.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-

Tabelle 4.5: Die Audiokomponente SoundComponent des Programms.

Verwendung im Programm: Es wird für den Authoring Modus nur eine Audiokomponente benötigt, die drei verschiedene Sounddateien abspielt (siehe Tabelle 3.2 in Abschnitt 3.2).

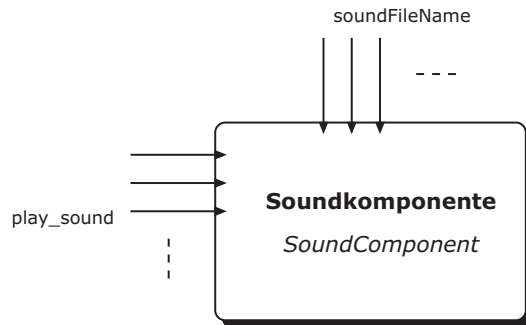


Abbildung 4.8: Aufbau der Audiokomponente.

²www.openal.org

4.3.4 ButtonComponent: Steuerelemente

Komponente des AMIRE Frameworks: Die Komponente des AMIRE Frameworks erlaubt das Laden von zwei Grafiken (im jpg oder tif Format), die als Bilder für einen Button im gedrückten bzw. ungedrückten Zustand verwendet werden. Die Größe und Position des Buttons wird über entsprechende Eigenschaften eingestellt. Die Sichtbarkeit des Buttons wird über einen Eingang gesteuert. Der Status des Buttons, der entweder gedrückt oder nicht gedrückt ist, wird an den Ausgang der Komponente gelegt.

Verwendung im Programm: Der Authoring Modus enthält entsprechend dem Konzept fünf verschiedene Schaltflächen-Komponenten:

- Bauteil auswählen,
- Kleinteil auswählen,
- Kleinteil montieren,
- Bauteil montieren,
- Anleitung beenden.

Für jeden dieser Buttons gibt es ein eigenes Symbol. Der Aufbau der Steuerleiste ist in Abbildung 4.9 dargestellt.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
imageFileName	String	Dateiname der Button-Grafik für den ungedrückten Zustand.
toggledImageFileName	String	Dateiname der Button-Grafik für den gedrückten Zustand.
position	Point2D	Position des Buttons auf der Bildfläche.
dimension	Dimension2D	Größe des Buttons.
depthLevel	String	Tiefenebene, auf der ein Button liegt (relevant, wenn z. B. dahinter eine Grafik einer Benutzeroberfläche eingeblendet wird).
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
visible	Boolean	Steuert die Sichtbarkeit eines Buttons.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
toggled	Boolean	Liefert true , wenn ein Button im gedrückten Zustand ist und false , wenn er im ungedrückten Zustand ist.

Tabelle 4.6: Bestandteile der ButtonComponent des AMIRE Frameworks.

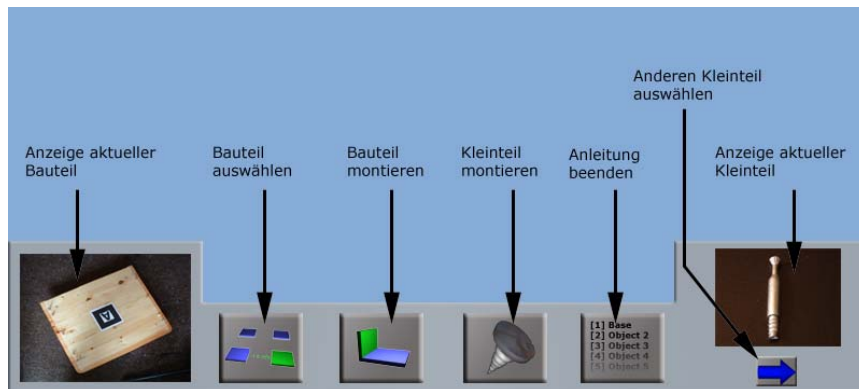


Abbildung 4.9: Steuerelemente des Authoring Modus.

4.3.5 SchematicComponent: Grafische Benutzeroberfläche und Bilder

Komponente des AMIRE Frameworks: Diese Komponente verwaltet ein oder mehrere Bilder, deren Dateinamen über eine Eigenschaft definiert sind. Der Name „Schema“ kommt daher, dass die einzelnen Bilder der Reihe nach angezeigt werden, wenn der Benutzer sie mit der Maus anklickt. Die Position und Größe der Bilder werden wie bei der `ButtonComponent` über eigene Eigenschaften eingestellt. Es gibt nur einen Eingang `visible`, der die Sichtbarkeit eines Schemas steuert (siehe Tabelle 4.7).

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
imageFileNames	String Vector	Dateiname(n) der Grafik(en) eines Schemas.
position	Point2D	Position des Schemas auf der Bildfläche.
dimension	Dimension2D	Größe des Schemas.
depthLevel	String	Tiefenebene, auf der ein Schema liegt (relevant, wenn z. B. dahinter eine andere Grafik eingeblendet wird).
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
visible	Boolean	Steuert die Sichtbarkeit eines Schemas.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-

Tabelle 4.7: Bestandteile der `SchematicComponent` des AMIRE Frameworks.

Verwendung im Programm: Diese Komponente findet im Authoring Modus zweierlei Anwendung. Einerseits werden die Hintergründe für die Benutzeroberfläche damit geladen und angezeigt. Andererseits wird die Anzeige der Grafiken von Bauteilen und Kleinteilen mit Hilfe von Schemas realisiert, wobei für jeden Bauteil zwei Grafiken geladen werden, die diesen von zwei verschiedenen Blickrichtungen zeigen. Es werden folgende Komponenten eingesetzt:

- Eine `SchematicComponent` für den Hintergrund der Benutzeroberfläche.
- Für jeden der vier Bauteile ein Schema, das zwei Bilder des jeweiligen Bauteils enthält. Dieses Schema wird am Bildschirmrand links unten angezeigt.
- Jeweils eine Schema-Komponente für die drei Kleinteile, zur Anzeige rechts unten.

4.3.6 `AMListComponent`: Speichern der Montageschritte

Diese Komponente dient als Schnittstelle zur Datenstruktur. Es wird eine Instanz der Klasse `VTInstructionList` angelegt. Entsprechend den Aktionen des Autors werden Bauteil-Objekte (`VTPart`) und Kleinteil-Objekte (`VTSmallPart`) erzeugt und in die entsprechenden Listen gespeichert. Es werden drei Benutzerinteraktionen verwaltet:

- **Ausgewählten Bauteil montieren** (`store_button_trigger`): Die Relativposition des aktuellen Bauteils zum zuvor montierten Bauteil wird in der Komponente berechnet. Diese Transformationsmatrix und die Bauteil-ID werden in die Datenstruktur gespeichert (siehe Abschnitt 3.3.2).
- **Ausgewählten Kleinteil montieren** (`smallPart_button_trigger`): Die relative Position des Kleinteils zum aktuellen Bauteil wird berechnet. Das Ergebnis der Berechnung, die Kleinteil-ID sowie der Anzeigemodus (d. h. die Art des Kleinteils) werden in die Kleinteil-Liste des aktuellen Bauteils eingetragen. Die bereits gespeicherten Kleinteile eines Bauteils (maximal vier) werden weiterhin in der Szene eingeblendet. Dafür werden die gespeicherten Daten (Transformationsmatrizen und Anzeigemodus) an Ausgänge gelegt. Vier separate `GeometryComponents` enthalten die Kleinteile als 3D-Objekte und werden entsprechend in die Szene transformiert und angezeigt. Sobald der Autor einen Bauteil-Montageschritt beendet (durch Betätigen der Schaltfläche „Bauteil montieren“) und einen neuen Bauteil auswählt, werden diese Kleinteile nicht mehr angezeigt.

- **Anleitung beenden/speichern (save_button_trigger):** Die gesamte Anleitung wird im spezifizierten XML-Format (Abschnitt 3.3.4) in einer Datei ausgegeben. Dafür wird die save-Methode der Klasse VTInstructionList verwendet.

Die genaue Aufstellung der Parameter dieser Komponente ist Abbildung 4.10 und den Tabellen 4.8 bzw. 4.9 zu entnehmen.

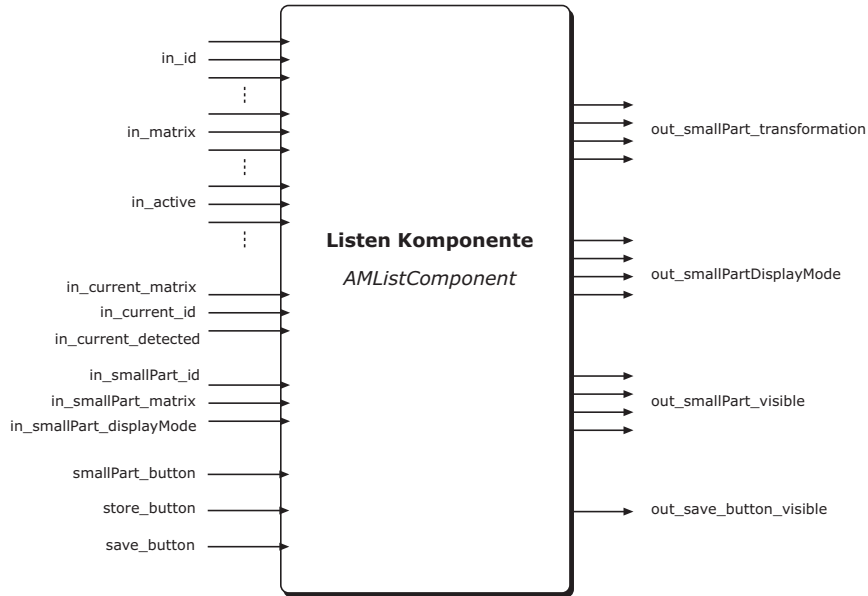


Abbildung 4.10: Die Listenkomponente und deren Ein- und Ausgänge.

<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
smallPart trans1-4	Double Vector	Positionen der bereits gespeicherten Kleinteile (maximal vier) relativ zum aktuellen Bauteil.
smallPart visible1-4	Boolean	Sichtbarkeit der bereits gespeicherten Kleinteile.
smallPart DisplayMode1-4	Integer	AnzeigeModus (bzw. Art) der bereits gespeicherten Kleinteile.
save_butto_active	Boolean	Steuert die Sichtbarkeit des Buttons, mit dem der Autor die Anleitung beendet/speichert. Der Button wird nach dem ersten Montageschritt sichtbar.

Tabelle 4.8: Ausgänge der Listenkomponente AMListComponent des Authoring Modus.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
active1-4	Boolean	Verbindung zur Logikkomponente; empfangen, ob ein Montageschritt gespeichert werden kann oder nicht.
matrix1-4	Double Vector	Verbindung zu den MarkerDetectionComponents der vier Bauteile; empfangen die Positionen der Bauteile (Transformationsmatrizen).
id1-4	String	Verbindung zu den MarkerDetectionComponents der vier Bauteile; empfangen die IDs der Bauteile.
current_id	String	Verbindung zur Logikkomponente; empfängt die ID des aktuell selektierten Bauteils.
current_detected	Boolean	Verbindung zur Logikkomponente; empfängt den Trackingstatus der aktuell selektierten Bauteils.
current_matrix	Double Vector	Verbindung zur Logikkomponente; empfängt die Transformationsmatrix der aktuell selektierten Bauteils als Grundlage für die Berechnung Relativposition zum vorigen Bauteil.
smallPart_matrix	Double Vector	Verbindung zur MarkerDetectionComponent des Kleinteil-Markers; empfängt die Transformationsmatrix des Kleinteiltrackings als Grundlage für die Berechnung der relativen Position eines Kleinteils zum aktuellen Bauteil.
smallPart DisplayMode	Integer	Verbindung zur Logikkomponente; empfängt den Anzeigemodus der Geometriekomponente der drei Kleinteile, d. h. das aktuelle Kleinteil.
store_button trigger	Boolean	Verbindung zum Button, mit dem der Autor die Position eines Bauteils speichert.
smallPart_button trigger	Boolean	Verbindung zum Button, mit dem der Autor die Position eines Kleinteils auf einem Bauteil speichert.
save_button trigger	Boolean	Verbindung zum Button, mit dem der Autor die Anleitung beendet/speichert.

Tabelle 4.9: Eingänge der Listenkomponente **AMListComponent** des Authoring Modus.

4.3.7 MatrixCalculateComponent: Berechnen der Zielposition bereits gespeicherter Kleinteile

Diese Komponente berechnet eine Zielposition aus einer absoluten Position (vom Markertracking generiert) und einer relativen Position (bereitgestellt aus der Datenstruktur). Diese beiden Position werden als Matrizen über die zwei Eingänge `base_matrix` und `relative_matrix` an die Komponente übertragen. Sobald der Eingang `active` den Wert `true` empfängt, wird die Berechnung durchgeführt und die Zielposition an den Ausgang `out_matrix` gesendet (siehe Tabelle 4.10 und Abbildung 4.11).

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
base_matrix	Double Vector	Absolute Position eines Bauteils.
relative_matrix	Double Vector	Relative Position zu einem Bauteil.
active	Boolean	Steuert, ob die Berechnung durchgeführt wird oder nicht.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
out_matrix	Double Vector	Berechnete Zielposition aus den beiden Eingangsmatrizen.

Tabelle 4.10: Parameter der MatrixCalculateComponent.

Die Berechnung basiert auf den Überlegungen aus Abschnitt 3.3.1 und wird mit Hilfe der `Matrix`-Klasse des AMIRE Frameworks im Programm wie folgt umgesetzt:

```

if (active) {
    /* Matrix für die Zielposition */
    Matrix result;
    /* absolute Position aus dem Eingangskanal in Matrix eintragen */
    result.set(base_matrix);
    /* Matrix der relativen Position */
    Matrix object_relative;
    /* Daten aus dem zweiten Eingangskanal in Matrix eintragen */
    object_relative.set(relative_matrix);
    /* Berechnung durchführen */
    result.dotRight(object_relative);
    /* Resultat in die Ausgangs-Matrix eintragen */
    out_matrix.set(result);
}

```

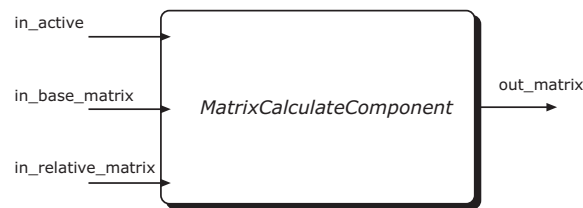


Abbildung 4.11: Komponente zur Berechnung der Zielposition eines Bauteils oder Kleinteils.

Verwendung im Programm: Die Listenkomponente liefert die Relativpositionen der gespeicherten Kleinteile eines Bauteils über ihre Ausgänge. Diese Positionen werden in vier `MatrixCalculateComponents`, ausgehend von der Position des aktuellen Bauteils (aus dem Markertracking), in Zielpositionen umgerechnet. Damit werden die 3D-Modelle der gespeicherten Kleinteile in die Szene transformiert und auf dem aktuellen Bauteil korrekt platziert dargestellt (siehe Abb. 4.12).

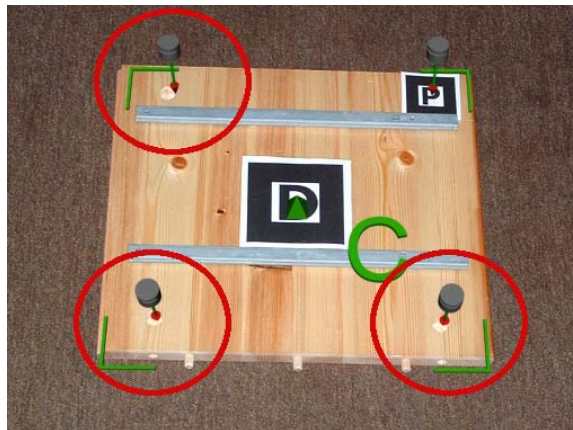


Abbildung 4.12: Die bereits gespeicherten Kleinteile eines Bauteils werden ebenfalls angezeigt.

4.3.8 AUTMLogicComponent: Zentrale Logik

Diese Komponente ist für die primäre Steuerung der Anzeige der grafischen Elemente zuständig:

- 3D-Markierungsrahmen der Bauteile: die Anzeigemodi werden entsprechend den Benutzeraktionen umgeschaltet.
- Das zum aktuell selektierten Bauteils passende Schema wird angezeigt.

- Das korrekte Kleinteil wird als 3D-Objekt angezeigt, wenn der Kleinteilmarker getrackt wird.
- Das zum Kleinteil passende Schema wird angezeigt.

Die Steuerung basiert auf den Daten des Markertrackings. Es wird registriert, welcher Bauteil gerade selektiert ist, welche Bauteile bereits montiert wurden und welche Bauteile noch verwendet werden müssen. Daraus resultieren die Farben der Markierungsrahmen und die Anzeige der Bauteilgrafiken. Darüber hinaus werden die Daten des Markers für das Kleinteiltracking verarbeitet. Aufbauend darauf wird die korrekte Kleinteilgrafik (2D und 3D) angezeigt.

Außerdem ist die Logikkomponente mit der Listenkomponente verbunden. Die Daten des aktuell selektierten Bauteils werden weitergeleitet, damit diese für die Speicherung in die Datenstruktur verfügbar sind:

- Bauteilposition: `current_transformation`.
- Bauteil-ID: `current_id`.
- Trackingstatus: `current_detected`.

Eine Aufstellung aller Ein- und Ausgänge ist in Abbildung 4.13 sowie den Tabellen 4.11 und 4.12 enthalten.

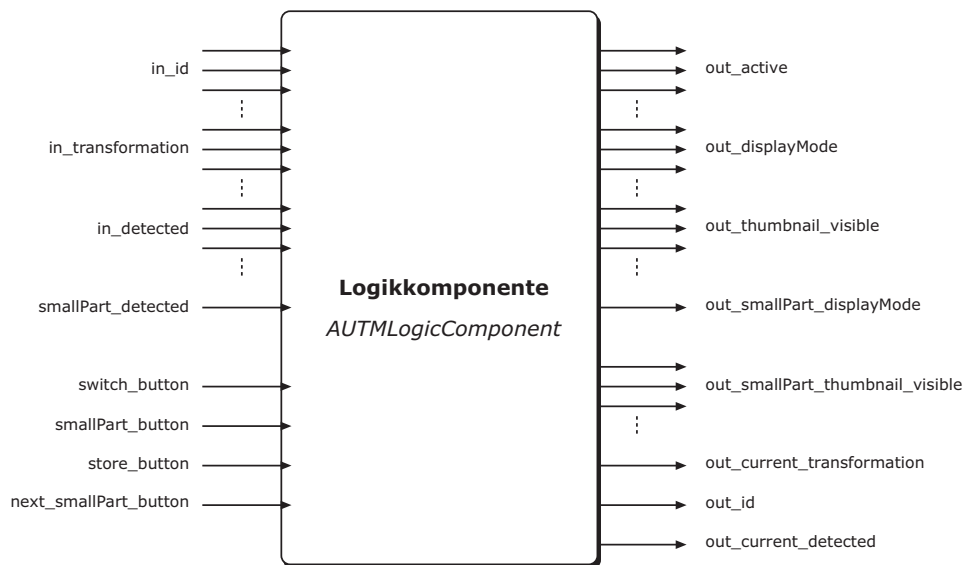


Abbildung 4.13: Die Logikkomponente des Authoring Modus und deren Ein- und Ausgänge.

<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
displayMode1-4	Integer	Verbindung zu den GeometryComponents der vier Bauteil-Markierungsrahmen; steuern die Anzeige des zum Bauteilstatus passenden Rahmens.
active1-4	Boolean	Verbindung zur Listenkomponeute; steuern, ob ein Bauteilmontageschritt gespeichert werden kann.
thumbnail visible1-4	Boolean	Verbindung zu den vier SchematicComponents (Bilder) der Bauteile; steuern, ob und welches Bauteil-Schema angezeigt wird.
smallPart displayMode	Integer	Verbindung zur GeometryComponent die alle drei Kleinteilgattungen des Testmöbelstücks enthält; steuert die Anzeige des aktuell ausgewählten Kleinteils (über den nextSmallPart Button).
smallPart thumbnail visible1-3	Boolean	Verbindung zu den drei SchematicComponents der Kleinteile; steuern, ob und welches Kleinteil-Schema angezeigt wird.
current transformation	Double Vector	Verbindung zur Listenkomponeute AMLListComponent zum Speichern der Montageschritte in die Datenstruktur; liefert die Position des aktuell selektierten Bauteils.
current_id	String	Verbindung zur Listenkomponeute AMLListComponent zum Speichern der Montageschritte in die Datenstruktur; liefert die ID des aktuell selektierten Bauteils.
current_detected	Boolean	Verbindung zur Listenkomponeute AMLListComponent zum Speichern der Montageschritte in die Datenstruktur; liefert, ob der aktuell selektierte Bauteil vom Markertracking erfasst wird.
switch_button active	Boolean	Steuert die Sichtbarkeit des Buttons, mit dem der Autor einen Bauteil auswählt.
store_button active	Boolean	Steuert die Sichtbarkeit des Buttons, mit dem der Autor die Position eines Bauteils speichert.
smallParts button active	Boolean	Steuert die Sichtbarkeit der beiden Buttons, mit dem der Autor einen anderen Kleinteil auswählt bzw. die Position eines Kleinteils auf einem Bauteil speichert.

Tabelle 4.11: Ausgänge der Logikkomponeute **AUTMLogicComponent** des Authoring Modus.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
transformation1-4	Double Vector	Verbindung zu den MarkerDetectionComponents der vier Bauteile; empfangen die Positionen der Bauteile (Transformationsmatrizen).
id1-4	String	Verbindung zu den MarkerDetectionComponents der vier Bauteile; empfangen die IDs der Bauteile.
detected1-4	Boolean	Mit den MarkerDetectionComponents der vier Bauteile verbunden; empfangen, ob ein Bauteil erfasst wird oder nicht.
smallPart detected	Boolean	Verbindung zur MarkerDetectionComponent des Kleinteil-Markers. empfängt, ob der Marker erfasst wird oder nicht.
switch_button	Boolean	Verbindung zum Button, mit dem der Autor einen anderen Bauteil auswählen kann.
store_button	Boolean	Verbindung zum Button, mit dem der Autor die Position eines Bauteils speichert.
smallPart_button	Boolean	Verbindung zum Button, mit dem der Autor die Position eines Kleinteils auf einem Bauteil speichert.
nextSmallPart button	Boolean	Verbindung zum Button, mit dem der Autor einen anderen Kleinteil auswählt.

Tabelle 4.12: Eingänge der Logikkomponente **AUTMLogicComponent** des Authoring Modus.

4.4 Application Modus

In den folgenden Absätzen werden die Komponenten des Application Modus beschrieben. Dieser Modus verwendet teilweise Bestandteile, die auch im Authoring Modus vorkommen. Die Aufgaben der Komponenten dieses Programmteiles sind im Einzelnen:

- *Markererkennung* für das Bauteiltracking,
- Darstellung der *3D-Objekte*: Bauteil-Markierungsrahmen, Bauteile an der Zielposition, Kleinteile),
- Anzeige von *Grafiken* der Bauteile, Kleinteile, Werkzeuge und der Benutzeroberfläche,

- Anzeige von *Erklärungstexten* zu den Montageschritten,
- *Berechnen der Zielpositionen* von Bauteilen und Kleinteilen,
- Anzeige eines *Pfeilobjekts* von einem Bauteil zu dessen Zielposition,
- Berechnen der Transformationsmatrizen eines *Animationspfades* auf Quaternionenbasis und Anzeige der Animation,
- Prüfung, ob ein *Bauteil korrekt montiert* wurde,
- *Steuerelemente* (Buttons) für die Benutzerinteraktion.
- Das *Laden der Montageschritte* aus dem spezifizierten XML Format, die Steuerung des Programmablaufes und Reaktionen auf Benutzeraktionen werden in einer Logikkomponente realisiert.

Die Verknüpfung dieser Komponenten ist in Abbildung 4.14 skizziert (genauer Verknüpfungsplan siehe Anhang A).

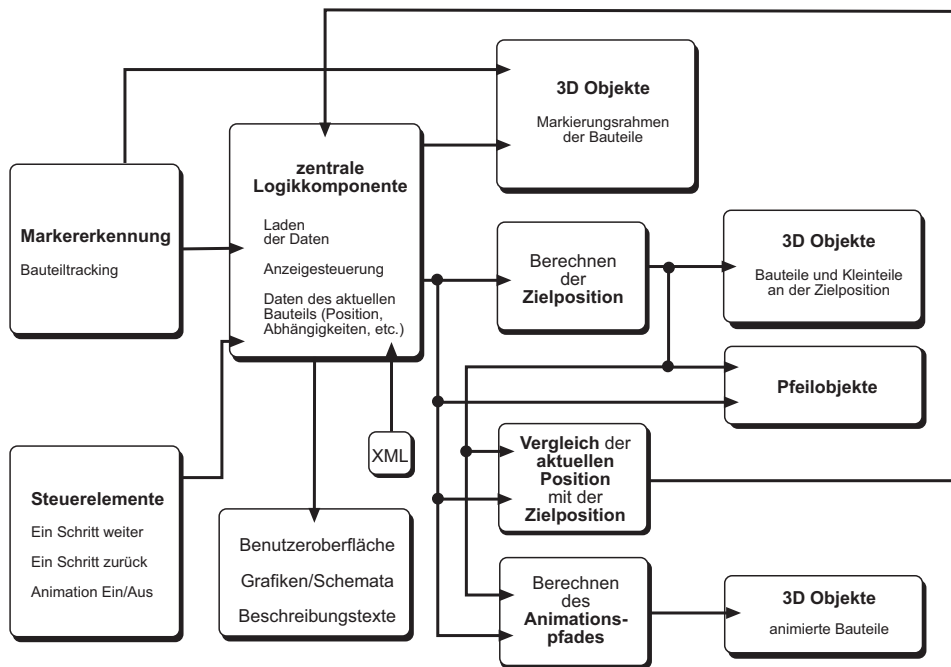


Abbildung 4.14: Komponenten des *Application* Modus.

4.4.1 MarkerDetectionComponent: Markererkennung für das Bauteiltracking

Für das Tracking der vier Bauteile des Beispielmöbelstücks wird jeweils eine `MarkerDetectionComponent` eingesetzt. Diese Komponenten sind sowohl mit den Geometrie-Komponenten zur Darstellung der Markierungsrahmen als auch mit der zentralen Logikkomponente des Application Modus verbunden (siehe Anhang B). Die Beschreibung dieser Komponente ist Abschnitt 4.3.1 zu entnehmen.

4.4.2 GeometryComponent: Darstellung der 3D-Objekte

Zur Darstellung der 3D-Objekte wird wie im Authoring Modus die Komponente `GeometryComponent` (siehe Abschnitt 4.3.2) verwendet. Es gibt im Application Modus drei verschiedene Arten von 3D-Objekten:

- 3D-Markierungsrahmen für die Bauteile: Die Positionen der Rahmen werden von den `MarkerDetectionComponents` übertragen. Es werden nur die zwei Rahmenfarben *grün* (aktueller Bauteil) und *grau* (sonstiger Bauteil) benötigt.
- 3D-Modelle der Bauteile und Kleinteile zur Anzeige an deren Zielpositionen,
- 3D-Modelle der Bauteile zur Anzeige in einer Animation von den realen Bauteilen zu deren Zielpositionen.

4.4.3 SchematicComponent: Anzeige von Grafiken

Die Grafiken werden wie im Authoring Modus mit `SchematicComponents` (siehe Abschnitt 4.3.5) geladen und angezeigt.

- Zwei Schemata für den Hintergrund der Benutzeroberfläche: Eine Grafik als Hintergrund für die Bedien- und Anzeigenleiste, die andere Grafik für die Anzeige der Erklärungen zu den Montageschritten.
- Je ein Schema für die Bauteile und Kleinteile.
- Zu jedem Kleinteil wird das Bild des benötigten Werkzeuges geladen.

4.4.4 TextComponent: Anzeige von Erklärungstexten

Ein beliebiger Text wird über eine Eigenschaft definiert. Dieser Text ist über zwei Koordinaten auf dem Bildschirm frei platzierbar. Ein Eingang (`visible`) steuert, ob der Text angezeigt wird oder nicht (siehe Tabelle 4.13 und Abbildung 4.15).

Die Darstellung des Texts wird mit Hilfe der Methode `glutBitmapCharacter` realisiert. Diese Methode ist Teil der OpenGL-Bibliothek *glut*.

Bevor ein Text am Bildschirm gerendert wird, muss der Anzeigemodus auf orthografische Projektion umgestellt werden:

```
void setOrthographicProjection() {
    /* Projektions Modus */
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    /* Matrix zurücksetzen */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, 0.0, 1.001);
    glMatrixMode(GL_MODELVIEW);
}
```

Die Darstellung des Texts geschieht in der Methode `renderBitmapString`.

```
void renderBitmapString(float x, float y, void *font, const char
*string[]) {
    const char *c;
    /* Startposition setzen */
    glRasterPos2f(x, y);
    /* Alle Elemente des Strings durchlaufen */
    for (c=*string; *c != '\0'; c++) {
        if (*c == '\n') {
            y-=0.05;
            glRasterPos2f(x, y);
            c++;
        }
        glutBitmapCharacter(font, *c);
    }
}
```

Danach wird der Anzeigemodus wieder auf perspektivische Projektion zurückgesetzt.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
text	String	Text, der angezeigt werden soll.
posX, posY	Double	Position des Textes am Bildschirm.
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
visible	Boolean	Steuert die Anzeige des Texts.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-

Tabelle 4.13: Parameter der `TextComponent`.

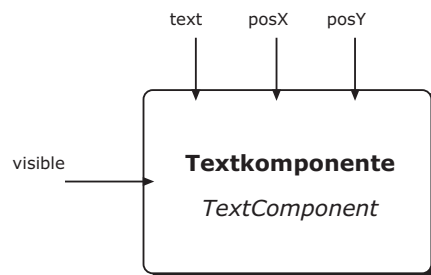


Abbildung 4.15: Aufbau der Textkomponente.

Verwendung im Programm: Der Application Modus ist laut Konzept in mehrere Teilschritte unterteilt (siehe Abbildung 3.3). Zu jedem Montageschritt wird ein Erklärungstext eingeblendet, bevor der Anwender die notwendige Aktion ausführt.

4.4.5 MatrixCalculateComponent: Berechnen der Zielposition von Bauteilen und Kleinteilen

Die Berechnung der Zielpositionen von Kleinteilen und Bauteilen wird wie im Authoring Modus von `MatrixCalculateComponents` (siehe Abschnitt 4.3.7) durchgeführt. Insgesamt werden acht Berechnungs-Komponenten verwendet: Für jeden Bauteil und vier Kleinteile (die auf einem Bauteil montiert sein können) berechnet je eine `MatrixCalculateComponent` die Zielposition. An dieser Position wird ein entsprechendes 3D-Modell eingeblendet.

4.4.6 ArrowComponent: Anzeige der Pfeilobjekte

Diese Komponente stellt einen 3D-Pfeil entsprechend den Vorgaben aus Abschnitt 3.3.7 dar. Dieser zeigt von der Position des aktuellen Bauteils zu dessen Zielposition. Dafür gibt es zwei Eingänge, die diese beiden Positionen als Transformationsmatrizen empfangen (`src_matrix` und `dest_matrix`). Ein weiterer Eingang `active` steuert die Sichtbarkeit des Pfeils und die damit verbundenen Berechnungen. Wird der Pfeil nicht dargestellt, erfolgt keine Berechnung (siehe Abb. 4.16 und Tab. 4.14).

Verwendung im Programm: Für jeden Bauteil des Testmöbelstückes gibt es eine eigene Pfeilkomponente. Es wird jeweils der Pfeil jenes Bauteils angezeigt, der gerade montiert werden muss. Entsprechend dem festgelegten Konzept (3.2) wird entweder der Pfeil oder eine Animation dargestellt. Der Anwender steuert die Darstellung über den Button „Animation Ein/Aus“.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
src_matrix	Double Vector	Transformationsmatrix des (aktuellen) Bauteils.
dest_matrix	Double Vector	Transformationsmatrix der Zielposition des (aktuellen) Bauteils.
active	Boolean	Steuert die Anzeige des Pfeilobjekts.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-

Tabelle 4.14: Parameter der ArrowComponent.

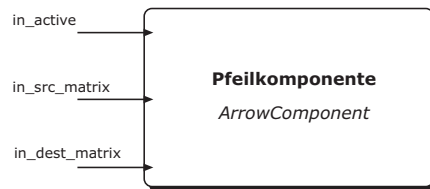


Abbildung 4.16: Die Komponente zur Anzeige eines Pfeilobjekts.

4.4.7 QAnimationComponent: Bauteilanimation

Die Animation des 3D-Modells des aktuellen Bauteils wird auf der Grundlage der Quaternionen realisiert (siehe Abschnitt 3.3.8). Die Position des aktuellen Bauteils (**src_matrix**) und dessen Zielposition (**dest_matrix**) werden für die Interpolation herangezogen. Als Basis für die Umsetzung wird die Klasse `Quat`³ herangezogen. Folgende Methoden werden verwendet:

- `Quaternion()`: Standardkonstruktor.
- `void convertMatrix (Matrix m)`: Umrechnen einer Transformationsmatrix (`Matrix`-Objekt) in ein Quaternion (`Quaternion`-Objekt).
- `Matrix getMatrix()`: Umrechnen eines Quaterions in eine Transformationsmatrix.
- `void Slerp(Quaternion q1, Quaternion q2, double t)`: Berechnen der Interpolation zweier Quaternionen mit dem Slerp-Algorithmus. Der Zeitpunkt $t = [0, 1]$ wird mit dem dritten Parameter angegeben.

³www.cs.wisc.edu/graphics/Courses/cs-838-1999/Students/ballard/proj1/

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
src_matrix	Double Vector	Transformationsmatrix des (aktuellen) Bauteils.
dest_matrix	Double Vector	Transformationsmatrix der Zielposition des (aktuellen) Bauteils.
active	Boolean	Steuert, ob die Berechnung der interpolierten Transformation durchgeführt wird oder nicht.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
out_matrix	Double Vector	Transformationsmatrix der interpolierten Transformation.

Tabelle 4.15: Parameter der QAnimationComponent.

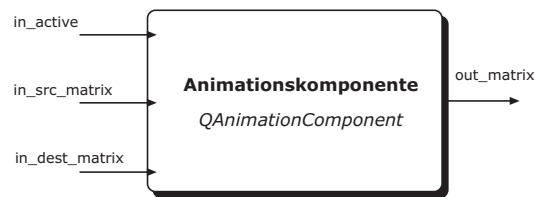


Abbildung 4.17: Die Komponente zur Berechnung der interpolierten Transformation für die Bauteil-Animation.

Sobald der Eingang **active** den Wert **true** empfängt, wird die Berechnung der interpolierten Transformation durchgeführt. Das Ergebnis wird an den Ausgang **out_matrix** gelegt. Es werden insgesamt 40 Teilschritte berechnet (innerhalb des Zeitintervalls $t = [0, 1]$), danach wird die Animation wiederholt.

Verwendung im Programm: Entsprechend der Anzahl der Bauteile gibt es vier **QAnimationComponents**. Eine Animation wird nur dann angezeigt, wenn der Button „Animation Ein/Aus“ aktiviert ist. Dann wird die Animation solange wiederholt, bis entweder der Button wieder deaktiviert ist, oder der nächste Montageschritt erfolgt.

4.4.8 CheckAssemblyComponent: Ermitteln, ob ein Bauteil korrekt montiert wurde

Diese Komponente ermittelt, ob ein Bauteil an der richtigen Stelle und korrekt angeordnet montiert wird. Dafür wird die Position des aktuellen

Bauteils mit dessen Zielposition verglichen. Diese Positionen werden als Transformationsmatrizen über zwei Eingänge empfangen. Wenn der Eingang `active` auf `true` gesetzt wird, vergleicht die Komponente die beiden Transformationen entsprechend den Vorgaben aus Abschnitt 3.3.9.

Die Translationsvektoren beider Matrizen werden direkt verglichen:

```

/* Transformationsmatrizen Bauteil, Zielposition */
Matrix in_matrix1, in_matrix2;
/* Translationsvektor Bauteil */
double x1 = in_matrix1[12];
double y1 = in_matrix1[13];
double z1 = in_matrix1[14];
/* Translationsvektor Zielposition */
double x2 = in_matrix2[12];
double y2 = in_matrix2[13];
double z2 = in_matrix2[14];
/* Schwellwert */
double thresh = 50.0;
/* Vergleich */
bool match_trans = ((x1 > (x2 - thresh) && x1 < (x2 + thresh)) &&
                    (y1 > (y2 - thresh) && y1 < (y2 + thresh)) &&
                    (z1 > (z2 - thresh) && z1 < (z2 + thresh)));

```

Für den Vergleich der Rotation werden die Rotationsmatrizen in Quaternionen umgerechnet und die Drehwinkel verglichen. Dafür wird die Methode `compare(Quaternion q, double thresh)` der Quaternion-Klasse verwendet.

```

/* Transformationsmatrizen Bauteil, Zielposition */
Matrix in_matrix1, in_matrix2;

Quaternion q1, q2;

/* Umrechnung in Quaternionen */
q1.convertMatrix (in_matrix1);
q2.convertMatrix (in_matrix2);

/* Schwellwert */
double threshR = 30 * PI / 180;

/* Vergleich */
bool match_rot = q1.compare(q2, threshR);

```

Die beiden ermittelten Boole'schen Werte werden schließlich verknüpft und damit der Vergleich beendet.

```
bool match = match_trans && match_rot;
```

Wird eine Übereinstimmung festgestellt, dann liefert der Ausgang `match` den Wert `true` (siehe Tabelle 4.16 und Abbildung 4.18).

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
matrix1	Double Vector	Transformationsmatrix des (aktuellen) Bauteils.
matrix2	Double Vector	Transformationsmatrix der Zielposition des (aktuellen) Bauteils.
active	Boolean	Steuert, ob die Überprüfung durchgeführt wird oder nicht.
<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
match	Boolean	Liefert true , wenn ein Bauteil als korrekt montiert eingestuft wird.

Tabelle 4.16: Parameter der `CheckAssemblyComponent`.



Abbildung 4.18: Komponente zum Ermitteln, ob ein Bauteil korrekt eingebaut wurde.

Verwendung im Programm: Die Montage jedes Bauteils wird mit einer `CheckAssemblyComponent` überprüft. Es gibt daher 4 solche Komponenten, deren Eingänge mit der Logikkomponente (aktuelle Position) und den `MatrixCalculateComponents` (Berechnung der Zielpositionen) verbunden sind. Die Ausgänge werden wieder in die Logikkomponente zurückgeführt, von der die entsprechenden Rückmeldungs-Anzeigen gesteuert werden.

4.4.9 ButtonComponent: Steuerelemente

Verwendung im Programm: Es gibt im Application Modus nur drei Möglichkeiten der Interaktion (siehe Abb. 4.19):

- Den nächsten Anleitungsschritt anzeigen,
- den vorhergehenden Anleitungsschritt anzeigen,
- zwischen Pfeildarstellung und Animationsdarstellung wechseln (Animation Ein/Aus).

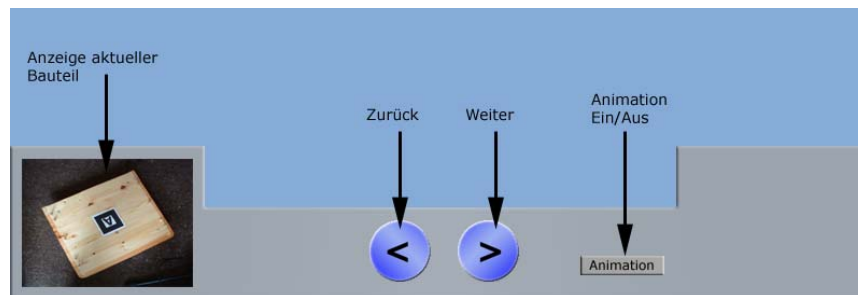


Abbildung 4.19: Steuerelemente des Application Modus.

Jede dieser Aktionsmöglichkeiten wird mit einer eigenen `ButtonComponent` (siehe Abschnitt 4.3.4) realisiert. Die Ausgänge der drei Komponenten sind mit der Logikkomponente verknüpft.

4.4.10 APPMLogicComponent: Zentrale Logikkomponente

Diese Komponente ist für die Steuerung der Sichtbarkeiten der grafischen Elemente zuständig. Außerdem reagiert sie auf die Aktionen des Anwenders. Beim Initialisieren dieser Komponente wird mit Hilfe der `XMLLoader`-Klasse eine gespeicherte Anleitung geladen und in einer `InstructionList` abgelegt. Diese Liste wird schrittweise durchgegangen. Der im Funktionsumfang festgelegte Ablauf wird in verschiedenen Anzeigemodi umgesetzt (siehe Abb. 4.20). Diese sind:

1. `NORMAL_PART_SEARCH_DESCRIPTION_MODE`: In diesem Modus werden ein Beschreibungstext sowie ein Bild zum aktuellen Bauteil angezeigt. Der Anwender wird aufgefordert, den Bauteil zu suchen und bereit zu halten.
2. `NORMAL_PART_SEARCH_VIDEO_MODE`: In diesem Modus sucht der Anwender mit der Kamera nach obigem Teil. Alle erkannten Bauteile werden mit grauen Rahmen überblendet, nur der Rahmen des aktuellen Bauteils ist grün.
3. `SMALL_PART_DESCRIPTION_MODE`: Hier wird ein Beschreibungstext und ein Bild des Kleinteils angezeigt, das an dem aktuellen Bauteil anzubringen ist.
4. `SMALL_PART_VIDEO_MODE`: Dieser Modus zeigt dem Anwender die Zielpositionen der Kleinteile auf dem aktuellen Bauteil an.
5. `NORMAL_PART_PLACE_DESCRIPTION_MODE`: Ein Text erklärt, dass der aktuelle Bauteil an der angezeigten Zielposition eingebaut werden muss.

6. **NORMAL_PART_PLACE_VIDEO_MODE**: In diesem Modus wird die Zielposition des Bauteils angezeigt. Es wird einerseits das 3D-Modell des aktuellen Bauteils dort eingeblendet (in der Farbe des realen Bauteils). Außerdem kann der Anwender zwischen der Pfeildarstellung und dem Animationsmodus wechseln. Sobald der Bauteil korrekt montiert wurde (siehe Abschnitt 4.4.8), wird das 3D-Modell an der Zielposition in grüner Farbe dargestellt.

Die Parameter dieser Komponente sind in Abbildung 4.21 und Tabelle 4.17 bzw. 4.18 zusammengefasst.

<i>Eigenschaft</i>	<i>Datentyp</i>	<i>Beschreibung</i>
-	-	-
<i>Eingang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
id1-4	String	Verbindung zu den MarkerDetectionComponents der vier Bauteile; empfangen die IDs der Bauteile.
transformation1-4	Double Vector	Verbindung zu den MarkerDetectionComponents der vier Bauteile; empfangen die Positionen der Bauteile (Transformationsmatrizen).
detected1-4	Boolean	Mit den MarkerDetectionComponents der vier Bauteile verbunden; empfangen den Trackingstatus.
match1-4	Boolean	Verbindung zu den CheckAssemblyComponents ; empfangen, ob der aktuelle Bauteil korrekt montiert wurde.
next_button_trigger	Boolean	Verbindung zum Button, mit dem der Anwender den nächsten Montageschritt anzeigen lässt.
previous_button_trigger	Boolean	Verbindung zum Button, mit dem der Anwender den vorigen Montageschritt anzeigen lässt.
animation_button_trigger	Boolean	Verbindung zum Button, mit dem der Anwender zwischen Pfeildarstellung und Animationsanzeige wechselt.

Tabelle 4.17: Eingänge der Logikkomponente **APPMLogicComponent** des Application Modus.

<i>Ausgang</i>	<i>Datentyp</i>	<i>Beschreibung</i>
active1-4	Boolean	Steuert, ob die Berechnungskomponenten der Zielposition/Pfeildarstellung/Animationspfade aktiv sind oder nicht.
displayMode SRC1-4	Integer	Anzeigemodus der einzelnen Bauteil-Markierungsrahmen (Farben grün oder grau): Verbunden mit den entsprechenden GeometryComponents .
displayMode DEST1-4	Boolean	Anzeigemodus des jeweiligen Bauteil-3D-Modells an der Zielposition.
relative_Matrix1-4	Double Vector	Liefert die Relativposition des aktuellen Bauteils zu einem anderen Bauteil; Verbunden mit den MatrixCalculateComponents .
base_trans	Double Vector	Liefert die absolute Position des aktuellen Bauteils; Verbunden mit den MatrixCalculateComponents der Bauteile.
smallPart_trans1-4	Double Vector	Liefert Relativpositionen der Kleinteile zum aktuellen Bauteil; Verbunden mit den MatrixCalculateComponents der Kleinteile.
smallPart DisplayMode1-4	Integer	Anzeigemodus der Kleinteile, die auf einem Bauteil montiert sein können; Verbunden mit den Kleinteil-GeometryComponents .
DESTvisible, ..., previous_button visible	Boolean	Sichtbarkeiten von: Bauteilen an der Zielposition, Pfeilen, Bauteilgrafiken links unten, große Bauteilgrafiken bei den Erklärungen, Erklärungstexte zum Suchen nach einem Bauteil, Erklärungstexte zum Montieren eines Bauteils, Kleinteile an der Zielposition, Kleinteilgrafiken links unten und dazugehöriges Werkzeug rechts unten, große Kleinteilgrafiken bei den Erklärungen, Buttons.

Tabelle 4.18: Ausgänge der Logikkomponente **APPMLogicComponent** des Application Modus.

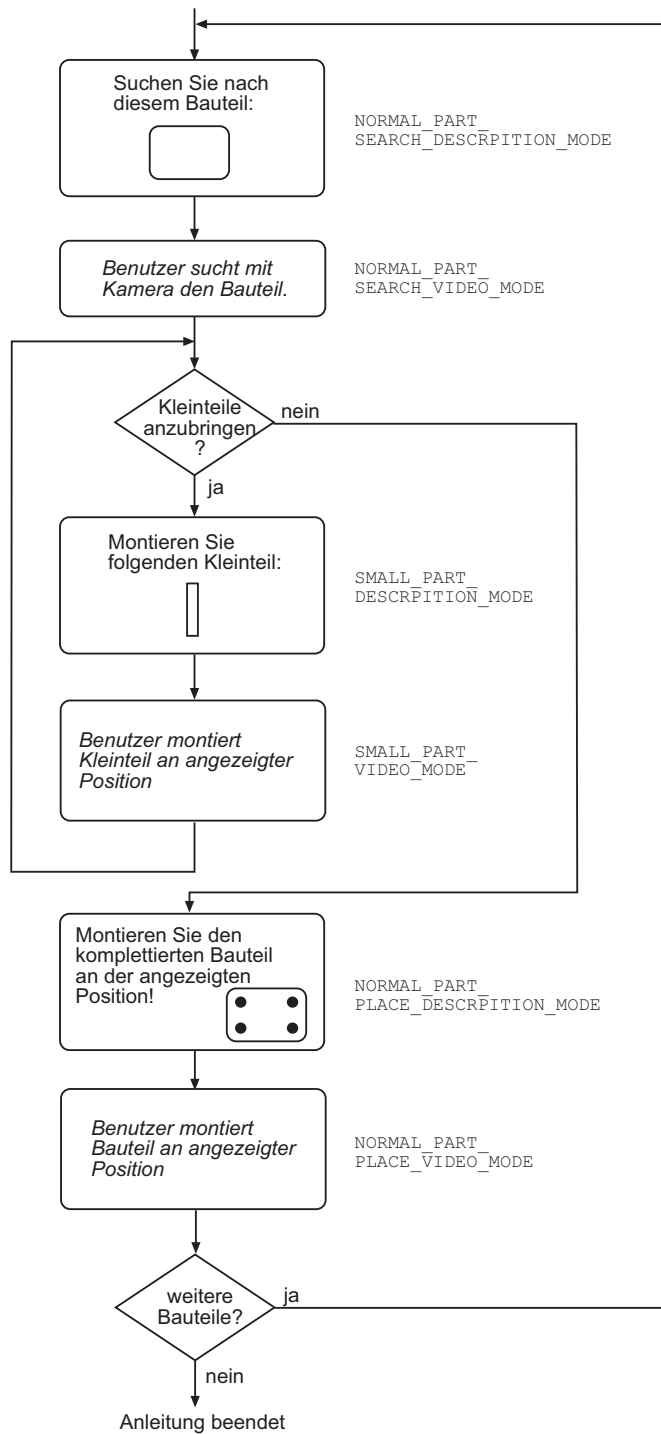


Abbildung 4.20: Die einzelnen Anzeigemodi im Ablaufdiagramm des *Application* Modus.

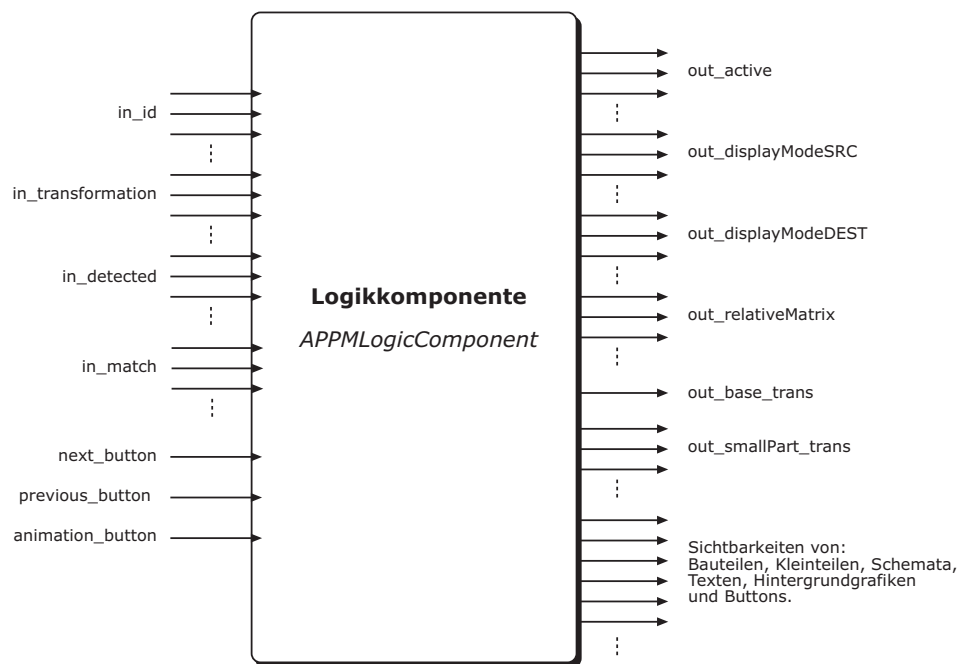


Abbildung 4.21: Die Logikkomponente des Application Modus und deren Ein- und Ausgänge.

Kapitel 5

Arbeiten mit dem FAI^{MR}

Das Programm wurde auf einem PC mit AMD Athlon 1000 MHz Prozessor, 256 MB RAM und nVidia GeForce 2 Grafikkarte entwickelt und getestet. Diese Konfiguration ermöglicht eine einigermaßen brauchbare Geschwindigkeit, wobei Frameraten höher als 15 fps bei einer Auflösung von 640×480 nicht zu erreichen sind.

Um den FAI^{MR} auf einem PC zu testen, müssen im Vorfeld einige Einstellungen im System vorgenommen werden. Eine genaue Anleitung zur Installation des AMIRE Frameworks ist auf der CD-Rom enthalten (der Inhalt der CD-Rom ist in den Dateien *Inhalt.txt* bzw. *content.txt* aufgelistet).

In den folgenden Absätzen wird die Verwendung des Prototyps erläutert. Es werden jene Schritte erklärt, die für das Entwerfen und Nachvollziehen einer Anleitung erforderlich sind.

5.1 Speichern einer Anleitung im Authoring Modus

Das Entwerfen einer Bauanleitung geschieht in fünf verschiedenen Schritten:

Nach dem aktuellen Bauteil suchen. Es hat sich als sinnvoll erwiesen, möglichst alle vorhandenen Bauteile auf einer größeren Fläche (Fußboden, Tisch) nebeneinander aufzulegen. So können alle Teile mit der Kamera einzeln erfasst werden. Erkennt das Programm einen Teil, wird er mit einem grauen Rahmen markiert. Durch Betätigen des Buttons „Bauteil auswählen“ werden alle erkannten Bauteile der Reihe nach selektiert und mit einem grünen Rahmen markiert.

Kleinteil auswählen. Um einen Kleinteil auszuwählen, der auf dem zuvor ausgewählten Bauteil montiert wird, muss der Kleinteilmarker auf dem Bauteil platziert und vom System erkannt werden. Am rechten unteren Bildschirmrand wird das Bild eines Kleinteils und ein Pfeil eingeblendet. Mit diesem Pfeil wird die Liste aller verfügbaren Kleinteile durchlaufen (siehe Abb. 5.1).



Abbildung 5.1: Auswählen eines Kleinteils.

Kleinteil montieren. Mit dem Button „Kleinteil montieren“ wird die Position des Kleinteils relativ zum aktuellen Bauteil gespeichert. Der Button wird nur dann angezeigt, wenn sowohl der aktuelle Bauteil als auch der Kleinteilmarker vom System erkannt werden (siehe Abb.5.2).

Bauteil montieren. Handelt es sich bei dem selektierten Bauteil um den ersten Teil des gesamten Möbelstücks, wird dieser mit Betätigen des Buttons „Bauteil montieren“ gespeichert. Bei allen nachfolgenden Bauteilen ist es wichtig, dass mindestens ein anderer bereits montierter Bauteil vom System erkannt wird. Nur dann kann auch die entsprechende Relativposition gespeichert werden (siehe Abb. 5.3).

Anleitung beenden. Mit Betätigen des Buttons „Anleitung beenden“ wird die Montage beendet und die Liste in der Datei „Instruction.xml“ gespeichert.

5.2 Nachvollziehen einer Anleitung im Application Modus

Die gespeicherte Anleitung wird beim Starten des Application Modus automatisch geladen. Mit den beiden Buttons „nächster Schritt“ und „voriger Schritt“ wird eine Anleitung schrittweise durchgegangen, wobei jeder Einzelschritt in die bereits beschriebenen Anzeigemodi unterteilt ist (siehe Abschnitt 3.2). Sobald ein Bauteil montiert werden muss, wird auch der Button „Animation Ein/Aus“ eingeblendet, mit dem man zwischen der Pfeildarstellung und dem Animationsmodus wechselt.

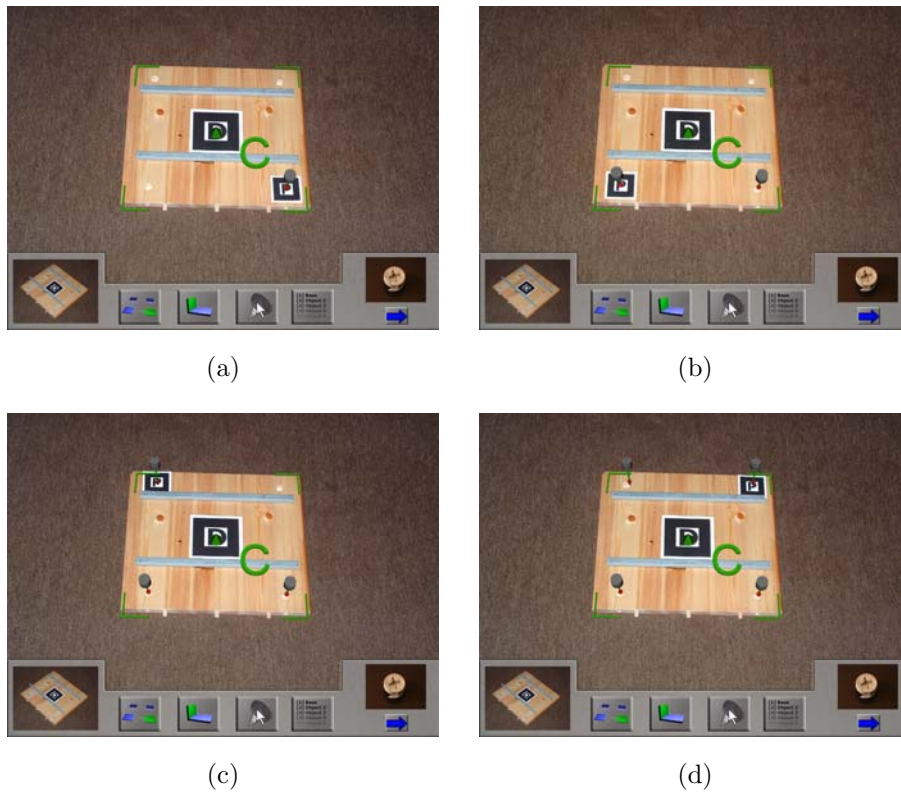


Abbildung 5.2: Montieren der Kleinteile.

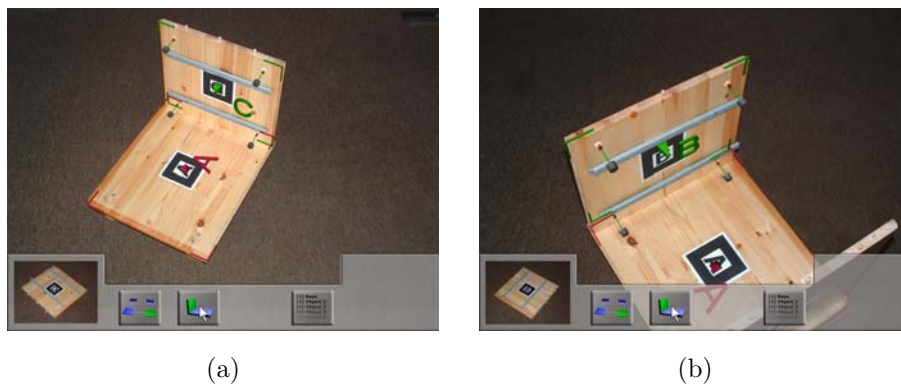


Abbildung 5.3: Bauteile montieren: (a) Bauteil A wurde bereits gespeichert. Nun wird Bauteil C (an dem vier Kleinteile angebracht sind) montiert und dessen Relativposition zu A gespeichert. (b) Auch an Bauteil B sind vier Kleinteile montiert. Schließlich wird auch dieser Bauteil in die Liste eingetragen.

Kapitel 6

Vergleich mit anderen Anwendungen

In diesem Kapitel werden einige Anwendungen betrachtet, die ebenfalls in den Themenkreis der computerunterstützten Anleitungen passen. Drei Beispiele werden herangezogen und mit dem Ansatz dieser Arbeit verglichen. Auf eine tiefere Beschreibung wird jedoch verzichtet und jeweils auf entsprechende Quellen verwiesen.

6.1 Animated Vision – Animierte Anleitungen

Die Firma Animated Vision¹ bietet die Erstellung animierter Anleitungen an. Die Anwendungen sind aber nicht auf Mixed Reality Basis, sondern als Flash Filme² implementiert. Das Spektrum reicht von Möbelbauanleitungen bis zu Anleitungen für Computer-Peripheriegeräte. Die Teile des jeweiligen Produkts werden als 3D-Grafiken visualisiert und animiert, wobei die meisten Anleitungen auch Schritt für Schritt abgearbeitet werden. Der Anwender navigiert vorwärts oder rückwärts und kann einen Teilschritt beliebig oft wiederholen. Durch das Anzeigen von Animationen zu jedem einzelnen Teilschritt (z. B. das Montieren einer Schraube oder eines ganzen Bauteils) wird die Vorgehensweise verdeutlicht. Außerdem werden Erklärungstexte eingeblendet, teilweise sogar mit akustischer Wiedergabe (siehe Abb. 6.1).

Trotz der animierten Teilschritte besteht keine direkte Verbindung zum realen Objekt, wie das in der Mixed Reality Anwendung über das Videobild der Fall ist. Außerdem ist der Prozess des Erstellens einer Anleitung weitaus zeitaufwändiger, da die Animationen einzeln produziert werden müssen. Auf der anderen Seite gibt es dadurch keine Beschränkung auf bestimmte Objekte oder einen bestimmten Satz von Bestandteilen. Darüber hinaus ist

¹www.animatedvision.com – Auf der Homepage sind einige Beispiele von Anleitungen zu finden.

²www.macromedia.com

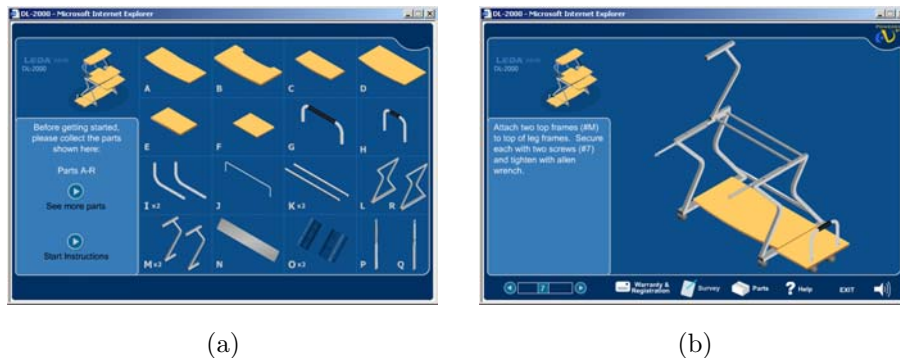


Abbildung 6.1: Beispiel einer Möbelbauanleitung von Animated Visions: (a) Liste der Bauteile und Kleinteile, (b) Erklärung eines Montageschritts anhand einer Animation und (vorgelesenem) Erklärungstext [Quelle: www.ledadesk.com/anim_learn_more.html bzw. www.animatedvision.com].

die Qualität der vorgefertigten Animationen besser, wodurch die tatsächliche Vorgehensweise bei einigen Montageschritten unter Umständen klarer ist als bei der zur Laufzeit berechneten Animation.

6.2 Proaktive Unterstützung

Ein Forschungsprojekt der ETH Zürich beschäftigt sich mit der Entwicklung eines sogenannten „proaktiven“ Systems [ANTIFAKOS et al. 2002]. „Proaktiv“ bedeutet in diesem Fall, dass die Anwendung bei jedem Montageschritt eines Möbelstücks den Zustand der montierten Bauteile überwacht und alle möglichen Folgeschritte kennt. Mit diesen Informationen werden erfahrene Benutzer, die eigene Wege innerhalb des Montagevorgangs wählen, rechtzeitig informiert, wenn eine Sackgasse erreicht wird oder der Aufbau instabil ist. Das Erkennen des Zustandes der Bauteile wird mittels Sensoren realisiert, die sowohl mit den Teilen als auch mit den Werkzeugen verbunden sind (siehe Abb. 6.2).

Der Vorteil dieser Anwendung liegt klar in der situationsabhängigen Unterstützung. Durch die verschiedenen Sensoren kann das System den Zustand der Bauteile sehr genau berechnen. Es werden Teilschritte (z. B. Montieren von Kleinteilen, Umdrehen des Möbelstücks) und komplette Schritte festgestellt.

Der größte Nachteil dieses Systems liegt in der Tatsache, dass die Sensoren weitaus kostenintensiver sind, als das Tracking mit Hilfer der Marker. Außerdem müssten entsprechend die benötigten Werkzeuge modifiziert werden. Im Weiteren ist das Erstellen einer Anleitung, in der alle möglichen Montageschritte erfasst werden, relativ aufwändig.

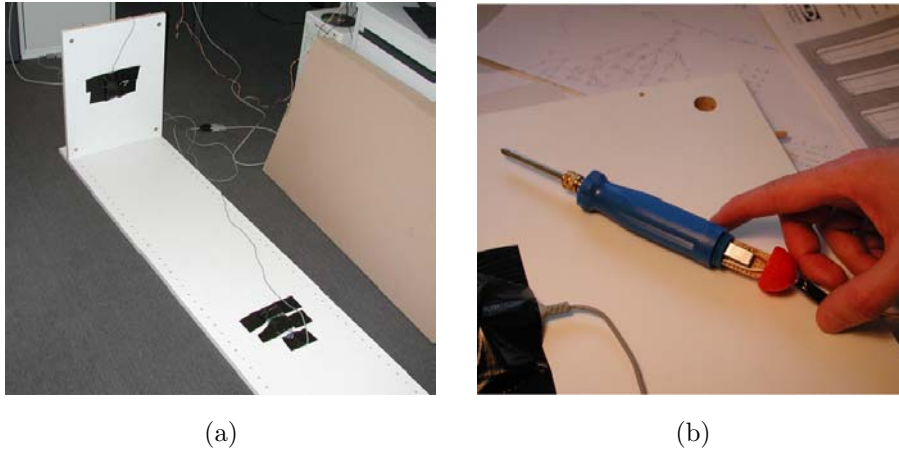


Abbildung 6.2: (a) Die Bauteile eines Möbelstücks sind mit Sensoren ausgestattet. (b) Der Schraubenzieher ist mit einem eigenen Sensor bestückt. Aus [ANTIFAKOS et al. 2002].

In [ANTIFAKOS et al. 2002] wird vorgeschlagen, bereits in der Entwurfsphase eines Möbelstücks mit dem Speichern der einzelnen Schrittfolgen zu beginnen. Außerdem könnte ein lernfähiges System die Vorgehensweise verschiedener Anwender erfassen.

6.3 AEKI

„AEKI“ heißt ein Projekt der IMS Gruppe (Interactive Media Systems) an der TU Wien [PINTARIC 2002]. Der Prototyp stellt die Montageanleitung eines Testmöbelstücks Schritt für Schritt dar. Zu diesem Zweck wird ARToolkit als Trackingsystem verwendet und die einzelnen Bauteile als 3D-Modelle über einem eigenen Marker eingeblendet (siehe Abb. 6.3). Ähnlich der Anwendung der vorliegenden Arbeit sind die einzelnen Bauteile mit Markern versehen, um das Tracking zu realisieren. Der AEKI-Prototyp beschränkt sich auf die Anzeige der Positionen der Bauteile, ohne auf zusätzliche Montageschritte (z. B. Montieren von Kleinteilen, benötigte Werkzeuge, Ausrichtung der Bauteile) einzugehen. Es geht aus [PINTARIC 2002] nicht hervor, wie das Erstellen (Authoring) einer Montageanleitung gelöst wurde.

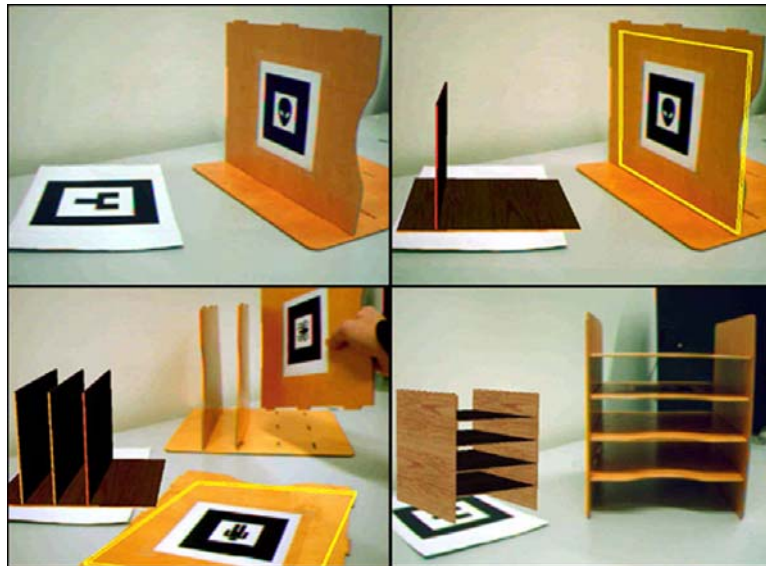


Abbildung 6.3: Bilder des AEKI-Prototyps: 3D-Modelle der Bauteile zeigen den aktuellen Montageschritt an. Auch bei dieser Anwendung werden Marker mit Hilfe von ARToolkit getrackt. Aus [PINTARIC 2002].

Kapitel 7

Schlußbemerkungen

Der entwickelte Prototyp stellt, wie bereits erwähnt, eine Konzeptstudie dar. Als solche erfüllt er die gestellten Anforderungen, eine Montageanleitung gänzlich auf Basis von Mixed Reality abzubilden. Entscheidend ist dabei die Tatsache, dass sowohl das Erstellen einer Anleitung als auch das Nachvollziehen einer solchen zur Gänze in der MR-Umgebung geschieht. Um festzustellen, ob der gewählte Ansatz einen tatsächlichen Vorteil gegenüber anderen Modellen bringt, müssten umfangreichere Tests mit verschiedenen Benutzergruppen durchgeführt werden. Für solche Tests war im Rahmen dieser Arbeit die Zeit zu knapp.

Es wurde mit der implementierten Anwendung eine Basis geschaffen, die als Ausgangspunkt diverser Weiterentwicklungen dient. Einige Ansätze dafür werden in der Folge kurz skizziert.

7.1 Mögliche Weiterentwicklungen

7.1.1 Anleitungen für beliebige Möbelstücke

Die wichtigste Erweiterung wäre, beliebige Möbelstücke mit dem Programm zu verwalten. Im Idealfall wären für jedes Möbelstück die Bauteile und Kleinteile mit allen nötigen Daten (Beschreibungstexte, 3D-Modelle, Grafiken etc.) in einer Datenbank abgelegt. Diese Daten könnte der Autor beim Starten des Authoring Modus laden und dann die Anleitung eines Möbelstücks erstellen. Um diese Erweiterung zu realisieren, müsste der Aufbau beider Modi dynamisch generiert werden, d. h. die Komponenten werden entsprechend initialisiert, deren Eigenschaften gesetzt und die Verknüpfungen automatisch hergestellt.

7.1.2 Komplexere Bauteile erfassen

Die Erweiterung auf das Verwalten beliebiger Möbelstücke bedeutet auch, dass komplexere Bauteile (z. B. Teile eines Sessels) vorkommen können. Um

solche Bauteile ebenfalls mit der Anwendung zu erfassen, müsste die Komponente `MarkerDetectionComponent` so erweitert werden, dass *beliebig viele* Marker (anstatt nur zwei) auf einem Bauteil angebracht werden können. Dazu müsste ein Weg gefunden werden, die Positionen der einzelnen Marker auf einem Bauteil festzulegen, da diese dann nicht mehr wie im *FAIMR* genau einander gegenüberliegend angeordnet sind. Eine Lösungsmöglichkeit wäre, einen der Marker als Referenzpunkt für alle anderen Marker festzulegen. Die Positionen (und Ausrichtungen) der anderen Marker müssten dann relativ zu diesem Referenzmarker definiert werden.

7.1.3 Auf verschiedene Benutzergruppen eingehen

Ähnlich dem Ansatz des in Abschnitt 6.2 beschriebenen Projekts könnte die Anleitung auf die jeweiligen Arten von Anwendern reagieren. Für einen unerfahrenen Anwender ist die bereits implementierte Vorgehensweise der Schritt-Für-Schritt Navigation optimal. Die weiterführende Überlegung ist dahingehend, einem fortgeschrittenen Anwender nur dann gezielt Informationen anzuzeigen, wenn er diese anfordert.

7.1.4 Mobilität

Es ist das Ziel der entwickelten Anwendung, das Hauptaugenmerk auf dem realen Möbelstück zu belassen. Da jedoch ein Computer benötigt wird, um die Anleitung überhaupt zu bedienen, besteht auch hier die Gefahr der Ablenkung vom eigentlichen Ziel, nämlich dem Zusammenbau eines Möbelstücks. Daher ist die Überlegung, das Programm auf einem mobilen Gerät einzusetzen, z. B. auf einem PALM oder einem Tablet-PC. Trotzdem muss der Anwender mit Hilfe des Mauszeigers die Abläufe steuern. Die Steuerung über Spracheingabe wäre die konsequente Weiterentwicklung, so dass der Anwender die Konzentration beim Montageprozess behalten kann. Als Grundlage dafür käme die Microsoft Speech API¹ in Frage. Im Application Modus wären die vier gesprochenen Befehle „weiter“, „zurück“ und „Animation Ein“ bzw. „Animation Aus“ ausreichend für die Steuerung.

7.1.5 Resümee

Die Betrachtungen dieser Arbeit zeigen die vielfältigen Möglichkeiten ausgehend vom erarbeiteten Ansatz auf. Mixed Reality Technologien können durch gezielten Einsatz in verschiedensten Gebieten eine Hilfe bei Fertigungs- und Reparaturprozessen sein. Durch die fortwährende Verbesserung der Trackingsysteme und Benutzerschnittstellen (Sprachsteuerung, intuitivere Interaktionsmittel) wird künftig der Einsatz von Mixed Reality in vielen Bereichen den Status der Science Fiction verloren haben.

¹www.microsoft.com/speech/download/sdk51/

Anhang A

Das XML Schema

```
<?xml version="1.0" encoding="UTF-8" ?> <!--Generated by XML
Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="instruction">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name" minOccurs="0" maxOccurs="1" />
        <xsd:element ref="list" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="name" type="xsd:string" />
  <xsd:element name="list">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="part" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="part">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="id" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="name" minOccurs="0" maxOccurs="1" />
        <xsd:element ref="relative_position" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="id" type="xsd:string" />
  <xsd:element name="smallParts">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="smallPart" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="smallPart">
    <xsd:complexType>
```

```
<xsd:sequence>
  <xsd:element ref="id" minOccurs="1" maxOccurs="1" />
  <xsd:element ref="name" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="DisplayMode" minOccurs="1" maxOccurs="1" />
  <xsd:element ref="relative_position" minOccurs="1" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="DisplayMode" type="xsd:int" />
<xsd:element name="relative_position">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="m" minOccurs="16" maxOccurs="16" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="m" type="xsd:double" />
</xsd:schema>
```

Anhang B

Komponenten der beiden Programm-Modi

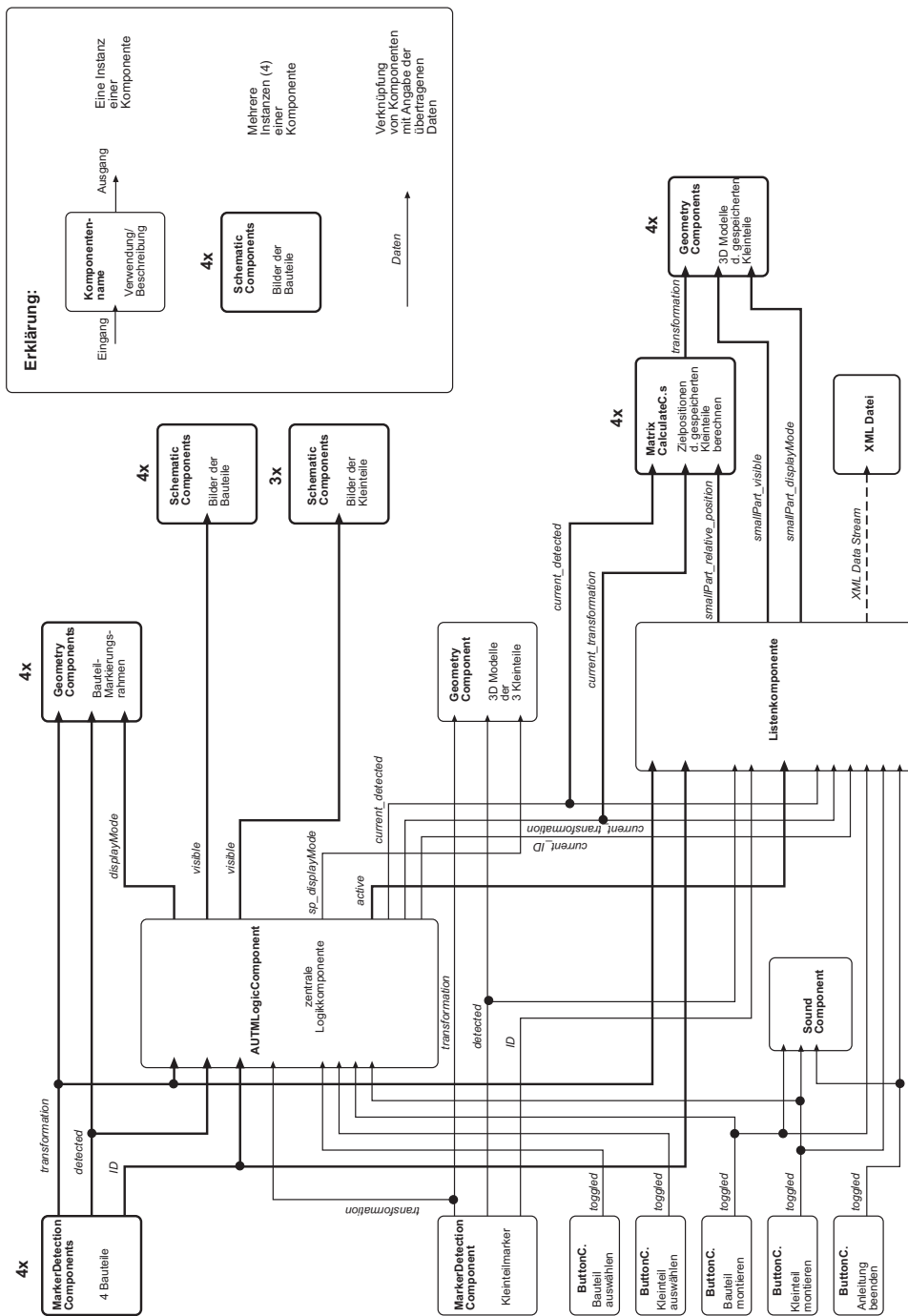


Abbildung B.1: Komponenten des Authoring Modus.

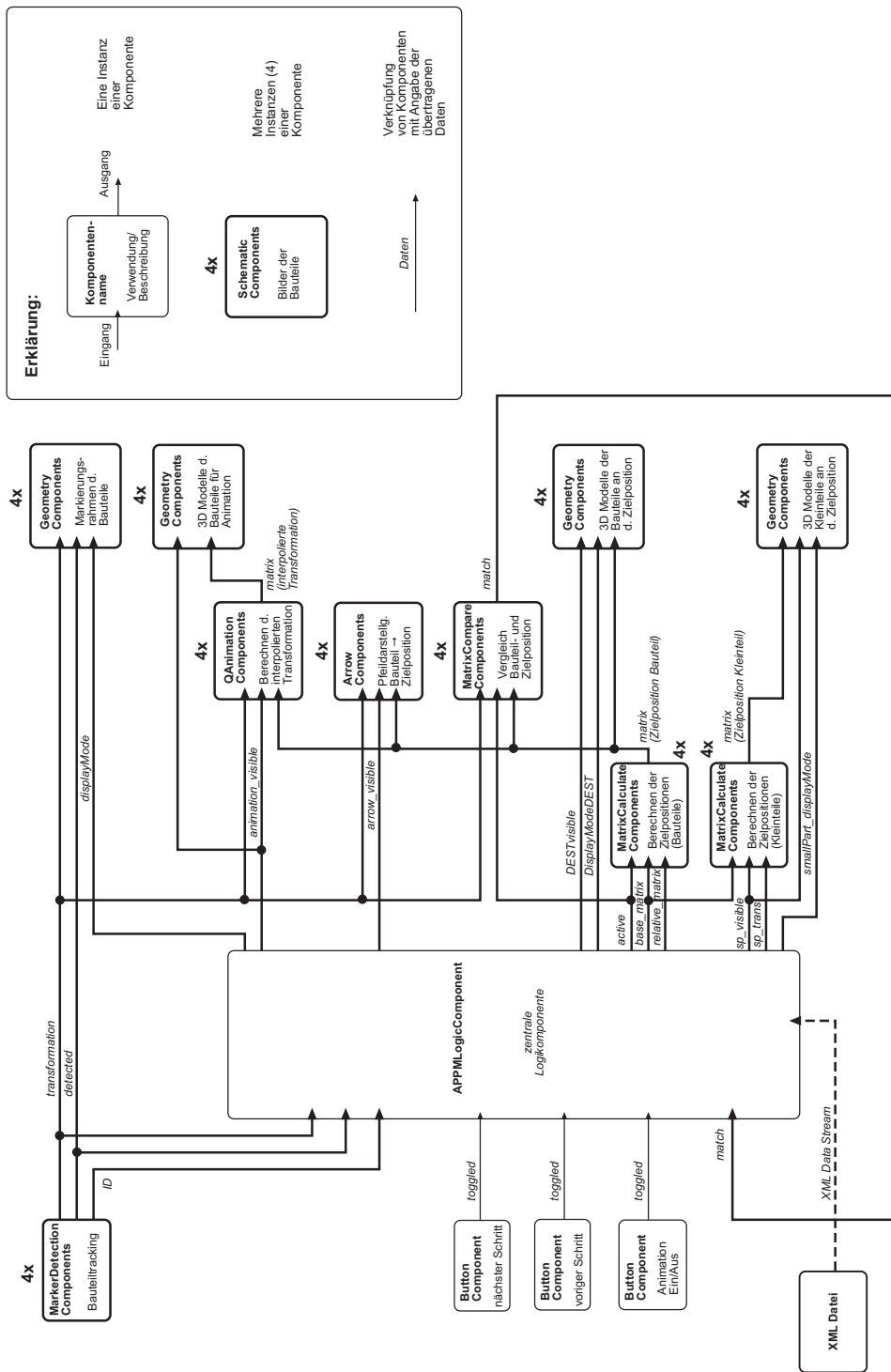


Abbildung B.2: Komponenten des *Application* Modus: Steuerung der Berechnungslogik.

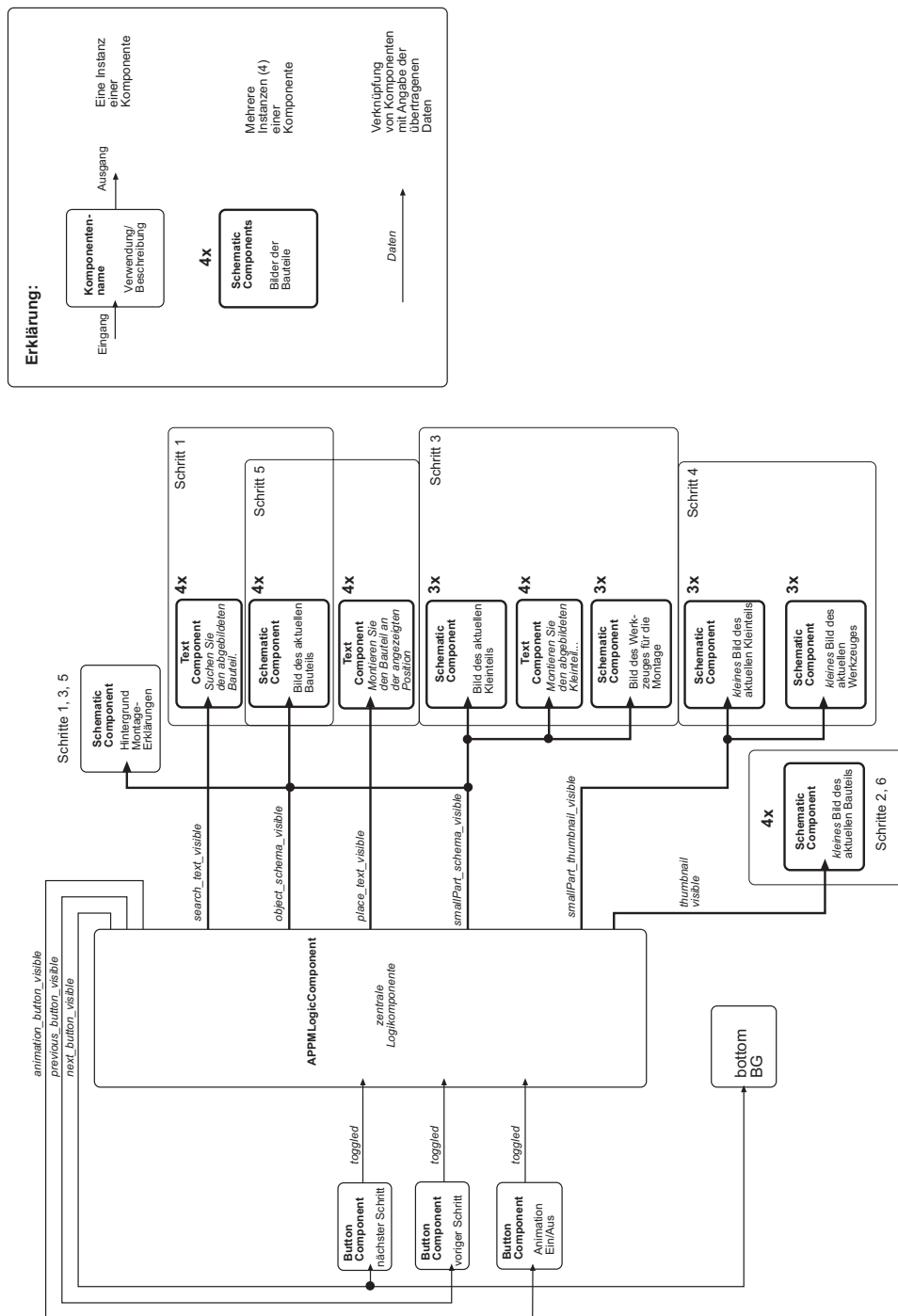


Abbildung B.3: Komponenten des *Application* Modus: Anzeigensteuerung. Die Angaben der Schritte 1 bis 6 beziehen sich auf den im Funktionsumfang definierten Ablauf der Applikation (s. Abb. 3.3).

Literaturverzeichnis

- [ANTIFAKOS et al. 2002] ANTIFAKOS, S., F. MICHAHELLES und B. SCHIELE (2002). *Proactive Instructions for Furniture Assembly*. In: *Proceedings of the The Fourth International Conference on Ubiquitous Computing (UbiComp 2002)*, Göteborg, Sweden. Kopie auf CD-Rom (ubicomp02e.pdf).
- [BILLINGHURST und KATO 1999] BILLINGHURST, M. und H. KATO (1999). *Collaborative Mixed Reality*. In: *Proceedings of International Symposium on Mixed Reality (ISMR '99). Mixed Reality – Merging Real and Virtual Worlds*, S. 261–284, Seattle, WA. Kopie auf CD-Rom (collaborative_MR.pdf).
- [BILLINGHURST et al. 1999] BILLINGHURST, M., H. KATO, S. WEGHORST und T. A. FURNESS (1999). *A Mixed Reality 3D Conferencing Application*. Technischer Bericht R-99-1, Human Interface Technology Laboratory, University of Washington, Seattle, WA. Kopie auf CD-Rom (r-99-1.pdf).
- [DAM et al. 1998] DAM, E. B., M. KOCH und M. LILLHOLM (1998). *Quaternions, Interpolation and Animation*. Technischer Bericht DK-2100 Kbh 0, Department of Computer Science, University of Copenhagen, Universitetsparken 1. Kopie auf CD-Rom (quaternion.pdf).
- [DÖRNER et al. 2002] DÖRNER, R., C. GEIGER, M. HALLER und V. PAELKE (2002). *Authoring Mixed Reality – A Component and Framework-Based Approach*. URL, <http://webster.fh-hagenberg.at/amire/research/amire@IWEC2002.pdf>. Kopie auf CD-Rom (amire@IWEC2002.pdf).
- [HALLER et al. 2003] HALLER, M., J. ZAUNER, W. HARTMANN und T. LUCKENEDER (2003). *A generic framework for a training application based on Mixed Reality*. Technical Report, Upper Austria University of Applied Sciences (MTD). Kopie auf CD-Rom (amireTR2003-1.pdf).
- [HAROLD 2002] HAROLD, E. R. (2002). *Die XML-Bibel*. MITP Verlag, 2. Aufl.

- [HERMANN 1976] HERMANN, U. (1976). *Fremdwörterlexikon*. Bertelsmann Verlagsgruppe.
- [KAISER 2001] KAISER, U. (2001). *C/C++ – Von den Grundlagen zur professionellen Programmierung*. Galileo Press. ISBN: 3-934358-03-9.
- [KATO et al. 2000] KATO, H., M. BILLINGHURST und I. POUPYREV (2000). *ARToolKit*. Technischer Bericht, Human Interface Technology Laboratory, Seattle, WA. Kopie auf CD-Rom (ARToolkit2.33doc.pdf).
- [KLOOCK und SPAHR 1996] KLOOCK, D. und A. SPAHR (1996). *Medientheorien – Eine Einführung*. Wilhelm Find Verlag, 2 Aufl. ISBN: 3-8252-1986-0.
- [MILGRAM und KISHINO 1994] MILGRAM, P. und F. KISHINO (1994). *A Taxonomy of Mixed Reality Visual Displays*. IEICE Transactions on Information Systems, E77-D(12). URL: http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html.
- [PINTARIC 2002] PINTARIC, T. (2002). *IMS Showcases Augmented Reality Technology at SIEMENS Forum Vienna*. URL, <http://www.ims.tuwien.ac.at/~thomas/siemensforum.html>. Kopie auf CD-Rom (AEKI.pdf).
- [SHOEMAKE 1985] SHOEMAKE, K. (1985). *Animating rotation with quaternion curves*. In: *Proceedings of SIGGRAPH'85*, S. 245–254.
- [TANG et al. 2002] TANG, A., C. OWEN, F. BIOCCA und W. MOU (2002). *Experimental Evaluation of Augmented Reality in Object Assembly Task*. In: *Proceedings of ISMAR 2002, IEEE and ACM International Symposium on Mixed and Augmented Reality*, Darmstadt, Germany. Kopie auf CD-Rom (2002_Pub_ARAssembly_ISMAR.pdf).
- [TANG et al. 2003] TANG, A., C. OWEN, F. BIOCCA und W. MOU (2003). *Comparative Effectiveness of Augmented Reality in Object Assembly*. In: *Proceedings of ACM CHI'2003*. Kopie auf CD-Rom (2003_Pub_ARAssembly_SIGCHI.pdf).
- [TOMIC-KOLUDROVIC et al. 2002] TOMIC-KOLUDROVIC, I., M. PETRIC und I. MITROVIC (2002). *Mixed Reality or One Reality: A Socio-Semiotic Approach to Hybrid Multiagent Environments*. *Journal of Artificial Societies and Social Simulation*, 5(1). URL: <http://jasss.soc.surrey.ac.uk/5/1/6.html>.