## Entwurf

### Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 08.03.2012 | 1.0 | Erste Version des Dokuments | LE |
| 09.03.2012 | 1.1 | Korrekturen und Review Frameworks | DT |
| 02.04.2012 | 1.2 | Interaktion durch Handtracking hinzugefügt | LE |
| 13.04.2012 | 1.3 | Architektur und PDF Darstellung | CH |
| 13.04.2012 | 1.4 | Review, Konvertierung mit Image Magick | DT |
| 16.04.2012 | 1.5 | Physische und logische Sicht, Auflösung | CH |
| 16.04.2012 | 1.6 | Review Auflösung | DT |
| 24.04.2012 | 1.7 | Review Auflösung, Performance | LE |
| 07.05.2012 | 1.8 | Anpassung aus Code Review 03.05.2012 | DT |
| 19.05.2012 | 1.9 | Review Begründung Nutzwertanalyse | DT |
| 29.05.2012 | 1.10 | Miniapps | CH |
| 29.05.2012 | 1.11 | Review Miniapps | DT |
| 30.05.2012 | 1.12 | Architektur | CH |
| 08.06.2012 | 1.13 | Architektur ergänzt, Betrieb und Lebenszyklus der Applikation hinzugefügt | LE |
| 09.06.2012 | 1.14 | Plug-in Möglichkeit, Prozesse und Threads, anklickbare Elemente hinzugefügt, kleine Korrekturen | LE |
| 10.06.2012 | 1.15 | Review Architektur | CH |
| 10.06.2012 | 1.16 | Review und Korrekturen | LE |
| 11.06.2012 | 1.17 | Applikationen | CH |

### Design Entscheide

#### Frameworks

Um eine Applikation mit Microsoft Kinect zu entwickeln, stehen die folgenden drei Frameworks zur Verfügung:

* Kinect for Windows SDK
* OpenNI
* OpenKinect

Nachfolgend einige Anmerkungen zu diesen Frameworks.

##### Framework 1: Kinect for Windows SDK[[1]](#footnote-1)

Das offizielle Kinect Framework von Microsoft für Windows wurde kurz vor Beginn dieser Arbeit, im Februar 2012, in der Version 1.0 herausgegeben. Wenn beachtet wird, dass andere Frameworks schon eher, als Beispiel OpenNI Ende 2010, veröffentlicht wurden, ist dies relativ spät. Entsprechend sind für dieses Framework viel weniger Beispiele und Bibliotheken im Internet zu finden, wobei dafür deren Qualität hoch ist.

Dieses Framework geht durch die Nutzwertanalyse (siehe Unterkapitel I.2.3 Nutzwertanalyse) klar als Sieger hervor.

##### Framework 2: OpenNI[[2]](#footnote-2)

Dieses Framework wurde in der Version 1.0.0.23 im Dezember 2010 erstmals freigegeben und konzentriert sich, im Gegensatz zum Microsoft Kinect SDK, nicht nur auf Kinect als Eingabemöglichkeit, sondern allgemein auf Natural User Interfaces (NUI).

Um weitere Geräte anzusprechen und gerätespezifische Funktionen zu implementieren, lässt sich im Framework zusätzliche Middleware einsetzen. So wird mit NiTE[[3]](#footnote-3) von PrimeSense[[4]](#footnote-4) entwickelt, um das Skeletal Tracking durchzuführen.

##### Framework 3: OpenKinect[[5]](#footnote-5) / libfreenect[[6]](#footnote-6)

OpenKinect ist eine Community, die den libreenect Treiber entwickelt. Leider gibt es dafür aber keine erweiterten Funktionen wie Gestenerkennung oder Skeletal Tracking.

##### Nutzwertanalyse

Um herauszufinden, welches dieser drei Framework das passende für die Entwicklung der Videowall-Applikation ist, wurde am 8. März 2012 eine Nutzwertanalyse durchgeführt.

Die Gewichtung der verschiedenen Kriterien lässt sich wie folgt begründen:

* Das Kriterium „Cooperate Support, Weiterentwicklung, Community“ ist sehr wichtig, da bei der Weiterentwicklung der Applikation sich auch die Libraries oder SDKs weiterentwickeln sollen oder dass neue Features zu Verfügung stehen.
* „Windows Integration und Installation“ ist bedingt wichtig, weil es zwar wünschenswert ist, die Integration in Windows und die Installation ohne grossen Aufwand durchführen zu können, jedoch andere Kriterienpunkte entscheidender für die Wahl des Frameworks sind.
* Das Kriterium „Linux / Mac Kompatibilität“ wurde als nicht wichtig eingestuft, da Kinect selbst schon von der Microsoft Corporation ist und es daher sinnvoll ist, mit Microsoft Technologien zu arbeiten.
* Das Kriterium „Skeletal Tracking Qualität“ ist essentiell für die Interaktion des Nutzers mit der Applikation und erhält daher eine hohe Wichtigkeit.
* Die „Libraries für Gestenerkennung“ sind bedingt wichtig, da Gesten zum jetzigen Zeitpunkt nicht verwendet werden.
* Die „Record / Replay Funktionalität“ ist wichtig, da damit ein Nutzer simuliert werden kann und so die Applikation vereinfacht getestet werden kann.
* Das Kriterium „Dokumentation“ ist wichtig, um die Features Framework zu kennen und zu verstehen.
* Zu den Punkten „Mit Framework realisierte Beispiele und Libraries (Quantität)“ und „Mit Framework realisierte Beispiele und Libraries (Qualität)“ ist zu erläutern, dass Beispiele dem Verständnis helfen, nicht aber notwendig sind und daher nur bedingt wichtigsind. Bei den Beispielen bedingt es nicht nur einer hohen Anzahl sondern auch einer guten Qualität.

Die Evaluation wurde manuell durchgeführt. Die Bewertung der einzelnen Kriterien mittels wenig wichtig (1), bedingt wichtig (3) und sehr wichtig (5) ist selbsterklärend und wird daher nicht begründet.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nutzwertanalyse: Auswahl Kinect Framework | | | | | | |  |
|  |  | **Framework 1** | | **Framework 2** | | **Framework 3** | |
|  |  | **Kinect for Windows SDK** | | **OpenNI** | | **OpenKinect / libfreenect** | |
| **Kriterium** | **Gewichtung** | **Bewertung** | **Total** | **Bewertung** | **Total** | **Bewertung** | **Total** |
| **Cooperate Support, Weiterentwicklung, Community** | 5 | 5 | 25 | 3 | 15 | 3 | 15 |
| **Windows Integration und Installation** | 3 | 5 | 15 | 3 | 9 | 3 | 9 |
| **Linux / Mac Kompatibilität** | 1 | 1 | 1 | 5 | 5 | 5 | 5 |
| **C# / .NET Framework / Visual Studio Integration** | 5 | 5 | 25 | 3 | 15 | 1 | 5 |
| **Skeletal Tracking Qualität** | 5 | 5 | 25 | 3 | 15 | 1 | 5 |
| **Libraries für Gestenerkennung** | 3 | 3 | 9 | 5 | 15 | 1 | 3 |
| **Record / Replay Funktionalität** | 5 | 3 | 15 | 5 | 25 | 1 | 5 |
| **Dokumentation** | 5 | 5 | 25 | 3 | 15 | 1 | 5 |
| **Mit Framework realisierte Beispiele und Libraries (Quantität)** | 3 | 1 | 3 | 5 | 15 | 5 | 15 |
| **Mit Framework realisierte Beispiele und Libraries (Qualität)** | 3 | 5 | 15 | 1 | 3 | 1 | 3 |
| **Total Punkte** |  |  | **158** |  | **132** |  | **70** |
| **Rang** |  |  | **1** |  | **2** |  | **3** |
|  |  |  |  |  |  |  |  |
| Bemerkung: Die Gewichtungs- / Bewertungsskala geht von wenig (1), bedingt (3) bis zu sehr wichtig (5). | | | | | | | |

Tabelle - Nutzwertanalyse: Auswahl Kinect Framework

Aus der Analyse (siehe Tabelle 1 - Nutzwertanalyse: Auswahl Kinect Framework) geht das Framework 1: Microsoft Kinect SDK als Sieger vor dem Framework 2: OpenNI hervor.

##### Sensitivitätsanalyse

In der Sensitivitätsanalyse wird untersucht, wie stark sich eine Änderung auf das Gesamtergebnis auswirken würde.

Das Framework 3 wird auch bei Änderungen der Bewertung nicht als Sieger hervorgehen.

Zwischen dem Framework 1 und 2 ist der Bewertungsunterschied einiges kleiner. Da das Framework 1 von Microsoft aber über ein ausgeklügeltes, vorhersehendes Skeletal Tracking System[[7]](#footnote-7) verfügt, eine bessere Dokumentation besitzt und perfekte Windows, Visual Studio, C# und .NET Integration bietet, würde sich dieses Framework trotz Anpassungen an einzelnen Gewichtungen oder Bewertungen gegenüber dem Framework 2 durchsetzen. Demensprechend ist diese Nutzwertanalyse nicht sensitiv gegenüber Änderungen.

##### Weiteres

Bei der Nutzwertanalyse wurden zwar möglichst viele nummerisch bewertbare Kriterien untersucht, es fehlt aber noch der persönliche Eindruck. Für das Projekt Videowall fällt die Entscheidung gefühlsmässig auf das Microsoft Framework, da auch die übrigen für die Arbeit eingesetzten Technologien von Microsoft sind und damit gerechnet werden muss, dass andere Frameworks nicht ohne Probleme mit den für das Projekt bereits festgesetzten Microsoft Technologien zusammenarbeiten können.

Weiter ist in der Bachelorarbeit „Kinect Bodyscanner“ von Felix Egli und Michael Schnyder [egli11] im Kapitel 3.3.1 Kinect Framework auf Seite 30 beschrieben, dass OpenNI für die Arbeit nur eine temporäre Lösung ist. Sie geben an, dass es geplant ist, auf das offizielle Framework von Microsoft zu wechseln, sobald dieses verfügbar ist. Auch diese Aussage spricht klar für das „Kinect for Windows SDK“.

#### PDF Darstellung

Die Poster liegen alle im PDF Format vor. Die unterschiedlichen Möglichkeiten, wie diese Dokumente in der Applikation dargestellt werden können, sind nachfolgend beschrieben. Anschliessend folgt eine Nutzwertanalyse zur Eruierung der am besten geeigneten Darstellungsvariante.

##### Varianten

###### Variante 1: PDF direkt darstellen

Wird diese Variante gewählt, so können die PDF Dokumente ohne zusätzlichen Umwandlungsaufwand verwendet werden. Jedoch bietet das WPF-Framework keine Komponente an, welche ein PDF Dokument direkt darstellen kann. Es besteht aber die Möglichkeit, einen Browser einzubinden, welcher zur Darstellung der PDFs den auf dem System installierten PDF Reader nutzt. Dabei sollten aber die Steuerelemente (Drucken, Verschicken, Zoom, Suche etc.) sowie der Standardhintergrund des PDF Readers nicht sichtbar sein. Dies kann jedoch nicht über WPF gesteuert werden. Des Weiteren sind die PDF Dokumente von unterschiedlicher Qualität, was sich zeigt, wenn in der Applikation von einem zum nächsten Dokument navigiert wird. Einige Dokumente benötigen sehr viel Zeit, bis sie geladen sind, andere wiederum haben nur eine kurze Ladezeit.

###### Variante 2: Umwandlung zu XPS

In WPF können XPS Dokumente mittels der DocumentViewer Klasse angezeigt werden. Das Layout des Viewers kann angepasst werden, so können beispielsweise die Steuerelemente ausblendet werden. Bei einem XPS Dokument handelt es sich, wie bei einem PDF Dokument, um eine Vektorgrafik. Ein XPS Dokument benötigt spürbar Zeit, um geladen zu werden, wenn von einem zum nächsten Dokument navigiert wird. Es ist schlecht möglich, die XPS-Dokumente im Voraus zu laden, da das Laden immer über den GUI-Thread geschieht, dieser aber gleichzeitig auch für Animationen und Ähnliches verwendet wird.

###### Variante 3: Umwandlung zu Bild

Die einfachste Möglichkeit der Darstellung der PDFs besteht darin, die Dokumente in Rastergrafiken umzuwandeln. Diese können mit minimalem Aufwand in eine WPF-Applikation eingebunden werden. Mit Hilfe verschiedenster Open Source Libraries ist die Umwandlung von einem PDF zu einem Bild problemlos möglich. Getestet wurde dies mit ImageMagick[[8]](#footnote-8), einer frei verfügbaren Software. Sie bietet die Umwandlung von Dokumenten zu Bildern. Eine Rastergrafik benötigt auch deutlich weniger Ladezeit als die Dokumente bei den beiden anderen Varianten. Nachteilig an dieser Lösung ist jedoch, dass das Dokument durch die Umwandlung auf eine Maximalgrösse beschränkt wird und dementsprechend bei einer Änderung der Auflösung der Bildschirme die Bilder neu umgewandelt werden müssen.

##### Nutzwertanalyse

Nachfolgende Nutzwertanalyse, welche am 13. April 2012 durchgeführt wurde, lässt die Variante 3: „Umwandlung zu Bild“ als Sieger hervorgehen.

Die Gewichtung der verschiedenen Kriterien lässt sich wie folgt begründen:

* Das Kriterium „Programmieraufwand“ ist nur bedingt wichtig, da der Aufwand für die Programmierung für eine der drei Varianten nur gering ist im Vergleich zum Aufwand, welcher für das Ausprogrammieren der gesamten Applikation betrieben wird.
* Die „Darstellungsqualität“ ist bedingt wichtig. Das Poster muss lesbar sein. Dazu bedarf es aber keiner überaus hohen Qualität.
* Die „Ladezeit bei Navigation“ darf nicht zu lange dauern, ansonsten würde die User Experience wesentlich darunter leiden. Das könnte dazu führen, dass die Nutzer nicht mehr mit der Wall interagieren wollen. Darum ist dieses Kriterium wichtig.
* Das Kriterium „Abhängigkeit von externen Komponenten“ ist ebenfalls wichtig. Sind Komponenten über eine Zeit lang nicht verfügbar, übt sich dies negativ auf den Betrieb aus.

Die Evaluation wurde manuell durchgeführt. Die Bewertung der einzelnen Kriterien mittels wenig wichtig (1), bedingt wichtig (3) und sehr wichtig (5) ist selbstsprechend und wird daher nicht begründet.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nutzwertanalyse: PDF Darstellung | | | | | | |  |
|  |  | **Variante 1: PDF direkt darstellen** | | **Variante 2: Umwandlung zu XPS** | | **Variante 3: Umwandlung zu Bild** | |
| **Kriterium** | **Gewichtung** | **Bewertung** | **Total** | **Bewertung** | **Total** | **Bewertung** | **Total** |
| **Programmieraufwand** | 3 | 1 | 3 | 3 | 9 | 5 | 15 |
| **Darstellungsqualität** | 3 | 5 | 15 | 5 | 15 | 3 | 9 |
| **Ladezeit bei Navigation** | 5 | 3 | 15 | 1 | 5 | 5 | 25 |
| **Abhängigkeit von externen Komponenten** | 5 | 1 | 5 | 3 | 15 | 5 | 25 |
| **Total Punkte** |  |  | **33** |  | **44** |  | **74** |
| **Rang** |  |  | **3** |  | **2** |  | **1** |
| Bemerkung: Die Gewichtungs- / Bewertungsskala geht von wenig (1), bedingt (3) bis zu sehr wichtig (5). | | | | | | | |

Tabelle - Nutzwertanalyse: PDF Darstellung

### Betrieb der Applikation

Die Ausführungen zum Betrieb der HSR Videowall sind im Kapitel (TODO: link Domain Analyse) zu finden.

### Lebenszyklus der Applikation

#### Videowall: Hauptapplikation und Framework

Zum Zeitpunkt der Abgabe dieser Bachelorarbeit besteht ein funktionsfähiger Prototyp. Bis sie für den produktiven Betrieb bereit ist, ist noch Weiterentwicklungsarbeit nötig. Diese Weiterentwicklung wird durch das Institut für Software an der HSR (IFS) erfolgen.

Wie im Unterkapitel I.1.3.2 Betrieb der Applikation beschrieben, ist ein automatisches Deployment der Applikation vorgesehen. Dabei handelt es sich um eine zentrale Funktion, die entwickelt werden soll. Zusätzlich ist die Möglichkeit der Administration der Inhalte über ein CMS wichtig, diese muss ebenfalls umgesetzt werden.

Für den Lebenszyklus werden die folgenden groben Meilensteine vorgeschlagen:

* Bis Herbst 2012: Beschaffung der Videowall Hardware
* Bis Oktober 2012: Weiterentwicklung der Videowall bis im Oktober 2012 (siehe auch TODO: ref Weiterentwicklung)
* Ende Oktober: erstes Deployment der Videowall und des CMS
* November 2012: Erfahrungen sammeln und wichtige Verbesserungen und Anpassungen implementieren

Danach soll eine Weiterentwicklung in einem halbjährlichen Zyklus erfolgen, jeweils immer nachdem die neuen Bachelorposter auf der HSR Videowall verfügbar sind.

Während dem Betrieb der Videowall ist es zudem notwendig, dass eine verantwortliche Instanz definiert wird, die bei Problemen und Fehlern der Videowall diese bearbeiten kann.

#### Plug-ins

Ein denkbarer Ablauf ist für das Erstellen und Deployen der Plug-ins ist im Kapitel (TODO ref domain analyse) beschrieben.

### Systemübersicht

In der Domain Analyse sind die Tiers der gewünschten Architektur grob beschrieben (TODO DomainAnalyse). Da für die Bachelorarbeit ein begrenzter Zeitrahmen zur Verfügung stand, wurde das Hauptaugenmerk auf die „HSR Videowall mit Kinect“ gelegt. Die weiteren Tiers konnten aus zeitlichen Gründen nicht implementiert werden. Abbildung 1 - Systemübersicht zeigt das in der Arbeit entwickelte System.



Abbildung 1 – Systemübersicht

Der Prototyp der Machbarkeitsstudie bietet folgende Funktionen:

Die wichtigste Funktion des Prototyps ist die dynamische Erweiterbarkeit, welche in Form eines Plug-in Frameworks (siehe I.1.7 Plug-in Framework) umgesetzt wurde. Für dieses wurden zwei Plug-in Applikationen erstellt. Mit der einen Applikation können die Bachelorposter angesehen und mit der anderen das Mittagsmenu der Mensa angezeigt werden.

### Architektur

#### Logische Sicht

Die grundlegende Architektur wurde im Team erarbeitet und durch Silvan Gehrig am 02.04.2012 validiert. Die verschiedenen Schichten sind in den nachfolgenden Unterkapiteln beschrieben.



Abbildung - Architektur Diagramm

Es wurde diskutiert, ob zwischen den Schichten ViewModels und Services (nicht im Diagramm ersichtlich) zusätzlich ein Business Layer eingefügt werden sollte. Da die bereitgestellten Daten jedoch nicht manipuliert sondern lediglich angezeigt werden, ist ein Business Layer überflüssig. Dieser würde nur das Service Interface kopieren und dadurch zu einem Durchlauferhitzer werden. Deshalb wurde der Business und Services Layer zu einem gemeinsamen ServiceModels Layer zusammengefasst.

##### Common

Im Common Layer befinden sich Klassen, welche von Klassen aus den meisten anderen Schichten verwendet werden.

##### ResourceLoader

In der Schicht ResourceLoader werden Resourcen, welche für die Videowall benötigt werden, geladen.

##### Interfaces

Die Interfaces werden von Klassen eines Plug-ins implementiert. Die Interfaces definieren Elemente, welche jedes Plug-in zu Verfügung stellen muss und noch weitere Interfaces, welche das Plug-in für die Nutzung weiterer Funktionalitäten (z.B. Zugriff auf Skelett-Daten des Kinect Sensors) nutzten kann.

##### Data

Der Data Layer regelt die Datenbankanbindung, lädt die benötigten Ressourcen und greift auf die Daten von Kinect zu (beispielsweise Skelett- oder Tiefendaten).

##### ServiceModels

Im ServiceModels Layer werden die vom Data Layer erhaltenen Daten in Models gespeichert. Diese werden dann über verschiedene Services den ViewModels zur Verfügung gestellt.

##### ViewModels

Die ViewModels stellen die von den Services erhaltenen Daten der View zur Verfügung. Auf diesem Layer befindet sich auch die Implementation des ICommand Interfaces. Diese Funktionen können somit von ViewModels und Views verwendet werden.

##### Views

Die Views stellen die Elemente aus den ViewModels grafisch dar.

#### Patterns

Nachfolgend sind die verwendeten Patterns beschrieben.

##### MVVM

Das MVVM („Model“, „View“, „ViewModel“) Pattern [microsoft09] wird benötigt, um die View vom Model zu entkoppeln. Deshalb wird als Zwischenglied ein ViewModel erzeugt, das die Commands des GUIs abarbeitet und die verfügbaren Elemente dem GUI zur Verfügung stellt.

Die grundlegende Idee dahinter ist, dass sich das GUI schneller ändert als die Businesslogik und deshalb die zwei Komponenten möglichst stark abzutrennen sind. Zusätzlich kann das ViewModel mit Unit Tests geprüft werden.

##### Inversion of Control / Dependency Injection mit Unity

Damit die Komponenten jederzeit und einfach ausgetauscht werden können, wurde mit Unity Containern[[9]](#footnote-9) gearbeitet, um Inversion of Control durch Dependency Injection zu ermöglichen. So können beispielsweise auf eine einfache Art und Weise Komponenten einer Software ausgetauscht werden, indem die Container ausgetauscht werden. Bei der Videowall wird dies benutzt, um bei der Entwicklung zwischen dem echten Kinect Sensor und einem simulierten Kinect Sensor zu wechseln. Dependency Injection kann auch beim Testen helfen, indem die Mock Objekte beim Unity Container registriert werden.

##### Extension Interface

Das Extension Interface zeigt auf, wie eine Architektur aufgebaut werden kann, damit Erweiterungen an der Applikation einfach vorzunehmen sind.

Wie das Pattern für die Videowall verwendet wird, kann im nachfolgenden Unterkapitel (I.1.5.4 Plug-in Framework) nachgelesen werden.

#### Prozesse und Threads

##### Allgemein

Grundsätzlich machen Multithreading und Multiprocessing dann Sinn, wenn die Performance einer Applikation erhöht werden soll. Da dies in der jetzigen Version noch nicht elementar ist, laufen alle Verarbeitungen in genau einem Prozess in genau einem Thread, dem GUI-Thread. Quasiparallele Verarbeitungen laufen über die Dispatcher Queue [microsoft12.3]:

Fenster neu zeichnen

Maus-Event verarbeiten

Tastatureingabe verarbeiten

Skelett neu zeichnen

Click auf Button verarbeiten

Handcursor-Animation starten

Der GUI-Thread führt die Aktion aus, die zuvorderst in der Queue liegt und löscht diese Aktion dann aus der Queue. Alles läuft seriell. Da die einzelnen Frames aber nur sehr wenig Zeit für die Verarbeitung benötigen, scheinen sie parallel abzulaufen.

…

Abbildung - Dispatcher Queue

Eine Ausnahme stellt die Klasse *Kinect.Toolbox.Record.SkeletonReplay* dar, die sich in einer Library befindet und in der Klasse *VideoWall.Data.Kinect.Implementation.AutoPlayFileSkeletonReader* benutzt wird. Da dieses SkeletonReplay in einem anderen Thread läuft, wird beim Feuern des Events die Verarbeitung sofort in die Dispatcher Queue des GUI-Threads verschoben, damit die Verarbeitung weiter seriell ablaufen kann.

Weiter ist es möglich, dass das .NET Framework oder das Kinect Framework von Microsoft im Hintergrund weitere Threads laufen lässt, diese Events aber durch die Frameworks in die Dispatcher Queue des GUI-Threads verschoben werden.

##### Kinect

Das Kinect SDK von Microsoft bietet auf der Klasse *KinectSensor* einen Event *SkeletonFrameReady*. Dieser wird ausgelöst sobald neue Skelett-Daten bereit sind. Dieser Event tritt bereits im GUI-Thread auf und muss dementsprechend nicht in die Dispatcher Queue verschoben werden.

##### Plug-ins

Da zurzeit auch die Plug-ins im GUI-Thread ablaufen ist es wichtig, dass die Plug-ins vor dem produktiven Einsatz einem Code Review unterzogen werden. So kann sichergestellt werden, dass die Plug-ins das GUI nicht blockieren und dass die Plug-ins keine eigenen Threads oder Prozesse starten.

Die Events, die an die Plug-ins gesendet werden, laufen auch im GUI-Thread ab.

##### Echte Parallele Verarbeitung

Sollte es in Zukunft nötig sein, mehrere Threads oder Prozesse einzusetzen, wird empfohlen, die Events nach der parallelen Verarbeitung sofort wieder in die Dispatcher Queue des GUI-Threads zu verschieben. Mit dieser Massnahme ist Programmieren ohne (Dead)Locks weiterhin möglich.

### Plug-in Framework

#### Grundlagen

Die erste Frage, die sich bei einem Framework stellt, ist, wie ein Plug-in in das Framework geladen wird (der Extension Point). Microsoft bietet für diesen Zweck das Managed Extensibility Framework (MEF)[[10]](#footnote-10) an. Technische Details dazu können in der MEF-Dokumentation[[11]](#footnote-11) nachgelesen werden.

Die wichtigste Funktionalität von MEF, die für die HSR Videowall gebraucht wird, ist die folgende:

* Das Schlüsselwort Export zeichnet eine Klasse (Einstiegspunkt), die ein von einem Framework (Videowall-Applikation) definierten Interface (IApp) implementiert, für den Export aus.
* Das Framework (Videowall-Applikation) importiert alle Klassen, die einen bestimmten Pfad haben (Ordner Extensions) und das Interface IApp implementieren.



Abbildung - Poster-Applikation (Extension) wird über [Export(typeof(IApp))] als IApp exportiert

Die obenstehende Abbildung zeigt die Klasse PosterApp, welche das Interface IApp implementiert. Der Ausdruck [Export(typeof(IApp))] markiert die Klasse für den Export.



Abbildung - AppController koordiniert den Import der Apps

Der Import der Apps wird vom AppController koordiniert.



Abbildung – Der ExtensionFolder, der sich im Framework befindet, importiert über das Attribut [Import] die Klassen, die das Interface IApp implementieren und sich in einem bestimmten Ordner (Directory) befinden.

Der Ausdruck [Import] im Framework (Videowall-Applikation) importiert die Klasse, welche das Interface IApp implementieren und sich in einem bestimmten Ordner befindet.



Abbildung - Der ExtensionManager führt den Import schliesslich mithilfe von MEF aus

Der ExtensionMagager führt den Import des Plug-ins schliesslich mithilfe von MEF11 aus.

#### Probleme

Beim Entwickeln eines Frameworks ist oftmals nicht vorhersehbar, wie dieses in der Zukunft aussehen wird, da sich die Anforderungen an das Framework stetig ändern. Würde nur ein einziges Interface (IApp), über das die Services des Frameworks angesprochen werden können, zur Verfügung gestellt, so müsste sich dieses ständig ändern. Folglich müssten die Plug-ins, zum Beispiel die PosterApp (siehe Unterkapitel I.1.2.1.1 Grundlagen), nach jeder Änderung am Interface (IApp) neu kompiliert werden. Mit nur einem Interface ist es also schwierig, den Plug-ins neue Funktionalität zur Verfügung zu stellen.   
Ein weiteres Problem eines einzigen Interfaces ist, dass dieses beliebig gross werden kann und dadurch die Kopplung steigt und die Kohäsion sinkt, was sehr unschön ist.

Das anfängliche IApp Interface wurde folgendermassen implementiert:



Abbildung - Anfängliche Implementation des Interfaces IApp

Wie in Abbildung 3 - Anfängliche Implementation des Interfaces IApp erkennbar ist, ist das Interface relativ gross und stellt verschiedenste Services zur Verfügung, die nichts miteinander zu tun haben. Beispielsweise das ResourceDirectory Property, welches die Plug-in-Dateien zur Verfügung stellt oder der SkeletonChangedEvent, der vom Framework aufgerufen werden soll, sobald sich das Skelett verändert hat.

Ändert sich dieses Interface, beispielsweise durch Hinzufügen neuer Funktionalität, müssen auch immer alle Plug-ins neu kompiliert werden. Dies ist suboptimal, speziell dann, wenn die Plug-ins von verschiedenen Personen gewartet werden.

#### Lösung

Die Lösung ist an das Extension Interface [schmidt00] angelehnt. Es bietet einen Ansatz, das Problem des ständig ändernden Interfaces zu lösen. In der Videowall-Applikation wurde das Extension Interface in abgeänderter Form angewendet, ohne die Vererbung des Root Interfaces. Zusätzlich wurde Unity12 verwendet um die Factory aus dem Extension Interface zu ersetzen. Die gegenwärtige Implementation des Interfaces sieht folgendermassen aus:



Abbildung - Das IApp Interface

Das IApp Interface (siehe Abbildung 4 - Das IApp Interface) bietet einen Einstiegspunkt. Da jede Applikation dieses Interface implementiert, sind hier nur die Anforderungen beschrieben, die jede Applikation anbieten muss. Speziell ist die Methode Activate, die auf jeder vom Framework zu ladenden Extension genau einmal aufgerufen wird (siehe auch Dependency Injection, [eilbrecht07]). In dieser Methode kann das Plug-in über das IVideoWallServiceProvider-Objekt weitere Services anfordern:



Abbildung - Durch den IVideoWallServiceProvider können weitere Extensions geladen werden

Über die Methode GetExtension des IVideoWallServiceProviders aus obiger Abbildung kann das Plug-in weitere Services (IVideoWallService) anfordern.

Das IVideoWallService Interface ist ein Marker-Interface. Es ist nicht vorgesehen, dass Applikationen weitere Plug-ins registrieren können. Dies stellt den Hauptunterschied zum Extension Interface Pattern dar.

##### Dynamische Sicht

Nachfolgend ein Sequenzdiagramm, welches den Ablauf des Ladens und Aktivierens der Applikationen durch das Framework (Videowall-Applikation) veranschaulicht.



Abbildung - Sequenzdiagramm, Ablauf des Ladens und Aktivierens von Applikationen durch das Framework

### Umsetzung des Demomodus

Die für den Demomodus gesammelten Ideen sind im Kapitel (TODO Link Domainanalyse Demomodus) zu finden.

##### Besprechung des Demomodus „Kraftfeld“

Am 07.05.2012 besprach das Team, wie bei der Umsetzung des ausgewählten Demomodus „Kraftfeld“ vorgegangen werden soll. Es handelt sich hierbei um den Demomodus, bei dem durch Vorbeilaufen die über alle Monitore verteilten Objekte (z.B. kleine Stücke eines Posters) bewegt werden können (siehe Domain Analyse, Demomodus, Sammlung der Ideen, Beschreibung zu Idee Nummer 4).

Die Abbildung 12 - Teilaufgaben des Demomodus "Kraftfeld" zeigt, dass die Applikation aus sechs Teilaufgaben bestehen müsste. Der erste Punkt ist das Generieren von Screenshots (1), welche dann in Teilchen zerschnitten werden. Weiter müssen diese Teilchen über den ganzen Bildschirm verteilt angezeigt werden(2). Damit bereits hier Dynamik im Spiel ist, benötigt jedes einzelne Teilchen eine Grundanimation (z.B. eine leichte Hin- und Herbewegung). Der dritte Punkt ist das Zusammenfügen der Teilchen (3) zu einem Ganzen, dem Ursprungsbild. Als Nächstes müssen die Bewegungen der Teilchen (4), die durch das Passieren der Videowall ausgelöst wird, festgelegt und implementiert werden. Dazu mehr im nachfolgenden Abschnitt, der die Abbildung 15 - Ideen zur Bewegungsart der Teilchen beschreibt. Abschliessend folgt das Wechseln vom Demomodus in den Interaktionsmodus (5) und umgekehrt (6).  
Ein Usability Test und das Umsetzen der allfällig dadurch entstandenen Verbesserungsansätze runden die Implementation ab.



Abbildung 12 - Teilaufgaben des Demomodus "Kraftfeld"

Die Umsetzung des im obigen Abschnitt aufgelisteten Punktes Nummer 4 benötigt mathematische Vorarbeit. Die Abbildung 13 - Ideen zur Bewegungsart der Teilchen zeigt Ansätze, wie die Bewegungen der Teilchen berechnet werden könnte.



Abbildung - Ideen zur Bewegungsart der Teilchen

Im oberen Teil der Abbildung soll mit den Abstandsangaben, die vom Tiefensensor der Kinect erfasst werden, gearbeitet werden. Der Tiefensensor misst für jedes Pixel, wie weit der darauf zu sehende Mensch oder Gegenstand vom Sensor entfernt ist. Mit Hilfe der daraus gewonnenen Zahlwerte könnten nun Geradensteigungen und Vektorrichtungen für die Bewegung der Teilchen, welche auf den Wall verteilt dargestellt werden, ausgerechnet werden.

Im unteren Teil der Abbildung wird im Hintergrund ein feines Raster über die Monitore gelegt. Ein Quadrat dieses Rasters beinhaltet mehrere Pixel. Wird nun das Skelett des Benutzers erkannt, so werden die Rasterquadrate, von denen ein oder mehrere Pixel im Bereich des Skeletts sind, als besetzt markiert (grün schraffierte Fläche). Teilchen, die sich auf diesen besetzen Rasterquadraten befinden, suchen sich nun den kürzesten Weg auf ein freies Quadrat. Teilchen, welche bereits auf einem freien Quadrat dargestellt werden, bewegen sich nicht.

###### Fazit

Beim Notieren der Teilaufgaben, welche alle erledigt werden müssen, um den Demomodus umsetzen zu können, wurde dem Team bewusst, dass die verfügbare Zeit nur für die Umsetzung des einfacheren Demomodus (Idee 2, siehe Domain Analyse Auswahl der besten Idee für den Demomodus) reicht.   
Das Team kam daher zum Schluss, den Demomodus „Kraftfeld“ aus zeitlichen Gründen nicht umzusetzen. Erklärungen zum alternativ umgesetzten Demomodus „Teaser“ sind im nachfolgenden Kapitel (I.1.3.2.2.2 Umsetzung des Demomodus „Teaser“) zu finden.

##### Umsetzung des Demomodus „Teaser“

Auch dieser Demomodus wurde in einzelne Teilaufgaben unterteilt. Wie bei der Idee I.1.7.1.1 Besprechung des Demomodus „Kraftfeld“ muss es möglich sein, zwischen dem Interaktions- und Demomodus zu wechseln. Sobald der Demomodus angezeigt wird, soll der Hintergrund auf eine zufällig ausgewählte Farbe gesetzt werden. Zudem soll auch ein Teaser-Text zur jeweilig im Hintergrund aktiven App angezeigt werden. Dabei könnte es sich, wie in der (TODO Ref Domain Analyse Bild Demomodus Ideen 1-3) ersichtlich, um einen Text wie „Hunger? – Dann stell dich hier hin“ handeln.

Die nachfolgende Abbildung 14 - Zustandsdiagramm Interaktions- und Demomodus zeigt das Zustandsdiagramm, welches den Wechsel vom Interaktionsmodus (Active) in den Demomodus (Teaser) und zurück aufzeigt.

Zu Beginn befindet sich die Applikation im Interaktionsmodus (Active). Solange ein Skelett erkannt wird, bleibt die Applikation in diesem Status. Wird über eine bestimmte Zeit (beispielsweise 10 Sekunden) kein Skelett mehr erkannt, wird in den Demomodus (Teaser) gewechselt. Die Applikation bleibt so lange im Demomodus, bis wieder ein Skelett erkannt wurde. Darauf folgt der Wechsel in den Countdown. Wird hier immer ein Skelett erkannt, so läuft Zähler von 5 Sekunden rückwärts bis auf 0 Sekunden und die Applikation wechselt in den Interaktionsmodus (Active). Falls im Countdown kein Skelett mehr erkannt werden sollte, so wird zurück in den Demomodus (Teaser) gewechselt.

Währenddem sich die Applikation im Demomodus befindet, werden im Hintergrund nach Ablauf einer bestimmten Zeit (zum Beispiel 20 Sekunden) die aktuelle Applikation und die Farbe des Demomodus-Hintergrunds gewechselt.

Während dem ganzen Ablauf speichert die Applikation jeweils die Zeit ab, bei der zuletzt ein Skelett erkannt wurde. Somit ist es möglich, die Zeitspanne, während der kein Skelett erkannt wurde, zu messen.



Abbildung - Zustandsdiagramm Interaktions- und Demomodus

### Interaktion durch Handtracking

In der Domain Analyse wurde evaluiert, dass der Benutzer mithilfe der Hand die Applikation bedienen kann („Meine Hand ist die Maus“) (TODO: Link). Wie das genau funktioniert, wird in diesem Kapitel erläutert.

#### Kinect Daten

Eines der wichtigsten Features des Kinect SDK ist das sogenannte Skeletal Tracking. Hierbei wird mit Hilfe der Sensoren (Tiefensensor, Bildsensor, Infrarotsensor) versucht, ein menschliches Skelett zu erkennen, und zwar in Echtzeit. Es ist möglich, gleichzeitig von zwei Personen das Skelett anzuzeigen. Für das Handtracking auf der Videowall ist aber nur das Tracken eines Skeletts vorgesehen.



Abbildung - Beispiel eines Skeletts

#### Handtracking

Wie aus der obigen Abbildung (Abbildung 15 - Beispiel eines Skeletts) ersichtlich ist, besteht das Skelett aus einzelnen Punkten, welche die Gelenke wie Schultern oder Knie der verfolgten Person darstellen. Es kann daher die rechte Hand eruiert und dargestellt werden.  
Die Position der Hand des Benutzers muss auf dem Bildschirm zeitgleich nachgestellt werden. Damit sich der Benutzer der Applikation nicht zu viel und weit bewegen muss, wird eine Grenze für das Tracken der Hand festgelegt. Das sieht schematisch folgendermassen aus:



Abbildung - Skelett mit Zone (rot) für das Handtracking

Der rote Bereich stellt den Bildschirm dar. Wenn nun der Benutzer seine Hand in der oberen rechten Ecke der roten Zone bewegt, so wird diese oben rechts auf dem Bildschirm angezeigt, wie Abbildung 17 - Beispiel Monitor mit Handtracking zeigt. Dort, wo sich die Hand im roten Bereich befindet, wird sie folglich auf dem Bildschirms angezeigt. Befindet sich die Hand ausserhalb des roten Bereichs, so wird sie (analog zur Maus auf dem Bildschirm) am Rand des Bildschirms oder gar nicht angezeigt.



Abbildung - Beispiel Monitor mit Handtracking

Wie die konkreten Masse des Bereichs für das Handtracking lauten und wo sich der Bereich genau befindet, ist in der Entwicklungsphase noch detailliert zu definieren und kann direkt dem Quellcode entnommen werden. Grundsätzlich ist klar, dass sich der Bereich über der Hüfte des Skelettes befinden und etwa bis zur Körpermitte gehen wird. Ebenfalls wird der Bereich nicht weit über die Position des Kopfes hinausragen.

#### Anklickbare Elemente

Für eine erste Version der Videowall, welche noch nicht mit Gesten gesteuert werden kann, ist es notwendig, dass gewisse Elemente angeklickt werden können, wie zum Beispiel das Menu zum Navigieren oder die Pfeile zum Browsen der Poster. Da diese Elemente alle mit Buttons realisiert wurden, müssen diese somit anklickbar sein. Falls neue Buttons hinzugefügt werden, sollen diese Buttons ebenfalls anklickbar sein.

Um also diese anklickbaren Elemente zu suchen, wird nach dem Starten der Applikation und den Plug-ins der gesamte Visual Tree[[12]](#footnote-12) nach Buttons durchsucht und in einer Liste gespeichert. Sobald sich dann der Handcursor bewegt wird, wird durch die Liste der Buttons iteriert und untersucht, ob sich der Handcursor über einem Button befindet. Falls ja, wird ein Timer gestartet, der auf diesen Button klickt, sobald eine bestimmte Dauer (ca. 1.5 Sekunden) abgelaufen ist.



Abbildung - Handcursor auf nicht anklickbarem Element

Abbildung - Ablauf eines Klicks auf einen Button

### Applikationen

#### Hauptapplikation

Die Videowall Applikation stellt die Hauptapplikation dar. Sie sorgt dafür, dass die verschiedenen Plug-ins geladen (diese Funktion wird im Kapitel I.1.5.4 Plug-in Framework erläutert) und visualisiert werden. Auch das Handtracking (siehe I.1.9 Interaktion durch Handtracking) ist darin umgesetzt. Zudem stellt sie die verschiedenen Interfaces zu Verfügungen, welche von den Plug-ins implementiert werden.

#### Plug-in-Applikationen

Durch das Team wurden zwei Inhalte für die Videowall definiert. Diese werden nachfolgend aufgezeigt.

##### Poster-Applikation

Die Poster-Applikation zeigt die Bachelorposter der verschiedenen Studiengänge. Es wird jeweils ein Poster in Grossansicht angezeigt und über die Pfeile kann zum nächsten oder vorhergehenden Poster navigiert werden.

##### Mittagsmenu-Applikation

Die Mittagsmenu-Applikation zeigt das aktuelle Menu der Mensa der HSR an.

#### Mini-Applikationen

Im Verlauf des Projektes wurden diverse kleine Anwendungen erstellt, welche Prototypen von essentiellen Teilen der ganzen Applikation sind. Diese befinden sich im Versionsverwaltungssystem unter code\miniapps\trunk und werden nachfolgend kurz erläutert.

##### DemoMode

Der Demomodus wurde zu Beginn ohne die Daten des Kinect Skeletts erstellt, die Skeletterkennung wird in dieser Mini-Applikation durch das Drücken eines Buttons simuliert. Sobald die Logik des Demomodus korrekt umgesetzt war, wurde dieser ins Hauptprojekt integriert.

##### DesignMenu

Die DesignMenu-Applikation wurde benutzt, um direkt in Expression Blend verschiedene Designs für die Tabs des Menus zu erstellen und untereinander zu vergleichen.

##### DesignPosterNavigationButtons

Um verschiedene Varianten der Navigationsschaltflächen für die Poster-Applikation zu prüfen, wurde ein separates Projekt erstellt. Die verschiedene Designs wurden in Expression Blend erstellt und verglichen.

##### HandCursorDemoApp

Für den Wizard of Oz Test musste in der Applikation anstelle des Maus-Cursors eine Hand dargestellt werden, mit welcher die Bewegungen des Skeletts der Testperson nachgeahmt wurden.

##### KinectHandTracker

Beim KinectHandTracker handelt es sich um ein Projekt, in welchem das Handtracking erstmals implementiert und erprobt wurde, bevor es in die Hauptapplikation eingebunden wurde.

##### KinectRecorder

In der KinectRecorder-Applikation lassen sich Skelettbewegungen aufzeichnen und wieder abspielen. Das wiederabspielbare File kann in der Hauptapplikation dazu verwendet werden, ein Skelett und desseen Bewegungen zu simulieren auch wenn kein Kinect Sensor am Computer angeschlossen ist.

##### KinectWpfViewers

Dies sind vordefinierte Klassen des Kinect SDKs. Sie wurden als Beispiel und als Hilfe, beispielsweise beim Einbinden des Skeletts in die View, verwendet.

##### ObjectTrackingVisualizer

Das Projekt ist eine Testapplikation, die zeigt, wann Personen erkannt werden und wann ihrem Skelett eine neue Identität zugewiesen wird. Der Wechsel der ID geschieht beispielsweise dann, wenn die Person den Erkennungsbereich verlässt und ihn danach erneut betritt.

###### PluginDemo

PluginDemo ist ein Beispielprojekt für ein mit MEF erstelltes Plugin und zeigt, wie dieses in die Hauptapplikation eingebunden werden kann.

##### VideoWithWPF

Dies ist eine Applikation, welche ein Video abspielt. Sie wurde benötigt, um feststellen zu können, wie flüssig Videos in einer WPF-Applikation auf der Videowall laufen.

##### WizardOfOzTest

Die WizardOfOzTest-Applikation wurde für den Wizard of Oz Test zu Beginn der Implementation (TODO: ref realisierung und test, wizard of oz test) verwendet.

1. <http://www.microsoft.com/en-us/kinectforwindows/> [↑](#footnote-ref-1)
2. <http://openni.org/> [↑](#footnote-ref-2)
3. <http://www.primesense.com/Nite/> [↑](#footnote-ref-3)
4. <http://www.primesense.com/> [↑](#footnote-ref-4)
5. <http://openkinect.org/wiki/Main_Page> [↑](#footnote-ref-5)
6. <https://github.com/OpenKinect/libfreenect> [↑](#footnote-ref-6)
7. <http://www.cs.dartmouth.edu/~cs104/BodyPartRecognition.pdf> [↑](#footnote-ref-7)
8. [www.imagemagick.org](http://www.imagemagick.org) [↑](#footnote-ref-8)
9. <http://unity.codeplex.com/> [↑](#footnote-ref-9)
10. <http://mef.codeplex.com/> [↑](#footnote-ref-10)
11. |  |  |
    | --- | --- |
    | [microsoft12.1] | Microsoft Corporation, Documentation for MEF,  <http://mef.codeplex.com/documentation>  letzter Zugriff: 22.05.2012 |

    [↑](#footnote-ref-11)
12. <http://msdn.microsoft.com/en-us/library/ms753391.aspx#two_trees> [↑](#footnote-ref-12)