## 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 man beachtet, 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 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

Da die Videowall in der Nacht nicht genutzt wird, ist es möglich, die Wall in dieser Zeit auszuschalten. Dadurch kann der Stromverbrauch gesenkt und die Abnutzung der Hardware, speziell der Monitore, verringert werden.

Da um etwa 7.30 Uhr die ersten Personen an der HSR eintreffen, wird der Videowall PC um etwa 7 Uhr hochgefahren. Damit verbleiben noch 30 Minuten für allfällige Updates und den Systemstart.

Falls eine neue Version der Software oder der Plug-Ins existiert, wird ein automatisches Deployment durchgeführt. Die Informationen und Dateien für dieses Deployment werden automatisch von einem Deployment Server (beispielsweise Team Foundation Server[[9]](#footnote-9)) heruntergeladen. Sollte beim Deployment etwas schief gehen, wird der Videowall PC wieder heruntergefahren.

Falls der PC nach dem automatischen Deployment noch läuft, wird die Videowall Applikation gestartet und die Plug-Ins werden geladen. Sobald dies fertig ist, werden die Bildschirme der Videowall um etwa 7.30 Uhr eingeschaltet. Die Videowall läuft dann den ganzen Tag lang ohne Unterbruch. Sollte ein Fehler auftreten, wird zuerst versucht, die Applikation neu zu starten. Falls dies fehlschlägt, wird die Videowall heruntergefahren.

Sobald die Videowall nicht mehr gebraucht wird (ca. um 20 Uhr), so wird der Videowall PC heruntergefahren und die Bildschirme ausgeschaltet.

Sollte zu irgendeinem Zeitpunkt ein Fehler auftreten, so wird automatisch ein Mail an die Verantwortlichen generiert mit dem Log und einem Stack Trace, um herauszufinden, um für einen Fehler es sich handelt. Zusätzlich wird die Videowall von einem externen Tool (Bsp. Nagios[[10]](#footnote-10)) überwacht, die ebenfalls die Verantwortlichen bei einem Problem notifiziert.

Abbildung - Betrieb der Videowall

### Lebenszyklus der Applikation

#### Videowall Hauptapplikation und Framework

Zum Zeitpunkt der Abgabe dieser Bachelorarbeit besteht ein funktionsfähiger Prototyp. Dieser muss für den Produktiven Betrieb aber noch weiterentwickelt werden. Diese Weiterentwicklung wird durch das Institut für Softare an der HSR (IFS) erfolgen.

Wie unter 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 Administration der Inhalte wichtig (CMS), welche ebenfalls umgesetzt werden muss.

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 Videowall und CMS
* November 2012: Erfahrungen sammeln und wichtige Verbesserungen und Anpassungen implementieren

Danach soll eine Weiterentwicklung in einem halbjährlichen Zyklus folgen, jeweils immer nachdem die neuen Bachelor Poster auf der 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

Durch das Plug-in System können Studenten Innovation in die Applikation einfliessen zu lassen. Da die Videowall aber an einem prominenten Ort steht und die HSR direkt repräsentiert, ist es wichtig, dass nicht beliebige Inhalte auf der Videowall publiziert werden (z.B. Gewalt verherrlichende oder erotische Inhalte). Ein weiteres Problem besteht darin, dass durch die Plug-Ins die Stabilität der Videowall negativ beeinträchtigt werden könnte.

Um also qualitativ hochwertige und politisch Korrekte Plug-ins sicherzustellen, ist es notwendig, die interessierten Studenten darauf aufmerksam zu machen. Deshalb wird vorgeschlagen, dass die Studierenden nach der Entwicklung eines Plug-ins zu einem Code Review eingeladen werden, bei dem der Quellcode des Plug-Ins analysiert wird. Zusätzlich müssen die interessierten Studenten mit ihrem Namen bürgen, dass durch ihre Erweiterung keine politisch unkorrekte Inhalte auf der Videowall erscheinen. Sollte dies doch passieren, sind im Vorherein Massnahmen zu definieren, welche bei einer Verletzung der Vorschriften eigeleitet werden.

Abbildung - Initialer Deployment Prozess

### Architektur

#### Systemübersicht

In der Domainanalyse wurden die Tiers grob beschrieben. Da ein begrenzter Zeitrahmen zur Verfügung stand, wurde das Hauptaugenmerk auf die „HSR Videowall mit Kinect“ gelegt. Die weiteren Tiers wurden nicht implementiert.



Abbildung - Systemübersicht

#### 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. Das nächste Kapitel I.1.3.3 Beschreibung der Assemblies beschreibt die Umsetzung der jeweiligen Schichten und deren Inhalte im Detail.



Abbildung - Architektur Diagramm

Es wurde diskutiert, ob zwischen den Layern ViewModels und Services 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

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

##### Interfaces

Die Interfaces werden von Klassen, welche als Plug-in dienen sollen, implementiert. Die Interfaces definieren hierfür Elemente welches jedes Plug-in zu Verfügung stellen muss und definieren noch weitere Interfaces, welche für die Nutzung weiterer Funktionalitäten genutzt werden kann (z.B. Zugriff aus Skelett-Daten).

##### 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 dadurch 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[[11]](#footnote-11) 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 man beim Unity Container die Mock Objekte registriert.

#### 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)[[12]](#footnote-12) an. Technische Details dazu können in der MEF-Dokumentation[[13]](#footnote-13) 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 (HSR Videowall Applikation) definierten Interface (IApp) implementiert, für den Export aus.
* Das Framework (HSR 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 in der HSR Videowall Applikation (Framework) befindet, importiert über das Attribut [Import] die Klasse, die das Interface IApp implementieren und sich in einem bestimmten Ordner (Directory) befinden

Der Ausdruck [Import] im Framework (HSR 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 MEF14 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 Extensions 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 HSR 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 die Extension weitere Services (IVideoWallService) anfordern.

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

###### Dynamische Sicht

Nachfolgend ein Sequenzdiagramm, welches den Ablauf, wie die Applikationen vom Framework (HSR Videowall Applikation) geladen und aktiviert werden, veranschaulicht.



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

#### Prozesse und Threads

##### Allgemein

Grundsätzlich macht 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[[14]](#footnote-14):

Fenster neu zeichnen

Maus Event verarbeiten

Tastatur Eingabe verarbeiten

Skelett neu zeichnen

Click auf Button verarbeiten

Handcursor Animation starten

Gui Thread führt 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 verschoben, damit die Verarbeitung weiter seriell ablaufen kann.

Weiter ist es natürlich möglich, dass das .NET Framework oder das Kinect Framework von Microsoft im Hintergrund weitere Threads laufen lässt, diese Events aber schon vom Framework 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 dann ausgelöst, sobald ein neues Skelett bereit ist. Dieser Event tritt bereits im Gui Thread auf und muss dementsprechend nicht dispatched 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 über den Gui Thread ab.

##### Echte Parallele Verarbeitung

Sollte es in der Zukunft nötig sein, mehrere Thread oder Prozesse einzusetzen, wird empfohlen, die Events so nach der parallelen Verarbeitung sofort wieder zu dispatchen. Damit kann weiter ohne (Dead)Locks programmiert werden.

### 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 4 - 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 Bereich (rot) für das Handtracking

Der rote Bereich stellt den Bildschirm dar. Wenn nun der Benutzer seine Hand in der oberen rechten Ecke des roten Bereichs bewegt, so wird diese oben rechts auf dem Bildschirm angezeigt, wie Abbildung 6 - 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) 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 ist es notwendig, dass gewisse Elemente angeklickt werden können, wie zum Beispiel das Menu zum Navigieren oder die Pfeile zum Wechseln 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[[15]](#footnote-15) nach Buttons durchsucht und in einer Liste gespeichert. Sobald sich dann der Handzeiger bewegt wird, wird durch diese Buttons iteriert und untersucht, ob sich der Handzeiger 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



Abbildung - Kurz vor einem Klick auf den Mittagsmenu 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.1.6 Interaktion durch Handtracking) wurde darin umgesetzt. Zudem stellt sie auch die verschiedenen Interfaces zu Verfügungen, welche von den Plug-ins genutzt 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 durch Pfeile kann zum nächsten oder vorherigen navigiert werden.

##### Mittagsmenu Applikation

Die Mittagsmenu Applikation zeigt die aktuellen Menus 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 wurde 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 Masterapplikation 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://msdn.microsoft.com/en-us/vstudio/ff637362.aspx> [↑](#footnote-ref-9)
10. <http://www.nagios.org/> [↑](#footnote-ref-10)
11. <http://unity.codeplex.com/> [↑](#footnote-ref-11)
12. |  |  |
    | --- | --- |
    | [microsoft12.1] | Microsoft Corporation, MEF Community Site,  <http://mef.codeplex.com/>  letzter Zugriff: 22.05.2012 |

    [↑](#footnote-ref-12)
13. |  |  |
    | --- | --- |
    | [microsoft12.2] | Microsoft Corporation, Documentation for MEF,  <http://mef.codeplex.com/documentation>  letzter Zugriff: 22.05.2012 |

    [↑](#footnote-ref-13)
14. <http://msdn.microsoft.com/en-us/library/ms741870> [↑](#footnote-ref-14)
15. <http://msdn.microsoft.com/en-us/library/ms753391.aspx#two_trees> [↑](#footnote-ref-15)