## Betriebskonzept

Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

### Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 25.05.2012 | 1.1 | Erste Version des Dokuments, Administration der Videowall | LE |
| 27.05.2012 | 1.2 | Review Administration der Videowall | CH |
| 27.05.2012 | 1.3 | Review Administration der Videowall | DT |
| 07.06.2012 | 1.1 | Installationsdokumentation | CH |
| 11.06.2012 | 1.2 | Review Installationsanleitung | DT |
| 12.06.2012 | 1.3 | Review | DT |

### Betrieb und Administration der Videowall

Im folgenden Kapitel wird beschrieben, wie die Videowall gewartet werden soll. Dazu gehören einerseits der allgemeine Betrieb sowie andererseits die Administration der Inhalte der Videowall.

Dieses Kapitel beschreibt den aktuellen Stand und muss bei einer Weiterführung des Projektes nochmals überarbeitet und verfeinert werden.

#### Betrieb

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 über eine Zeitschaltuhr 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[[1]](#footnote-1)) heruntergeladen. Sollte beim Deployment etwas schief gehen, wird der Videowall PC wieder heruntergefahren.

Nach dem automatischen Deployment wird die Videowall-Applikation gestartet und die Plug-ins werden geladen. 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. Schlägt dies fehl, so wird die Videowall heruntergefahren.

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

Sollte zu irgendeinem Zeitpunkt ein Fehler auftreten, so wird automatisch ein Mail mit dem Log und einem Stack Trace generiert und an die Verantwortlichen geschickt, um herauszufinden, um für einen Fehler es sich handelt. Zusätzlich wird die Videowall von einem externen Tool (Bsp. Nagios[[2]](#footnote-2)) überwacht, welches bei einem Problem ebenfalls die Verantwortlichen benachrichtigt.

Abbildung - Betrieb der Videowall

#### Datenadministration

Bis zum Ende des Projektes sind folgende zu verwaltende Inhalte vorgesehen:

##### Plug-ins

Plug-ins sind Applikationen, die auf der Videowall dargestellt werden können und müssen verwaltet werden.

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. gewaltverherrlichende oder erotische Inhalte). Ein weitere Problematik 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 auf gewisse Restriktionen und Regeln aufmerksam zu machen. Es 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 Studenten mit ihrem Namen dafür bürgen, dass durch ihre Erweiterung keine politisch unkorrekten Inhalte auf der Videowall erscheinen. Sollte dies doch passieren, sind im Vorhinein Massnahmen zu definieren, welche bei einer Verletzung der Vorschriften eingeleitet werden.

Folgender Ablauf ist für das Erstellen und Deployen eines Plug-ins denkbar:

Abbildung - Initialer Deployment Prozess

##### Poster-Applikation

Jedes Semester entstehen neue Bachelorposter, welche für die Videowall aufbereitet werden müssen. Es ist noch nicht bestimmt, wer diese Aufgabe übernehmen wird.   
Folgender Ablauf ist denkbar:

1. Die Bachelorposter werden von den Studierenden erstellt.
2. Die Studiengangleiter sind verantwortlich, diese Poster in elektronischer Form entgegenzunehmen und inhaltlich zu kontrollieren.
3. Die Poster werden von den Studiengangleitern dem Sekretariat der HSR übergeben.
4. Das Sekretariat pflegt die Inhalte über ein CMS Interface in die Videowall-Applikation ein.
5. Berichtigungen können dem Sekretariat gemeldet werden, welches die Korrekturen ins CMS Interface einpflegt.

##### SV Group Mensa Mittagsmenu

Das Mittagsmenu ist an den Wochentagen auf der Internetseite der SV Group verfügbar (Typo3 CMS der SV Group) (siehe Unterkapitel I.1.3.3.2 Mittagsmenu). Leider bietet die SV Group keine Schnittstelle ausser der HTTP/HTML-Version für das Menu an. Um das Mittagsmenu trotzdem aktuell zu halten, wird der Menuplan beim Start der Applikation in HTML heruntergeladen, die nötigen Informationen herausgelesen und in einer Form zwischengespeichert, die sich für WPF eignet. Die HTTP/HTML-Schnittstelle ist technisch gesehen labil. Eine Design-Änderung an der Website der SV Group könnte dazu führen, dass das Menu nicht mehr richtig eingelesen werden kann. Daher wurden spezifisch für das Mensamenu Unit Tests geschrieben. Sie bieten dem Entwickler die Möglichkeit, die Schnittstelle einfach und schnell zu testen. Dazu wird eine aktuelle Version des HTML heruntergeladen und im Unit Test eingebunden.

Um das Menu zu aktualisieren, gibt es zwei Möglichkeiten:

* Die Applikation wird neu gestartet.
* Es wird ein Cronjob oder ein (Dispatcher-)Timer mit hohem Aktualisierungsintervall programmiert, der beispielsweise alle zwei Stunden das Mittagsmenu aktualisiert.

#### Datenverwaltung der Plug-ins

Jedes Plug-in kann über eigene Daten verfügen. Die Poster-Applikation benötigt beispielsweise die Bilder der anzuzeigenden Poster. Das Framework könnte ein Interface zur Verfügung stellen, über welches die Daten der Plug-ins verwaltet werden können. Die Entwickler eines Plug-ins definieren die Objekte, welche verwaltet werden sollen. Das Framework generiert dann automatisch eine Benutzeroberfläche für deren Bearbeitung.

Alternativ könnte jedes Plug-ins eine eigene Administrationsoberfläche anbieten. Da bei dieser Variante Funktionen (beispielsweise das Speichern der Daten in einer Datenbank) redundant programmiert werden müssten und die Bedienung nicht einheitlich wäre, ist sie jedoch weniger geeignet.

#### Administrationsoberfläche

Für die Administrationsoberfläche der Inhalte gibt es verschiedene Möglichkeiten.

##### Evaluation der verschiedenen Möglichkeiten

Die drei wichtigsten sind in den nachfolgenden Unterkapiteln kurz beschrieben.

###### Administration über Typo3 CMS

Das Sekretariat der HSR[[3]](#footnote-3) arbeitet bereits mit einem Typo3 CMS[[4]](#footnote-4). Eine Integration der Administration der Videowall in dieses System wäre eine Option. Dabei könnte auf zwei Arten vorgegangen werden:

Typo3 Extension mit Typo3 Datenbank

Es wird eine Typo3 Extension[[5]](#footnote-5) für den Administrationsbereich entwickelt. Diese Applikation wird sehr einfach gehalten und zeichnet sich vor allem durch XML Konfigurationen aus, die das Datenbankschema beschreiben. Durch die Installation der Extension im Typo3 wird die Datenbank automatisch erzeugt und die Inhalte können sofort über den Administrationsbereich von Typo3 bearbeitet werden. Die grafische Oberfläche des Administrationsbereichs wird automatisch vom Framework generiert und sieht so wie eine gewöhnliche Typo3 Extension aus. Gewisse Standardfunktionalitäten wie die Archivierungsoption oder das Ausblenden von einzelnen Datensätzen werden ebenfalls durch das Framework angeboten.

Der Nachteil an dieser Lösungsvariante ist, dass die Daten von der Typo3 Datenbank zur Videowall migriert werden müssen. Eine Synchronisation des unidirektionalen Informationsflusses von der Typo3 Datenbank zur Videowall kann aber auf einfache Art per Cronjob (z.B. alle 15 Minuten) eingerichtet werden.

Diese Variante eignet sich gut, wenn alle Personen, die an der Videowall etwas ändern müssen, Zugriff auf Typo3 haben. Speziell für das Sekretariat ist diese Art von Interface einfach zu bedienen, es wird bereits täglich benutzt.

Web Interface und Typo3 Extension mit Iframe

Wie in der ersten Lösungsvariante (siehe Unterkapitel I.1.5.5.1.1.1 Typo3 Extension mit Typo3 Datenbank) ist auch hier eine Typo3 Extension vorgesehen. Dieses Mal wird die Extension aber so erstellt, dass nur ein Iframe programmiert wird, das auf einen anderen Web Server verweist. Somit kann die Administrationsoberfläche Typo3-technologieunabhängig entwickelt werden, zum Beispiel mit ASP.NET MVC3.

Die Hauptvorteile dieses Ansatzes sind, dass die Administrationsoberfläche nicht mit Typo3 programmiert werden muss und trotzdem ins Typo3 integriert ist. Die Administrationsoberfläche kann auch ohne Typo3 bearbeitet werden, gegebenenfalls mit einem SSO. Die Implementierung dieser Lösungsvariante ist allerdings etwas zeitaufwändiger.

###### Administration über Web Server

Ähnlich wie in der zweiten Typo3-Lösungsvariante I.1.5.5.1.1.2 Web Interface und Typo3 Extension mit Iframe beschrieben ist, wird bei dieser Lösung auf einem Web Server (z.B. mit ASP.NET MVC3) eine Administrationsoberfläche entwickelt, die gegebenenfalls mit dem SSO der HSR gekoppelt wird. Als Transportprotokoll dient HTTPS/HTML.

Die Vorteile liegen darin, dass das System klar von anderen Applikationen abgegrenzt ist. Auch ist es einfach möglich, eine mobile Applikation mit HTML5 zu entwickeln. Als Nachteil ist jedoch aufzuführen, dass ohne Typo3 Extension die Benutzer auf eine separate URL zugreifen müssen und ihnen das System nicht sofort bekannt vorkommt. Für diese Variante könnte statt ASP.NET auch Silverlight eingesetzt werden.

###### Administration über WPF-Applikation

In dieser Variante geht es darum, einen WPF Client zu schreiben, mit dem die Inhalte bearbeitet werden können. Als Transportprotokoll würde WCF eingesetzt werden.

Bei dieser Variante können grosse Teile aus den Daten- und Serviceschichten (TODO: Ref Dokument Entwurf, Kapitel Logische Sicht) der bestehenden Software wiederverwendet werden, was ein Vorteil ist. Nachteilig ist, dass durch WPF die Plattform eingeschränkt wird und die Entwicklung einer mobilen Applikation so nicht möglich ist.

Diese Variante wird nicht empfohlen.

##### Umsetzungsempfehlung

Das Bachelorteam hätte bei genügend Zeit die Lösungsvariante, welche im Unterkapitel I.1.5.5.1.1.2 Web Interface und Typo3 Extension mit Iframe beschrieben ist, für die Videowall umgesetzt.

Vor der Umsetzung ist die ausgewählte Lösungsmöglichkeit mit der Kommunikationsstelle der HSR zu validieren.

### Installationsdokumentation

#### Dokumentation für Administrator

Damit die Applikation verwendet werden kann wird folgendes benötigt:

* Kompilierte Version des Projektes
* .NET, Kinect Runtime
* Internetverbindung

Die kompilierten Dateien müssen als erstes in den gewünschten Zielordner kopiert werden.   
Weiter können verschiedene Plug-ins zur Applikation hinzugefügt werden. Damit dies möglich ist, muss die Ordnerstruktur nach der Anleitung im Kapitel I.1.8.3 Plug-ins aufgebaut werden.

#### Dokumentation für Entwickler

Damit die Applikation weiterentwickeln werden kann, muss ein SVN Checkout auf dem gesamten SVP Repository durchgeführt werden. Dazu wird ein SVN Tool wie z.B. TortoiseSVN benötigt (siehe Tools TODO).

Zudem werden folgende Werkzeuge benötigt:

* Visual Studio 10
* Expression Blend
* ReSharper
* Kinect for Windows SDK
* NuGet

Die Konfiguration der Applikation findet in der app.config-Datei statt. Wie die Konfiguration angepasst werden kann, wird im Kapitel I.1.8.4 Konfigurationsdatei erläutert.   
Weiter können der Applikation Plug-ins hinzugefügt werden, dies wird im nachfolgenden Kapitel (I.1.8.3 Plug-ins) aufgezeigt.

#### Plug-ins

Im app.config-File kann der Pfad zum Ordner, welcher die Plug-ins enthalten soll, konfiguriert werden. Wie dieser Pfad gesetzt und angepasst werden kann, wird im Unterkapitel TODO pFad zu den Plugins erläutert.

Für jede Plug-in-Applikation, die man auf der Videowall anzeigen möchte, muss ein entsprechender Unterordner im Plug-ins Ordner (todo pfad zu den plugins) erstellt werden, in welchem dann die Files des Plug-ins gespeichert werden. Ein solcher Ordner darf nie leer sein, da sonst beim Starten der Videowall-Applikation eine entsprechende Fehlermeldung angezeigt wird.   
Bestimmte Plug-in-Applikationen benötigen zusätzliche Dateien wie beispielsweise Bilder. Diese müssen in einem Unterordner mit dem Namen „Files“ im Ordner des Plug-ins abgelegt werden. Fehlt dieser Ordner, so wird eine entsprechende Fehlermeldung beim Starten der Hauptapplikation angezeigt.

#### Konfigurationsdatei

Die Einstellungen, welche in der Konfigurationsdatei getätigt werden können, sind nachfolgend aufgezeigt.

##### Konfigurationssektionen

Im Abschnitt configSections können verschiedene Konfigurationssektionen definiert werden.

  <configSections>

    <section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection, Microsoft.Practices.Unity.Configuration"/>

  </configSections>

Abbildung - app.config, Konfigurationssektionen

##### Unity

In der Konfigurationssektion Unity werden die Namespaces angegeben, in denen die Klassen gesucht werden sollen, welche über Unity instanziiert werden sollen. Zu den Namespaces muss weiter angegeben werden, in welchem Assembly diese sich befinden.

    <namespace name="VideoWall.Data.Kinect.Implementation"/>

    <namespace name="VideoWall.Data.Kinect.Interfaces"/>

    <assembly name="VideoWall.Data"/>

    <namespace name="VideoWall.ServiceModels.Apps.Implementation"/>

    <namespace name="VideoWall.ServiceModels.Apps.Interfaces"/>

    <namespace name="VideoWall.ServiceModels.DemoMode.Implementation"/>

    <namespace name="VideoWall.ServiceModels.DemoMode.Interfaces"/>

    <namespace name="VideoWall.ServiceModels.HandCursor.Implementation"/>

    <namespace name="VideoWall.ServiceModels.HandCursor.Interfaces"/>

    <namespace name="VideoWall.ServiceModels.Player.Implementation"/>

    <namespace name="VideoWall.ServiceModels.Player.Interfaces"/>

    <assembly name="VideoWall.ServiceModels"/>

    <namespace name="VideoWall.ViewModels"/>

    <namespace name="VideoWall.ServiceModels.Apps"/>

    <namespace name="VideoWall.ViewModels.Cursor"/>

    <namespace name="VideoWall.ViewModels.HitButton"/>

    <namespace name="VideoWall.ViewModels.Menu"/>

    <namespace name="VideoWall.ViewModels.Skeletons"/>

    <assembly name="VideoWall.ViewModels"/>

Abbildung 37 - app.config, Sektion Unity, Namespaces&Assemblies

###### Interfaces auf konkrete Klassen mappen

Weiter können in der Sektion Unity mithilfe des Tags <container> Interfaces auf konkrete Klassen gemappt werden. Dies wird wie folgt angegeben:

        <type type="IDemoModeService" mapTo="DemoModeService"/>

        <type type="IAppController" mapTo="AppController"/>

        <type type="IHandCursorPositionCalculator" mapTo="HandCursorPositionCalculator"/>

        <type type="IPlayer" mapTo="Player"/>

Abbildung - app.config, Sektion Unity, Mapping von Interfaces auf Klassen

###### Skelettdaten

Der *ISkeletonReader* dient dazu, die Skelettdaten zu Lesen.   
Der *KinectSkeletonReader* wird verwendet, wenn Kinect angeschlossen ist und die Videowall normal betrieben wird.   
Für Testzwecke werden der *FileSkeletonReader* und der *AutoPlayFileSkeletonReader*, welche mit Kinect aufgenommene Skelettdaten abspielen, angeboten. Der *FileSkeletonReader* spielt das File einmal, der *AutoPlayFileSleketonReader* spielt das File immer wieder von Neuem ab. Das abzuspielende File mit den Skelettdaten kann über den Typ *KinectReplayFile* definiert werden, was im Unterkapitel I.1.5.3.9 KinectReplayFile erläutert wird.

        <!-- FileSkeletonReader -->

        <!-- AutoPlayFileSkeletonReader -->

        <!-- KinectSkeletonReader -->

        <type type="ISkeletonReader" mapTo="KinectSkeletonReader">

          <lifetime type="singleton"/>

        </type>

Abbildung 39 - app.config, Sektion Unity, Mapping für ISkeletonReader

###### Cursor

Ist der Kinect Sensor am Computer angeschlossen, so ist für das Mapping vorzugsweise das *KinectCursorViewModel* zu verwenden. Der Cursor, welcher in der Applikation als kleine Hand dargestellt wird, kann so durch Handbewegungen vor dem Kinect Sensor gesteuert werden. Ist das Mapping für den *ISkeletonReader* (siehe Unterkapitel I.1.8.4.2.2 Skelettdaten) auf *FileSkeletonReader* oder *AutoPlayFileSkeletonReader* gesetzt, so wird der Handcursor durch das Skelett im abzuspielenden File gesteuert.

Wird das Schlüsselwort *MouseCursorViewModel* verwendet, so lässt sich die Applikation mit der Maus bedienen. Trotzdem wird auch hier eine Hand angezeigt.

        <!-- KinectCursorViewModel -->

        <!-- MouseCursorViewModel -->

        <type type="ICursorViewModel" mapTo="MouseCursorViewModel">

          <lifetime type="singleton"/>

        </type>

Abbildung - app.config, Sektion Unity, Mapping für ICursorViewModel

##### Padding

Damit die Applikation angenehm mit Kinect zu bedienen ist, wurde einer Interaktionszone für die Hand definiert. Das Prinzip dieser Zone wird im Kapitel (TODO. Entwurf, Handtracking) genauer erläutert. Die rote Fläche in der Abbildung 41 - Interaktionszone ist auf den Arm des Nutzers ausgerichtet und stellt die eigentliche Interaktionszone dar. Die schwarz umrahmte Fläche stellt den Erkennungsbereich von Kinect dar. Das *RelativePadding* wird verwendet, um diese rote Zone zu definieren. Möchte man die Position und Grösse der Zone verändern, so kann dies durch Anpassungen an den Werten der aufgelisteten Schlüsselwörter *left*, *top*, *right* und *bottom* getan werden. Welchen Abstand diese Schlüsselwörter verändern ist in der Abbildung 41 - Interaktionszone ersichtlich.

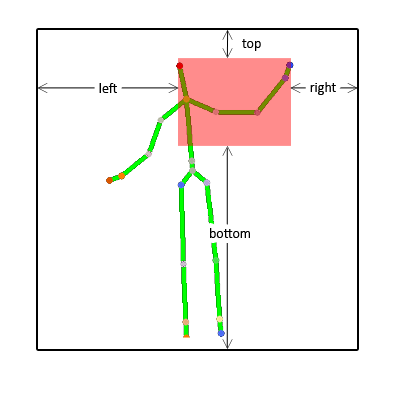


Abbildung 41 - Interaktionszone

        <type type="RelativePadding" mapTo="RelativePadding">

          <constructor>

            <param name="left" value="0.45"/>

            <param name="top" value="0.1"/>

            <param name="right" value="0.3"/>

            <param name="bottom" value="0.49"/>

          </constructor>

        </type>

Abbildung 42 - app.config, Sektion Unity, Padding

###### Pfad zu den Plug-ins

Die Applikation kann durch Plug-ins dynamisch erweitert werden. Diese müssen in einem bestimmten Ordner abgelegt werden. Der Parameter *extensionsDirectoryPath* muss auf den gewünschten Pfad gesetzt werden.

        <type type="ExtensionsConfig" mapTo="ExtensionsConfig">

          <constructor>

            <param name="extensionsDirectoryPath" value="../../../Extensions"/>

          </constructor>

        </type>

Abbildung 43 - app.config, Sektion Unity, Pfad zu den Plug-ins

###### Demomodus

Der Demomodus wird aktiv, wenn Kinect keinen Nutzer erkennt. Einstellungen für den Demomodus können im Bereich *IDemoModeConfig* vorgenommen werden.

        <type type="IDemoModeConfig" mapTo="DemoModeConfig">

          <constructor>

            <param name="backgroundColors">

              <array>

                <value value="#ff0065a3"/>

                <value value="#ff6e1c50"/>

                <value value="#ff548c86"/>

                <value value="#ff7b6951"/>

                <value value="#ff00738d"/>

                <value value="#ffbabd5d"/>

              </array>

            </param>

            <param name="skeletonCheckTimeSpan" value="00:00:00.01"/>

            <param name="changeAppTimeSpan" value="00:00:05"/>

            <param name="fromActiveToDemoModeTimeSpan" value="00:00:15"/>

            <param name="countdownTimeSpan" value="00:00:04.999"/>

            <param name="skeletonTrackingTimeoutTimeSpan" value="00:00:00.5"/>

          </constructor>

        </type>

Abbildung 44 - app.config, Sektion Unity, Demomodus

Als Werte für den Parameter *backgroundColors* können Farben für den Hintergrund des Demomodus angegeben werden.

Des Weiteren können die Werte der verschiedenen Timer angepasst werden.   
Der *skeletonCheckTimeSpan* gibt an, nach wie vielen Millisekunden erneut geprüft wird, ob gerade ein Nutzer erkannt wurde.  
Über den *changeAppTimeSpan* kann die Zeitspanne bis zum Wechsel des Teaser-Textes und der Hintergrundfarbe im aktiven Demomodus definiert werden.  
Wenn die Applikation aktiv ist aber kein Skelett mehr erkennt, wechselt sie nach Ablauf des *fromActiveToDemoModeTimeSpan* in den Demomodus.   
Wird im Demomodus ein Nutzer erkannt, so wird ein Countdownzähler angezeigt, welcher den Übergang in den Interaktionsmodus andeutet. Die Länge dieses Zählers kann über den *countdownTimeSpan* verändert werden.   
Nachdem ein Nutzer von Kinect erkannt wurde, wird immer wieder geprüft, ob der Nutzer immer noch erkannt wird. Die Zeitspanne bis zur nächsten Überprüfung wird über den *skeletonTrackingTimeoutTimeSpan* angegeben.

###### Singelton

Von bestimmten Klassen darf es immer nur eine Instanz geben, welche dann an mehreren Orten verwendet werden kann.   
Die Klasse Player muss ein Singleton sein: Das Skelett des Kinect Players wird zum einen beim Übergang vom Demomodus in den Interaktionsmodus und schliesslich im Interaktionsmodus selbst angezeigt. Die Problematik mit Kinect ist, dass das Gerät nur von einer Instanz angesteuert werden kann. Bestünden nun zwei verschiedene Skelette, die das Kinect ansteuern und Daten vom Gerät verlangen, würde das in einem Fehler enden.   
Im Falle des MenuViewModels darf nur eine Instanz existieren, welche alle Plug-ins im Menu aufzeigt und den Inhalte des aktuell angewählten auf der Videowall anzeigt.

        <type type="Player" mapTo="Player">

          <lifetime type="singleton"/>

        </type>

        <type type="MenuViewModel" mapTo="MenuViewModel">

          <lifetime type="singleton"/>

        </type>

Abbildung 45 - app.config, Sektion Unity, Singleton

###### KinectReplayFile

Wird im app.config beim Mapping auf den ISkeletonReader (siehe I.1.8.4.2.2 Skelettdaten) der *FileSkeletonReader* oder der *AutoPlayFileSkeletonReader* verwendet, so muss ein *KinectReplayFile* angegeben werden. Dieses beinhaltet Daten zu Skelettbewegungen. Der Pfad zu dieser Datei kann an folgender Stelle beim Schlüsselwort *path* angegeben werden:

 <type type="KinectReplayFile" mapTo="KinectReplayFile">

          <lifetime type="singleton"/>

          <constructor>

            <param name="path" value="..\..\..\..\..\..\kinect\_records\20120312\_lukas\\_1.replay"/>

          </constructor>

        </type>

Abbildung 46 - app.config, Sektion Unity, KinectReplayFile

Der Pfad zum Replay-File ist relativ anzugeben.

##### Runtime Version

Über das Schlüsselwort *runtime* können die Pfade zu den neuen Assembly-Versionen und deren Speicherort angegeben werden. Diese Angaben werden benötigt um Assembly-Versionskonflikte zu vermeiden.

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="Microsoft.Kinect" publicKeyToken="31BF3856AD364E35" culture="neutral"/>

<bindingRedirect oldVersion="1.0.0.0-1.5.0.0" newVersion="1.5.0.0"/>

</dependentAssembly>

</assemblyBinding>

</runtime>

Abbildung - app.config, Runtime

1. <http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx> [↑](#footnote-ref-1)
2. <http://www.nagios.org/> [↑](#footnote-ref-2)
3. <http://www.hsr.ch/typo3/index.php> [↑](#footnote-ref-3)
4. <http://typo3.org> [↑](#footnote-ref-4)
5. <http://typo3.org/extensions/repository/> [↑](#footnote-ref-5)