
ASGAL
Informatik GmbH

# .NET Application Container

## Bachelor Arbeit

## Abteilung Informatik

## HSR – Hochschule für Technik Rapperswil

**Frühjahrssemester 2010**

**Autor**
Andrea Berweger
aberwege@hsr.ch

**Betreuer**
Prof. Hansjörg Huser
hhuser@hsr.ch

**Projektpartner**
Asgal Informatik GmbH
Thomas Gall
thomas.gall@asgal.ch

**Experte**
Stefan Zettel

**Gegenleser**
Markus Stolze

Version: 17. Juni 2010

**Abstract**
Ein Kunde der ASGAL Informatik GmbH hat ein neues Projekt lanciert, bei dem vorgesehen ist, dass mehrere von einander unabhängige Funktionen in einer Applikation vereint werden. Um dieses Problem zu lösen, wird ein Basissystem in Form eines kleinen Frameworks entwickelt. Das Basissystem stellt eine Umgebung dar, die mittels Komponenten um Funktionen erweitert werden kann. Für das Basissystem besteht die Möglichkeit, dass es auch für weitere Kunden, bzw. Projekte eingesetzt werden kann. Das Framework ist klein und damit einfach gehalten.

Aus diesen Anforderungen ist der „.NET Application Container“entstanden. Er besteht hauptsächlich aus einem User Interface, dass durch die Komponenten verwendet werden kann und sorgt dafür, dass Fehler einer Komponente isoliert behandelt und geloggt werden. Der Container stellt ein Subsystem zur Verfügung mit dem Authentisierung und Autorisierung für die Komponenten ermöglicht wird. Zum Bearbeiten der Berechtigungen existiert eine Komponente für den Container.

Der Container wird mit den neusten Technologien umgesetzt. Mit anderen Worten kommt das .NET Framework 4.0 von Microsoft zum Einsatz. Im Bereich User Interface wird die Windows Presentation Foundation (WPF) mit Model-View-ViewModel- (MVVM) und Command-Pattern sowie das Ribbon-Menü eingesetzt. Der Zugriff auf die Datenbank wird mittels Entity Framework (EF) realisiert und das dynamische Laden der Komponenten wird durch das Managed Extensibility Framework unterstützt.

# Inhaltsverzeichnis

# 1. Einführung

## 1.1. Zweck

Ein neues Projekt soll die Verarbeitung verschiedener Prozesse optimieren und vereinfachen. Da die Prozesse von einander unabhängig sind, und verschiedene Personen unterschiedliche Prozesse verarbeiten, spricht dies für eine modulare Applikation. Die Applikation kann somit auf verschiedenen Arbeitsstationen mit unterschiedlichen Funktionen, je nach Anforderung, mit minimalem Aufwand realisiert werden. Die verschiedenen Funktionen werden mit einem Berechtigungssystem geschützt, damit die Applikation, bzw. die verschiedenen Funktionen nicht von jedem gelesen und ausgeführt werden können. Das Design resultierte in Form des .NET Applikcation Container, der mit Hilfe von Komponenten (Plugins) erweitert werden kann.

Der .NET Application Container unterstützt die Entwicklung in der ASGAL Informatik GmbH für Projekte und ermöglicht zudem, dass Projekte in Zukunft effizienter entwickelt und erweitert werden können. Eine Portierung der aktuellen Projekte der ASGAL Informatik GmbH ist bis auf weiteres nicht geplant, da der Aufwand für die Portierung zu gross ist. Begründet ist das dadurch, da die aktuellen Projekte mit dem .NET Framework 2.0 entwickelt sind und eine objektorientiert Programmierung nur bedingt besteht. Die Projekte müssten somit neu entwickelt werden.

Der implizite Zweck des Projektes bringt der ASGAL Informatik GmbH neue Erkenntnisse und Knowhow zu den aktuellen Technologien und Möglichkeiten des .NET Framework. Dabei werden folgende Programmierparadigmen untersucht und eingesetzt:

- Windows Presentation Foundation (WPF)
- Model-View-ViewModel (MVVM)
- Entity Framework (EF)
- Managed Extensibility Framework (MEF)
- Office 2007 Ribbon Menü

## 1.2. Übersicht

Dieses Dokument umschreibt Design Entscheidungen sowie die funktionalen und nicht funktionalen Anforderungen für den .NET Application Container. Um den Text in diesem Dokument einfach und übersichtlich zu halten, wird anstelle von „.NET Application Container„nur „Container“geschrieben und Komponenten, die von Container geladen werden, werden als „Plugins“deklariert.

Das Dokument wurde mit Latex erstellt und sollte die Formatierung an gewissen Stellen Fehler oder Unschönheiten aufweisen, wird um Verständnis gebeten, da Latex manchmal auch Schwierigkeiten hat die Formatierung sauber zu gestalten.

# 2. .NET Application Container

## 2.1. Allgemeine Beschreibung

Der Container stellt ein minimales Basissystem in Form eines Frameworks dar. Der Hauptbestandteil des Containers liegt im Frontend, sprich dem User Interface, dass bei der Entwicklung innerhalb der ASGAL Informatik GmbH für ein einheitliches Design sorgt. Zukünftige Projekte der ASGAL Informatik GmbH werden auf diese Weise immer mit einem ähnlichen User Interface Aufbau entwickelt. Der Container bietet einem Plugin insgesamt fünf Bereiche im User Interface die genutzt werden können. Diese Bereiche sind somit einheitlich definiert und bilden ein immer wiederkehrendes Bild. Neue Applikationen können als Plugins in Form von Programmbibliotheken (DLL's) ausgeliefert werden, die Grundapplikation bleibt immer die selbe. Der Container bietet zudem ein Subsystem an, dass genutzt werden kann um Benutzer zu identifizieren und deren Berechtigungen abzufragen. Die dafür benötigten Daten werden in einer Datenbank gespeichert.
Der Container behandelt allfällige nicht behandelte Fehler von Plugins, isoliert sie, und sorgt dafür, dass das Plugin neu geladen wird.

### 2.1.1. Controller

Der Contoller ist die implizite Basiseinheit des Containers, der die gesamten Abläufe koordiniert und dafür sorgt, dass die Plugins geladen werden und diese mit dem User Interface kommunizieren können.

- Startet Container
- Lädt alle verfügbaren Plugins
- Lädt den Exception-Handler

Optional: Der Controller unterstützt eine Schnittstelle zur Kommunikation zwischen den Plugins. Optional: Der Controller bietet Funktionen an um das optische aussehen des User Interfaces zu verändern (Anwendung von Themes).

### 2.1.2. Komponenten Schnittstelle

- Dynamisches Einbinden von Plugins
- Plugins als Klassenbibliotheken (DLL's)
- Erkennung der Plugins über definierte Schnittstellen

Das komponentenbasierende Konzept bringt die Möglichkeit, kundenspezifische Anforderungen und Wünsche möglichst einfach zu gewährleisten. Der Kunde hat so auch die Möglichkeit verschiedene Projekte die von der ASGAL Informatik GmbH entwickelt werden in ein und der selben Applikation zu betreiben. Dies bietet maximalen Komfort bei Personen die mehrere Applikation (aus verschiedenen Projekten) gleichzeitig nutzen.

### 2.1.3. Datenbank Mapping

Der Container bietet eine einheitliche Schnittstelle an eine Datenbank. Dabei ist abzuklären, ob es mit Hilfe des Entity-Framework realisiert werden kann, oder ob es mittels einer speziellen Implementierung umgesetzt wird. Als Beispiel für diese Umsetzung hat man in der ASGAL Informatik GmbH bereits vor dem Projekt einen Prototypen eines Datenbankmanagers entwickelt, der mittel Dataset und OleDb den Zugriff auf eine Datenbank kapselt.

### 2.1.4. User Interface

- Corporate Design Richtlinie (Der Grundaufbau einer Applikation innerhalb der ASGAL Informatik wird durch den Container definiert.)
- Mögliche Darstellung für das GUI

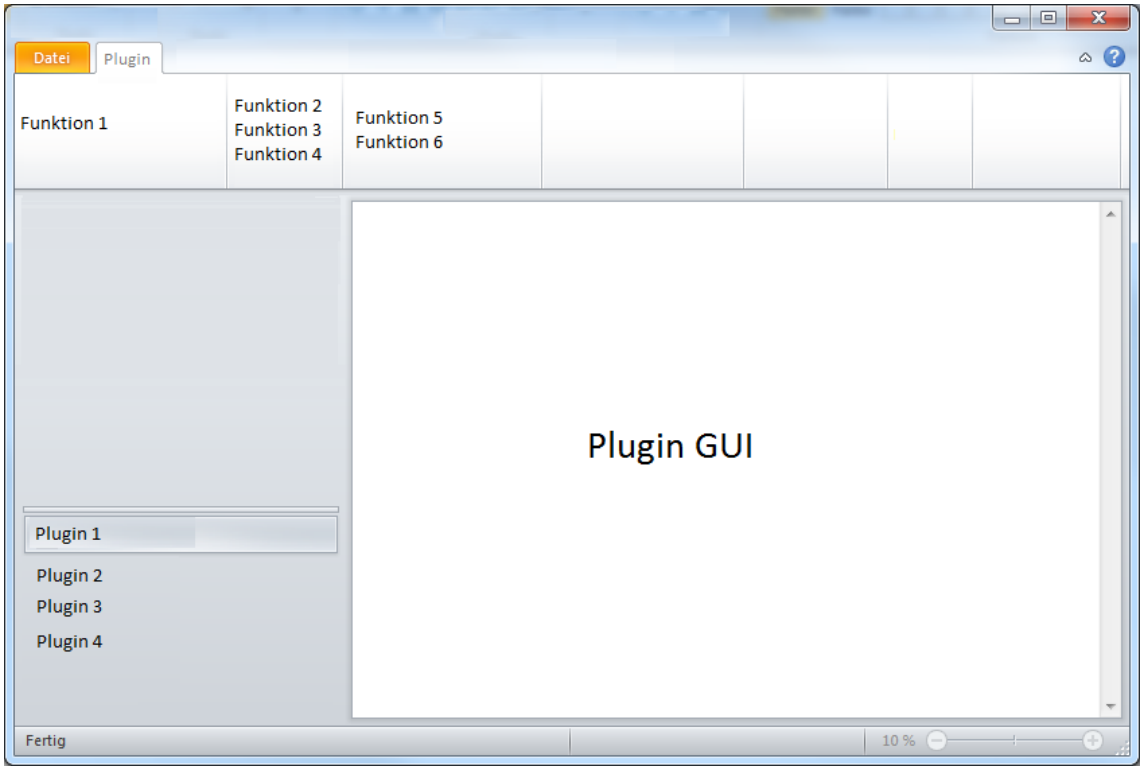


Abbildung 2.1.: Container User Interface Konzept

- Am oberen Rand eine Menüliste, ähnlich der Ribbon-Liste von Office 2007/2010
- Am linken Rand ein Menü, dass alle geladenen Plugins zugreifbar macht
- Die restlichen Bereiche dient für die Darstellung des Plugin spezifischen Inhalts

## 2.2. Analyse

### 2.2.1. Funktionale Anforderungen

#### 2.2.1.1. Brief Use Cases

**UC01: Container laden**

Der Benutzer startet die Applikation. Dadurch wird ein Controller aufgerufen, der mit Hilfe von UC2, UC3, UC4 die notwendigen Komponenten und Funktionen ladet und dem Benutzer zur Verfügung stellt.

**UC02: User Interface anzeigen**

Der Container lädt das Standard-User Interface, dass ohne aktives Plugin leer ist, d.h. keinen Inhalt besitzt. Erst durch das aktivieren eines Plugin, kann dieses die Bereiche nutzen um die eigenen Funktionen darzustellen.

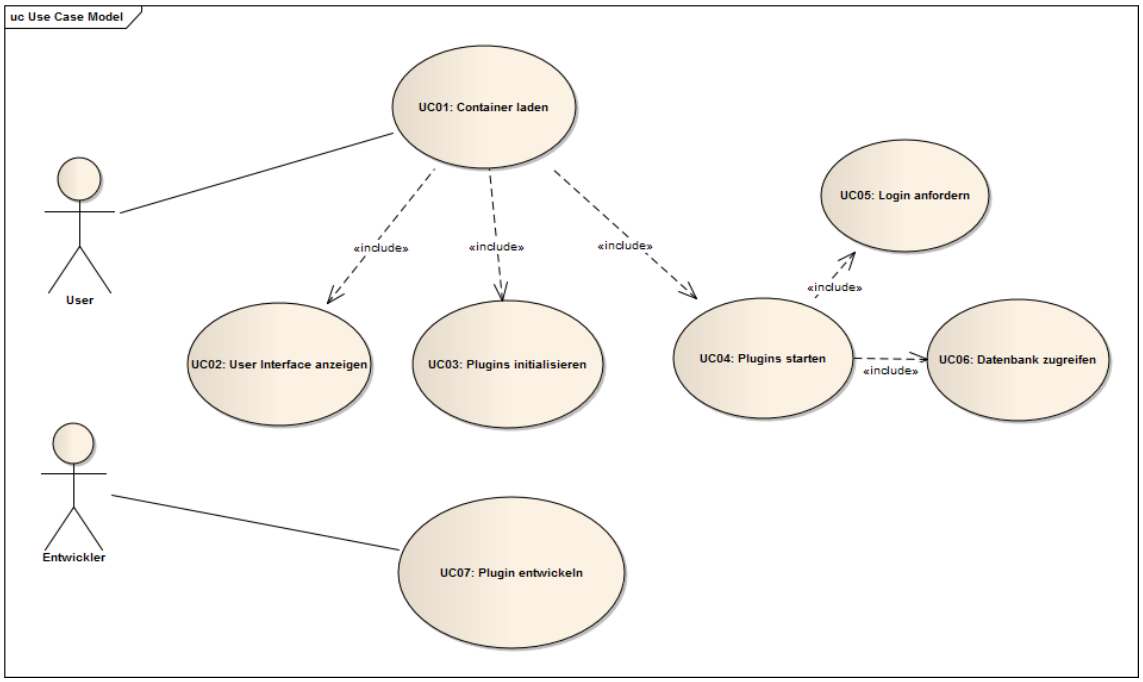


Abbildung 2.2.: Container Use Case Model

### UC03: Plugins initialisieren

Der Controller lädt den Pluginmanager, der alle in einem spezifizierten Ordner hinterlegten Plugins in Form von Klassenbibliotheken (DLL) analysiert und startet (UC04).

### UC04: Plugins starten

Der Container startet alle Plugins. Jedes Plugin kann während des starten Benutzerinformationen beschaffen (UC06, UC07) um dem Container mitzuteilen, ob der aktuelle Benutzer berechtigt ist das Plugin zu nutzen. Ist der Benutzer autorisiert, so wird es in das User Interface des Containers integriert, andernfalls wird es nicht angezeigt. Im Fehlerfall wird es ebenfalls nicht angezeigt und ignoriert.

### UC05: Login anfordern

Das Plugin fordert vom Container eine Liste von bereits gespeicherten Benutzerinformationen. Wird mit dieser Liste kein gültiges Login gefunden, so ist das Plugin selbst dafür verantwortlich, die Benutzerinformationen zu besorgen. Der Container bzw. das Framework stellt aber optional eine Standard Abfrage für Benutzerinformationen zur Verfügung. Nach Abfrage der Benutzerinformationen kann das Plugin die erhaltenen Informationen dem Container übergeben. Dieser erweitert somit seine Benutzerliste.

### UC06: Datenbank zugreifen

Das Plugin benötigt Informationen von einer Datenbank, wobei es selbst für den Zugriff verantwortlich ist. Optional steht eine Klasse zur Verfügung, die den Zugriff kapselt. Validierung von Daten muss ebenfalls durch das Plugin erfolgen.

#### UC07: Plugin entwickeln

Der Entwickler erstellt die benötigte Funktionalität in Form einer Klassenbibliothek (DLL). Die Klassenbibliothek muss eine spezifizierte Schnittstelle aufweisen damit der Container fähig ist sie zu laden. Das entwickelte Plugin wird in einem definierten Ordner hinterlegt, der beim starten des Containers geladen wird (UC03).

### 2.2.1.2. Detailtiere Use Cases

Der Use Case 01 (Container laden) ist der initiale Use Case, der mit dem Starten des Containers beginnt. Durch die Abarbeitung des ersten Use Case werden alle anderen Use Cases, bis auf UC07, der nur für die Entwicklung eines Plugins von Bedeutung ist, durchlaufen. Aus diesem Grund wird der Ablauf unten nochmals detailliert beschrieben.

#### UC01: Container laden

| | |
|---|---|
| Primary Actor | Benutzer |
| Stackholders/Interest | Benutzer möchte eines oder mehrere Funktionen (Plugins) nutzen können. |
| Preconditions | Application Container ist nicht gestartet. |
| Postconditions | Applikation Container ist gestartet und initialisiert. Der Benutzer kann die Funktionen der geladenen Plugins nutzen. |

**Basis Verlauf**

1. Benutzer benötigt eine Funktion eines Plugin.
2. Benutzer startet den Applikation Container.
3. Applikation Container initialisiert den Controller.
4. ≪UC02≫ Der Controller initialisiert das Standard-User Interface, zeigt es aber noch nicht an.
5. ≪UC03≫ Der Controller initialisiert den Pluginmanager, welcher wiederum den Ordner mit den Plugins durchsucht und analysiert.
6. ≪UC04≫ Der Pluginmanager startet jedes Plugin und integriert es in das User Interface des Containers.
7. Der Controller macht das User Interface des Containers sichtbar.
8. Der Benutzer kann die Funktionen der vorhandenen Plugins nutzen.

**Alternativer Verlauf**

a. Systemfehler tritt auf (jeder Zeit)

1. Fehler tritt in Container auf
   a) Fehler wird gespeichert in Error-Log (Datei)
   b) Allen Plugin wird falls möglich ein „Shutdown Event“geschickt
   c) Container muss neu gestartet werden
2. Fehler tritt in Plugin auf
   a) Das Plugin ist selber verantwortlich für Error-Log, kann es aber mit Hilfe des Exception-Handler des Containers bewerkstelligen
   b) Das Plugin ist selber verantwortlich für Data-Recovery
   c) Der Container speichert den Fehler des Plugin in eigenem Error-Log (Vorausgesetzt dem Controller steht die Fehlerinformation zur Verfügung)
   d) Der Container versucht das Plugin neu zu laden

6a. ≪UC3≫ Pluginmanager startet Plugins

1. Das Plugin benötigt Identifikation des Benutzers
   a) Das Plugin holt vom Container, bereits eingegebene Nutzerdaten
   b) Das Plugin selbst ruft Identifikation vom Benutzer ab
   c) Das Plugin nutzt das Subsystem um die Identifikation des Benutzers abzurufen
2. Das Plugin ermittelt Berechtigung vom Benutzer
   a) Das Plugin ermittelt die Berechtigungen selber
   b) Das Plugin ermittelt die Berechtigungen mit Hilfe des Subsystems
3. Benutzerinformationen speichern
   a) Das Plugin kann die abgerufenen Informationen an den Container zur weiteren Nutzung von anderen Plugins weitergeben
   b) Das Plugin gibt die abgerufenen Informationen nicht an den Container weiter
4. Plugin Berechtigung
   a) Der Benutzer ist berechtigt $->$ Plugin propagiert „Ready“
   b) Der Benutzer ist nicht berechtigt $->$ Plugin propagiert „Fehler“
5. Plugin verursacht beim starten ein Fehler $->$ Plugin propagiert „Fehler“

| **Spezielle Anforderungen** |
| --- |
| - Sprachbasierendes User Interface abhängig vom aktiven Plugin<br>- Mehrere Fenster<br>- Mehrere Instanzen des selben Plugin |
| **Technologie** |
| - Betriebssystem: Windows<br>- .NET Framework 4.0 |
| **Häufigkeit** |
| 0 bis 10 mal pro Tag |
| **Bekannte Probleme** |
| ≪keine≫ |

### 2.2.2. Nicht funktionale Anforderungen

#### 2.2.2.1. Portability

- Installierbar als Applikation pro Arbeitsplatz oder auf Netzlaufwerk
- Installierbar als Terminal Service Server Applikation

#### 2.2.2.2. Maintainability

- Optional: User Interface für Webbrowser

#### 2.2.2.3. Efficiency

- User Interface Reaktionszeit $< 1$ Sekunde

#### 2.2.2.4. Reliability

- Exceptions ausgelöst durch den Container werden in einer Datenbank geloggt
- Unbehandelte Exceptions von Plugins werden durch den Container, mittels neu laden des Plugins, behandelt
- Wiederherstellen verlorener Daten ist Sache des Plugin

#### 2.2.2.5. Functionality

- Das User Interface des Containers kann beliebig viel mal geöffnet werden (Ist aber nur eine Instanz)
- Jedes Plugin kann mehrere Instanzen innerhalb des Container haben

- Der Container benötigt keinerlei Konfiguration
- User spezifische Konfiguration (z.B. Sprache) wird vom Plugin verwaltet
- Der Container stellt globale Benutzerinformationen zur Verfügung, damit nicht jedes Plugin den Benutzer anfragen muss
- Der Container erhält die Benutzerinformationen von den Plugins
- Ist keine gültige Benutzerinformation vom Container erhalten worden, so ist das Plugin selbst verantwortlich für das Ermitteln der Benutzerinformationen
- Für die Abfrage von Benutzerdaten wird optional ein Subsystem zur Verfügung gestellt
- Jedes Plugin hat sein eigenes Konfigurationsfile (Falls notwendig)
- Für den Datenbankzugriff wird optional eine Klasse zur Verfügung gestellt um den Zugriff zu standardisieren
- Die Sprache des Containers wird vom aktiven Plugin definiert. Der Container besitzt eine Datei mit den für den Container notwendigen Textbausteinen in vordefinierten Sprachen. Der Standard ist Deutsch falls keine Sprachinformationen vom Plugin vorhanden sind
- Optional: Kommunikationsschnittstelle Plugin-Plugin

### 2.2.3. Anforderungen an Software

- Betriebssystem

  Minimum: Windows XP

  Empfohlen: Windows 7
- Microsoft .NET Framework 4.0

### 2.2.4. Anforderungen an Hardware

| System requirements | | |
|---|---|---|
| | Minimum | Empfohlen |
| Arbeitsspeicher | 512 MB (WindowsXP) | 2 GB (Windows Vista, 7) |
| CPU | 1GHz | 2GHz |
| Netzwerk (Internet Zugang) | ADSL 5000KBit | LAN 100MBit |

## 2.3. Design

### 2.3.1. Designentscheidungen

Da der Container das Design der Plugins stark beeinflusst werden nachfolgen die wichtigsten Designentscheidungen erläutert. Dabei wird begründet warum und wieso gewisse Designentscheidungen gefällt worden sind. Eine detaillierte Übersicht und Vergleich der Technologien sowie erfahrene Probleme sind im Anhang zu finden.

#### 2.3.1.1. User Interface

Beim User Interface hat man auf die Windows Presentation Foundation gesetzt, da WPF im Vergleich zu den Winforms keine Nachteile hat sondern nur Vorteile. Die saubere Strukturierung durch Anlehnung an XML, sowie umfangreiche Bindings zum reduzieren des Code-Behind stellen bereits einen grossen Vorteil dar. Auch die Unterstützung für die Implementierung der Commands reduziert den Code-Behind und fördert somit die Testbarkeit. Die Grafischen Aspekte kann man als Overhead bezeichnen könnten jedoch im weiteren Verlauf für den Container nützlich werden. Insofern wird in einem nächsten Schritt der Container erweitert, dass jedes Plugin die Farbe das User Interface ändern kann. Somit entsteht für den Benutzer eine besser optische Trennung, welches Plugin aktiv ist. Die Nachteile von WPF liegen einzig in der Komplexität.
Localization ist ein schwerwiegendes Requirement für den Container, dass bislang noch nicht integriert ist. Begründet ist das Fehlen durch die hohe Komplexität und dafür benötigten Zeitaufwand um das Knowhow zu erlangen. Diese Funktionalität wird eine der nächsten Funktion sein, die in den Container integriert wird.

Der Einsatz von WPF hatte zur Folge, dass das MVVM Pattern untersucht wurde. Es hat sich dabei herausgestellt, dass der Einsatz ungewohnt, sich aber im späteren Verlauf, als sehr praktisch erwies. Durch das MVVM kann ein Teil der GUI Funktionalität mittels Unittests getestet werden, was wiederum den manuellen Testaufwand reduziert und die Qualität der Software verbessert.

Das Design des User Interface ist an das des Outlook 2010 von Microsoft Office angelehnt. Dabei ist zu erwähnen, dass die Ribbon Bibliothek erst im Style von Office 2007 verfügbar ist. Office 2010 ist kürzlich erschienen und man kann davon ausgehen, dass die Bibliothek in nicht all zu langer Zeit ein Update erfährt. Der Entscheid, dass Design dem des Outlook anzulehnen, geht davon aus, dass Outlook als Mail Klient mit 43% am meisten genutzt wird und somit von vielen Personen, vor allem im Businessbereich, bekannt ist. Das Zurechtfinden innerhalb der Applikation wird somit erleichtert.

#### 2.3.1.2. Dynamisches Laden von Plugins

Beim Laden der Plugins hat man sich auf das im „.NET Framework 4.0“erschienene „MEF“konzentriert. Der Vorteil liegt darin, dass das laden mit Hilfe von MEF relativ einfach geht. Im weiteren Verlauf der Entwicklung hat sich jedoch gezeigt, dass das MEF die falsche Wahl war. Diese wird dadurch begründet, dass die Plugins in einer eigenen Applikationsdomäne gestartet werden müssen, damit der Exception-Handler des Containers unbehandelte Exceptions einem Plugin zuordnen kann. Die Requirements spezifizieren, dass im Falle eines Fehlers von einem Plugin, dasjenige Plugin neu geladen wird und die anderen davon nicht betroffen sind. Werden die Plugins mit dem MEF geladen, so laufen sie alle in der selben Applikationsdomäne. Dies hat zur Folge, das wenn in einem Plugin eine Exception geworfen und nicht behandelt wird, sie vom Exception-Handler verarbeitet wird. Diese Exception wird über den Thread an den Exception-Handler geleitet, welcher somit nicht mehr darauf zurück schliessen kann, wer den Fehler verursacht hat. Es bleibt aktuell nichts anderes übrig, als den Container und alle darin enthaltenen Plugins zu beenden. Dieses Problem wird ebenfalls als eines der nächsten behandelt.

Ein Lösungsansatz ist die Verwendung von „Reflection“, die es möglich macht Assemblies in verschiedenen Applikationsdomänen zu starten. Somit muss die ApplicationContainer-PluginBase alle unbehandelten Exceptions der eigenen Applikationsdomäne abfangen und an den Exception-Handler weiterleiten. Somit weiss dieser, wo der Fehler auftrat. Dieses System bedingt, falls die Plugins in Zukunft mit einander Kommunizieren können müssen, dass diese nur noch via „Remoting“kommunizieren dürfen. Die Umstellung auf des Ladens bedingt keinem all zu grossen Eingriff in das aktuelle Konzept. Es muss lediglich der Pluginmanager ersetzt werden.

#### 2.3.1.3. Datenbankanbindung

Bei der Datenbankanbindung hat man sich für das Entity Framework entschieden. Diese Entscheidung beruht darauf, dass der Container, bzw. das integrierte Prinzipal, keine komplexen Anfragen an die Datenbank stellt sondern hauptsächlich CRUD-Operationen durchführt. Im Vergleich zum Entwickeln mit ADO.NET DataSet sowie OleDbDataProvider ist das Entity Framework sehr einfach und intuitiv. Der benötigte Programcode wird ebenfalls stark reduziert. Sollte trotzdem der Fall auftreten, dass eine komplexe Datenbankabfrage notwendig ist, kann für diese immer noch auf den OleDbDataProvider zurück gegriffen werden.

### 2.3.2. Architecture Design

Das nachfolgende Architecture Design zeigt den groben Aufbau des Containers und dessen wichtigsten Komponenten. Das Diagramm zeigt die drei wichtigsten Komponenten. Dabei ist der Container, bzw. die ausführende Assembly des Container nur von der PluginInterface Assembly abhängig. In dieser Konfiguration stellt der Container aber nur eine leere Hülle dar und besitzt keinerlei nutzbarer Funktionalität. An dieser Stelle tritt die dritte Komponente, die Plugins, ins Spiel, von der beliebig viele vorhanden sein können. Erst wenn der Container beim Starten Plugins laden kann, wird eine für den Benutzer sinnvolle Funktionalität zur Verfügung gestellt.
Der Container erstreckt sich lediglich über den User Interface-Layer und den Application-Layer. Er stellt ein Grundelement des Frontend der Applikation dar, welche von den geladenen Plugins erweitert werden kann.

Die PluginInterface Assembly erstreckt sich bis auf den Data-Layer, da das integrierte Prinzipal via Entity Provider auf eine Datenbank zugreifen muss. Wird das integrierte Prinzipal nicht genutzt, so erstreckt sich die PluginInterface Assembly lediglich über den Application-Layer und stellt hauptsächlich Interfaces und Basisklassen, die notwendig sind um mit dem Container zu kommunizieren, zur Verfügung.

Die Plugins erstrecken sich Abhängig vom Plugin über alle Layer. Es ist dem Plugin überlassen, ob eine WCF Schnittstelle genutzt wird oder nicht.

### 2.3.3. Domain Model

#### 2.3.3.1. Application Container

Das Domain Model des Plugin Interfaces ist im Anhang C.2 zu finden.

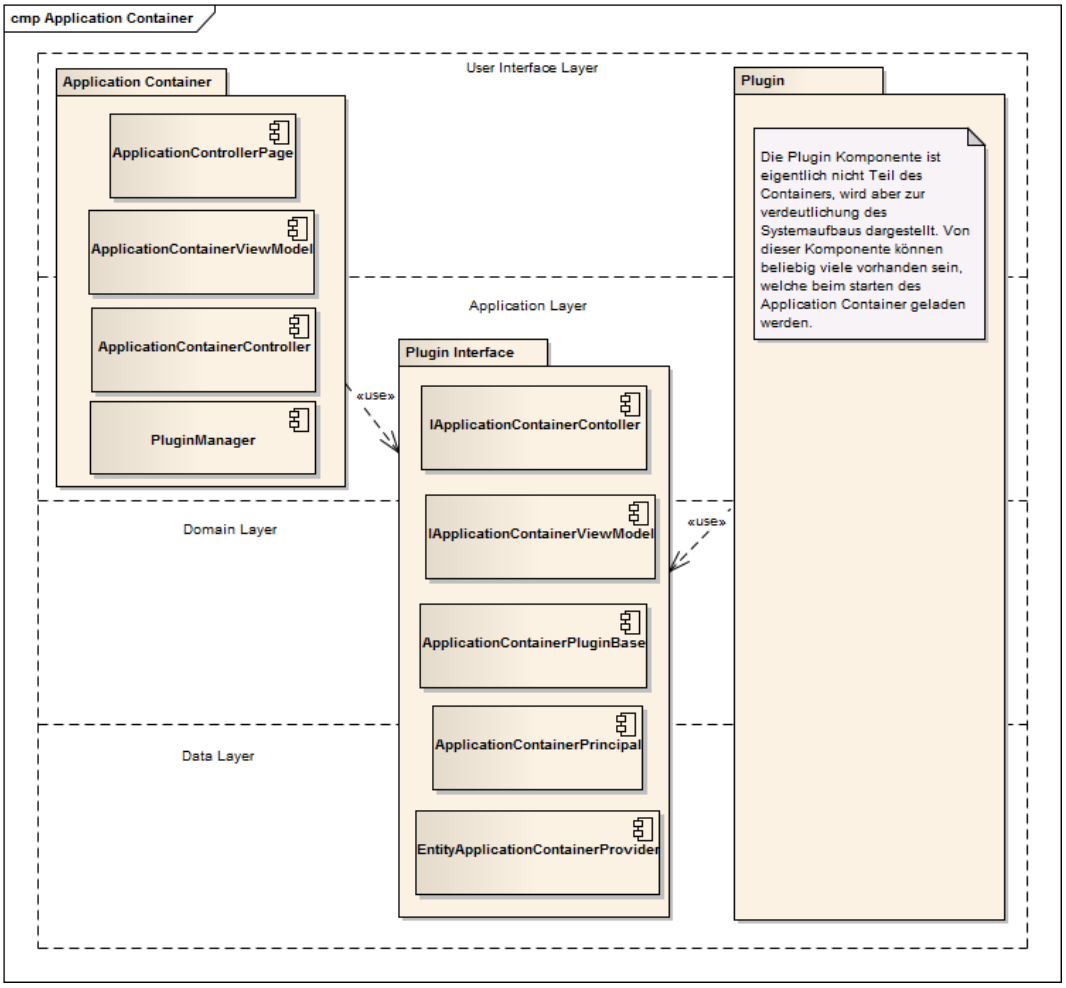


Abbildung 2.3.: Container Architekturübersicht

### ApplicationContainerController

Der Controller ist das Herzstück der Applikation und implementiert das IApplicationContainerController des PluginInterface. Dieses Interface stellt Methoden zum Austausch von Prinzipalen zwischen dem Controller und den Plugins zur Verfügung. Beim Einsatz von mehreren unterschiedlichen Plugins verhindert das Austauschen von Prinzipalen, dass der Benutzer sich bei jedem Plugin separat identifizieren muss. Der Controller startet eine Instanz des PluginManager sowie mindestens eine View. Beim beenden der Applikation sorgt der Controller dafür, dass alle Plugins kontrolliert beendet werden.

### ApplicationContainerViewModel

Das ViewModel des Containers stellt die Funktionalität zur Interaktion mit dem User Interface zur Verfügung. Über das ViewModel können die fünf Bereiche des Windows durch die Plugins angesprochen werden. Es beinhaltet folgende Commands:

- AddInstanceCommand: Erstellt eine neue Instanz eines Plugin und integriert es in alle offenen Fenster. Das Plugin kann vorgeben, ob dieser Command ausgeführt

werden darf oder nicht. Wird aufgerufen über das Kontextmenü der Plugin-Buttons.

- ActivatePluginCommand: Aktiviert das Plugin und stellt das User Interface zur Verfügung. Wird aufgerufen durch das Anklicken eines Plugin-Buttons.
- AddViewCommand: Öffnet eine neues zusätzliches Fenster. Wird aufgerufen über das Kontextmenü der Plugin-Buttons.
- ExitCommand: Schliesst alle Fenster und beendet den Container. Wird aufgerufen durch das Menü des Ribbon-Menüs.

### ApplicationContainerWindow

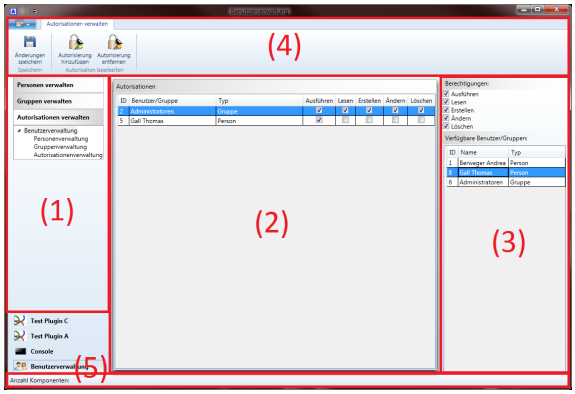


Abbildung 2.4.: Aufteilung Container User Interface

Der Container ist in fünf Bereiche unterteilt. Die Plugins haben via ViewModel des Containers Zugriff. Links, der (1) Menübereich, in der Mitte der (2) Hauptbereich sowie rechts der (3) Detailbereich. Zudem stehen ein (4) Menü Oben und eine (5) Statusleiste unten zur Verfügung. Der Detailbereich ist optional und wird, falls er nicht durch das Plugin verwendet wird ausgeblendet.

### PluginManager

Der Pluginmanager wird beim starten der Applikation durch den Controller instanziiert. Er verwaltet die vorhandenen Plugins und stellt eine Methode zur Verfügung mit deren Hilfe sogenannte Kataloge hinzugefügt werden können. Diese Kataloge können auf einen Ordner im System oder aber auch auf eine Assembly verweisen. Beim Hinzufügen eines Katalogs, durchsucht der Manager den Katalog nach Plugin Signaturen und fügt sie der Liste hinzu.

### ExceptionHandler

Der ExceptionHandler hat zwei Funktionen. Einerseits behandelt er alle Exceptions, die nicht behandelt worden sind und andererseits stellte er Methoden zur Verfügung um Nachrichten und Exceptions zu loggen. In der aktuelle Version wird das Log als Datei auf dem System hinterlegt. In Zukunft wird die Funktionalität, so erweitert, dass die Meldungen in einer Datenbank geloggt werden. Somit können sie schneller verarbeitet und korrigiert werden.

#### 2.3.3.2. Plugin Interface: Interaktion

Das Domain Model des Containers ist im Anhang C.2 zu finden.

Dieser Teil stellt die Interaktionslogik für das Zusammenspiel zwischen Container und Plugin dar. Der grösste Teil sind Interfaces um eine Kommunikation mit dem Container zu ermöglichen. Die ApplicationConatinerPluginBase stellt das Herzstück jedes Plugins dar. Jedes Plugin muss von dieser Klasse abgeleitet werden und diesen Typ mittels Komponent Model exportieren. Weiter Informationen zum Entwickeln von Plugins findet man unter „Plugin entwickeln“.

#### 2.3.3.3. Plugin Interface: Prinzipal

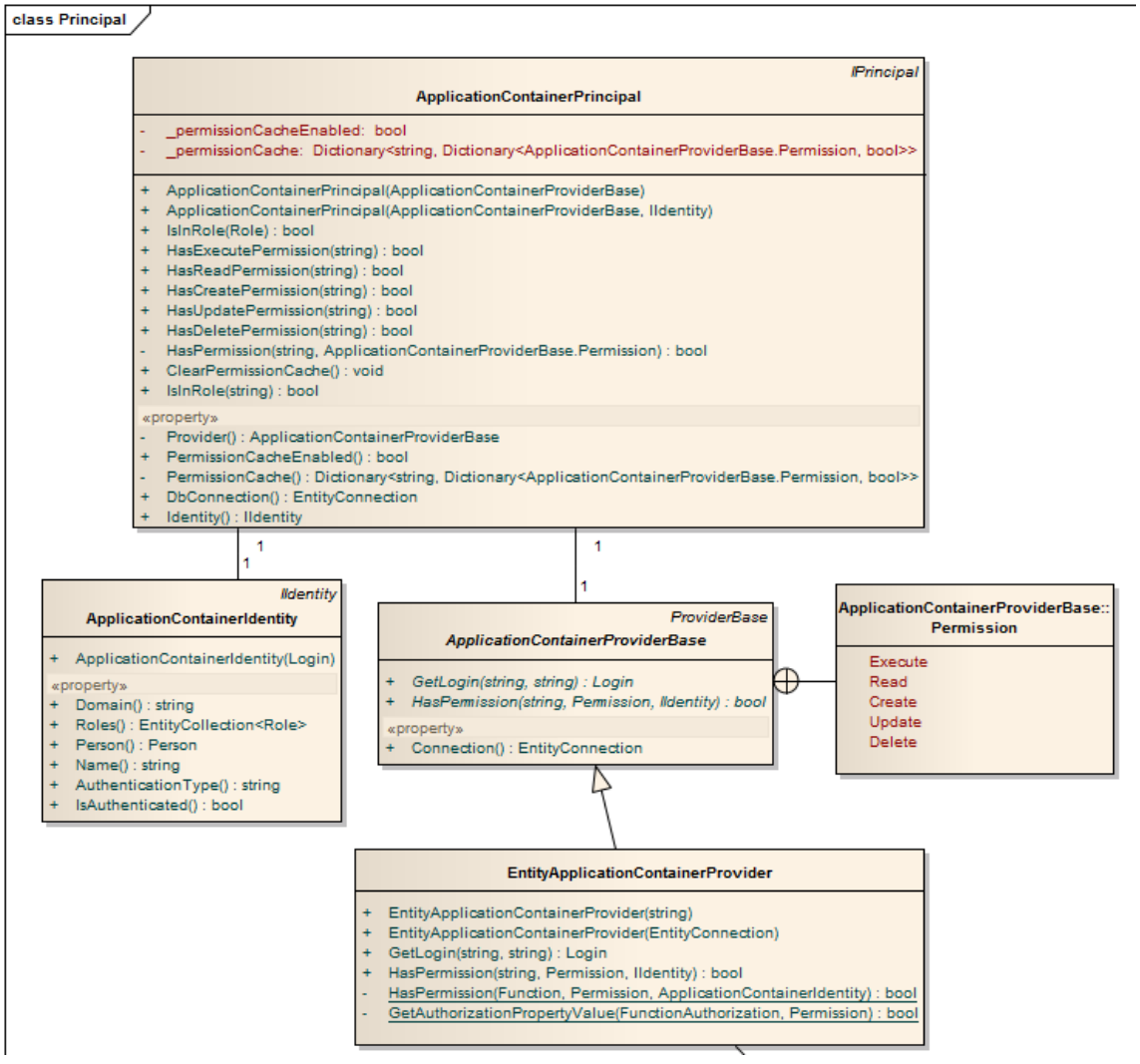


Abbildung 2.5.: Klassen für Subsystem Prinzipal

Dieser Teil stellt die Logik für das Subsystem, in Form eines integrierten Prinzipal, zur Verfügung. Das Subsystem beinhaltet das Prinzipal selbst (Implementiert das IPrinci-

pal Interface), ein entsprechendes Identity Objekt (Implementiert das IIdentity Interface) sowie eine ApplicationContainerProviderBase (Abgeleitet von ProviderBase) die für das Prinzipal den Datenspeicher zur Verfügung stellt. Als einen ersten Provider steht der EntityProvider zur Verfügung um die Berechtigungsdaten verfügbar zu machen. Der EntityProvider, wie der Name schon sagt, greift mittels Entity Framework auf eine Datenbank zu. Das unten stehend Bild zeigt den Aufbau des Datenbankmodels. Mit Hilfe des integrierten Prinzipals kann über eine Methode die Berechtigung für ein Plugin oder eine Funktion abgefragt werden. Die Berechtigungen für die Plugins sowie deren Funktionen sind hierarchisch aufgebaut. Bei der Abfrage einer Berechtigung via Prinzipal wird die entsprechende Funktion übergeben. Der Prinzipal ermittelt nun von der angegebenen Funktion aus der Hierarchie nach oben, ob der Prinzipal eine Berechtigung hat. Ist er an der obersten Hierarchiestufe, sprich dem Plugin, angelangt ohne eine Berechtigung für den Prinzipal zu finden, wird der Zugriff verweigert.

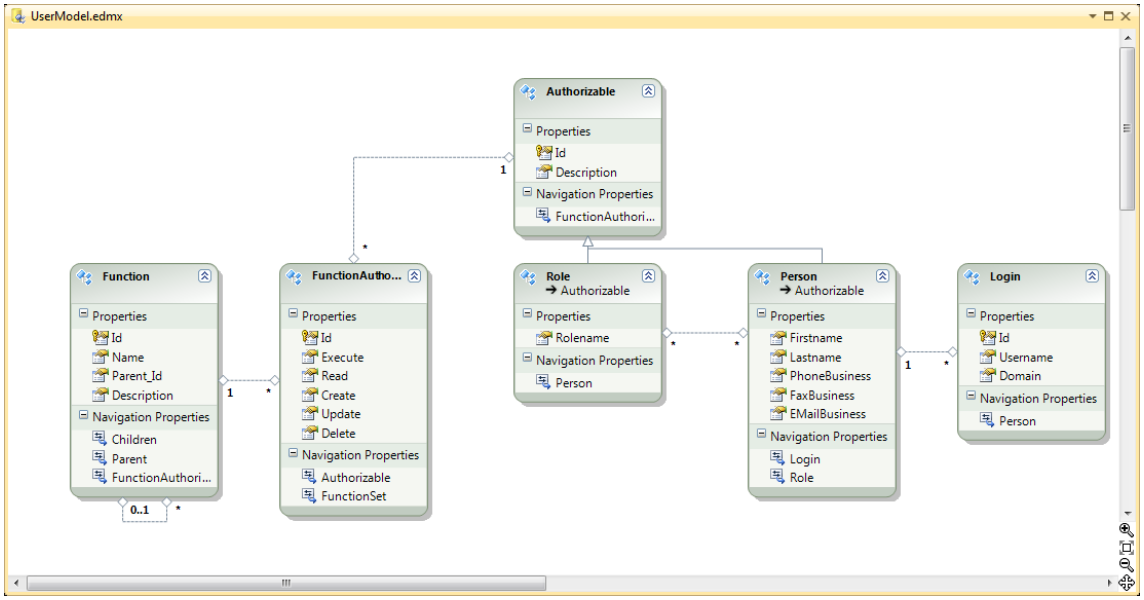


Abbildung 2.6.: Datenbankmodel des Subsystems

Das Datenbankmodel das durch den Prinzipal genutzt wird besteht aus sechs Entities die hier kurz beschrieben werden:

- **Function:** Die „Function“stellt ein Plugin oder eine Funktion eines Plugins dar, der eine Berechtigung zugewiesen werden kann. Die oberste Hierarchiestufe ist das Plugin. Ein Plugin kann unter sich auf Funktionen aufgeteilt werden. Die Funktionen können wiederum in Unterfunktionen aufgeteilt werden. Aus diesem Grund hat die Funktion eine Beziehung zu sich selbst. Mit anderen Worten kann eine Funktion eine Funktion als Elternknoten haben.
- **FunctionAuthorization:** Die „FunctionAuthorization“stellt eine Verbindung zwischen einem Plugin, bzw. einer Funktion um einem autorisierbaren Objekt her. Sie beinhaltet die Berechtigungen (Ausführen, Lesen, Erstellen, Ändern, Löschen) für dieses Objekte.

- **Authorizable:** Das „Authorizable“stellt eine abstrakte Basisklasse für eine Rolle oder Person dar. Dies ist notwendig um einer Funktion eine Rolle oder eine Person zuweisen zu können.

- **Role:** Die „Role“kann einer Funktion zugewiesen werden und es können ihr beliebige Personen angehören.

- **Person:** Die „Person“kann einer Funktion zugewiesen werden und sie kann beliebigen Rollen angehören.

- **Login:** Das „Login“identifiziert eine Person und ist eindeutig. Dies bedeutet, dass nicht zweimal der selbe Username und Domain zusammen vorkommen darf. Eine Person kann mehrere Logins haben um sich an mehreren Domänen anmelden zu können. Da das Entity Framework bislang noch keine Unterstützung für die Definition von „Unique“kennt, muss nachträglich in der Datenbank die Spalten „Username“und „Domain“als „Unique“definiert werden. Tritt in der Datenbank zweimal das selbe Login auf, so wird es nicht mehr durch den Prinzipal erkannt und die Person kann nicht ermittelt werden.

### 2.3.4. Interaction Diagram

#### 2.3.4.1. Startup

Im nachfolgenden Sequenzdiagramm ist ersichtlich was bei starten des Containers, bzw. der Applikation vor sich geht und wie die Klassen des Containers miteinander interagieren.

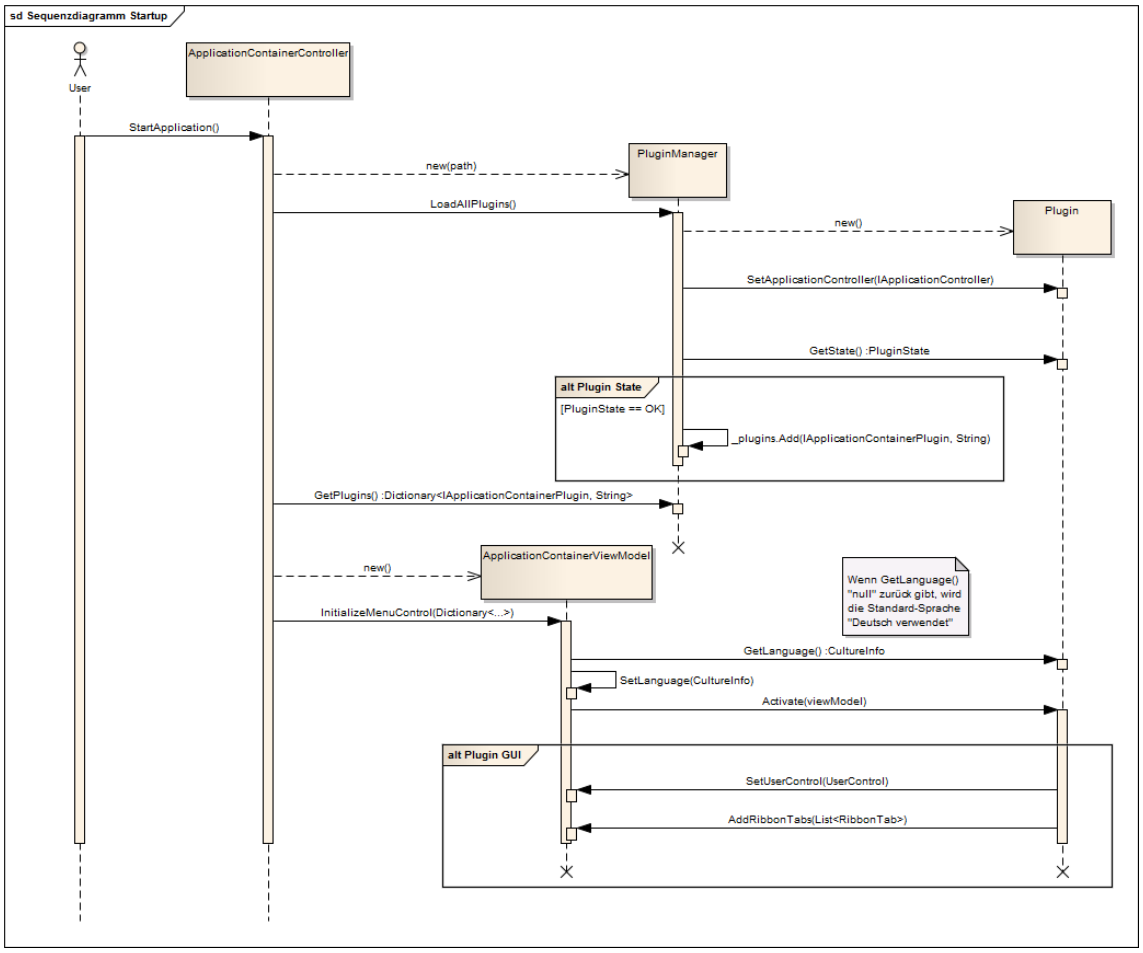


Abbildung 2.7.: Sequenzdiagramm Startup Container

Beim Startvorgang wird eine Instanz des ApplicationContainerController erzeugt, welcher wiederum eine Instanz des Pluginmanager erzeugt. Nach dem Erzeugen des Pluginmanager wird vom Controller ein Catalog mit dem Standardpfad übergeben, in welchem sich die Plugin Assemblies befinden. Nach jeder Initialisierung eines Plugin überprüft der Pluginmanager den Status des Plugin. Entspricht der Status eines Plugin nicht dem Status „Ready“, so wird es nicht in die Plugin Liste aufgenommen und verworfen. Entspricht der Status, so wird dem Plugin eine Referenz auf den Controller übergeben, dieser kann vom Plugin genutzt werden, um die Authorisationen auszutauschen. Danach wird das Plugin in einer Liste hinterlegt.
Nach der Initialisierung der Verfügbaren Plugins, erzeugt der Controller eine View in Form eines ApplicationContainerViewModel, welches für die Interaktion mit dem User Interface zuständig ist. Dieses wird in einer Liste, da mehrere Views unterstützt werden, hinterlegt. Jede View erzeugt ein Fenster, das als User Interface dient. Bei jedem hinzufügen einer View wird vom Controller das Initialisieren des PluginMenu aufgerufen. Dabei werden alle Plugins vom Pluginmanager an das ViewModel weitergereicht, welche diese mit Hilfe der PluginMenuButton Klasse in das Plugin-Menü integriert. Um ein Plugin korrekt im User Interface anzeigen zu können muss das Plugin mindesten einen Namen besitzen der nicht null oder leer ist. Optional kann auch ein Image zur Verfügung gestellt werden, welches

im Plugin-Menu angezeigt wird. Bereits der Pluginmanager überprüft, ob der Name des Plugin den Anforderungen entspricht, andernfalls wird das Plugin bereits durch den Pluginmanager verworfen.

#### 2.3.4.2. Shutdown

Das nachfolgende Sequenzdiagramm illustriert das Beenden des Container. Beim herunterfahren des Containers stellt der Controller sicher, dass jedes Plugin mittels der „Shutdown“-Methode die Möglichkeit erhält sich zu beenden. Dies wird insbesondere wichtig, wenn das Plugin eine Verbindung zu einer Datenbank hat, oder nicht gespeicherte Daten vorhanden sind.

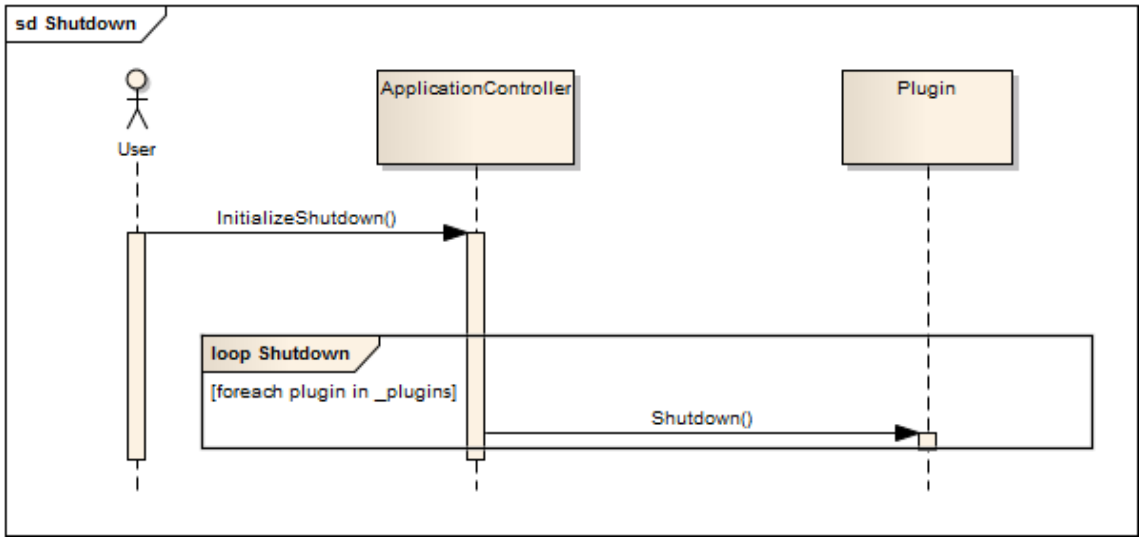


Abbildung 2.8.: Sequenzdiagramm Shutdown Container

#### 2.3.4.3. Plugin Wechsel

Beim Wechseln eines Plugin wird das derzeitig aktive Plugin mittels „Deactivate“aufgefordert seinen Zustand zu speichern und das User Interface freizugeben. Danach wird das User Interface durch des ViewModel zurückgesetzt, so dass das aktivierte Plugin die notwendigen Controls wieder laden kann. Auch hier wird bei „Deactivate“und „Activate“das ViewModel übergeben, da ja das Plugin in einer anderen View immer noch aktiv sein kann. Vor dem „Activate“wird durch das ViewModel die Sprache des zu aktivierenden Plugin ermittelt, und auf das User Interface der Container relevanten Objekte angewendet.

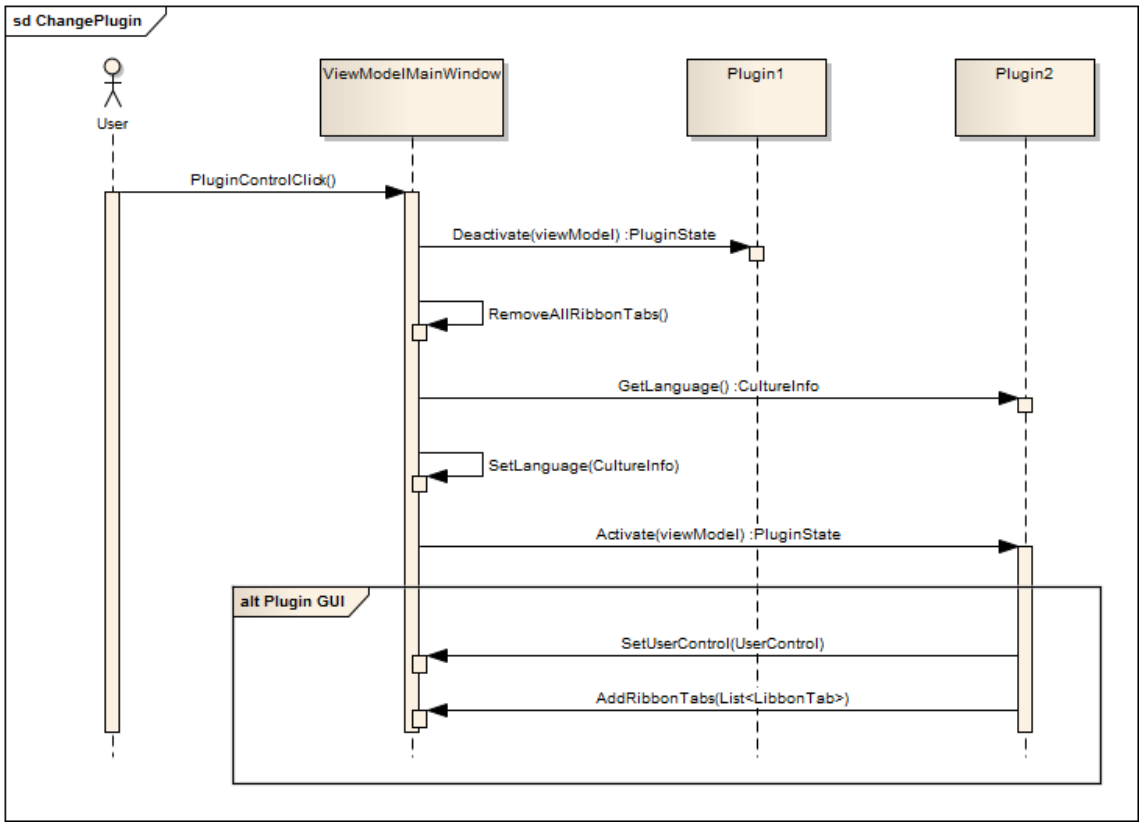


Abbildung 2.9.: Sequenzdiagramm Wechseln eines Plugin

#### 2.3.4.4. Austausch von Authorisationen zwischen Plugins

Um zu verhindern, dass ein Benutzer für jedes Plugin ein Login benötigt, können von Plugins ermittelte Benutzerinformationen mit Hilfe des Container ausgetauscht werden. Somit muss nicht jedes Plugin den Benutzer mit lästigen Benutzerabfragen plagen. Voraussetzung dafür ist natürlich, dass ein Benutzer, der mehrere Plugins nutzt, auf allen dasselbe Login besitzt. Sind diese Einschränkungen gegeben, so zeigt das nachfolgende Diagramm den Ablauf des Austausches.

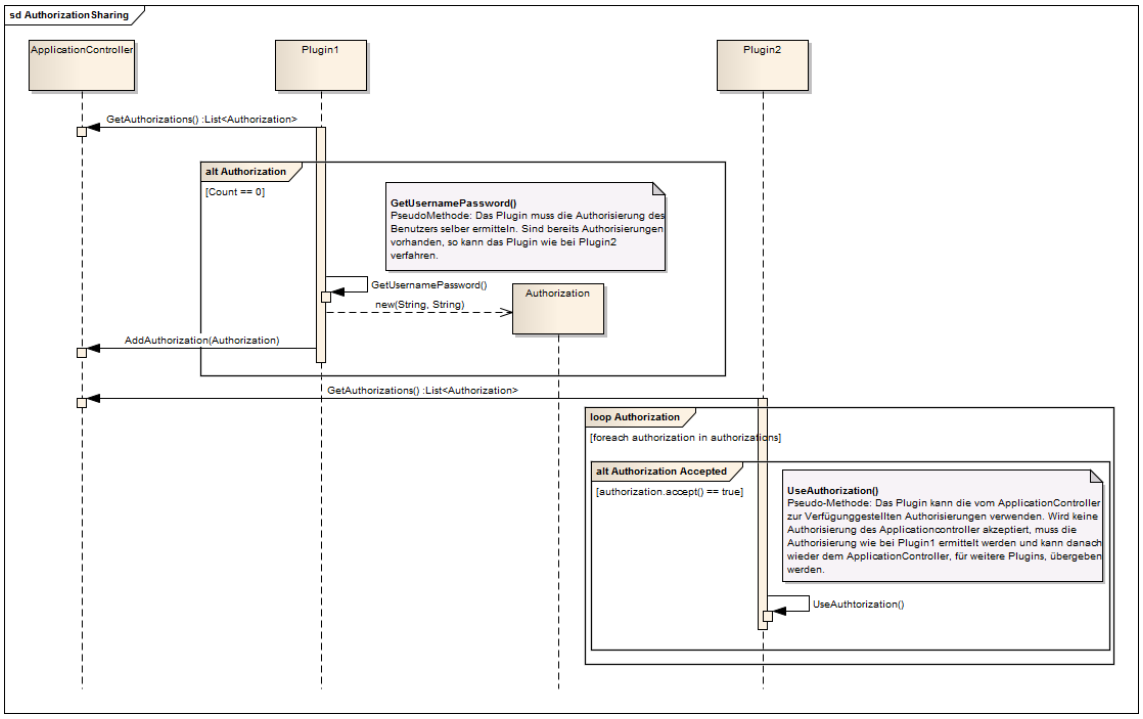


Abbildung 2.10.: Sequenzdiagramm Austausch Berechtigungen

Benötigt ein Plugin Benutzerinformationen, so hat es die Möglichkeit beim Controller eine Liste von vorhanden Authorisationen abzufragen. Ist diese leer, oder keines der Authorisationen entspricht einem korrekten Login, so muss das Plugin die Benutzerinformationen selber besorgen. Hat das Plugin die korrekten Benutzerinformationen erhalten, so liegt es im ermessen des Plugin, ob diese Benutzerdaten an den Controller weitergeleitet werden um anderen Plugins zur Verfügung gestellt werden oder nicht.
Für das Ermitteln des Benutzers oder dessen Berechtigungen steht im Framework ein Subsystem zur Verfügung.

## 2.4. Datenbank Mapper

### 2.4.1. Übersicht

In den Requirements des Container, wurde spezifiziert, dass der Container eine standardisierte Schnittstelle an eine Datenbank anbieten muss. Es wurde deklariert, dass ein eigenständiges Assembly geben muss, dass den Zugriff auf die Datenbank abstrahiert. Dieses Requirement entspringt der konzeptionellen Idee des Datenbank Managers.

Das Domain Model des Datenbank Managers ist im Anhang C.3 zu finden.

Die DatabaseManager-Klasse ist eine abstrakte Klasse und wird in der Anwendung als abgeleitete Klasse, je nach Datenbank (MSSQL, Oracle, usw.), instanziiert. Die abgeleiteten Klassen sorgen dafür, dass die SQL-Statement ins richtige Format für die jeweilige Datenbank konvertiert werden. Der DatabaseManager stellt verschiedene Command-Klassen zur

Verfügung die jeweils einer Aufgabe auf der Datenbank entspricht. Das DatabaseManager-Objekt, wird in der Anwendung einmalig erzeugt und danach den Klassen, welche einen Datenbankzugriff benötigen, zur Verfügung gestellt. Diese Klassen können mit Hilfe der Commands verschiedene Aufgaben auf der Datenbank erledigen lassen. Der Manager verarbeitet ein Command nach dem anderen asynchron und gibt den abgearbeiteten Command mittels Callback an den Antragsteller zurück. Der zurückgegebene Command enthält das Resultat des Antrags und/oder allfällige Exceptions. Somit wird auch das Dataset gekappselt und kann nicht von überall manipuliert werden.

### 2.4.2. Fazit

Da man sich nun dazu entschlossen hat, das Entity Framework für Datenbankzugriffe zu verwenden, fällt das Entwickeln der Datalayerabstraktion weg. Die Funktion der Abstraktion wird durch das Entity Framework bewerkstelligt. Der Datenbank Manager ist vom Verwendungszweck her mit dem Entity Framework sehr ähnlich. Diese Tatsache kann mit einem Quervergleich deutlich gemacht werden.
Betrachtet man die Klasse „DatabaseManager“des DatenbankManager-Konzeptes, so kann man diese mit dem Kontext des Entity-Framework vergleichen, der den Klassen zur Verfügung gestellt wird. Die entsprechenden SQL-Statements werden beim Datenbank-Manager als SQL-String mittels einem Command an den Manager gereicht, was beim Entity-Framework mittels Linq-Expression auf dem Kontext geschieht. Beim Datenbank-Manager wird das Resultat innerhalb des Commmand zurück geliefert, was beim Entity-Framework als Query geschieht.

Aus dieser Erkenntnis kann man schliessen, dass das Entity-Framework bereits das anbietet, was man eigentlich als standartisierte Schnittstelle auf eine Datenbank für den Container erzeugen wollte.

## 2.5. Plugin entwickeln

Die Entwicklung eines Plugin gestaltet sich recht einfach. Beim Design des Containers wurde explizit darauf geachtet, dass die Entwicklung einfach ist. In einem ersten Schritt wird ein neues Bibliothekenprojekt angelegt. Dabei spielt es keine Rolle ob mit C# oder Visual Basic entwickelt wird. Dem neu erzeugten Projekt muss ein Verweis auf das „PluginInterface“und „System.ComponentModel.Composition“, hinzugefügt werden. Mindestens eine Klasse im Projekt muss nun von der ApplicationContainerPluginBase abgeleitet werden, die im „PluginInterface“zu finden ist. Zusätzlich muss dieser Klasse ein Export-

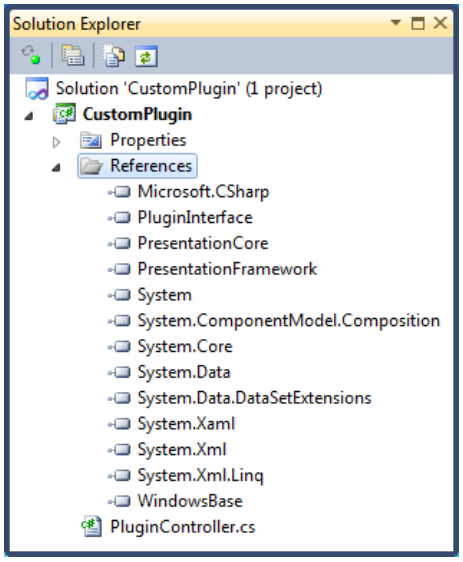


Abbildung 2.11.: Projektmappe Plugin Entwicklung

Attribut für den Typ „ApplicationContainerPluginBase“angehängt werden. Diese Klasse wird durch den Container erkannt und geladen. Damit das Plugin angezeigt wird, müssen ein paar Dinge beachtet werden. Das abstrakte Property „Name“muss überschrieben werden und darf weder leer, null noch eine Exception werfen. Im Konstruktor des Plugin muss sichergestellt sein, dass das Property „State“in der Basisklasse auf „Ready“gesetzt wird. Wird beim instanziieren eine Exception geworfen oder ist der Status nicht „Ready“so wird das Plugin ignoriert und nicht geladen.

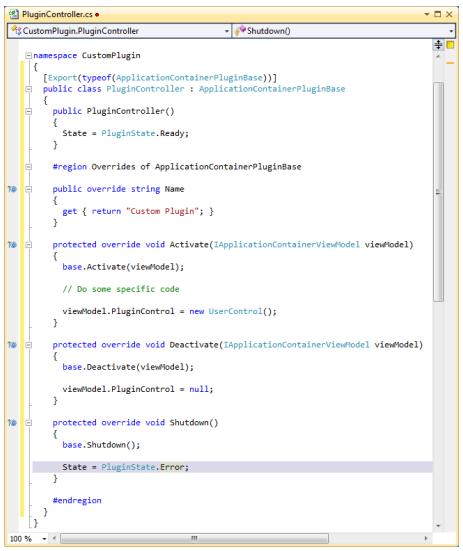


Abbildung 2.12.: Controller Klasse Plugin Entwicklung

Die Kommunikation zwischen dem Container und dem Plugin geschieht über Interfaces. Die Basisklasse stellt überschreibbare Methoden zur Verfügung, die durch den Container beim Aktivieren, Deaktivieren sowie Herunterfahren aufgerufen werden. Beim Aktivieren und Deaktivieren wird das ViewModel des Containers, das den Aufruf ausgelöst hat, übergeben. Das Container ViewModel implementiert das IApplicationContainerViewModel Interface das Methoden zur Verfügung stellt um das User Interface des Containers zu manipulieren.

Bei der Entwicklung muss beachtet werden, dass der Container mehrere Fenster anzeigen kann. Da bei der Windows Presentation Foundation ein Control nur einen Parent zugewiesen werden kann, muss dafür gesorgt werden, dass das Plugin für jedes Fenster ein neues Control instanziiert. Andernfalls wird eine Exception ausgelöst, dass das Control nur einem Parent angehängt werden kann.

Im Installationsverzeichnis des Containers ist ein Ordner „Plugins“zu finden. Dieser Ordner wird standardmässig nach Plugins durchsucht. Das erstellte Plugin muss lediglich in diesen Ordner kopiert werden. Dabei müssen natürlich alle vom Plugin benötigten Dateien kopiert werden.

## 2.6. Konsolen Plugin

Das Konsolenplugin ist nebenbei entwickelt worden und stellt ein kleines aber spezielles Plugin dar. Das spezielle am diesem Plugin ist, dass es mittels dem IConsolePlugin Interface im PluginInterface-Assembly durch den Container angesprochen werden kann. Wird das Plugin beim Laden des Containers erkannt, so wird es dem Exception-Handler des Containers angehängt. Alle Fehler und/oder Meldungen die über den Exception-Handler behandelt werden, werden im Plugin aufgelistet. Dies kann zur Erkennung und Behandlung von Fehlern von Vorteil sein und dient somit hauptsächlich administrativen Zwecken.

Das Plugin hat einen simplen Aufgebau und wird in der dargestellten Abbildung gezeigt. Das Plugin besteht aus dem Controller der wiederum das Plugin darstellt, dem ViewModel, dass den Zustand des User Interfaces verwaltet und einem Control, dass lediglich aus einer Textbox besteht, die die Meldungen anzeigt. Zusätzlich existiert ein RibbonTab mit einem Button, der es ermöglicht, die Konsole zu löschen.

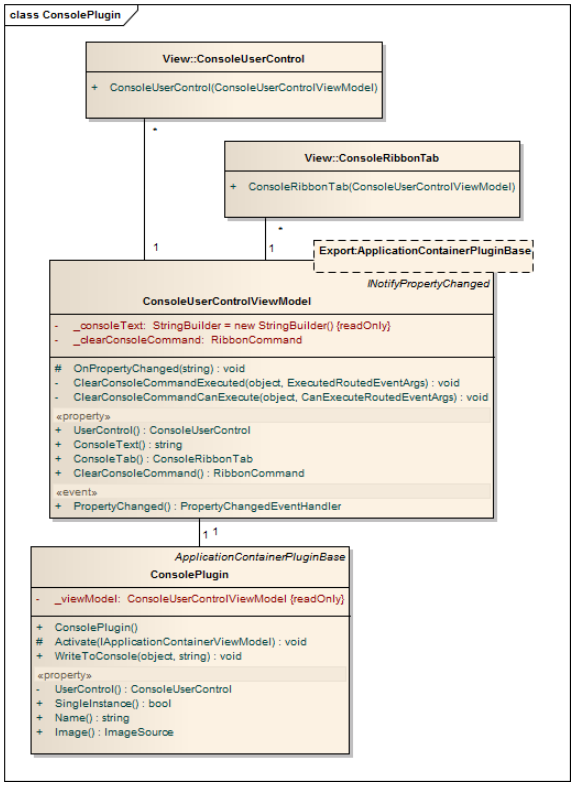


Abbildung 2.13.: Console Plugin Domain Model

# 3. Benutzerverwaltung

## 3.1. Allgemeine Beschreibung

Das integrierte Subsystem des Containers, mit dem die Authentisierung sowie Autorisierung bewerkstelligt werden kann, benötigt eine Datenquelle, von der es die Informationen über den aktuellen Benutzer beziehen kann. Diese Daten werden in einer Datenbank gespeichert und mittels Entity Framework abgerufen. Um diese Daten bearbeiten zu können, bedingt es einer Verwaltung. Die Verwaltung für diese Daten wird als Plugin für den Container realisiert.

Das Plugin bestätigt einerseits die Funktionalität des Containers und andererseits die Funktionalität des Subsystems

Die Verwaltung bietet Funktionen zum erstellen und bearbeiten sowie löschen von Personen, Rollen und Berechtigungen.

## 3.2. Analyse

### 3.2.1. Funktionale Anforderungen

In der Vergangenheit, hat sich gezeigt, dass es von den Benutzern geschätzt wird, wenn sie sich nicht mittels Passwort einloggen müssen, trotzdem aber eine minimale Sicherheit besteht, dass nicht jeder Zugriff auf das System hat. Um das zu Realisieren, hat man die Identifikation der Benutzer über deren Arbeitsstation realisiert. Das heisst, dass der Benutzer mittels Windows-Benutzername und Domäne identifiziert wird. Damit ein Benutzer auf verschiedenen Arbeitsstationen erkannt werden kann, muss die Möglichkeit bestehen, einem Benutzer mehrere Logins zuzuweisen. Ansonsten werden die Berechtigungen dem Windows nachempfunden. Es gibt Personen und Rollen wobei die Personen den Rollen zugeordnet werden können. Die Rollen und/oder Personen können wiederum Funktionen zugeordnet werden und deren Berechtigung definieren.

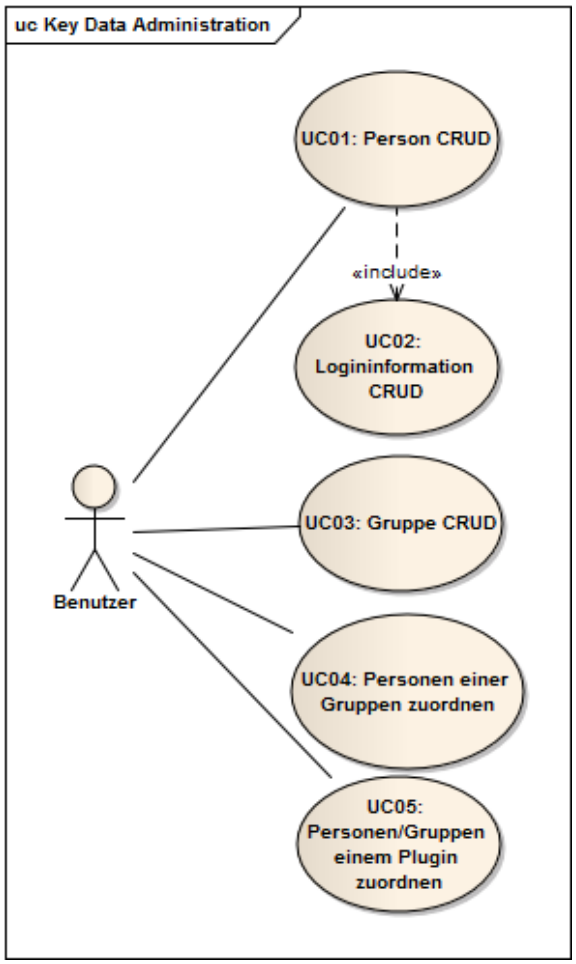


Abbildung 3.1.: User Administration Use Case Model

Das Use Case Diagramm besteht mehrheitlich aus Create-, Read-, Update- und Delete-Operationen (CRUD). Es existiert nur ein Typ von Benutzer der die Benutzerverwaltung benutzt. Dieser Benutzer kann aber je nach Berechtigung des Benutzerverwaltungs-Plugin und/oder Gruppenzugehörigkeit unterschiedliche Aufgaben innerhalb der CRUD-Aufgaben erledigen. Die Berechtigungen werden wie CRUD aufgeteilt. Es kann einem Benutzer oder einer Gruppe die Berechtigung für Ausführen, Erstellen, Lesen, Ändern oder Löschen zugeordnet werden. Die Zuordnung der Benutzer/-Gruppen zu einem Plugin ist Hierarchisch gegliedert, d.h. es kann nicht nur einem Plugin als ganzes einem Benutzer/Gruppe zugeordnet werden, sondern das Plugin kann auch noch Berechtigungen von Funktionen innerhalb sich selber definieren.

#### 3.2.1.1. Brief Use Cases

**UC01: Person CRUD**

Ein berechtigter Benutzer startet die Benutzerverwaltung und navigiert zur Personenverwaltung. Er erzeugt eine neue Person und gibt gibt mindestens einen Vor- und Nachnamen ein. Andernfalls kann die neue Person nicht gespeichert werden. Nach Eingabe von Vor- und Nachname sowie optionalen zusätzlichen Informationen kann der neue Benutzer gespeichert werden und sie steht im System zur Verfügung.

**UC02: Logininformation CRUD**

Ein bereits vorhandener Benutzer möchte die Benutzerverwaltung an einer zusätzlichen Arbeitsstation ausführen. Er meldet sich an einer Station an, auf der er bereits berechtigt ist und startet die Benutzerverwaltung. Er sucht in den Personen nach seiner eigenen Identifikation. Nach Auffinden der Person, fügt er ein neues Login hinzu und bearbeitet das Login entsprechend den Informationen der zusätzlichen Arbeitsstation. Er speichert die Änderungen.

#### UC03: Gruppe CRUD

Ein berechtigter Benutzer startet die Benutzerverwaltung und navigiert zur Gruppenverwaltung. Er erzeugt eine neue Gruppe und gibt ihr einen entsprechenden Name. Nach dem speichern der Änderung, ist die neue Gruppe im System verfügbar.

#### UC04: Personen einer Gruppen zuordnen

Ein berechtigter Benutzer startet die Benutzerverwaltung und navigiert zur Gruppenverwaltung. Er sucht in der Gruppenauswahl die Gruppe, der eine zusätzliche Person angefügt werden soll. Nach auffinden der Gruppe sucht er in den vorhanden Personen die Person, die hinzugefügt werden soll. Nach auffinden der Person wird diese markiert und hinzugefügt.

#### UC05: Personen/Gruppen einem Plugin zuordnen

Ein berechtigter Benutzer startet die Benutzerverwaltung und navigiert zur Berechtigungsverwaltung. Er sucht nach dem entsprechenden Plugin bzw. einer Funktion und markiert dieses. Im Berechtigungsbereich werden alle Berechtigungen aufgelistet. Um einer Person oder Gruppe eine Berechtigung hinzuzufügen, markiert er eine Person oder Gruppe in der Auflistung aller vorhandenen Personen und Gruppen. Durch einen Klick auf Berechtigung hinzufügen, wird die neue Berechtigung erzeugt. Er speichert die Änderung.

### 3.2.1.2. Detailierte Use Cases

UC01 bis UC03 werden nicht genauer beschrieben, da sie alle die Bearbeitung (CRUD) von jeweils verschiedenen Daten darstellen. Die Funktionalität von UC04, das Zuordnen von Benutzern zu Gruppen, ist zwar kein CRUD, jedoch trotzdem selbsterklärend. UC05 lässt ein wenig Spielraum offen und wird deshalb genauer definiert.

#### UC05: Personen/Gruppen einem Plugin zuordnen

| | |
|---|---|
| Primary Actor | Benutzeradministrator |
| Stackholders/Interest | Benutzeradministrator möchte einer Gruppe und dem stellvertretender Mitarbeiter die Berechtigung auf das Benutzerverwaltungs-Plugin oder eine Funktion dessen erteilen. |
| Preconditions | Gruppe und stellvertretender Mitarbeiter hat beim laden des Containers keinen Zugriff auf die Benutzerverwaltung |
| Postconditions | Gruppe und stellvertretender Mitarbeiter hat beim laden des Containers Zugriff auf die Benutzerverwaltung |

**Basis Verlauf**

1. Der Benutzeradministrator startet den Container, bzw. er aktiviert die Benutzerverwaltung mit seiner Berechtigung, sprich an seiner Arbeitsstation.
2. Er aktiviert die Berechtigungsverwaltung für die Plugins.
3. Nach dem er die Berechtigungsverwaltung aktiviert hat selektiert er das Plugin, für das eine Berechtigung erstellt werden soll.
4. Er sucht und selektiert die Person des stellvertretenden Mitarbeiters in der Liste der vorhandenen Personen und Gruppen.
5. Er klickt auf Berechtigung hinzufügen. Es wird eine neu Berechtigung erzeugt und selektiert.
6. Er definiert die Berechtigungen (Ausführen, Lesen, Erstellen, Ändern, Löschen) für die neue bereits selektierte Berechtigung.
7. Um der Gruppe den Zugriff zu erlauben sucht er diese in der List der vorhanden Gruppen und Personen und selektiert diese.
8. Nach einem Klick auf Berechtigung hinzufügen, wird wieder eine Berechtigung erzeugt und selektiert.
9. Auch hier kann er die Berechtigungen (Ausführen, Lesen, Erstellen, Ändern, Löschen) definieren.
10. Um den Vorgang abzuschliessen, speichert er die Änderungen mit einem Klick auf Speichern und schliesst den Container, bzw. die Benutzerverwaltung

| **Alternativer Verlauf** |
|---|
| a. Systemfehler tritt auf (jeder Zeit)<br><br>1. Das Plugin schickt dem Container den Event zum loggen<br><br>2. Der Ursprungszustand der Berechtigungen werden wiederhergestellt $-> $ Datenbankkonsistenz<br><br>3. Das Plugin wird neu geladen<br><br>3a. Die Berechtigung wird grundsätzlich auf alle Funktionen bis auf die Personenverwaltung angewendet.<br><br>1. Er erzeugt zuerst Plugin übergreifend eine Berechtigung für den stellvertretenden Mitarbeiter und die Gruppe mittels des Standardverlaufs.<br><br>2. Zusätzlich erzeugt er eine Berechtigung für den Mitarbeiter und die Gruppe für die Personenverwaltung nach dem Standardverlauf, verweigert aber das Ausführen. Somit wird die Berechtigung für die Gruppenverwaltung und Berechtigungsverwaltung vererbt. Die Personenverwaltung jedoch durch die einschränkende Berechtigung verweigert.<br><br>3b. Benutzer ist noch nicht im System vorhanden.<br><br>1. Der Benutzeradministrator öffnet die Personenverwaltung. ≪UC01≫<br><br>2. Er erzeugt eine neue Person und gibt dessen Informationen ein.<br><br>3. Er klickt auf Login hinzufügen. ≪UC02≫<br><br>4. Er gibt den Benutzername und Domäne des Logins ein.<br><br>5. Er speichert die neue Person.<br><br>5a. Die Gruppe ist noch nicht im System vorhanden.<br><br>1. Der Benutzeradministrator öffnet die Gruppenverwaltung. ≪UC03≫<br><br>2. Er klickt auf Gruppe hinzufügen und gibt ihr den entsprechenden Namen.<br><br>3. Für jede Person, die der Gruppe angefügt werden soll, selektiert er diese in der Liste der vorhanden Personen und klickt auf hinzufügen.<br><br>4. Er speichert die neue Gruppe. |

| **Spezielle Anforderungen** |
|---|
| ≪Keine speziellen Anforderungen≫ |
| **Technologie** |
| ≪Keine speziellen Anforderungen≫ |
| **Häufigkeit** |
| 0 bis 1 mal pro Monat |
| **Bekannte Probleme** |
| ≪keine≫ |

### 3.2.2. Nicht funktionale Anforderungen

#### 3.2.2.1. Portability

- Realisiert als Plugin für den Container
- Keine Installation $->$ Drag & Drop der Klassenbibliothek in Pluginordner des Containers
- Verfügbarkeit des Plugin kann gesteuert werden über Vorhandensein der Klassenbibliothek als DLL im Pluginordner oder mittels Berechtigung.

#### 3.2.2.2. Maintainability

- Prüfbar mittels UnitTests
- User Interface wird in Container integriert

#### 3.2.2.3. Efficiency

- User Interface Reaktionszeit $< 1$ Sekunde (Optional: Asynchroner Datenbankzugriff)

#### 3.2.2.4. Functionality

- Autorisierung wird an die Windows-Autorisierung angelehnt
- Erlernbarkeit: Learning by doing

#### 3.2.2.5. Reliability

- Exceptions werden durch das Plugin verarbeitet jedoch an den Container mitgeteilt
- Datenkonsistenz muss erhalten bleiben
- Optional: Wiederherstellen von eingegebenen, nicht gespeicherten Daten

#### 3.2.2.6. Functionality

- Die Berechtigung für die Benutzerverwaltung wird von sich selbst (Benutzerverwaltungs-Plugin) verwaltet.
- Speichert die Daten in einer Datenbank
- Datenbanken: MSSQL, Optional: Oracle, CompactSQL
- Datenbankzugriff mittels Entity-Framework
- Es werden sensible Daten Verwaltet $->$ Datenbankzugriff Security
- Benutzer wird ohne Passwort mittels Windows-Benutzername und Domäne bzw. Arbeitsgruppe ermittelt
- Benutzer kann mehrere Login haben für mehrere verschiedene Arbeitsstationen

### 3.2.3. Anforderungen an Schnittstellen

#### 3.2.3.1. Datenbank

- Primär Microsoft SQL
- Optional: Oracle SQL, Microsoft SQL Compact

### 3.2.4. Anforderungen an Software

- Betriebssystem

  Minimum: Windows XP

  Empfohlen: Windows 7

- Microsoft .NET Framework 4.0

### 3.2.5. Anforderungen an Hardware

| System requirements | | |
|---|---|---|
| | Minimum | Empfohlen |
| Arbeitsspeicher | 512 (WindowsXP) | 1024 (Windows Vista, 7) |
| CPU | 1GHz | 2GHz |
| Netzwerk (Internet Zugang) | ADSL 5000KBit | LAN 100MBit |

## 3.3. Design

### 3.3.1. User Interface

Die User Administration nutzt aktuell vier der fünf Bereiche. Die Statusbar wird bislang nicht verwendet. Das Plugin Menü bleibt immer das selbe Control, egal ob die Personenverwaltung, die Rollenverwaltung oder die Berechtigungsverwaltung aktiv ist. Die restlichen Controls werden durch die verschiedenen Verwaltungen geladen. In allen Verwaltungen werden Validierungsfehler in den Übersichten und im Detailbereich mit einer roten Umrandung gekennzeichnet.

#### 3.3.1.1. Personenverwaltung

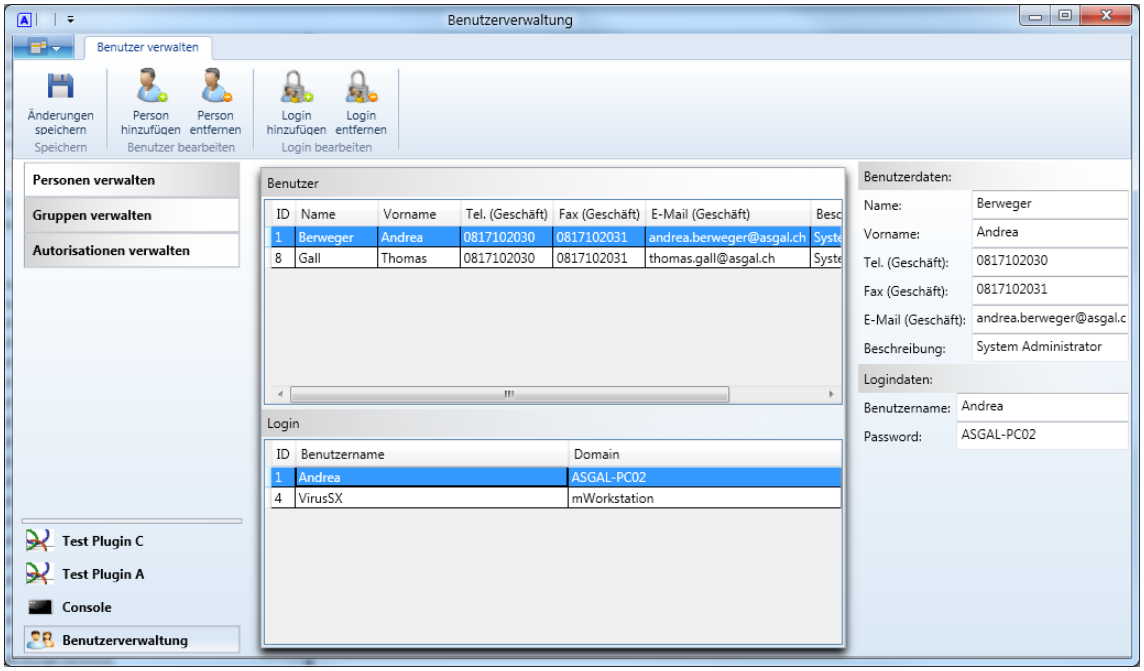


Abbildung 3.2.: User Administration mit aktiver Personenverwaltung

Bei der Personenverwaltung ist im mittleren Bereich oben die Übersicht über alle Personen auf der Datenbank. Gleich darunter ist die Übersicht der Logins der selektierten Person. Im Detailbereich werden die Daten der selektierten Person sowie die des selektierten Logins angezeigt und können da auch geändert werden.

Validierungsfehler:

- Eine Person muss ein Vor- und Nachname definiert haben
- Ein Login muss Benutzername und Domäne definiert haben

Ribbon-Buttons:

- **Speichern:** Speichert alle aktuellen Änderungen in der Personenübersicht.
  Aktiv wenn:
  * Keine Validierungsfehler innerhalb der Personenverwaltung.
- **Person hinzufügen:** Fügt der Personenübersicht einen neuen leeren Datensatz hinzu.
  Aktiv wenn:
  * Berechtigt zum Erstellen von Datensätzen.
- **Person entfernen:** Löscht die selektierte Person aus der Übersicht.
  Aktiv wenn:
  * Berechtigt zum Löschen von Datensätzen.
  * Person in der Übersicht selektiert.

- **Login hinzufügen:** Fügt der selektierten Person ein neues leeres Login hinzu.
  Aktiv wenn:
  * Berechtigt zum Erstellen von Datensätzen.
  * Person in der Übersicht selektiert.
- **Login entfernen:** Löscht das selektierte Login aus der Übersicht.
  Aktiv wenn:
  * Berechtigt zum Löschen von Datensätzen.
  * Person in der Übersicht selektiert.
  * Login in der Übersicht selektiert.

#### 3.3.1.2. Rollenverwaltung

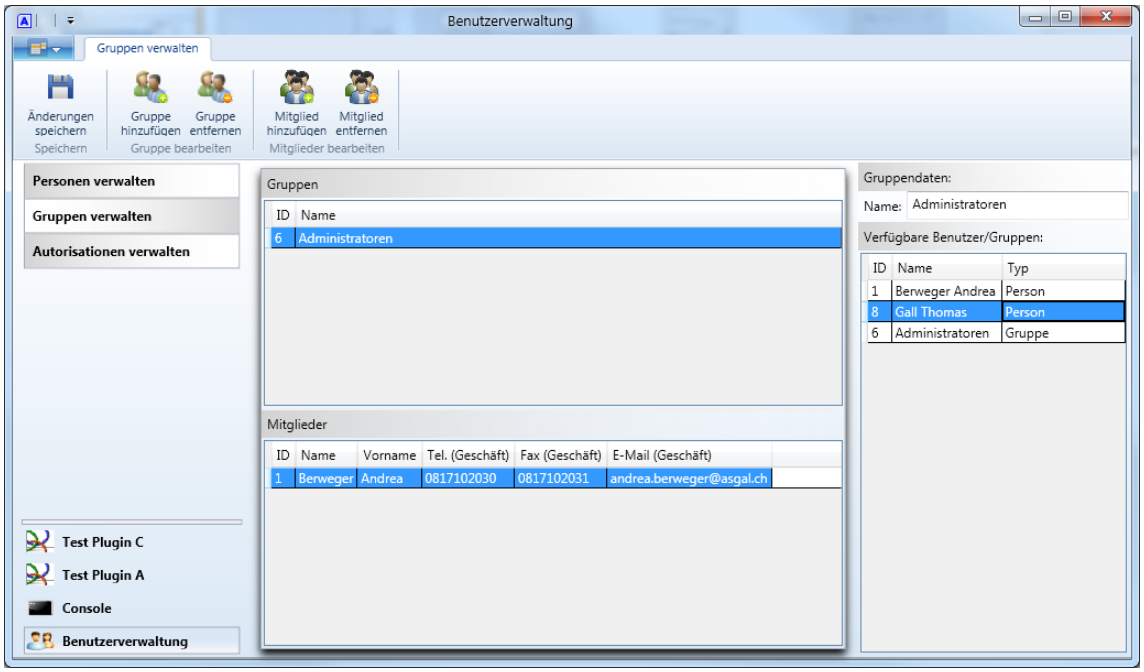


Abbildung 3.3.: User Administration mit aktiver Rollenverwaltung

Bei der Rollenverwaltung ist im mittleren Bereich oben die Übersicht über alle Rollen auf der Datenbank. Gleich darunter ist die Übersicht der Mitglieder der selektierten Rolle. Im Detailbereich wird der Name der selektierten Rolle und eine Auflistung aller verfügbaren Personen und Rollen in der Datenbank angezeigt. Diese Liste wird verwendet, um der selektierten Rolle ein neues Mitglied hinzuzufügen. Ist eine Person selektiert, wird diese, wenn sie nicht bereits als Mitglied definiert ist, hinzugefügt. Ist eine Rolle selektiert, werden alle Mitglieder, die nicht bereits als Mitglied definiert sind, dieser Rolle hinzugefügt.

Validierungsfehler:

- Eine Rolle muss Name definiert haben

Ribbon-Buttons:

- **Speichern:** Speichert alle aktuellen Änderungen in der Personenübersicht.
  Aktiv wenn:
  * Keine Validierungsfehler innerhalb der Rollenverwaltung.
- **Rolle hinzufügen:** Fügt der Rollenübersicht einen neuen leeren Datensatz hinzu.
  Aktiv wenn:
  * Berechtigt zum Erstellen von Datensätzen.
- **Rolle entfernen:** Löscht die selektierte Rolle aus der Übersicht.
  Aktiv wenn:
  * Berechtigt zum Löschen von Datensätzen.
  * Rolle in der Übersicht selektiert.
- **Mitglied hinzufügen:** Fügt der selektierten Rolle die im Detailbereich selektierte Person oder Rolle hinzu.
  Aktiv wenn:
  * Berechtigt zum Erstellen von Datensätzen.
  * Person in der Übersicht selektiert.
  * Person oder Rolle im Detailbereich selektiert.
- **Mitglied entfernen:** Löscht das selektierte Mitglied aus der Übersicht.
  Aktiv wenn:
  * Berechtigt zum Löschen von Datensätzen.
  * Mitglied in der Übersicht selektiert.

#### 3.3.1.3. Berechtigungsverwaltung

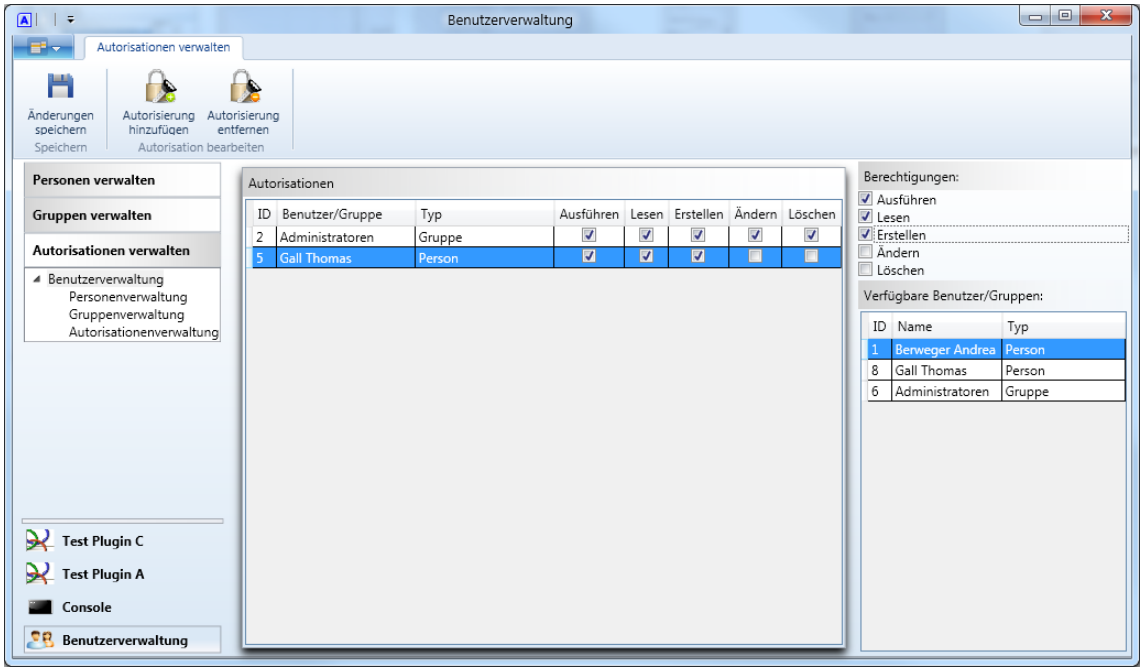


Abbildung 3.4.: User Administration mit aktiver Berechtigungsverwaltung

Bei der Berechtigungsverwaltung wird im Plugin-Menü die Übersicht über alle verfügbaren Funktionen als Baumstruktur dargestellt. Im mittleren Bereich sind alle Personen und Rollen aufgelistet, die für die selektierte Funktion berechtigt sind. Im Detailbereich sind Berechtigungsstufen sowie eine Übersicht über alle verfügbaren Personen und Rollen.

Validierungsfehler:

- Keine

Ribbon-Buttons:

- **Speichern:** Speichert alle aktuellen Änderungen in der Personenübersicht.
  Aktiv wenn:
  * Keine Validierungsfehler innerhalb der Berechtigungsverwaltung.
- **Berechtigung hinzufügen:** Fügt der Berechtigungsübersicht eine neue Person oder Rolle hinzu.
  Aktiv wenn:
  * Berechtigt zum Erstellen von Datensätzen.
  * Funktion in der Baumstruktur selektiert.
  * Person oder Rolle im Detailbereich selektiert.
- **Berechtigung entfernen:** Löscht die selektierte Berechtigung aus der Übersicht.
  Aktiv wenn:
  * Berechtigt zum Löschen von Datensätzen.
  * Berechtigung in der Übersicht selektiert.

### 3.3.2. Domain Model

Das Domain Model der User Administration ist im Anhang C.4 zu finden.

Das oben dargestellte Diagramm zeigt nur den funktionalen Bereich des User Administration Plugins. Das Plugin hat zusätzlich eine Abhängigkeit zum PluginInterface, die hier aus Platzgründen nicht dargestellt ist. Einerseits werden vom PluginInterface die notwendigen Klassen und Interfaces, die zum Entwickeln eines Plugin notwendig sind, zur Verfügung gestellt. Andererseits enthaltet das PluginInterface das Datenbankmodel, mit dessen Hilfe das integrierte Prinzipal die Berechtigungen ermitteln kann. Um die Verwaltung für dieses Datenbankmodel zu realisieren, wird dieses natürlich benötigt.

#### 3.3.2.1. UserAdministrationPluginController

Der Hauptbestandteil ist der Controller, der auch die Basisklasse für die Plugins ableitet. Der Controller ist das Herzstück, das beim Laden des Containers geladen wird. Das Design ist so konzipiert, dass der dargestellte Inhalt des User Administration Plugin bei mehreren Fenster synchronisiert ist und überall die selben Informationen vorhanden sind. Damit alle Fenster mit dem selben Inhalt versorgt werden, wird das laden der Daten aus der Datenbank im Controller bewerkstelligt. Die Daten werden mittels des MVVM Pattern durch die entsprechenden ViewModel gekapselt. Die Hauptaufgabe des Controllers ist somit die Funktionalität des Plugin, sowie das bereitstellen der Daten aus der Datenbank.

#### 3.3.2.2. IUserAdministrationViewModel

Das IUserAdministrationViewModel Interface wird dazu benötigt, dass das PluginMenuControlViewModel die aktuell angezeigte View speichern kann. Bei einem internen Wechsel einer Funktion, zum Beispiel der Wechsel von der Personenverwaltung zur Gruppenverwaltung, wird die Personenverwaltung durch das PluginMenuControlViewModel deaktiviert und die Gruppenverwaltung aktiviert. Nach dem aktivieren wird die Gruppenverwaltung im PluginMenuControlViewModel als aktiv hinterlegt. Das Interface stellt dabei die Methoden zum Aktivieren, bzw. Deaktivieren zur Verfügung.

#### 3.3.2.3. PluginMenuControlViewModel

Dieses ViewModel verwaltet den Zustand des UserAdministrationPlugin und speichert sich, welcher Bereich (Personenverwaltung, usw.) zur Zeit aktiv ist. Es hat eine Referenz zu jedem anderen ViewModel und instanziiert sie gegebenenfalls. Als einziges ViewModel hat es eine Referenz zum Plugin Controller. Diese wird verwendet, um die notwendigen Daten von der Datenbank für die Instanziierung der anderen ViewModel abzurufen.

#### 3.3.2.4. PersonAdminControlViewModel, RoleAdminControlViewModel, AuthorizationAdminControlViewModel

Diese drei ViewModel sind für den Zustand der drei Bereiche zuständig. Sie erhalten bei der Instanziierung die notwendigen Daten vom Controller. Der Kontext, der für den Zugriff auf die Datenbank via Entity Framework notwendig ist, wird ebenfalls durch den Controller zur Verfügung gestellt. Dies ist notwendig, damit bei mehreren Fenster diese synchronisiert sind. Somit wird beispielsweise ein Speichervorgang in einem Fenster automatisch auf allen

angewendet. Die ViewModel stellen eine Bearbeitungstechnik zur Verfügung, die mit dem von Office Word verglichen werden kann. Es können beliebige Änderungen vorgenommen werden. Diese werden erst mit dem Befehl „Speichern“bestätigt und in die Datenbank gespeichert. Sind Validierungsfehler vorhanden, so wird das Speichern unterbunden.

Die ViewModel referenzieren jeweils drei Controls. Diese drei Controls halten sich als DataContext das entsprechende ViewModel, das hauptsächlich zum Binden von Controls benötigt wird. Diese drei Controls werden beim aktivieren jeweils in den Container integriert. Eines der drei Controls ist jeweils vom RibbonTab abgeleitet und erweitert das RibbonMenü des Container. Die Commands für die RibbonButtons sind ebenfalls im ViewModel definiert und werden mittels Binding verlinkt.

### 3.3.3. Interaction Diagramm

Das Sequenzdiagramm der User Administration ist im Anhang C.5 zu finden.

Der UserAdministrationPluginController wird beim Laden des Container durch den PluginManager instanziiert und geladen. Der Controller überprüft bereits bei der Instanziierung, ob der Benutzer berechtigt ist die Benutzerverwaltung zu starten. Kann der Benutzer nicht eindeutig identifiziert werden, wird der Status des Plugin auf „AuthorizationFailed“gesetzt und der PluginManger verwirft es. Es wird somit nicht im User Interface des Containers angezeigt. Kann der Benutzer identifiziert und autorisiert werden, so wird das Plugin geladen. Das User Interface des Plugins ist noch nicht sichtbar. Erst mit der Aktivierung des Plugins durch den Benutzer instanziiert das Plugin das View Model für das Menü. Das ViewModel lädt mit der Instanziierung das PluginMenuControl und setzt es im Container auf den Menübereich. Durch die Aktivierung eines der drei Bereiche durch den Benutzer, wird von diesem das ViewModel erzeugt und aktiviert. Durch die Aktivierung wird der restliche Teil, sprich Hauptbereich, Detailbereich sowie das Ribbon-Menü erweitert. Das obige Diagramm zeigt den Ablauf für den Fall, das der Benutzer berechtigt ist das User Administration Plugin zu laden.

### 3.3.4. Datenbankanbindung

Die Datenbankanbindung für die User Administration wird mittels Entity Framework realisiert und nutzt den DataContext des integrierten Prinzipal des Containers. Aktuell existiert in der Benutzerverwaltung ein statischer Connection String um auf eine Testdatenbank zugreifen zu können. Im weiteren Verlauf des Projektes wird die User Administration mehrere Connection Strings verwalten um mit mehreren Instanzen im Container auf verschiedene Datenbanken zugreifen zu können. Zur Zeit ist es vorgesehen, dass beim Instanziieren je nach Rollenangehörigkeit die Verbindung zur Datenbank gewählt werden kann.
Ein noch offenes Problem im Zusammenhang mit Connection Strings liegt darin, wie man es bewerkstelligen kann, die Anmeldeinformation, die notwenig ist um auf die Datenbank zu verbinden, zu verschlüsseln. Die Anmeldeinformation in den Code integrieren ist kein Ansatz, da der IL (Intermediate Language) mit einfachen Mitteln eingesehen werden kann. Um eine Verschlüsselung zu bewerkstelligen benötigt man einen Key. Dieser Key muss aus einer Quelle stammen die nicht mit dem Code zusammenhängt, mit anderen Worten durch den Benutzer eingegeben werden muss. Das möchte man aber aus Anwendungsfreundlich-

keit vermeiden.
Ein Lösungsansatz besteht darin, die Connection Strings in einer externen Datei, zum Beispiel in einer Konfigurationsdatei zu hinterlegen. Bei der Installation, ist in der Konfigurationsdatei die Anmeldeinformation im Klartext enthalten. Bei der Installation werden die Anmeldeinformationen mittels der Arbeitsstationsidentifikation verschlüsselt und in der Konfigurationsdatei überschrieben. Beim laden des Plugin kann die Anmeldeinformation mittels der Arbeitsstationsidentifikation entschlüsselt werden. Der Nachteil liegt darin, dass bis zur Installation die Anmeldeinformation immer noch im Klartext vorliegt. Um diese Problem ebenfalls zu umgehen, kann man einen Webservice in Erwägung ziehen, bei dem eine Anfrage mit Übermittlung der Arbeitsstationsidentifikation das Passwort verschlüsselt abgerufen wird. Das Plugin kann somit nicht mehr auf eine andere Arbeitsstation verschoben werden, da in dem Fall die Anmeldeinformation nicht mehr korrekt entschlüsselt werden kann.

# 4. Code Test

## 4.1. Unittest

Die Entwicklung des Container sowie dem User Administration Plugin wird mit Hilfe von Unittests unterstützt. Durch den Einsatz von WPF und MVVM kann auch ein Teil der User Interface Funktionalität getestet werden, was den manuellen Testaufwand reduziert, Zeit einspart und die Qualität der Software verbessert.

## 4.2. Usertest

Da zu Beginn der Projektplanung eine Prototypenplugin geplant war, dass dem Kunden nutzbare Funktionen bietet, wurden Usertests eingeplant, die ein Feedback über die Neuentwicklung bringen sollte. Da während der Entwicklung das Prototypenplugin ersetzt wurde und dieses dem Kunden keine nutzbare Funktionalität liefert, machte es keinen Sinn, Usertests durchzuführen. Die Usertest können erst Sinnvoll durchgeführt werden, wenn weitere Plugins entwickelt sind.

# 5. Ergebnis und Zukunft

## 5.1. Ergebnis

Das Resultat des Projektes ist ein Container der mittels Plugins erweitert werden kann. Der Hauptbestandteil ist das User Interface, das von den Plugins genutzt werden kann sowie ein Subsystem, dass die Authentisierung sowie die Autorisierung der Plugins unterstützt. Die Authentisierung des Subsystems identifiziert den Benutzer Anhand dem Windowsbenutzername und der Domäne. Somit ist es für den Benutzer nicht notwendig ein Passwort einzugeben.
Der Container behandelt Fehler die nicht eigens durch ein Plugin behandelt werden, in dem er den Fehler isoliert, loggt und das verursachende Plugin neu startet. Somit bleiben die andere Plugins unberührt.
Das User Interface besitzt Feel & Style vom Microsoft Outlook 2010.
Zusätzlich wurde eine erstes Prototypenplugin entwickelt mit dessen Hilfe die Authentisierungen und Autorisierungen des Subsystems verwaltet werden kann. Das Plugin hat gezeigt, dass sowohl der Container sowie auch das Subsystem funktionsfähig sind und den Erwartungen entsprechen.

Als implizites Resultat hat die ASGAL Informatik GmbH eine Technologiesprung erfahren. Es ist viel Erfahrung und Knowhow über die neuen Technologien von Microsoft gesammelt worden.

## 5.2. Zukunft und Vision

Für die Zukunkt des .NET Application Container sind bereits konkrete Erweiterungen geplant:

Container:

- Das Laden der Plugins wird verbessert, damit sie bei Exceptions isoliert behandelt werden können.
- Das User Interface wird mehrsprachig gestaltet. Dies ist zwingend notwendig, da bereits italienisch und französisch sprechende Kunden von der ASGAL Informatik GmbH bedient werden.
- Konzept um eine Kommunikation unter den Plugins zu ermöglichen.

Plugins:

- Benutzerverwaltung wird erweitert, damit mehrere Instanzen mit unterschiedlichen Datenbanken im Container geladen werden können.

- Plugin um Daten manuell zu erfassen.
- Plugin um vorhandene Daten in Excel-Tabellen zu importieren.
- Plugin um vorhandene Daten aus Datenbank zu importieren.
- Plugin um importierte Daten auszuwerten.

# A. Technologien

## A.1. User Interface

Das .NET Framework von Microsoft bietet entweder Winforms oder WPF zu erstellen von User Interfaces an. Bei beiden sind vordefinierte Controls vorhanden um die Entwicklung zu erleichtern. Natürlich ist es auch möglich eigene Controls zu zeichnen, was aber in den meisten Fällen zu aufwändig und beim Angebot der Verfügbaren Controls auch nicht notwendig ist.

### A.1.1. WinForms

WinForms wurden mit dem ersten Framework (.NET Framework 1.0) im Jahre 2002 eingeführt. Diese erste Version wurde für Windows 98, NT 4.0, Windows 2000 und Windows XP veröffentlicht. Es enthält die wichtigsten Controls, die man seit je her von Windows kennt. Mit .NET 2.0 wurde der Umfang der Controls nochmals überarbeitet und wird bis heute in der Version 2.0 genutzt. Es bietet nur wenig Möglichkeiten Controls zu binden. Die Grafiken der Controls sind Rastergrafiken. Das erstellen von Formen wird von Visual Studio mittels eines Designers unterstützt. Bei WinForms wird der Designcode wie Code-Behind in der gewählten Sprache (C# oder VB) erzeugt.

### A.1.2. WPF - Windows Presentation Foundation

Da bei Betriebssystemen in den letzten 5 Jahren immer mehr Wert auf schönes Design und Animationen gelegt wurde, hat Microsoft mit dem .NET 3.0 WPF eingeführt, welches die WinForms ablöst. WPF bietet viel detailliertere grafische Aspekte. Der Code, welches das Design darstellt, wurde auf eine an XML angelehnte Beschreibungssprache mit dem Namen XAML ausgelagert. Der XAML Code bietet sehr ausgeprägte Möglichkeiten um den Code-Behind mittels Bindings zu minimieren. Dies bietet den Vorteil, dass der Code der Applikation durch einen professionellen Software Entwickler erzeugt werden kann und das Design der Applikation von einem professionellen Designer. Aus diesem Grund wurde auch die Entwicklungsumgebung aufgeteilt in Coding mittels Visual Studio und Desgin mittels Expression Blend. Visual Studio bietet zwar auch das designen von User Interfaces an, ist aber in den Möglichkeiten wie zum Beispiel Animationen beschränkt. Der XAML Code bringt zusätzlich die Möglichkeit, dass das User Interface in einem Webbrowser angezeigt werden kann. Beim Anzeigen in einem Browser wird die gesamte Applikation auf den Client geladen (Zum Vergleich siehe Silverlight). Der Webbrowser benötigt aber trotzdem ein entsprechendes Plugin um die Applikation darzustellen. Der XAML Code wird zusätzlich auf einen logischen Baum und einen visuellen Baum aufgeteilt. Dies ermöglicht das Traversieren über den XAML Code um zum Beispiel die grafischen Aspekte des User Interface zu ändern (Themes, Skins, usw.).
WPF wurde mit dem .NET Framework 4.0 weiterentwickelt und mit zusätzlichen Controls ausgestattet.

#### Aufgetretene Probleme

- Es kann vorkommen das eine Bindingdefinition im XAML Visual Studio zum Absturz bringt. Im dümmsten Fall, wenn das XAML so gespeichert wird, stürzt Visual Studio bei jedem Versuch das File wieder zu öffnen ab. In diesem Fall muss die Bindingdefinition in einem externen Editor entfernt werden.

- Dieses Problem bezieht sich auf Visual Studio 2010 und wird mehr durch den Kompiler erzeugt, als dass es ein Problem seitens WPF ist. Es gibt in WPF eine Konstellation, die dazu führt, dass ein mittels XAML definiertes UserControl, wenn es in einem anderen XAML eingesetzt wird, nicht mit einem Name, bzw. x:Name versehen werden kann. In diesem Fall muss das UserControl ohne XAML in einer normalen cs-Datei via Code erzeugt werden. Einen besseren Workaround konnte noch nicht gefunden werden.

### A.1.3. Silverlight

Silverlight wurde für RIA (Rich Internet Application) entwickelt. Silverlight ist ein Subset von WPF und kann nur innerhalb einer virtuellen Umgebung für Silverlight genutzt werden (Gleichzustellen mit Flash & ActionScript). Diese virtuelle Umgebung wird von Webbrowsern zur Verfügung gestellt, ist aber auch als eigenständiger Container verfügbar. Silverlight ist nur im Zusammenhang mit einem Webserver lauffähig. Es benötigt immer eine Remote Kommunikation wie zum Beispiel WCF (Windows Communication Framework). Es wird lediglich das User Interface vom Server geladen und angezeigt (Zum Vergleich siehe WPF). Die Entwicklung von Silverlight kann auch in Visual Studio oder Expression Blend erfolgen, wobei auch hier der grafische Umfang nur von Expression Blend voll ausgenutzt werden kann.

### A.1.4. Fazit

Da Business Applikationen häufig keinen Wert auf Design und Animationen legen, sondern auf Funktionalität, kann WPF als Overkill angesehen werden. Jedoch in Anbetracht der Tatsache, dass bei WPF problemlos auf die grafischen Aspekte verzichtet werden kann und einfach nur die vom .NET Framework zur Verfügung gestellten Controls mittels Visual Studio auf gleiche Art und Weise wie WinForms genutzt werden kann, macht es keinen Unterschied welche Technologie genutzt wird. Es kann aber damit gerechnet werden, dass WinForms nicht mehr weiter entwickelt wird und stetig durch WPF ersetzt wird. Als bestes Beispiel kann man Visual Studio nehmen, welches in der Version 2010 mittels WPF realisiert wurde. WPF bietet keine Nachteile gegenüber WinForms, auch wenn die Komplexität abschreckend wirken kann in Anbetracht der Tatsche, dass viel grafische Funktionalität von WPF häufig nicht gebraucht wird. Die Trennung von User Interface mittels XAML ist auf jeden Fall ein Vorteil der das Design der gesamten Applikation flexibler gestaltet.

Durch die Nutzung von WPF sowie MVVM und Bindings wird die Testbarkeit der Application deutlich erhöht. Zu Begin wirkt MVVM ein wenig unübersichtlich, ist jedoch wenn man die Prinzipien, zum Beispiel Commands, verstanden hat, sehr hilfreich. Auch die offene Möglichkeit zum Einsatz von grafischen Aspekten kann sehr praktisch sein. Umfangreiche Applikationen können beispielsweise durch farbliche Aspekte übersichtlicher gestaltet werden um verschiedene, oder eventuell sogar sehr ähnlichen Fenster, optisch

von einander differenzieren zu können.

Der Einsatz von Silverlight macht nur Sinn, wenn eine Rich Internet Application (RIA) entwickelt werden soll.

## A.2. Datenbank

Im Framework von Microsoft findet man hauptsächlich zwei Ansätze für die Anbindung an eine Datenbank. Das eine Konzept existiert schon seit das Framework existiert und verarbeitet die Anfragen mittels OleDBDataProvider, welcher DataTable-Objekte in einem Dataset ablegt. Der zweite Ansatz wurde mit dem Framework 3.5 eingeführt und ist unter dem Name Microsoft Entity-Framework bekannt.

### A.2.1. OleDbDataProvider Datenbank Anbindung

Dieses verfahren ist schon seit geraumer Zeit im Framework integriert. Die Verbindung zur Datenbank wird mit Hilfe von OleDBConnection-Objekten realisiert. Das Abrufen der Daten geschieht über OleDBDataAdapter oder OleDbDataReader, welche wiederum mittels OleDbCommand mit den nötigen Informationen versorgt werden. Die OleDbCommands enthalten die SQL-Strings, welche den Inhalt der Abfrage darstellen. Die Commands unterstützen parameterisierte Abfragen. Bei verschiedenen Datenbanken ist der Programmierer selbst dafür verantwortlich, dass der SQL-String für die entsprechende Datenbank ausgelegt ist. Transactions werden ebenfalls unterstützt, müssen jedoch auch explizit durch den Entwickler mittels OleDbTransaction erzeugt werden. Mit Hilfe des OleDbDataReader können Daten seriell via Schleife abgerufen werden, oder es kann mittels OleDbDataAdapter ein DataTable-Ojekt erzeugt werden, welches zusätzlich noch in einem Dataset hinterlegt werden kann. Das Konzept ist im Vergleich zum Entity-Framework nicht typisiert. Dies bietet aber die Möglichkeit für sehr komplexe Abfragen und Zusammenstellungen von Tabellen. Als ein komplexes Beispiel kann man das übersetzen von Zeilen in Spalten nennen. Da solche komplexen Abfragen jedoch eher selten sind und meist nur einfache Abfragen notwendig sind, ist die nicht typisierte Darstellung der Tabellen mühsam und erfordern häufiges casten. Ebenfalls müssen Verbindung, Commands und DataAdapter und/oder DataReader manuell erzeugt werden.

### A.2.2. Entity-Framework

In letzter Zeit sind bei verschiedensten Programmiersprachen die OR-Mapper immer beliebter geworden. Diese OR-Mapper, unter welches auch das Entity-Framework fällt, übernehmen einen grossen Teil, der zur Verbindung einer Datenbank notwendig ist. Zusätzlich werden die Tabellen einer Datenbank auf Klassen, bzw. Objekte gemappt, was eine stark typisierte Abfrage zur folge hat. Das Entity-Framework von Microsoft entspricht solch einem OR-Mapper. Der Vorteil liegt darin, dass die typisierten Tabellen ein einfaches programmieren erlauben. Man muss sich nicht um Verbindungen oder Transactions kümmern. Es wird ein Kontext erzeugt, der über die verschiedenen Tabellen, bzw. Klassen/Objekte informiert ist. Der Kontext sowie dessen Klassen kann mit Hilfe von Designer-Tool oder direkt aus einem Datenbankschema erzeugt werden. Sämtliche Abfragen auf die Datenbank werden über den Kontext mittels Linq-Expressions erledigt. „Linq to Entity“ist eine standardisierte Abfragesprache, die mittels Kontext automatisch auf die darunterliegende

Datenbank übersetzt wird. Voraussetzung dafür ist natürlich, das der Provider der jeweiligen Datenbank unterstützt wird. Es entsteht der Nachteil, dass wenn die Datenbank nicht unterstützt wird, diese natürlich auch nicht angesprochen werden kann. Der Vorteil ist, dass man sich als Entwickler nicht um korrekte SQL-Statements für die Datenbank kümmern muss, sonder nur die Linq-Expressions verstehen muss.

Nachfolgend werden zwei „Best Practices “aufgelistet.

1. Der Kontext wird geshared. Die Klassen, die einen Zugriff auf die Datenbank benötigen, können mittels des zugänglichen Kontext die Abfrage durchführen.

2. Der Kontext wird mit Hilfe eines Repositories gekappselt. Es wird eine Repository-Klasse erzeugt, die, die von der Applikation benötigten Abfragen mittels Methoden zur Verfügung stellt. Dies hat den Vorteil, dass der Kontext von nur noch einer Klasse zugegriffen wird. Hat aber natürlich den Nachteil, dass für alle verschiedenen Datenbankzugriffe eine Methode zur Verfügung gestellt werden muss. Dieser Ansatz wird auch im Zusammenhang mit WCF genutzt, da über WCF eine Schnittstelle von Methoden zur Verfügung gestellt werden kann.

#### Aufgetretene Probleme

- Vorsicht: Beim Generieren der Datenbank aus dem Model, werden alle Tabellen gelöscht und neu erstellt. Dieser Vorgang ist höchstens initial sinnvoll, andernfalls muss ein Backup von Inhalt der Datenbank gemacht werden.

- Das definieren von unique Properties, bzw. einer Datenbankspalte ist nicht möglich über das Model. Es kann nachträglich in der Datenbank gesetzt werden, muss dann jedoch im Programmcode die Überprüfung vornehmen, da sonst eine Exception auftritt.

- Beim Ändern des Model kann ein Verlust der Zuordnungen zwischen Property und Datenbankspalte auftreten. (Ist weniger ein Problem jedoch trotzdem mühsam.)

- Beim Entwickeln mit Entities kann es zu Problemen beim Vergleichen kommen. Bei EntityCollections wird bei der Methode Contains nicht die „Equals“Methode des EntityObjects genutzt. Bei EntityCollections wird zum Vergleich direkt die interne „GetHashCode“Methode genutzt. Möglicher Workaround ist die Iteration oder besser „Linq to Collection“mit Hilfe eines entsprechenden Where-Teils.

### A.2.3. Fazit

Das Entity-Framework ist gut in die Entwicklungsumgebung Visual Studio integriert und bietet ein einfaches und schnelles Entwickeln einer Datenbankanbindung. Neu kann im Framework 4.0 und Visual Stuido 2010 auch die Datenbank mittels Model erzeugt werden. Dabei ist jedoch Vorsicht geboten, da das erzeugt SQL-Skript vorhandene Tabellen löscht und neu erzeugt. Dies kann bei schon vorhandener Daten zu enormem Datenverlust führen. Besser ist es in so einem Fall, die Datenbank anzupassen, und die Änderungen ins Model übernehmen lassen.

Da ein grosser Teil aller Anfragen einer Datenbank aus eher einfacheren Skripts besteht, reicht die Funktionalität des Entity-Frameworks aus, etwaige komplexe Statements müssen explizit mittels OleDb ausgeführt werden. Ein Nachteil, welches man jedoch zugleich auch als Vorteil nehmen kann, ist, dass nicht jede Datenbank unterstützt wird. Der Vorteil, der aber daraus entsteht ist, dass man sich nicht um die Notationen der jeweiligen Datenbanksprache kümmern muss, sondern alles mittels Linq-Expressions erledigen kann.

## A.3. Komponentenbasiertes Entwickeln

Komponenten basierendes Entwickelt stellt die Tatsache von der dynamischen Einbindung von Klassenbibliotheken (DLL's) dar. Es existieren schon lange Techniken und Konzepte zum dynamischen Einbinden von Klassenbibliotheken. Diese wurden jedoch mit dem Framework 4.0 deutlich vereinfacht.

### A.3.1. Managed Extenisbility Framework (MEF)

Mit dem .NET Framework 4.0 wurde MEF eingeführt, was für „Managed Extensibility Framework "steht. Dieses Framework bietet Methoden an um Klassenbibliotheken nachladen zu können. Dabei wird mit einem Katalog gearbeitet, der die Klassenbibliotheken beinhaltet. Es existieren verschiedene Typen von Katalogen und sie könne beliebig mit einander kombiniert werden. Es gibt Kataloge, die die Klassenbibliotheken von einem Laufwerk, bzw. aus einem Ordner auf dem Dateisystem laden, oder zum Beispiel aus dem aktuell laufenden Assembly usw. Aus diesen Katalogen kann mittels Export die exportierten Klassen, bzw. Methoden extrahiert werden. Alle Klassen, bzw. Methoden die exportiert werden möchten, müssen mittels Anotations markiert werden. Alle Exporte können zusätzlich mit Metadaten versehen werden. Das MEF unterstützt lazy loading und ist einfach zu bedienen.

### A.3.2. Reflection

Ein etwas älteres Konzept ist das Laden von Klassenbibliotheken mittels Reflection. Es wird aus einer Klassenbibliothek zur Laufzeit ein Assembly erzeugt, aus welchem wiederum mittels Reflection die vorhanden Typen, Felder, Properties oder Methoden ausgelesen werden können. Reflection ist eine Technik bei welcher mittels Methodenname als String die entsprechende Methode innerhalb einer Klasse gesucht wird und im Fall eines fündig werdens ein MethodInfo-Objekt zurückgegeben wird. Dieses Objekt kann anschliessend zum aufrufen dieser Methode genutzte werden. Das Prozedere funktioniert wie schon erwähnt auch bei Typen, Properties oder Feldern. Der Vorteil dieser Strategie ist, dass man das Laden in eine andere Applikationsdomäne auslagern kann.

### A.3.3. Fazit

Dynamisches einbinden von Klassenbibliotheken gibt es bereits schon lange. Das Managed Extensibility Framework (MEF) vereinfacht das vorgehen massiv, schränkt dadurch jedoch auch die Möglichkeiten ein. Benötigt man keine komplexen Elemente beim laden einer Klassenbibliothek, so ist das MEF sicherlich eine vorteilhaftes Konzept, da es einfach und schnell angewendet werden kann. Bei komplexeren Sachen muss alternativ auf Reflection ausgewichen werden.

## A.4. User Interface Menüs

Über lange Zeit bestand bei den meisten Applikationen das Standardmenü (Datei, Bearbeiten, Ansicht, usw.). Diese wurde auch durch die Designspezifikationen von Microsoft propagiert und war ein Muss für eine Applikation um Microsoft zertifiziert zu sein. Mit Office 2007 führte Microsoft erstmals seit langer Zeit ein neues Menüsystem ein, die Ribbonleiste.

### A.4.1. Standardmenü

Der grosse Vorteil dieses Menüs ist, dass dieses von den meisten Benutzern als gewohnt eingestuft wird und der Aufbau sehr simpel ist. Ansonsten gibt es keine gravierenden Vorteile.

### A.4.2. Office Ribbon

Erstmals mit Office 2007 wurde die Ribbonleiste eingeführt. Diese führte zu grossem Gesprächsstoff und wurde meistens von alt eingesessenen Benutzern boykottiert, da sie sich auf einmal nicht mehr in ihrer gewohnten Umgebung befanden und alle Funktionen wie ein blutiger Anfänger wieder suchen musste. Die Ribbonliste ist definitiv gewöhnungsbedürftig, ist aber wenn man sich daran gewöhnt einfacher und schneller zu bedienen als die alten Menüs. Sie bringt auch für die immer wichtiger werdende Optik der Applikation einen gewissen Touch. Da die Ribbonliste nun auch in Office 2010 wieder integriert wurde und ebenfalls seinen Platz im Windows Paint gefunden hat (Erst ab Windows 7), macht es den Anschein, als würde sich diese Menüliste langsam etablieren. Um die eigene Applikation mit der offiziellen Ribbonliste von Microsoft auszustatten, muss zuerst die RibbonControlLibrary mittels Registrierung heruntergeladen werden. Die Registrierung bezieht sich nicht nur auf die Person, sondern schränkt die Nutzung gleich auch noch auf die in der Registrierung zwangsläufige anzugebende Applikation ein. Dies ist so realisiert, da Microsoft anscheinend ein Patent auf dieses Control besitzt. Nach dem Herunterladen kann die eigene Applikation mit der Ribbonliste versehen werden. Dadurch entsteht durchaus ein etwas neuzeitlicher optischer Eindruck. Die Entwicklung zeigt sich aber manchmal etwas harzig, da das Control doch noch da und dort Kinderkrankheiten beinhaltet.

#### Aufgetretene Probleme

- Das RibbonComboboxControl kann nur mit RibbonComboboxItems gefüllt werden. Falls andere Objekte abgefüllt werden müssen, dient der Workaround, in dem einfach ComboboxItems hinzufügt und deren DataContext-Property auf das gewünschten Objekt gesetzt wird. Der Zugriff auf das Objekt erfolgt dann auch wieder via DataContext.

- Vorsicht beim verwenden von ContextualTabGroups. Beim entfernen kann es zu inkonsistentem Zustand zwischen normalen und gruppierten Tabs („ContextualTabGroup“) kommen. Es kann sein, dass die Tabs sich nicht mehr entfernen lassen.

- Wird das RibbonWindowControl anstatt einem normalen WindowControl eingestzt, um die Ribbonliste in das Fenster zu integrieren, so wird beim Maximieren der Applikation der obere Rand schwarz und die Buttons zum Minimiern, Maximieren und

Schlissen des Fensters sind nicht mehr sichtbar. Dieses Verhalten ist nur, wenn mehr als ein Bildschirm angeschlossen ist. Ist am Computer nur ein Bildschirm angeschlossen oder aktiv, so funktioniert es korrekt.

- Die Tooltipps der RibbonCommands lassen sich setzen und sie werden im User Interface angezeigt. Jedoch beim Rücksetzen, sprich auf null setzen, werden sie immer noch angezeigt. Mit anderen Worten lassen sie sich nicht mehr entfernen.

- Das entfernen des letzten Tab aus der Liste hat zu folge, dass die Gruppen, die im Tab definiert waren, in der Ribbon zurück bleiben (Siehe Bild unten). Als Workaround muss die Visibility aller Gruppen auf „Hidden“gesetzt werden.


Abbildung A.1.: RibbonTab Error: Kein Tab mehr vorhanden, aber die Gruppen werden noch angezeigt.

# B. Administratives

## B.1. Projektplannung

Wie es häufig der Fall ist, ändert sich der Projektplan ständig. Durch unberechenbare Vorkommnisse muss der Plan immer wieder angepasst werden.

- Der Zeitaufwand für das erarbeiten des Knowhow für die neuen Technologien wurde zu Beginn zu wenig Zeit eingeplant. Die zu kurze Planung für das Erarbeiten der Technologien dient aber auch als Einschränkung, da sich für Technologiestudien sonst keine Grenze bietet. Das erarbeiten der Technologien ist aber allgemein sehr wichtig und bringt grosse Vorteile für weitere Projekte.
- Die Planung für das erstellen eines Abstraktion-Layer wurde ganz aus dem Projekt gestrichen. Die Begründung für die Streichung ist im Design des Containers unter „Datenbank“beschreiben.
- Ursprünglich, war im Projektplan eingeplant, dass das Prototypenplugin als Neuentwicklung einer Komponente eines bereits vorhandenen Projektes realisiert wird. Da sich ein komplettes Redesign des vorhandenen Projektes nicht lohnen würde, hat man diese Idee verworfen und dafür ein universelles Plugin zur Verwaltung der Benutzer für das Subsystem entwickelt.
- Der Benutzertest konnte noch nicht durchgeführt werden. Die Begründung dafür ist im Kapitel „Code Test“zu finden.
- Die geplante Bedienungsanleitung für die Benutzerverwaltung wurde aus zeitlichen Gründen noch nicht erstellt.

### B.1.1. Projektplan

Der Projektplan ist im Anhang C.6 zu finden.

## B.2. Arbeitsreport

| Datum | Startzeit | Endzeit | Differenz | Beschreibung |
|---|---|---|---|---|
| 22.02.2010 | 15.30 Uhr | 17.00 Uhr | 01:30 | Kickoff Meeting |
| 23.02.2010 | 9.00 Uhr | 12.00 Uhr | 03:00 | Server aufsetzen (SVN, SSH) |
| 23.02.2010 | 13.30 Uhr | 17.00 Uhr | 03:30 | Server aufsetzen (SVN, SSH) |
| 25.02.2010 | 9.00 Uhr | 12.00 Uhr | 03:00 | Doku vorbereitet |
| 25.02.2010 | 13.00 Uhr | 17.00 Uhr | 04:00 | Doku vorbereitet |
| 26.02.2010 | 13.00 Uhr | 17.00 Uhr | 04:00 | TechResearch |
| 01.03.2010 | 9.00 Uhr | 12.00 Uhr | 03:00 | TechResearch |
| 01.03.2010 | 13.30 Uhr | 16.30 Uhr | 03:00 | TechResearch |
| 05.03.2010 | 9.00 Uhr | 12.00 Uhr | 03:00 | SQL-Server aufsetzen & Konfigurieren |
| 05.03.2010 | 13.30 Uhr | 17.00 Uhr | 03:30 | TechResearch |
| 12.03.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | TechResearch |
| 12.03.2010 | 13.00 Uhr | 18.00 Uhr | 05:00 | TechResearch |
| 15.03.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | TechResearch |
| 15.03.2010 | 13.30 Uhr | 15.30 Uhr | 02:00 | TechResearch |
| 16.03.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | TechResearch |
| 16.03.2010 | 13.00 Uhr | 18.00 Uhr | 05:00 | TechResearch |
| 17.03.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | TechResearch |
| 17.03.2010 | 13.00 Uhr | 15.30 Uhr | 02:30 | TechResearch |
| 18.03.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | TechResearch |
| 18.03.2010 | 14.00 Uhr | 15.00 Uhr | 01:00 | Meeting |
| 19.03.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | TechResearch |
| 19.03.2010 | 13.30 Uhr | 17.50 Uhr | 04:20 | TechResearch |
| 22.03.2010 | 8.10 Uhr | 8.36 Uhr | 00:26 | Zeiterfassung nachgetragen in TimeCatcher |
| 22.03.2010 | 8.36 Uhr | 11.56 Uhr | 03:20 | Requirements Document |
| 22.03.2010 | 13.31 Uhr | 14.40 Uhr | 01:08 | Requirements Document |
| 22.03.2010 | 14.40 Uhr | 15.06 Uhr | 00:26 | Documentation |
| 23.03.2010 | 7.53 Uhr | 11.39 Uhr | 03:45 | Requirements Document |
| 23.03.2010 | 13.30 Uhr | 17.44 Uhr | 04:14 | Requirements Document |
| 24.03.2010 | 8.07 Uhr | 9.45 Uhr | 01:37 | Requirements Document |
| 24.03.2010 | 9.46 Uhr | 11.23 Uhr | 01:37 | Documentation |
| 26.03.2010 | 8.18 Uhr | 8.40 Uhr | 00:21 | Meeting Protokoll 24.03.2010 |
| 29.03.2010 | 10.15 Uhr | 11.09 Uhr | 00:54 | Requirement Document |
| 29.03.2010 | 11.35 Uhr | 11.55 Uhr | 00:19 | Requirements Document |
| 29.03.2010 | 13.24 Uhr | 15.02 Uhr | 01:37 | - Requirements Document - Outlook Style & Feel mit WPF |
| 30.03.2010 | 13.30 Uhr | 16.50 Uhr | 03:20 | - Use Cases (Fully Dressed) - Requirements angepasst |
| 30.03.2010 | 17.45 Uhr | 18.18 Uhr | 00:33 | -Requirement Document angepasst - Use Case (Fully Dressed) |
| 30.03.2010 | 18.19 Uhr | 18.22 Uhr | 00:03 | - Requirements Document angepasst |
| 31.03.2010 | 14.27 Uhr | 15.00 Uhr | 00:33 | - MEF - Domain Model |
| 01.04.2010 | 8.16 Uhr | 10.50 Uhr | 02:33 | - MEF analysiert für Domainmodel vom Controller<br>- MEF beispiel programmiert<br>- Documentation Biblihography mit MEF Websites erweitert |
| 01.04.2010 | 11.45 Uhr | 13.45 Uhr | 02:00 | - Domain Model (Problem Domain, Plugin)<br>- RelayCommand Pattern |
| 06.04.2010 | 8.00 Uhr | 9.00 Uhr | 01:00 | - Meeting Protokoll<br>- Projektplan |
| 06.04.2010 | 9.45 Uhr | 11.58 Uhr | 02:13 | -Ribbon design analysiert |
| 06.04.2010 | 13.12 Uhr | 13.24 Uhr | 00:11 | - Ribbon design analysiert |
| 06.04.2010 | 13.44 Uhr | 14.36 Uhr | 00:52 | - Ribbon design analyse |
| 06.04.2010 | 14.43 Uhr | 17.46 Uhr | 03:03 | - Ribbon design analyse<br>- Relay-/Routed-/Ribbon-Commands |
| 07.04.2010 | 8.23 Uhr | 9.36 Uhr | 01:12 | - CommandParameter binding Probleme analysiert |
| 07.04.2010 | 14.37 Uhr | 15.48 Uhr | 01:10 | - Datagrid analyse |
| 08.04.2010 | 8.21 Uhr | 10.48 Uhr | 02:27 | - Ribbon Command & Bindings |
| 08.04.2010 | 10.52 Uhr | 10.54 Uhr | 00:01 | - Duration Property (Transient) Probleme bearbeitet |
| 08.04.2010 | 10.55 Uhr | 11.13 Uhr | 00:18 | - Duration TimeCatcher |
| 08.04.2010 | 11.13 Uhr | 11.41 Uhr | 00:28 | - Database TimeCatcher verschoben |
| 08.04.2010 | 11.41 Uhr | 11.46 Uhr | 00:05 | - kleine Änderung betreffend Funktionalität |
| 08.04.2010 | 13.34 Uhr | 15.52 Uhr | 02:17 | - Domain Model ApplicationContainer (MEF) - Sequenzdiagramm ApplicationContainer (MEF) |
| 09.04.2010 | 8.43 Uhr | 11.48 Uhr | 03:05 | - Domain Model ApplicationContainer |
| 09.04.2010 | 13.41 Uhr | 18.04 Uhr | 04:23 | - Domain Model |
| 13.04.2010 | 10.36 Uhr | 11.46 Uhr | 01:09 | - DomainModel verbessert: Component Factory eingeplant |
| 13.04.2010 | 13.37 Uhr | 16.14 Uhr | 02:37 | - PluginInterface implementiert |
| 14.04.2010 | 9.15 Uhr | 10.00 Uhr | 00:45 | - Meeting Protokoll 12.04.2010 |
| 12.04.2010 | 14.00 Uhr | 15.30 Uhr | 01:30 | - Domainmodel - Sequenzdiagramme |
| 12.04.2010 | 16.00 Uhr | 17.00 Uhr | 01:00 | - Meeting |
| 04.03.2010 | 16.00 Uhr | 17.00 Uhr | 01:00 | - Meeting |
| 18.03.2010 | 16.00 Uhr | 17.00 Uhr | 01:00 | Meeting |
| 24.03.2010 | 16.00 Uhr | 17.00 Uhr | 01:00 | Meeting |
| 01.04.2010 | 16.30 Uhr | 17.30 Uhr | 01:00 | - Meeting |
| 15.04.2010 | 11.03 Uhr | 11.17 Uhr | 00:13 | - Domainmodel anpassungen |
| 20.04.2010 | 8.30 Uhr | 9.29 Uhr | 00:59 | - Domainmodel anpassungen - Implementation TestPlugin |
| 20.04.2010 | 12.38 Uhr | 17.00 Uhr | 04:21 | - Implementation PluginManager |
| 20.04.2010 | 20.48 Uhr | 20.53 Uhr | 00:04 | - Projekt auf Notebook vorbereitet |
| 20.04.2010 | 21.16 Uhr | 21.18 Uhr | 00:01 | - Projekt angepasst |
| 21.04.2010 | 7.37 Uhr | 16.00 Uhr | 08:22 | - ApplicationController implemented - GUI Design started |
| 21.04.2010 | 13.10 Uhr | 16.00 Uhr | 02:50 | - ApplicationController implemented - GUI Design started |
| 22.04.2010 | 8.12 Uhr | 8.53 Uhr | 00:41 | - PluginMenuButton in PluginInterface verschoben |
| 22.04.2010 | 10.45 Uhr | 12.06 Uhr | 01:20 | - Anwendung von Themes |
| 23.04.2010 | 13.46 Uhr | 17.27 Uhr | 03:41 | - GUI Design Themes integriert |
| 24.04.2010 | 13.00 Uhr | 17.54 Uhr | 04:54 | - Tests angepasst<br>- Code Refactored<br>- Activation Prinzip überdenkt |
| 26.04.2010 | 7.55 Uhr | 12.05 Uhr | 04:09 | - Implementation GUI Commands |
| 26.04.2010 | 13.40 Uhr | 15.00 Uhr | 01:20 | - Umgang mit URI studiert für Image laden von Plugins |
| 27.04.2010 | 8.00 Uhr | 12.00 Uhr | 04:00 | - Design Document vorbereitet |
| 27.04.2010 | 13.15 Uhr | 17.54 Uhr | 04:39 | - Design Document bearbeitet |

| | | | | |
|---|---|---|---|---|
| 27.04.2010 | 19.40 Uhr | 22.13 Uhr | 02:33 | - Kurzpräsentation vorbereitet<br>- ViewModel UnitTests<br>- Reflection Knowhow gesammelt |
| 27.04.2010 | 22.13 Uhr | 22.47 Uhr | 00:34 | - Shutdown implementiert<br>- Statusbar loading implementiert |
| 28.04.2010 | 9.00 Uhr | 9.56 Uhr | 00:56 | - Image loading Problem gelöst |
| 30.04.2010 | 13.37 Uhr | 15.33 Uhr | 01:55 | - Meeting Plugin Benutzermanager Kennzahlenerfassung/Auswertung Thurgau<br>- Doku angepasst |
| 28.04.2010 | 11.00 Uhr | 12.00 Uhr | 01:00 | - Meeting Zwischenbericht |
| 28.04.2010 | 14.30 Uhr | 17.00 Uhr | 02:30 | - Informationen gesammelt Best Practices Entity Framework |
| 30.04.2010 | 16.45 Uhr | 19.59 Uhr | 03:14 | - Dokumentation Datenbank-Layer<br>- Dokumentation Datenbank Technologien<br>- Dokumentation Extensibility Technologien |
| 01.05.2010 | 15.00 Uhr | 18.30 Uhr | 03:30 | - Console/Exception-Handler |
| 02.05.2010 | 22.30 Uhr | 0.45 Uhr | 02:15 | - Console<br>- Exception Handling |
| 03.05.2010 | 8.02 Uhr | 12.13 Uhr | 04:10 | - RibbonTabs Add and Remove mit ContextualTabs implementiert |
| 03.05.2010 | 13.00 Uhr | 16.02 Uhr | 03:02 | - Console implemented |
| 04.05.2010 | 8.04 Uhr | 12.05 Uhr | 04:00 | - Benutzerverwaltung-Plugin Analyse |
| 04.05.2010 | 13.35 Uhr | 17.49 Uhr | 04:14 | - Benutzerverwaltung Analyse |
| 05.05.2010 | 8.02 Uhr | 11.50 Uhr | 03:47 | - Design Document User Administration |
| 05.05.2010 | 13.30 Uhr | 18.27 Uhr | 04:57 | - Design User Administration<br>- class diagram<br>- gui design |
| 05.05.2010 | 19.00 Uhr | 20.00 Uhr | 01:00 | - Projektplan aktualisiert |
| 06.05.2010 | 8.25 Uhr | 12.06 Uhr | 03:40 | - Gui Design erstellt<br>- Domain-Model erstellt |
| 06.05.2010 | 13.29 Uhr | 15.50 Uhr | 02:21 | - Validation erarbeitet<br>- GUI design Änderung |
| 07.05.2010 | 13.28 Uhr | 17.55 Uhr | 04:26 | - Projekt Benutzerverwaltung und PluginKlasse vorbereitet<br>- GUI Design (PluginMenuControl) der Benutzerverwaltung begonnen<br>- Principal Abklärungen getroffen |
| 10.05.2010 | 8.29 Uhr | 10.43 Uhr | 02:14 | - PluginMenuControl Benutzerverwaltung<br>- ContentPresenter bringt VS2010 ständig zum Absturz |
| 10.05.2010 | 10.45 Uhr | 12.07 Uhr | 01:22 | - PluginMenuControlButton<br>- Versuch den Content des UserControl auf das Stackpanel innerhalb des UserControl umzuleiten |
| 10.05.2010 | 13.37 Uhr | 16.06 Uhr | 02:28 | - Custom Dependency Properties um die Commands Der UserControl auf die darin enthaltenen Buttons zu mappen. |
| 11.05.2010 | 8.13 Uhr | 11.37 Uhr | 03:23 | - Gui Design erstellt |
| 11.05.2010 | 13.37 Uhr | 18.14 Uhr | 04:37 | - GUI Design: RibbonMenu erweitert |
| 12.05.2010 | 13.34 Uhr | 16.08 Uhr | 02:34 | - Verlust von UserControls, da sie durch den SVN nicht commited wurden.<br>- GUI Design neu programmiert |
| 13.05.2010 | 11.28 Uhr | 14.06 Uhr | 02:38 | - GUI Design vervollständigt |
| 17.05.2010 | 9.39 Uhr | 12.00 Uhr | 02:20 | - Principal in Application Container integriert |
| 18.05.2010 | 8.12 Uhr | 16.26 Uhr | 08:14 | - Principal und Identity Model für ApplicationContainer Benutzer<br>- Entity Model für Benutzerverwaltung |
| 18.05.2010 | 13.33 Uhr | 18.19 Uhr | 04:46 | - Entity Model für Benutzerverwaltung erstellt<br>- Principal und Identity für ApplicationContainer implementiert |
| 19.05.2010 | 8.05 Uhr | 11.52 Uhr | 03:47 | - Principal und Identity Test erstellt<br>- Tests mit System.Windows.Application laufen nicht, da im selben AppDomain nur eine Application gestartet werden kann, der UnitTest jedoch den Test mehrfach instanziert und somit eine Exception verursacht |
| 19.05.2010 | 14.18 Uhr | 18.37 Uhr | 04:18 | - Entity Model beendet<br>- Functionen werden im UserAdmin Plugin geladen<br>- Eintrag in DB für Andrea Berweger erstellt zu Testzwecken |
| 20.05.2010 | 8.18 Uhr | 9.21 Uhr | 01:03 | - Permissionfunktionalität für Principal implementiert. |
| 20.05.2010 | 15.05 Uhr | 16.00 Uhr | 00:55 | - Event Logging PluginManager<br>- VisualSVN neu installiert, da noch im Beta Stadion für VS2010 und die Source-COntrol im VS2010 versagte. Neuinstallation hat das Problem behoben. |
| 17.05.2010 | 15.30 Uhr | 16.30 Uhr | 01:00 | - Meeting |
| 17.05.2010 | 9.21 Uhr | 12.45 Uhr | 03:24 | - Permissionfunktionalität für Principal implementiert. |
| 21.05.2010 | 8.10 Uhr | 12.07 Uhr | 03:56 | - Permission Caching implementiert<br>- PersonControl ViewModel begonnen mit Funktionalität implementieren |
| 21.05.2010 | 13.35 Uhr | 18.04 Uhr | 04:28 | - Personenverwaltung GUI functionalität implementiert<br>- Problem mit Custom Control und namedContent gelöst mit Hilfe von einem UserControl ohne xaml |
| 25.05.2010 | 8.08 Uhr | 11.41 Uhr | 03:33 | - Funktionalität für Personen und Login implementiert<br>- Datavalidation begonnen zu implementieren bei Personen |
| 25.05.2010 | 13.39 Uhr | 19.19 Uhr | 05:39 | - Gruppenfunktionalität implementiert<br>- Datavalidation Konzepte studiert und abgeschlossen |
| 26.05.2010 | 8.09 Uhr | 11.50 Uhr | 03:41 | - AuthorizationControl: DataGrids initialisiert, Treeview initialisiert, Checkboxen Bindings erstellt |
| 26.05.2010 | 13.17 Uhr | 14.26 Uhr | 01:09 | - AuthorizationControl: RibbonCommands implementiert |
| 27.05.2010 | 7.46 Uhr | 11.50 Uhr | 04:03 | - Parallelisierung des UserAdminPlugin -> Mehrfach-Views möglich |
| 27.05.2010 | 13.00 Uhr | 18.03 Uhr | 05:03 | - Refactorings und Optimierung des UserAdminPlugin<br>- Noch offen: Gleiche Login dürfen nicht möglcih sein |
| 28.05.2010 | 8.10 Uhr | 12.00 Uhr | 03:49 | - Refactoring & Verbesserungen ApplicationContainer |
| 28.05.2010 | 13.00 Uhr | 18.25 Uhr | 05:25 | - Refactoring & Verbesserungen ApplicationContainer |
| 31.05.2010 | 9.56 Uhr | 11.08 Uhr | 01:11 | - Refactoring Person Administration DataGrid Validation |
| 31.05.2010 | 9.00 Uhr | 12.00 Uhr | 03:00 | - Validation von Login auch wenn Person nicht selektiert |
| 31.05.2010 | 13.30 Uhr | 16.19 Uhr | 02:48 | - Validation bei Personen erweitert, das fehlerhafte Login auch geprüft werden |
| 01.06.2010 | 9.58 Uhr | 20.17 Uhr | 10:19 | - Datavalidation: GUI wird nicht richtig aktualisiert, Login wird auf Vorhandensein geprüft und als Error markiert |
| 02.06.2010 | 8.11 Uhr | 12.10 Uhr | 03:58 | - Refactoring und Verbesserungen sowie Bereinigungen |

| | | | | |
|---|---|---|---|---|
| 02.06.2010 | 13.05 Uhr | 14.52 Uhr | 01:47 | - Refactoring und Verbesserungen sowie Bereinigungen |
| 03.06.2010 | 7.50 Uhr | 12.07 Uhr | 04:17 | - Codebereinigung und Codedokumentierung |
| 03.06.2010 | 13.02 Uhr | 16.42 Uhr | 03:39 | - Code Comments eingefügt (xml-Kommentare) |
| 04.06.2010 | 7.31 Uhr | 12.07 Uhr | 04:36 | - Codebreinigung |
| 04.06.2010 | 13.02 Uhr | 18.01 Uhr | 04:59 | - UnitTest Bereinigung |
| 07.06.2010 | 8.14 Uhr | 12.04 Uhr | 03:49 | - administrative Dokumente gesammelt und Vorbereitet - Poster Design entworfen |
| 07.06.2010 | 13.20 Uhr | 17.49 Uhr | 04:29 | - Poster beendet - Bilder für Abstract designt, benötigen noch bessere Qualität |
| 08.06.2010 | 7.52 Uhr | 12.01 Uhr | 04:09 | - UnitTest optimiert im Bezug auf mehrere Klassen die als Basis die System.Windows.Application haben. Pro UnitTest-Lauf darf nur eine Application pro Domain erzeugt werden. Alle Application Controller müssen in separaten Domains geladen werden. |
| 08.06.2010 | 13.23 Uhr | 18.29 Uhr | 05:06 | - Poster verbessert<br>- Abstract erstellt |
| 09.06.2010 | 8.02 Uhr | 11.58 Uhr | 03:56 | - Poster überarbeitet - Unittests für UserAdministrationPlugin überarbeitet |
| 09.06.2010 | 12.55 Uhr | 18.28 Uhr | 05:33 | - Unittest überarbeitet |
| 10.06.2010 | 7.37 Uhr | 12.06 Uhr | 04:28 | - Poster beendet - Abstract überarbeitet |
| 10.06.2010 | 13.46 Uhr | 17.36 Uhr | 03:50 | - Abstract beendet - Unittest überarbeitet |
| 11.06.2010 | 8.03 Uhr | 12.03 Uhr | 03:59 | - Administrative Dokumente erstellt |
| 11.06.2010 | 13.00 Uhr | 17.28 Uhr | 04:28 | - UserAdministrationPlugin überarbeitet<br>- ViewModel und Befehlsparameter via Selected-DependencyProperties von Datagrid's |
| 11.06.2010 | 17.30 Uhr | 17.39 Uhr | 00:09 | - Abstract für Studienarbeit korrigiert |
| 13.06.2010 | 11.45 Uhr | 13.17 Uhr | 01:32 | - Person & Role ViewModel Command Implementation überarbeitet |
| 13.06.2010 | 13.24 Uhr | 15.04 Uhr | 01:40 | - Authorization ViewModel Command Implementation überarbeitet<br>- UnitTest für UserAdministration überarbeitet |
| 13.06.2010 | 16.31 Uhr | 19.10 Uhr | 02:39 | - Unittest überarbeitet |
| 13.06.2010 | 19.30 Uhr | 22.12 Uhr | 02:42 | - Unittest überarbeitet und fertiggestellt |
| 14.06.2010 | 7.41 Uhr | 12.00 Uhr | 04:18 | - Poster und Abstract überarbeitet<br>- Domainmodel mit Code synchronisiert |
| 14.06.2010 | 13.20 Uhr | 17.26 Uhr | 04:06 | - Domain Model mit Code synchronisiert<br>- Dokumentation überarbeitet |
| 14.06.2010 | 17.30 Uhr | 18.30 Uhr | 01:00 | - Dokumentation überarbeitet |
| 14.06.2010 | 19.08 Uhr | 22.55 Uhr | 03:46 | - Dokumentation überarbeitet |
| 15.06.2010 | 7.45 Uhr | 10.05 Uhr | 02:20 | - Konvertierungstool für xml Codebeschreibungen: Einlesen des xml erstellt |
| 15.06.2010 | 11.00 Uhr | 12.00 Uhr | 01:00 | - Meeting |
| 15.06.2010 | 13.00 Uhr | 15.00 Uhr | 02:00 | - Administrative Dokumente abgeschlossen - Poster gedruckt |
| 15.06.2010 | 16.05 Uhr | 18.22 Uhr | 02:17 | - Kovertierungstool beendet: Tex-Files für Latex erstellt und in Doku aufgenommen |
| 15.06.2010 | 19.00 Uhr | 23.55 Uhr | 04:55 | - Dokumentation überarbeitet |
| 16.06.2010 | 10.28 Uhr | 11.59 Uhr | 01:30 | - Dokumentation überarbeitet |
| 16.06.2010 | 13.09 Uhr | 20.00 Uhr | 06:50 | - Dokumentation überarbeitet |
| 17.06.2010 | 8.00 Uhr | 18.00 Uhr | 10:00 | [Pseudoeintrag da bereits in die Doku integriert]<br>- Abschliessende Arbeiten:<br>- Dokumentation Feinschliff<br>- Drucken & Binden der Dokumentation<br>- Code & Umgebung CD bereitstellen |
| | **Gesamtaufwand:** | | | **461h** |

Die vorgegebenen 360h Aufwand wurden um 100h überschritten. Somit liegt der Gesamtaufwand bei ca. 460h und ist 30% über dem Soll. Der massive Mehraufwand ist auf das umfangreiche, jedoch notwendige erarbeiten des Knowhow über die neuen Technologien zurückzuführen. Für das Projekt und das Knowhow ist der Mehraufwand sicherlich positiv und es konnten viele Erfahrungen gesammelt werden. Wirtschaftlich gesehen kann man es negativ werten, da der Kunde 30% Mehrkosten tragen muss.

## B.3. Persönlicher Bericht

Da ich bis zum Beginn hauptsächlich mit .NET 2.0 und Visual Basic entwickelt habe, hat mir das Projekt sehr viel Erfahrung gebracht. Mittlerweile habe ich mühe, um in VB-Projekten die Syntax richtig zu schreiben, da ich mich sehr an C# gewöhnt habe. Ich würde auch kein weiteres Projekt mit Visual Basic entwickeln. Vom Funktionsumfang sind sie zwar sehr ähnlich, aber die Syntax von C# wirkt ausgereifter und es gibt so gar ein paar Funktionen die es in VB nicht gibt. Als Beispiel möchte ich hier mal das Schlüsselwort „yield“und eigen entwickelte Iteratoren erwähnen. Diese können nämlich in VB nicht entwickelt werden.
Alles in allem hat das .NET Framework 4.0 trotz einigen Problemen einen guten und soliden Eindruck hinterlassen.
Vor diesem Projekt habe ich alle privaten Projekte mit .NET 2.0 und Winforms entwickelt. Das war natürlich auch die einfachere Variante, da ich mich das gewohnt war. Mittlerweile

würde ich jedes noch so kleine Programm mit .NET 4.0 sowie WPF entwickeln. WPF mag bei kleinen Applikationen eine Overkill darstellen, ist aber meiner Meinung nach trotzdem besser und übersichtlicher.

Das Entwickeln hat auch gezeigt, dass die neuen Technologien noch so ihre Probleme haben. Das ist aber immer so und wird sich mit der Zeit noch bessern. Um solche Probleme zu umgehen ist auch immer eine Innovation gefragt und verlangt nach Lösungsansätzen die einem vielleicht das System besser verstehen lässt. Technologien wie das Entity Framework erleichtern das Entwickeln enorm und gäbe es da keine Probleme bräuchte es bald keine Softwareentwickler mehr.

Wenn ich so auf das Projekt zurück schaue, habe ich immer noch das Gefühl, dass ich eigentlich gar nicht so viel entwickelt habe, trotz allem ich Woche für Woche am Projekt gearbeitet habe. Meiner Meinung nach ist das schon darauf zurück zuführen, dass das erlernen von neuen Technologien sehr aufwendig ist. Sie werden einem ja auch nicht wie in den Vorlesung auf dem Silbertablett serviert. Hängt man an irgend einem Problem, so kann das schon mal länger dauern bis man es versteht und eine Lösung hat. Klar ist der Aufwand bei der selbständigen Lösungssuche viel Zeit intensiver. Der Grad an Erfahrung dem man aber in diesem Aufwand einsetzt ist unersetzlich. Man lernt die Probleme und Technologien wirklich kennen.

Neben den Erfahrungen auf Seite Technologien war das alleinige Bestreiten der Arbeit eine sehr gute Erfahrung, da man jeden Schritt selber unternehmen muss und alle Details erfährt. Man erfährt das es rund um das Programmieren noch sehr viel administrative Arbeit zum Erledigen gibt. Es war schon recht aufwendig, die Technologien zu studieren, nebenbei eine saubere Dokumentation der Arbeit zu erstellen und gleichzeitig auch noch eine Anwendung auf die Beine zu stellen, bei der man die Technologie erst gerade Ansatzweise erfahren hat.

In einem weiteren Projekt mit Deadline würde ich nächstes mal mehr Zeit einplanen für allgemeine abschliessende Arbeiten. Die letzte Woche hat sich als doch sehr gedrängt und stressig erwiesen.

### B.3.1. Danke

In diesem Zusammenhang möchte ich mich bei allen Personen die das Projekt ermöglicht und unterstützt haben bedanken. Speziell ein Dankeschön an Hansjörg Huser, der das Projekt begleitet hat. Sowie ein Dankeschön an den Industrie Partner ASGAL Informatik GmbH, sprich Thomas Gall, der das Projekt lanciert hat.

# C. Diagramme

Nachfolgend werden alle übergrossen Diagramme in Grossformat zur besseren Übersicht dargestellt.

# D. Klassenübersicht

Um die Dokumentation nicht unnötig zu vergrössern ist die Klassenübersicht auf der beigelegten CD unter „Dokumentation\.NET Application Container (Klassenübersicht).pdf“zu finden.

Die Klassenübersicht wurde automatisch aus den XmlDocs von Visual Studio ins Latex-Format konvertiert. Von da her kann es sein, das sich allfällige Fehler und unschöne Formatierungen eingeschlichen haben.

# Literaturverzeichnis

[Entity Framework] Create a Database using Model-First Development in .NET Entity Framework 4.0, by Vince Varallo
Website: http://aspalliance.com/1912_ASPNET_40_and_the_Entity_Framework_4__Part_1__Create_a_Database_using_ModelFirst_Development.all, 2010-02-09

[Entity Framework] The ADO.NET Entity Framework Overview, by Microsoft msdn
Website: http://msdn.microsoft.com/en-us/library/bb399572.aspx, 2006-06-01

[Entity Framework] The ADO.NET Entity Framework Best Practicess, by Tim Mallalieu
Website: http://msdn.microsoft.com/en-us/magazine/ee236639.aspx#id0400056, 2006-06-01

[MEF] Managed Extensibility Framework Programming Guide, by haveriss, gblock, nblumhardt
Website: http://mef.codeplex.com/wikipage?title=Guide&referringTitle=Overview, 2010-02-17

[MEF] Managed Extensibility Framework Crash Course, by Daniel Plaisted
Website: http://blogs.msdn.com/dsplaisted/archive/2009/06/08/a-crash-course-on-the-mef-primitives.aspx, 2009-06-08

[MEF] Managed Extensibility Framework Dynamic Part Instantiation in MEF, by Nicholas Blumhardt
Website: http://blogs.msdn.com/nblumhardt/archive/2009/08/28/dynamic-part-instantiation-in-mef.aspx, 2009-08-28

[MEF] Managed Extensibility Framework Building Composable Apps in .NET 4 with the Managed Extensibility Framework, by Glenn Block
Website: http://msdn.microsoft.com/en-us/magazine/ee291628.aspx, 2010-02

[WPF Trees] MVC Pattern mit WPF Tutorial, by Norbert Eder
Website: http://dotnet-gui.com/forums/t/162.aspx, 2009-02-27

[MVVM Pattern] MVVM Pattern mit WPF Tutorial, by Norbert Eder
Website: http://dotnet-gui.com/forums/t/216.aspx, 2009-04-07

[WPF Concepts] Logical- and Visual Trees
Website: http://en.csharp-online.net/WPF_Concepts-Logical_and_Visual_Trees, 2007

[WPF Trees] Trees in WPF, by Microsoft msdn
Website: http://msdn.microsoft.com/en-us/library/ms753391.aspx, 2008

[WPF Trees] Understanding the Visual Tree and Logical Tree in WPF, by Josh Smith
Website: http://www.codeproject.com/KB/WPF/WpfElementTrees.aspx, 2007-12-05

[WPF Command] WPF Apps With The Model-View-ViewModel Design Pattern (RelayCommand), by Josh Smith
Website: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx#id0090030, 2007-12-05

[Mail Client Statistics] Email client market share, by fingerprint
Website: http://fingerprintapp.com/email-client-stats, February 2010

[Mail Client Statistics Source] Source for Email client market share, by Salted Services
Website: http://litmusapp.com/, 2005-2010

# Glossar

**EF** Das **Entity Framework** ist ein Framework um Datenbanktabellen auf Klassen abzubilden. Das ermöglicht eine einfacher und objektorientierte Programmierung. Es wurde mit dem .NET Framework 3.5 eingeführt., 1

**MEF** Das **Managed Extensibility Framework** ermöglicht eine Komponenten basierte Programmierung. Das Framework wurde mit .NET Framework 4.0 eingeführt. Mit Hilfe dieses Framework können Bibliotheken dynamisch in eine Applikation eingebunden werden., 1

**MVVM** Das **Model-View-ViewModel** ist ein Konzept, bzw. Pattern, dass im Zusammenhang mit WPF erschienen ist. Es spaltet den Code vom User Interface ab. Einerseits kann somit das User Interface getrennt vom Code entwickelt werden. Dies hat den Vorteil, dass das User Interface von einem Design Professional entwickelt werden kann und der Code von einem Programmer Professional. Andererseits wird durch die Trennung ein Teil der User Interface Funktionalität mittels Unittests testbar., 1

**WPF** Die **Windows Presentation Foundation** wurde mit Windows Vista eingeführt. Es ist eine neue Technologie um das User Interface zu gestalten. Es hat einen bietet mehr Möglichkeiten um ein grafisch anspruchsvolles User Interface zu entwickeln., 1