Office Communication Server Adressbucherweiterung

Dokumentation DF GUI

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
18.12.2008	1.0	Finalversion erstellt	Müller	Dietrich

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. DirectoryFrameworkGUI 4
2. Testdokumentation 58
3. Benutzerhandbuch 71

Directory Framework GUI

Dokumentation

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
30.09.2009	1.0	erstellt	Müller	
02.10.2009	1.1	Erweiterung der UseCases	Müller	
08.12.2009	2.0	Merge aller bisher vorhandenen Dokumente	Müller	
	2.1	Externes Design erstellt	Müller	
09.12.2009	2.2	Architektonische Ziele und Einschränkungen	Müller	
10.12.2009	2.3	Architekturkonzepte	Müller	
	2.3	Logische Architektur	Müller	
11.12.2009	2.4	Design Pakete	Müller	
	2.4	Architekturkonzepte	Müller	
14.12.2009	2.5	Design Entscheidung	Müller	
15.12.2009	2.6	Implementierung	Müller	
	2.6	Aussicht	Müller	
	2.7	Überarbeitung Layout	Müller	
17.12.2009	3.0	Finalversion	Müller	Dietrich

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. Anforderungsspezifikationen 6
1.1. Allgemeine Beschreibung 6
1.1.1. Produkt Funktion 6
1.1.1.1. Layout des GUI 6
1.1.1.2. Erweiterbarkeit des GUI 6
1.1.1.3. Erweiterte Eigenschaften des GUI 6
1.1.2. Benutzer Charakteristik 6
1.1.3. Einschränkungen 6
1.1.4. Abhängigkeiten 6
1.1.5. Use Case Überblick 6
1.2. Spezifische Anforderungen 7
1.2.1. Funktionale Anforderungen 7
1.2.1.1. Richtigkeit 7
1.2.1.2. Interoperabilität 7
1.2.1.3. Sicherheit 7
1.2.2. Bedienbarkeit 7
1.2.2.1. Niedrige Lernzeit 7
1.2.2.2. Verständlichkeit 7
1.2.3. Zuverlässigkeit 7
1.2.3.1. Fehlertoleranz 7
1.2.4. Wartbarkeit 7
1.2.4.1. Installation 7
1.2.5. Schnittstellen 7
1.2.5.1. Benutzerschnittstelle 7
1.2.5.2. Softwareschnittstellen 7
1.2.5.3. Datenbankschnittstelle 7
1.2.6. Lizenzanforderungen 7
1.2.7. Verwendete Standards 7
1.3. Use Cases 8
1.3.1. Use Case Model 8
1.3.2. Use Case brief 8
1.3.2.1. UC 1: Directory Framework neu konfigurieren 8
1.3.2.2. UC 2: Directory Framework Konfiguration erweitern 8
1.3.2.3. UC 3: Directory Framework Konfiguration ändern 9
1.3.2.4. UC 4: Directory Framework Konfiguration verringern 9
1.3.2.5. SUC 1: Mainconfig erstellen 9
1.3.2.6. SUC 2: SearchModuleInstance erstellen/hinzufügen 9
1.3.2.7. SUC 3: Mainconfig ändern/erweitern/verringern 9
1.3.2.8. SUC 4: SearchModuleInstance ändern 9
1.3.2.9. SUC 5: SearchModuleInstance löschen/entfernen 9
1.3.3. Use Case fully dressed 9
1.3.3.1. UC 1: Directory Framework neu konfigurieren 9
1.3.3.2. UC 2: Directory Framework Konfiguration erweitern 10
1.3.3.3. UC 3: Directory Framework Konfiguration ändern 10
1.3.3.4. UC 4: Directory Framework Konfiguration verringern 11
1.3.3.5. SUC 1: Mainconfig erstellen 11
1.3.3.6. SUC 2: SearchModule Instance erstellen/hinzufügen 12
1.3.3.7. SUC 3: Mainconfig ändern / erweitern / verringern 12
1.3.3.8. SUC 4: SearchModule Instance ändern 13
1.3.3.9. SUC 5: SearchModule Instance löschen / entfernen 13
2. Domainanalyse 14
2.1. Domainmodel 14
2.1.1. Strukturdiagramm 14
2.1.2. Konzeptbeschreibung 14
2.2. Systemsequenzdiagramm 15
2.2.1. SSD UC 1: Directory Framework neu konfigurieren 15
2.2.2. SSD UC 2: Directory Framework Konfiguration erweitern 16

2.2.3. SSD UC 3: Directory Framework Konfiguration ändern 17
2.2.4. SSD UC 4: Directory Framework Konfiguration verringern 18
2.2.5. Sub Use Cases SUC1 – SUC5 18
2.3. Systemoperationen 18
2.3.1. Contract UC1: Directory Framework Neukonfiguration 18
2.3.1.1. CO 1: configureMainconfig 18
2.3.1.2. CO2 : addSearchModuleType 19
2.3.1.3. CO 3: configureSearchModuleType 19
2.3.1.4. CO 4: addInstances 19
2.3.1.5. CO 5: configureInstances 19
2.3.1.6. CO 6: saveConfiguration 19
2.3.2. Contract UC2: Directory Framework Konfiguration erweitern 20
2.3.2.1. CO 7: loadConfiguration 20
2.3.2.2. CO 8: saveConfiguration 20
2.3.3. Contract UC3: Directory Framework Konfiguration ändern 20
2.3.4. Contract UC4: Directory Framework Konfiguration verringern 20
2.3.4.1. CO 9: removeSearchModuleType 20
2.3.4.2. CO 10: removeInstance 20
2.4. Activity Diagram 21
2.5. Paperprototype 22
3. Externes Design 22
3.1. GUI Navigation Map 22
3.2. Window 23
3.3. Tabs (User Controls) 24
3.3.1. Csv Tab 25
3.3.2. Exchange Public Tab 26
3.3.3. Exchange Private Tab 27
3.3.4. Ldap Tab 28
3.3.5. Sql Tab 29
3.3.6. Twixtel Tab 30
4. Software Architektur Dokument 31
4.1. Umgebung 31
4.2. Architektonische Einschränkungen und Ziele 31
4.2.1. Ziele 31
4.2.2. Einschränkungen 32
4.3. Architektonische Entscheidungen 33
4.3.1. Schichtenmodel, Model View ViewModel Pattern (MVVM) 33
4.3.1.1. Faktoren 33
4.3.1.2. MVVM – Model View ViewModel 33
4.3.1.3. MVP – Model View Presenter 33
4.3.1.4. Designentscheid 33
4.3.2. Verwendung von Commands 34
4.3.2.1. Faktoren 34
4.3.2.2. Lösung, Verwendung von Commands 34
4.3.2.3. Erwogene Alternativen 34
4.3.3. Verwendung von DF Komponenten 34
4.3.3.1. Faktoren 34
4.3.3.2. Lösung 35
4.3.3.3. Erwogene Alternativen 35
4.3.4. Fehlerbehandlung 35
4.3.4.1. Faktoren 35
4.3.4.2. Lösung 35
4.3.4.3. Erwogene Alternativen 35
4.3.5. GUI Farb- und Formanpassung 35
4.3.5.1. Faktoren 35
4.3.5.2. Lösung 35
4.3.5.3. Erwogene Alternativen 35
4.3.6. Konstanten 36
4.3.6.1. Faktoren 36
4.3.6.2. Lösung 36
4.3.6.3. Erwogene Alternativen 36
4.4. Architekturkonzepte 36

4.4.1. Schichtenmodell 36
4.4.1.1. Übersicht 37
4.5. Design Entscheidungen 38
4.5.1. RelayCommands 38
4.5.2. ConfigurationChangeController 39
4.6. Logische Architektur 40
4.6.1. Übersicht 40
4.6.2. Packagestruktur 41
4.6.3. Solutionübersicht 41
4.6.4. View 41
4.6.5. ViewModel 41
4.6.6. Model 42
4.6.7. Persistenz 42
4.7. Design Pakete 42
4.7.1. Package View 42
4.7.1.1. Beschreibung des Packages 42
4.7.1.2. Klassendiagramm 43
4.7.1.3. Klassenbeschreibung 44
4.7.2. Package ViewModel 45
4.7.2.1. Beschreibung des Packages 45
4.7.2.2. Klassendiagramme 45
4.7.2.2.1. ViewModel 45
4.7.2.2.2. ViewModel SearchModuleInstance 45
4.7.2.3. Klassenbeschreibung 45
4.7.3. Package Model 46
4.7.3.1. Beschreibung des Packages 46
4.7.3.2. Klassendiagramm 47
4.7.3.3. Klassenbeschreibung 47
4.7.3.4. SD ConfigReader 48
4.7.3.5. Object Graph Config Reader 48
4.7.3.6. Operationen 49
4.8. Prozesse und Threads 50
4.9. Datenspeicherung 50
5. Implementierung 50
5.1. Helperklassen bei den SearchModuleInstance Klassen 50
5.1.1. CsvFieldMappingHelper 50
5.1.2. ExchangePublicPathHelper 50
5.1.3. ExchangePrivateContactFolderHelper 51
5.2. Pathmanager 51
5.3. Constants Klassen 51
6. Ausblick 51
6.1. Verwendung von Templates 51
6.2. Write Funktionalität 51
6.3. Vereinfachung von Konfigurationskomponenten 51
6.4. Pfadmanagement und Verwendung von Template Konfigurationsdateien 52
7. Literaturverzeichnis 53
8. Abbildungsverzeichnis 53
9. Tabellenverzeichnis 54

1. Anforderungsspezifikationen

1.1. Allgemeine Beschreibung

1.1.1. Produkt Funktion

Ziel des Graphical User Interface (GUI) ist es, die Konfiguration des Directory Frameworks (DF) derart zu vereinfachen bzw. übersichtlich und intuitiv zu gestalten, dass es auch ungeübten Nutzern möglich ist, dieses zu konfigurieren.

Der Zweck des GUIs besteht darin, XML Konfigurationsfiles zu erstellen, anzupassen oder zurück zu setzen. Diese Benutzerinteraktionen, die im Abschnitt UseCases genauer definiert sind, sollten möglichst einfach zu handhaben sein. Das GUI sollte die Konfiguration des DF vereinfachen und auf keinen Fall verkomplizieren. Damit der Anwender sich in diesem GUI nicht verliert, stehen immer nur die aktuell selektierten und benötigten UI Komponenten zur Verfügung. Es wird auf weitere Einstellungsmöglichkeiten des GUIs verzichtet, um das Wesentliche in den Vordergrund zu stellen.
So sollten bestehende Konfigurationen geladen werden können, und allfällige Änderungen mittels eines Save-Buttons gespeichert werden. Beim Beenden sollte der User auf zwei Möglichkeiten zurückgreifen können, ein Beenden mit integrierter Save-Funktion und ein herkömmliches Beenden. Zusätzliche Hilfestellung wie Daten-Validierung sollte wo notwendig eingesetzt werden, um dem Benutzer das Anwenden des GUIs zu erleichtern und auch Fehlverhalten der Applikation zu vermeiden.
Zudem sollte dem Benutzer die Möglichkeit geboten werden das DF nach der Konfiguration neu zu starten.

1.1.1.1. Layout des GUI

Das Layout sollte intuitiv bedienbar sein und keine unnötigen oder verwirrende Controls beinhalten. Beim Starten sollte die Konfiguration geladen werden und beim Beenden sollte diese wieder abgespeichert werden.

Um den Anwender nicht unnötig zu verwirren wird auf die klassische Menüführung verzichtet. Es wird keine versteckten Features oder alternative Konfigurationswege für das GUI geben. Der gesamte Funktions- und Konfigurationsumfang wird auf ein Tab (Reiter) pro SearchModuleType beschränkt.

1.1.1.2. Erweiterbarkeit des GUI

Das GUI sollte einfach erweiter bar sein. Wenn ein neues Searchmodul hinzukommt, kann man dies in einem eigenen Tab verwirklichen, dass das Aussehen der Anderen über ein Template übernimmt. Somit müssen nur die modul-bezogenen Controls eingebunden und anhand des Architekturmodels, Model-View-ViewModel (MVVM), implementiert werden. Zudem wird noch die Einbindung ans Mainconfig Tab benötigt. Änderungen innerhalb eines Konfigurationsfiles müssen natürlich in allen Architekturschichten angepasst werden, sofern neue Eigenschaften hinzukommen oder entfernt werden. Anderenfalls kann man die Eigenschaft auch an ein anderes Control binden.

1.1.1.3. Erweiterte Eigenschaften des GUI

Zusätzliche Konfigurationsmöglichkeiten sind nicht vorgesehen, um die Einfachheit des GUIs sicherzustellen. Dazu gehören die individuelle Anpassung der Ansicht des GUIs sowie die Umstellung der Sprache. Allenfalls könnte man bei der Einrichtung des GUIs (Installation) die Sprache auswählen aber auch dies ist vorerst nicht vorgesehen. Zudem sind keine Seitenansichten, Druckfunktionen und weitere externe Schnittstellen geplant.

1.1.2. Benutzer Charakteristik

Die Zielgruppe dieses Projektes sind Firmen, welche sich für einen Einsatz des Directory Framework entschieden haben.

1.1.3. Einschränkungen

Da das DF mittels C# entwickelt wurde und der Office Communications Server (OCS) ebenfalls ein Microsoft spezifisches Produkt ist, ist das Projekt auf reine Microsoftumgebungen beschränkt.

1.1.4. Abhängigkeiten

Der Einsatz des DF- GUI ist direkt abhängig von der Verwendung des DFs.

1.1.5. Use Case Überblick

Im Kapitel *1.3 Use Cases* genauer beschrieben.

1.2. Spezifische Anforderungen

1.2.1. Funktionale Anforderungen

1.2.1.1. Richtigkeit

Das GUI bildet die eingegebene Konfiguration 1:1 auf die entsprechenden Konfigurationsfiles ab. Die Struktur dieses Abbilds muss identisch den Vorgaben des DFs entsprechen, damit dieses die vom GUI erstellten, oder geänderten Files auch korrekt einlesen und verwenden kann.

1.2.1.2. Interoperabilität

Der Einsatz des Projekts ist nur in einer Microsoftumgebung, die das Directory Framework verwendet vorgesehen.

1.2.1.3. Sicherheit

Durch die Verwendung des DF- GUI sollten keine Sicherheitslöcher auf Arbeitsstationen oder Servern auftreten.

1.2.2. Bedienbarkeit

1.2.2.1. Niedrige Lernzeit

Die Anlernzeit zur Bedienung des GUI soll sich auf maximal 10 Minuten beschränken, die Problematik des Konfigurierens sollte nicht das „Wie“ sondern das „Was“ sein.

1.2.2.2. Verständlichkeit

Das GUI muss intuitiv bedienbar sein, d.h. dass mind. 80% der Anwender keine Hilfe bei der Bedienung benötigen.

1.2.3. Zuverlässigkeit

1.2.3.1. Fehlertoleranz

Das GUI hat den Anspruch zu 99,9% fehlerfrei zu laufen.

1.2.4. Wartbarkeit

1.2.4.1. Installation

Die Installation soll von einer Person ohne spezielle Vorkenntnisse innerhalb von weniger als 10 Minuten vollzogen werden können.

1.2.5. Schnittstellen

1.2.5.1. Benutzerschnittstelle

Eingaben des Benutzers erfolgen über Maus und Tastatur, das Feedback des Programmes wird dem Benutzer via Bildschirm veranschaulicht.

1.2.5.2. Softwareschnittstellen

Stellt die Anforderung an das DF, dass eine Serviceschnittstelle zur Verfügung steht, welche erlaubt dem DF den Befehl zum erneuten Laden der Konfigurationsdateien zu geben.

1.2.5.3. Datenbankschnittstelle

Es wird keine Datenbankschnittstelle realisiert, da keine Daten in einer Datenbank persistiert werden. Die Daten werden direkt in die jeweiligen XML Configuration Files persistiert.

1.2.6. Lizenzanforderungen

Es wird eine rechtmässig erworbene Directory Framework Lizenz benötigt. Diese wird jedoch nicht vom System geprüft.

1.2.7. Verwendete Standards

Entwicklung des Projektes mit dem .net Framework 3.5 und C#. Darauf aufbauend WPF und WCF.

1.3. Use Cases

In diesem Abschnitt und in den folgenden Kapitel *2.2 Systemsequenzdiagramm*, *2.3 Systemoperationen* wird der Begriff User verwendet. Die genaue Spezifikation des Users ist in diesem Fall der System-Administrator, welcher das DF administriert.

1.3.1. Use Case Model

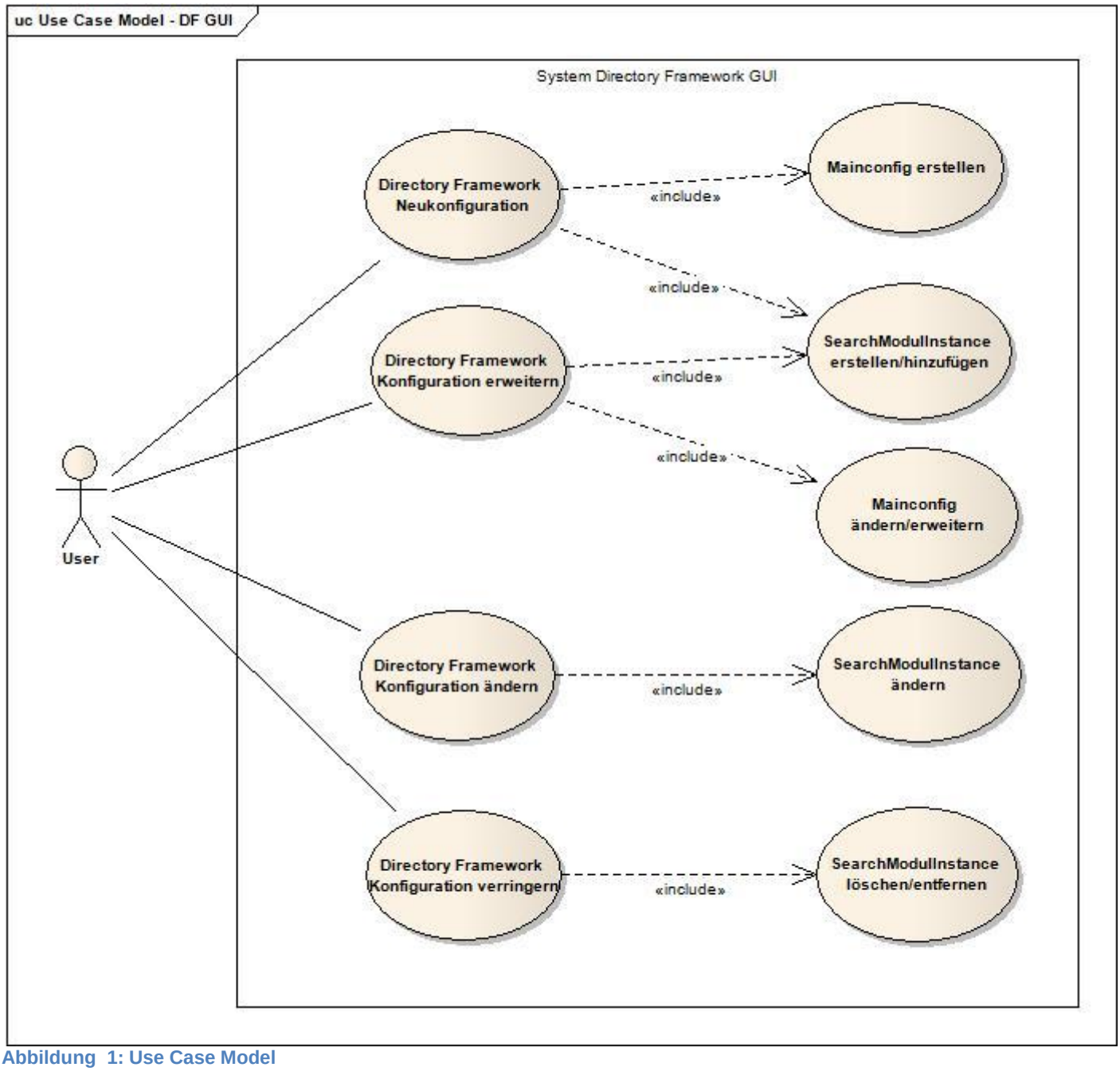


Abbildung 1: Use Case Model

1.3.2. Use Case brief

1.3.2.1. UC 1: Directory Framework neu konfigurieren

Der Anwender erstellt eine komplett neue DF Konfiguration. Er kann SearchModule hinzufügen und die Einstellungen vornehmen.

1.3.2.2. UC 2: Directory Framework Konfiguration erweitern

Der Anwender öffnet die bereits vorhandene Konfiguration des DF und kann diese um weitere Searchmodule erweitern.

1.3.2.3. UC 3: Directory Framework Konfiguration ändern

Der Anwender öffnet die bereits vorhandene Konfiguration des DF und kann innerhalb der vorhandenen Searchmodule Einstellungen verändern oder anpassen.

1.3.2.4. UC 4: Directory Framework Konfiguration verringern

Der Anwender öffnet die bereits vorhanden Konfigurationen des DF und kann diese durch entfernen der gewünschten Searchmodule anpassen.

1.3.2.5. SUC 1: Mainconfig erstellen

Der Anwender definiert die Searchmodultypen, welche er benutzen möchte und zu konfigurieren wünscht. Zudem kann er das Logging und den Webservice einstellen.

1.3.2.6. SUC 2: SearchModuleInstance erstellen/hinzufügen

Falls mehrere Instanzen eines Searchmodules gewünscht sind, kann der Anwender dies hier anpassen. Die erste Instanz eines Searchmoduls wird automatisch erzeugt, wenn in der Mainconfig die entsprechende Auswahl getätigt wurde. Weitere Instanzen können mittels eines Anzahlfeldes definiert werden.

1.3.2.7. SUC 3: Mainconfig ändern/erweitern/verringern

Der Webservice wie auch das Logging kann angepasst werden. Auch das Hinzufügen bzw. Entfernen von Searchmodul- Typen kann vorgenommen werden.

1.3.2.8. SUC 4: SearchModuleInstance ändern

Die Einzelnen, bereits vorhandenen, Searchmodule-Instanzen können neu konfiguriert werden.

1.3.2.9. SUC 5: SearchModuleInstance löschen/entfernen

Einzelne Searchmodul- Instanzen können gelöscht werden.

1.3.3. Use Case fully dressed

1.3.3.1. UC 1: Directory Framework neu konfigurieren

Use Case ID	UC 1
Umfang	Directory Framework GUI
Ebene	Anwenderziel
Primärakteur	User
Stakeholders und Interessen	User: Will möglichst einfach eine neue und komplette Directory Framework Konfiguration erstellen.
Nachbedingungen	Die Konfiguration des Directory Frameworks entspricht den getätigten Einstellungen.
Standardablauf	1. Benutzer startet das GUI. 2. Benutzer tätigt die Konfiguration des Mainfiles, diese umfasst grundsätzlich drei Punkte. a) Auswahl der Searchmodules. b) Definition der Basisadresse des Webservices. c) Festlegung des Logging verhalten. 3. Benutzer konfiguriert die spezifischen Searchmodul Instanzen. 4. Benutzer speichert die vorgenommene Konfiguration in ein gewünschtes Ziel.
Erweiterungen	3a. Je nach ausgewählten Searchmodulen müssen verschiedene Komponenten konfiguriert werden. 4a. Benutzer überlegt es sich anders und verwirft die vorgenommene Konfiguration.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Grundsätzlich tritt dieser Use Case am Anfang der Benutzung des Directory Frameworks auf. Somit ist er eher als selten bis mittel einzustufen.

Tabelle 1: UC 1 Directory Framework neu konfigurieren

1.3.3.2. UC 2: Directory Framework Konfiguration erweitern

Use Case ID	UC 2
Umfang	Directory Framework GUI
Ebene	Anwenderziel
Primärakteur	User
Stakeholders und Interessen	User: Möchte eine bestehende und funktionierende Konfiguration erweitern, mit neuen Instanzen von bereits bestehenden Searchmodulen, oder mit neuen Searchmodultypen.
Nachbedingungen	Die Konfiguration des Directory Frameworks ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Benutzer startet GUI. 2. Benutzer wählt den Pfad, in welchem die vorhandenen Konfigurationen liegen. 3. System lädt bestehende Konfiguration und zeigt diese an. 4. Benutzer wählt neuen Searchmodultyp, danach tätigt er die spezifische Searchmodulkonfiguration. 5. Benutzer speichert die vorgenommene Konfiguration.
Erweiterungen	4a. Benutzer ändert Anzahl Instanzen von bereits ausgewählten Searchmodulen und tätigt danach die spezifische Searchmodulkonfiguration. 5a. Benutzer überlegt es sich anders und verwirft die vorgenommenen Änderungen.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann öfters auftreten, zum Beispiel bei Updates und/oder Erweiterungen der Software/DB die, die Stammdaten enthält.

Tabelle 2: UC2 Directory Framework Konfiguration erweitern

1.3.3.3. UC 3: Directory Framework Konfiguration ändern

Use Case ID	UC 3
Umfang	Directory Framework GUI
Ebene	Anwenderziel
Primärakteur	User
Stakeholders und Interessen	User: Will eine bestehende und funktionierende Konfiguration ändern. Es sind Änderungen in einem oder mehreren vorhandenen Searchmodule-Instanzen erforderlich, die zum Beispiel auf neue Pfade verweisen oder ein neuer Benutzeraccount, der für die Konnektivität des Modules zuständig ist. Diese Einstellungen sind von Searchmodule zu Searchmodule unterschiedlich.
Nachbedingungen	Die Konfiguration des Directory Frameworks entspricht dem Kundenwunsch und ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Benutzer startet GUI. 2. Benutzer wählt den Pfad, in welchem die vorhanden Konfigurationen liegen. 3. System lädt bestehende Konfiguration und zeigt diese an. Die Schritte 4 und 5 werden wiederholt bis alle gewünschten Änderungen ausgeführt wurden. 4. Benutzer navigiert zu gewünschtem Searchmodule, welches er ändern möchte. 5. Benutzer nimmt gewünschte Konfigurationsänderungen vor. 6. Benutzer speichert die vorgenommene Konfiguration.
Erweiterungen	4a. Benutzer möchte nur die Mainconfig ändern. 6a. Benutzer überlegt es sich anders und verwirft die Änderungen.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann öfters auftreten, zum Beispiel bei Updates und/oder Erweiterungen der Software/DB die, die Stammdaten enthält.

Tabelle 3: UC 3 Directory Framework Konfiguration ändern

1.3.3.4. UC 4: Directory Framework Konfiguration verringern

Use Case ID	UC 4
Umfang	Directory Framework GUI
Ebene	Anwenderziel
Primärakteur	User
Stakeholders und Interessen	User: Möchte die Konfiguration verringern, dass heisst er möchte bestehende konfigurierte Searchmodule aus der Gesamtkonfiguration entfernen.
Nachbedingungen	Die Konfiguration des Directory Frameworks ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Benutzer startet GUI. 2. Benutzer wählt den Pfad in welchem die vorhandenen Konfigurationen liegen. 3. System lädt bestehende Konfiguration und zeigt diese an. Die Schritte 4 – 6 werden wiederholt bis alle gewünschten Instanzen entfernt wurden. 4. Benutzer navigiert zu gewünschtem SearchModule in welches er verringern möchte. 5. Benutzer wählt eine SearchModule Instanz aus, welche er entfernen möchte. 6. Benutzer löscht diese Instanz. 7. Benutzer speichert die vorgenommene Konfiguration.
Erweiterungen	4a. Benutzer möchte ein komplettes SearchModule entfernen. (Achtung: entfernt alle Instanzen dieses Typs.) 7a. Benutzer überlegt es sich anders und verwirft die Änderungen.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann auftreten wenn vorhandene Systeme durch neue ersetzt werden. Also ist er eher als selten bis mittel einzustufen.

Tabelle 4: UC 4 Directory Framework Konfiguration verringern

1.3.3.5. SUC 1: Mainconfig erstellen

Use Case ID	SUC 1
Umfang	Directory Framework GUI
Ebene	Anwenderziel, Sub Use Case von UC 1
Primärakteur	User
Stakeholders und Interessen	User: Möchte eine neue Konfiguration erstellen, dies umfasst die Konfiguration des Mainconfig File sowie die, der spezifischen Searchmodule Files. Mit diesem SUC möchte er das Mainconfig File korrekt konfigurieren und somit die weitere Konfiguration einleiten.
Nachbedingungen	Das Mainconfig File ist nach Kundenwunsch und korrekt konfiguriert. Durch diese Konfiguration stellt das GUI die weiter benötigten konfigurations Views zur Verfügung.
Standardablauf	1. Benutzer wählt die gewünschten Searchmodule aus. 2. System generiert die zusätzlichen konfigurations Views. 3. Benutzer konfiguriert die HostAdresse des Webservices. 4. Benutzer wählt die Logging Verhalten aus.
Erweiterungen	Der Benutzer kann auch nach Schritt 1, bereits die Searchmodule spezifische Konfiguration vornehmen, jedoch sollten die Informationen vom Mainconfig nicht vergessen gehen, da sonst das DF mit Sicherheit nicht funktioniert.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Grundsätzlich tritt dieser Use Case am Anfang der Benutzung des Directory Frameworks auf. Somit ist er eher als selten bis mittel einzustufen.

Tabelle 5: SUC 1 Mainconfig erstellen

1.3.3.6. SUC 2: SearchModule Instance erstellen/hinzufügen

Use Case ID	SUC 2
Umfang	Directory Framework GUI
Ebene	Anwenderziel, Sub Use Case von UC 1 und UC 2
Primärakteur	User
Stakeholders und Interessen	User: Will eine bestehende und funktionierende Konfiguration ändern. Durch Hinzufügen neuer Searchmodule oder einer neuen Anzahl eines Searchmodultypes, werden die neu benötigten Komponenten geladen und zur Verfügung gestellt. Mittels dieses SUC kann eine Konfiguration abgeschlossen werden.
Nachbedingungen	Die Konfiguration des Directory Frameworks entspricht dem Kundenwunsch und ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Durch die Auswahl der Searchmodulen und deren Anzahl Instanzen werden diese automatisch hinzugefügt, dieser SUC arbeitet sehr eng mit SUC 1 zusammen. 2. System generiert die zusätzlichen Konfigurations Views. 3. System generiert die zusätzlichen Konfigurationskomponenten und zeigt die Standardkonfiguration derselben an. 4. Benutzer konfiguriert die searchmodulespezifischen Komponenten.
Erweiterungen	Dieser SUC beschreibt weiter noch die Konfiguration der Searchmodule-Instanzen, da diese aber sehr unterschiedlich sind wird nicht konkret darauf eingegangen.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann öfters auftreten, zum Beispiel bei Updates und/oder Erweiterungen der Software/DB die, die Stammdaten enthält.

Tabelle 6: SUC 2 SearchModule Instance erstellen / hinzufügen

1.3.3.7. SUC 3: Mainconfig ändern / erweitern / verringern

Use Case ID	SUC 3
Umfang	Directory Framework GUI
Ebene	Anwenderziel, Sub Use Case von UC 2
Primärakteur	User
Stakeholders und Interessen	User: Möchte grundsätzliche Änderungen an der bestehenden Konfiguration vornehmen. Mittels Eingriffen im Mainconfig File kann er SearchModules hinzufügen oder entfernen, oder die Hostadresse des Webservice und das Logging verändern.
Nachbedingungen	Die Konfiguration des Directory Frameworks entspricht dem Kundenwunsch und ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Je nach Konfigurationswunsch, kann der Benutzer die Änderungen vornehmen 2. Wenn es die Searchmodule betrifft, erweitert das System die Konfiguration, oder vermindert diese. 3. Wenn es die beiden Punkte Webservice oder Logging betrifft, werden die Neukonfigurationen gespeichert.
Erweiterungen	2a. Wenn die Searchmodule erweitert werden, führt dies zu zusätzlichen Konfigurationen.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann öfters auftreten, zum Beispiel bei Updates und/oder Erweiterungen der Software/DB, welche Stammdaten enthalten.

Tabelle 7: SUC 3 Mainconfig ändern / erweitern / verringern

1.3.3.8. SUC 4: SearchModule Instance ändern

Use Case ID	SUC 4
Umfang	Directory Framework GUI
Ebene	Anwenderziel, Sub Use Case von UC 3
Primärakteur	User
Stakeholders und Interessen	User: Will eine oder mehrere bestehende und funktionierende Searchmodule Instanz Konfigurationen ändern.
Nachbedingungen	Die Konfiguration des Directory Frameworks entspricht dem Kundenwunsch und ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Benutzer navigiert zur gewünschten Searchmodule-Instanz. 2. Benutzer ändert die vorhandene Konfiguration, je nach Searchmodul ist diese Konfiguration sehr unterschiedlich. Dieser Ablauf kann sich mehrmals wiederholen, wenn mehrere Instanzen von den Änderungen betroffen sind.
Erweiterungen	keine
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann öfters auftreten, zum Beispiel bei Updates und/oder Erweiterungen der Software/DB, welche Stammdaten enthalten.

Tabelle 8: SUC 4 SearchModule Instance ändern

1.3.3.9. SUC 5: SearchModule Instance löschen / entfernen

Use Case ID	SUC 5
Umfang	Directory Framework GUI
Ebene	Anwenderziel, Sub Use Case von UC 4
Primärakteur	User
Stakeholders und Interessen	User: Möchte eine bestehende Searchmodule-Instanz löschen, weil diese nicht mehr verwendet wird oder durch eine neue ersetzt wurde.
Nachbedingungen	Die Konfiguration des Directory Frameworks entspricht dem Kundenwunsch und ist weiterhin vollumfänglich und funktionsfähig.
Standardablauf	1. Benutzer navigiert zur gewünschten Searchmodul Instanz. 2. Benutzer löscht diese Instanz. Dieser Ablauf kann sich mehrmals wiederhohlen, wenn die gewünschte Instanz nicht vom gleichen Searchmodultyp ist.
Erweiterungen	Wenn er alle Instanzen von einem gewissen Searchmodule löschen möchte, kann er alle Instanzen löschen, dann geht die gesamte Konfiguration verloren, oder er entfernt die Bindung im Mainconfig File, somit wird die Konfiguration nicht mehr geladen, sie ist aber noch in der spezifischen Searchmodule Konfigurationsdatei gespeichert. Diese könnte zu einem späteren Zeitpunkt wieder eingebunden werden.
Spezielle Anforderungen	Directory Framework ist angebunden.
Häufigkeit des Auftretens	Dieser UseCase kann öfters auftreten, zum Beispiel bei Updates und/oder Erweiterungen der Software/DB die, die Stammdaten enthält.

Tabelle 9: SUC 5 SearchModule Instance löschen / entfernen

2. Domainanalyse

2.1. Domainmodel

2.1.1. Strukturdiagramm

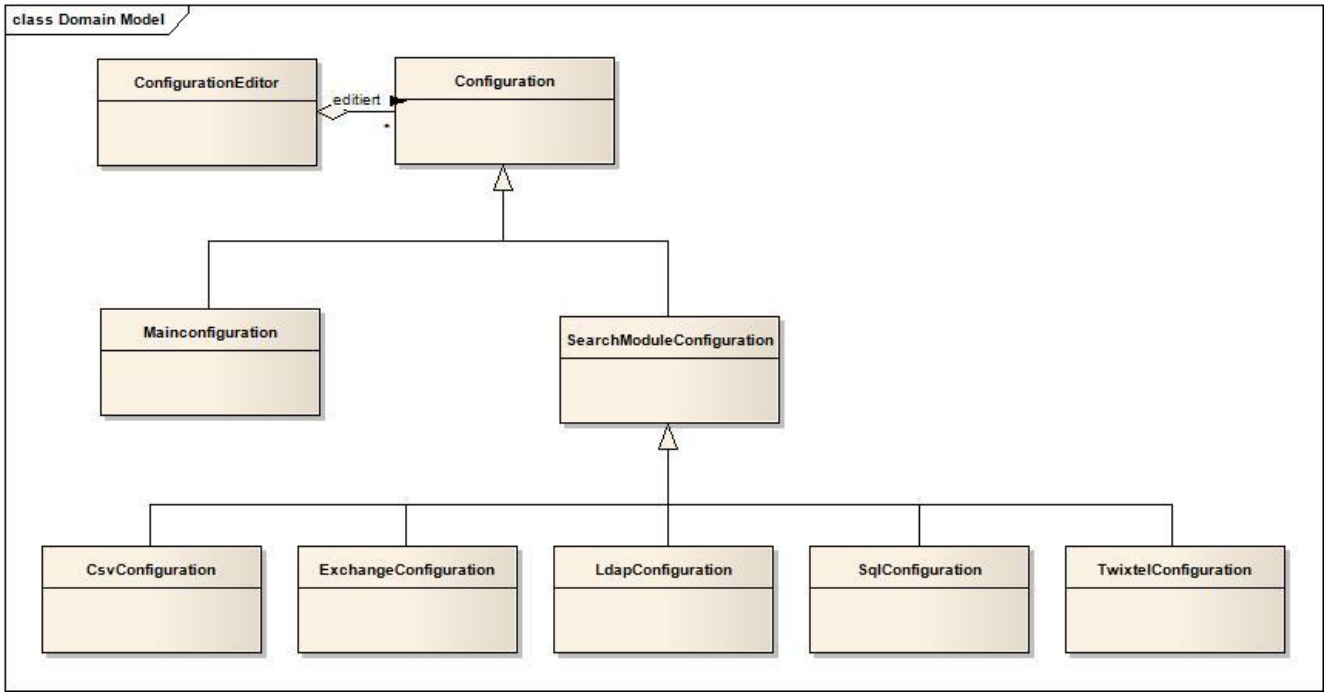


Abbildung 2: Strukturdiagramm

2.1.2. Konzeptbeschreibung

Da das Domainmodel nur die Kernfunktionalität beschreibt, sind die technische Architektur (Layer Struktur) und der genauere Editiervorgang der Konfigurationen, nicht in diesem Model ersichtlich.

Klasse	Beschreibung	Attribute	Beziehungen
ConfigurationEditor	Enthält die Bearbeitungs-Funktionalitäten, lesen, schreiben		Editiert eine Konfiguration
Configuration	Gemeinsame Superklasse aller Configurations Klassen		Superklasse aller Configurations Klassen
MainConfiguration	Repräsentiert die Klasse der Mainconfig (DFService.exe.config)	- Liste der SearchModuleTypen - HostbaseAddress - RegularLogLevel - RequestLogLevel	Erbt von Configuration
SearchModule Configuration	Gemeinsame Superklasse aller SearchModuleConfigurations Klassen	- InstanceID - InstanceName - RequestTransformationRules - ResponseTransformationRules	Superklasse aller SearchModule Configurations Klassen
CsvConfiguration	Repräsentiert die Konfiguration des CsvSearchModule (Csv.dll.config)	- Liste der typspezifischen Instanzen - DataFilePath - LineSeparator - FieldMapping	Erbt von SearchModule Configuration
ExchangeConfgiuration	Repräsentiert die Konfiguration der ExchangeSearchModules (Exchange.dll.config)	- Liste der typspezifischen Instanzen - ExchangeServer - WindowsDomain - User - Password	Erbt von SearchModule Configuration

		- Path (Public) - PathPrefix (Private) - ContactFolders (Private)	
LdapConfiguration	Repräsentiert die Konfiguration des LdapSearchModules (Ldap.dll.config)	- DirectoryEntry - GlobalCatalog - User - Password	Erbt von SearchModule Configuration
SqlConfiguration	Repräsentiert die Konfiguration des SqlSearchModule (Sql.dll.config)	- DataProvider - ConectionStirng - TransforamtionSQLStatement	Erbt von SearchModule Configuration
TwixTelConfiguration	Repräsentiert die Konfiguration des TwixtelSearchModule (Twixtel.dll.config)	- DataPath	Erbt von SearchModule Configuration

Tabelle 10: Konzeptbeschreibung

2.2. Systemsequenzdiagramm

2.2.1. SSD UC 1: Directory Framework neu konfigurieren

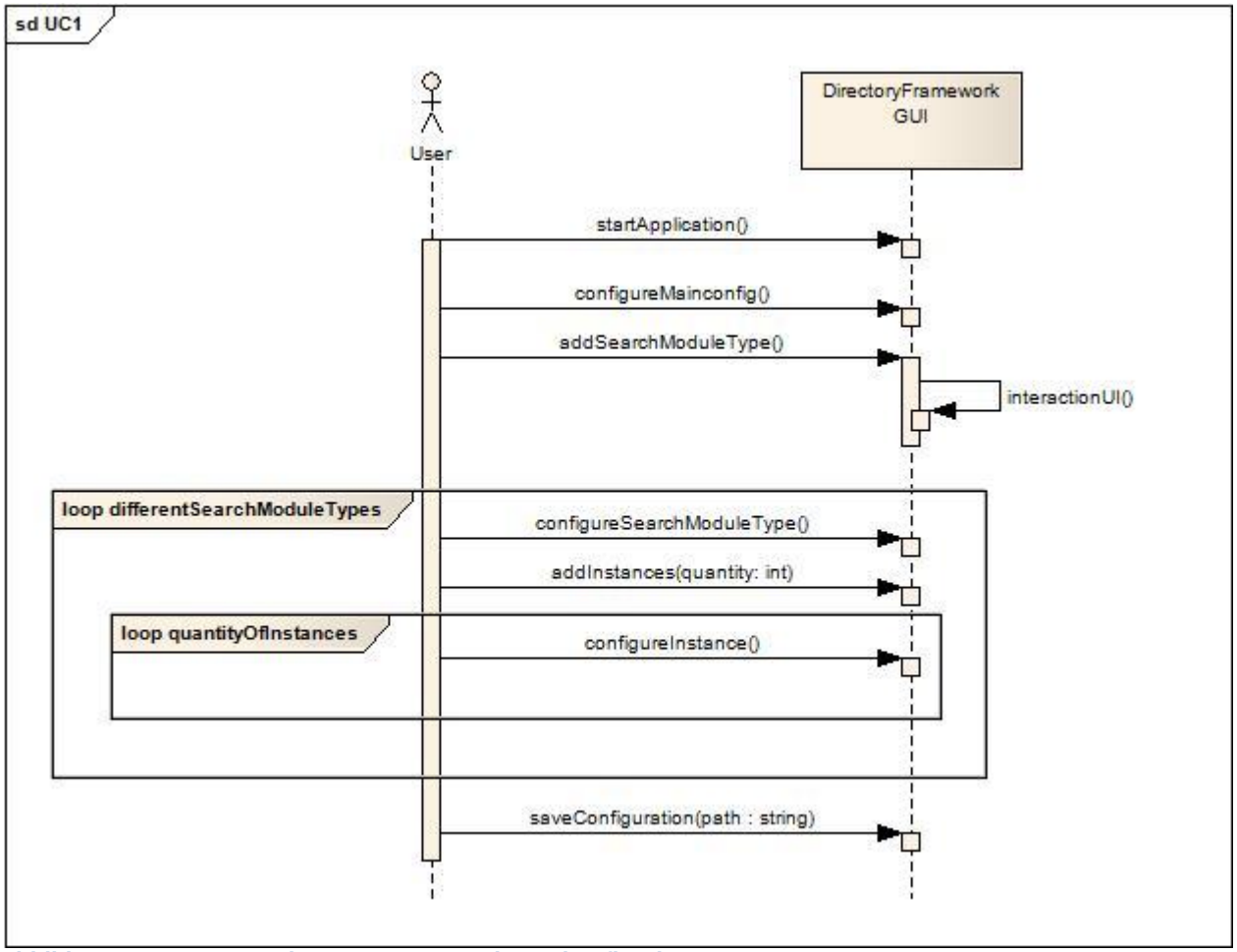


Abbildung 3: SSD UC 1 Directory Framework neu konfigurieren

2.2.2. SSD UC 2: Directory Framework Konfiguration erweitern

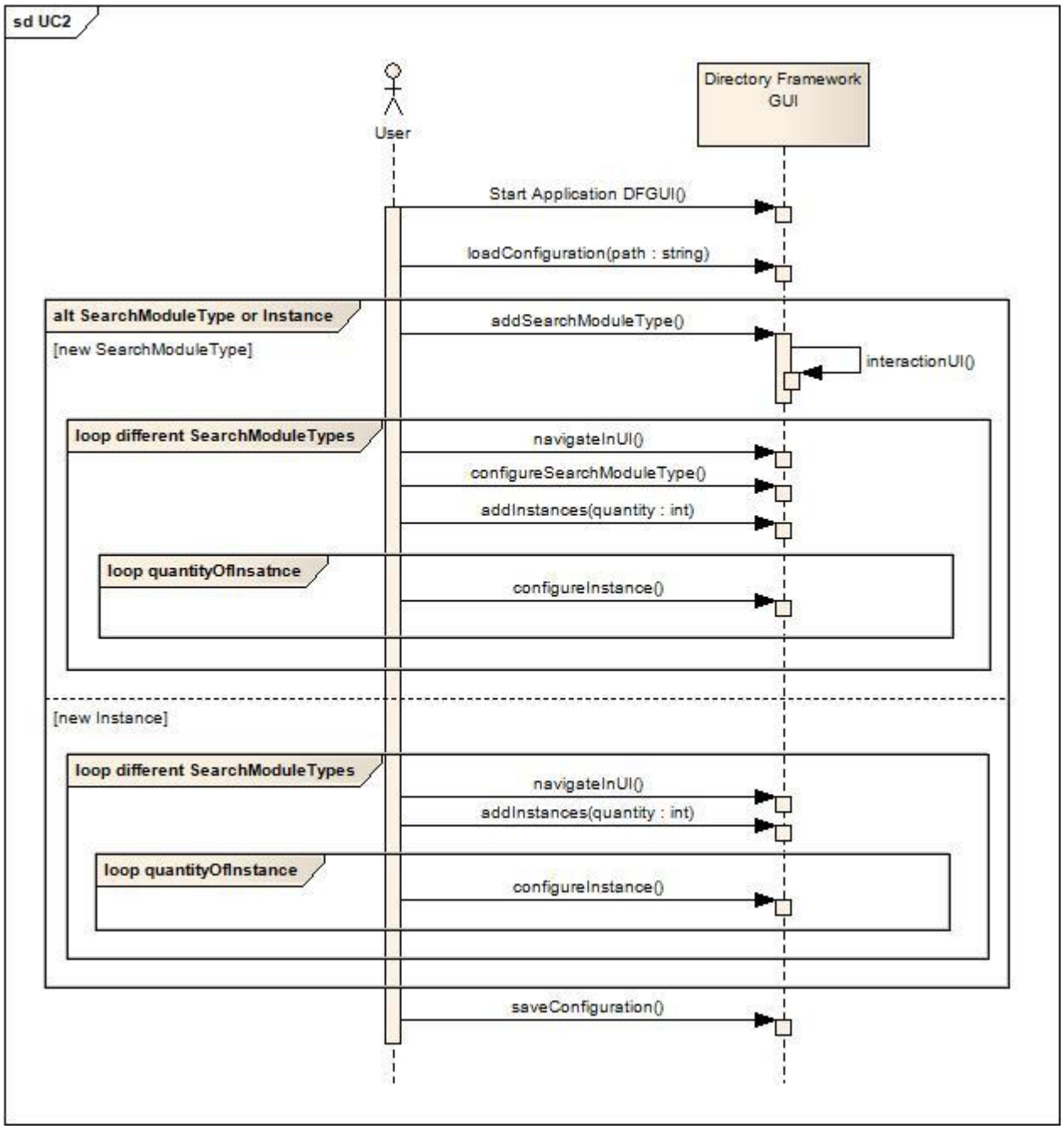


Abbildung 4: SSD UC 2 Directory Framework Konfiguration erweitern

2.2.3. SSD UC 3: Directory Framework Konfiguration ändern

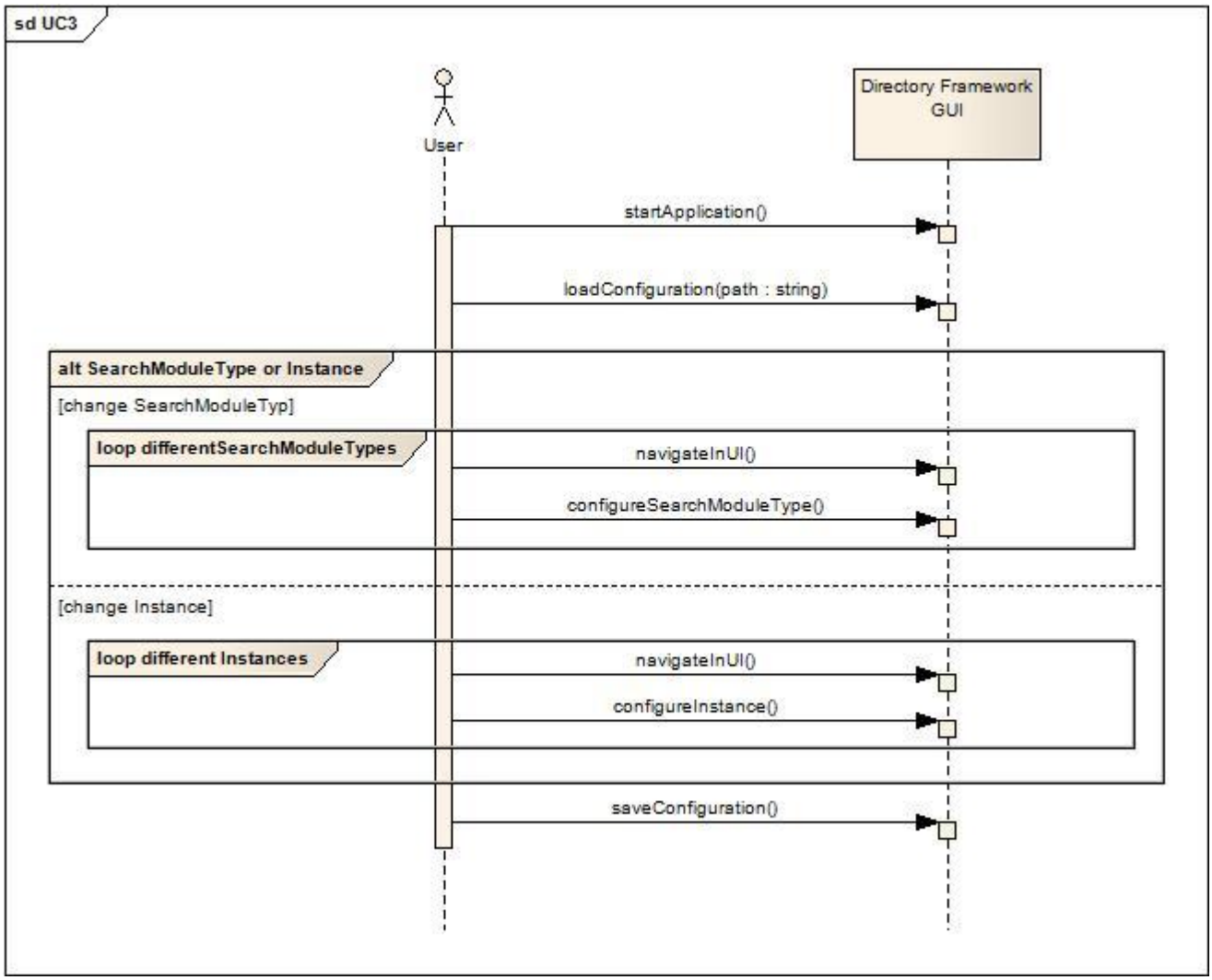


Abbildung 5: SSD UC 3 Directory Framework Konfiguration ändern

2.2.4. SSD UC 4: Directory Framework Konfiguration verringern

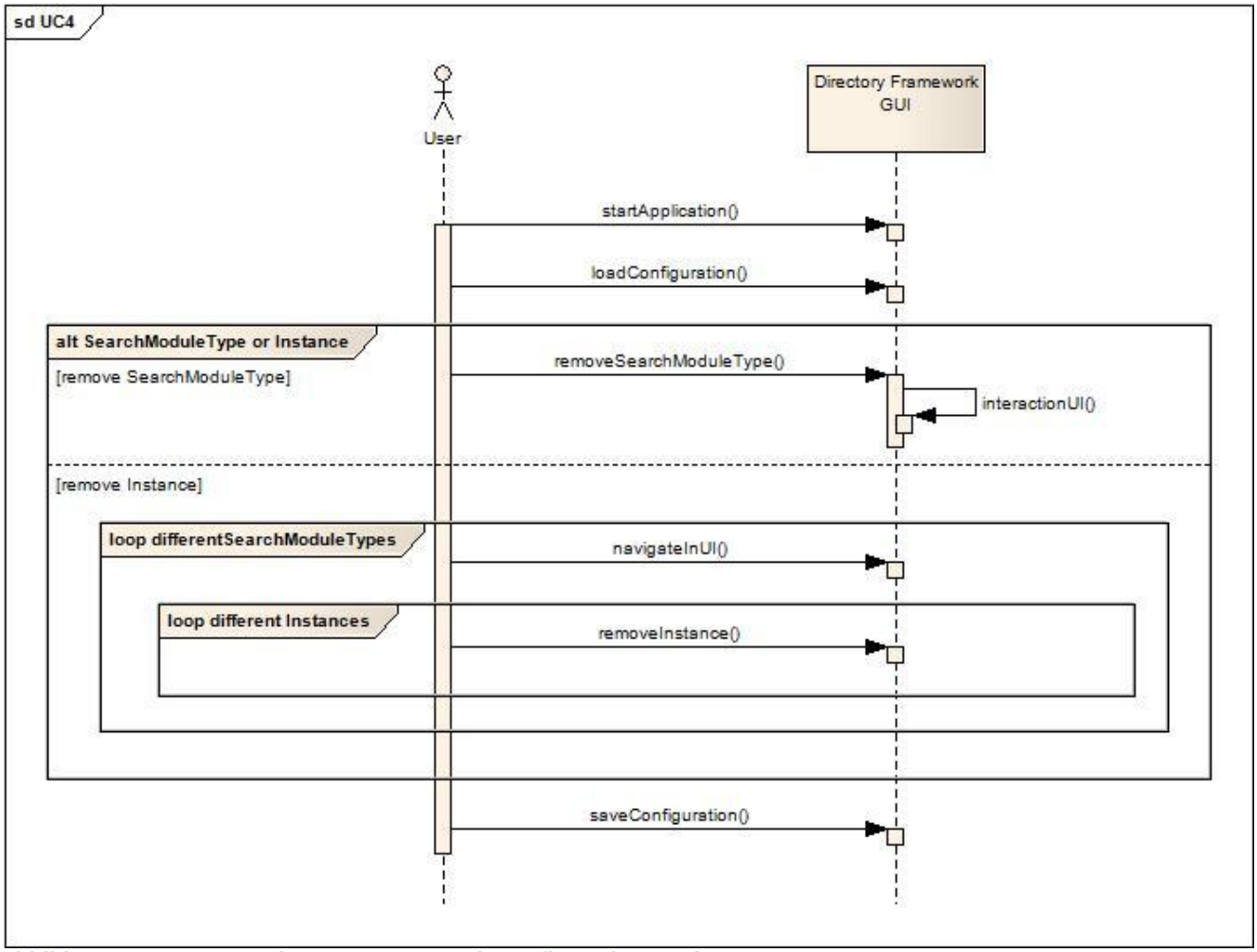


Abbildung 6: SSD UC 4 Directory Framework Konfiguration verringern

2.2.5. Sub Use Cases SUC1 – SUC5

Die Sub Use Cases sind sehr abstrakt gehalten, da sie sehr differenziert sind je nach SearchModuleType. Daher wird auf die einzelnen System Sequenz Diagramme verzichtet, da diese denjenigen der vier Haupt Use Cases sehr ähneln.

2.3. Systemoperationen

2.3.1. Contract UC1: Directory Framework Neukonfiguration

2.3.1.1. CO 1: configureMainconfig

Operation:	configureMainconfig()
Querverweise:	UC1: Directory Framework neu konfigurieren UC3: Directory Framework ändern
Vorbedingungen:	- Die Directory Framework GUI Applikation wurde gestartet.
Nachbedingungen:	- Das Property *HostBaseAddress* wurde gesetzt. - Der *RegularLogLevel* wurde definiert. - Der *RequestLogLevel* wurde definiert.

Tabelle 11: CO 1 configureMainconfig

2.3.1.2. CO2 : addSearchModuleType

Operation:	addSearchModuleType()
Querverweise:	UC1: Directory Framework neu konfigurieren UC2: Directory Framework erweitern
Vorbedingungen:	- Das DF GUI wurde gestartet. - Eine Mainconfiguration wurde geladen oder erstellt.
Nachbedingungen:	- Die UI Interaction hat die neu hinzugefügten UI Komponenten zur Verfügung gestellt. - Die Standard Einstellungen der ausgewählten SearchModules wurde erstellt.

Tabelle 12: CO2 addSearchModuleType

2.3.1.3. CO 3: configureSearchModuleType

Operation:	configureSearchModuleType()
Querverweise:	UC1: Directory Framework neu konfigurieren UC2: Directory Framework erweitern UC3: Directory Framework ändern
Vorbedingungen:	- Das DF GUI wurde gestartet. - SearchModules sind vorhanden, durch das Laden einer Konfiguration oder das Erstellen einer neuen Konfiguration und dem erfolgreichem Hinzufügen von SearchModules.
Nachbedingungen:	- Die Konfiguration ist dem Kundenwunsch entsprechend konfiguriert und erstellt worden.

Tabelle 13: CO3 configureSearchModuleType

2.3.1.4. CO 4: addInstances

Operation:	addInstances(quantity : int)
Querverweise:	UC1: Directory Framework neu konfigurieren UC2: Directory Framework erweitern
Vorbedingungen:	- Das DF GUI wurde gestartet. - SearchModules sind vorhanden, durch laden einer Konfiguration oder erstellen einer Konfiguration inklusive SearchModules.
Nachbedingungen:	- Eine Instanz Liste des ausgewählten SearchModules ist vorhanden mit der richtigen Anzahl von Instanzen. - Die Standardkonfiguration der einzelnen Instanzen wurde geladen.

Tabelle 14: CO 4 addInstances

2.3.1.5. CO 5: configureInstances

Operation:	configureInstances()
Querverweise:	UC1: Directory Framework neu konfigurieren UC2: Directory Framework erweitern UC3: Directory Framework ändern
Vorbedingungen:	- DF GUI wurde gestartet. - SearchModules sind vorhanden. - Instanz Liste ist vorhanden und enthält Instanz Objekte. - Eine Instanz Objekt wurde selektiert
Nachbedingungen:	- Die zu konfigurierende Instanz wurde nach Kundenwunsch geändert und in der Instanz Liste abgelegt.

Tabelle 15: CO 5 configureInstances

2.3.1.6. CO 6: saveConfiguration

Operation:	saveConfiguration(path : string)
Querverweise:	UC1: Directory Framework neu konfigurieren

Vorbedingungen:	- DF GUI wurde gestartet. - Komplette Konfiguration wurde vorgenommen, Mainconfig, SearchModules, Instanzen.
Nachbedingungen:	- Die gesamte Konfiguration wurde korrekt im gewünschten Zielpfad in den einzelnen Konfigurationsdateien gespeichert.

Tabelle 16: CO 6 saveConfiguration

2.3.2. Contract UC2: Directory Framework Konfiguration erweitern

2.3.2.1. CO 7: loadConfiguration

Operation:	loadConfiguration(path : string)
Querverweise:	UC2: Directory Framework erweitern UC3: Directory Framework ändern UC4: Directory Framework verringern
Vorbedingungen:	- DF GUI wurde gestartet.
Nachbedingungen:	- Die gewünschte Konfiguration wurde eingelesen. - Die UI Interaction hat die geladenen UI Komponenten zur Verfügung gestellt. - Die einzelnen Konfigurationen sind in den UI Komponenten verfügbar. - Bei einem Fehler muss der User informiert werden und das Laden abgebrochen werden.

Tabelle 17: CO 7 loadConfiguration

2.3.2.2. CO 8: saveConfiguration

Operation:	saveConfiguration()
Querverweise:	UC2: Directory Framework erweitern UC3: Directory Framework ändern UC4: Directory Framework verringern
Vorbedingungen:	- DF GUI wurde gestartet. - Komplette Konfiguration wurde vorgenommen, Mainconfig, SearchModules und Instanzen
Nachbedingungen:	- Die gesamte Konfiguration wurde korrekt in den einzelnen Konfigurationsdateien gespeichert. (Im selben Pfad von dem die Konfiguration geladen wurde.)

Tabelle 18: CO 8 saveConfiguration

2.3.3. Contract UC3: Directory Framework Konfiguration ändern

Alle Operationen, die in diesem Use Case vorkommen, wurden in den vorangehenden Contracts behandelt, beschrieben und referenziert mittels Querverweis.

2.3.4. Contract UC4: Directory Framework Konfiguration verringern

2.3.4.1. CO 9: removeSearchModuleType

Operation:	removeSearchModuleType()
Querverweise:	UC4: Directory Framework verringern
Vorbedingungen:	- DF GUI wurde gestartet. - Konfigurierte SearchModules sind vorhanden.
Nachbedingungen:	- Das oder die zu entfernenden SearchModules sind aus dem Mainconfiguration File entfernt. - Die nicht mehr benötigten UI Komponenten wurden durch die UI Interaction entfernt und werden nicht mehr angezeigt.

Tabelle 19: CO 9 removeSearchModuleType

2.3.4.2. CO 10: removeInstance

Operation:	removeInstance()
Querverweise:	UC4: Directory Framework verringern

Vorbedingungen:	- DF GUI wurde gestartet. - Konfigurierte SearchModules sind vorhanden. - Instanz Listen der vorhandenen SearchModules enthalten Instanz Objekte. - Ein Instanz Objekt ist selektiert.
Nachbedingungen:	- Das gewünschte Instanz Objekt wurde aus der Instanz Liste gelöscht und wird nicht mehr gespeichert beim Persistieren.

Tabelle 20: CO 10 removeInstance

2.4. Activity Diagram

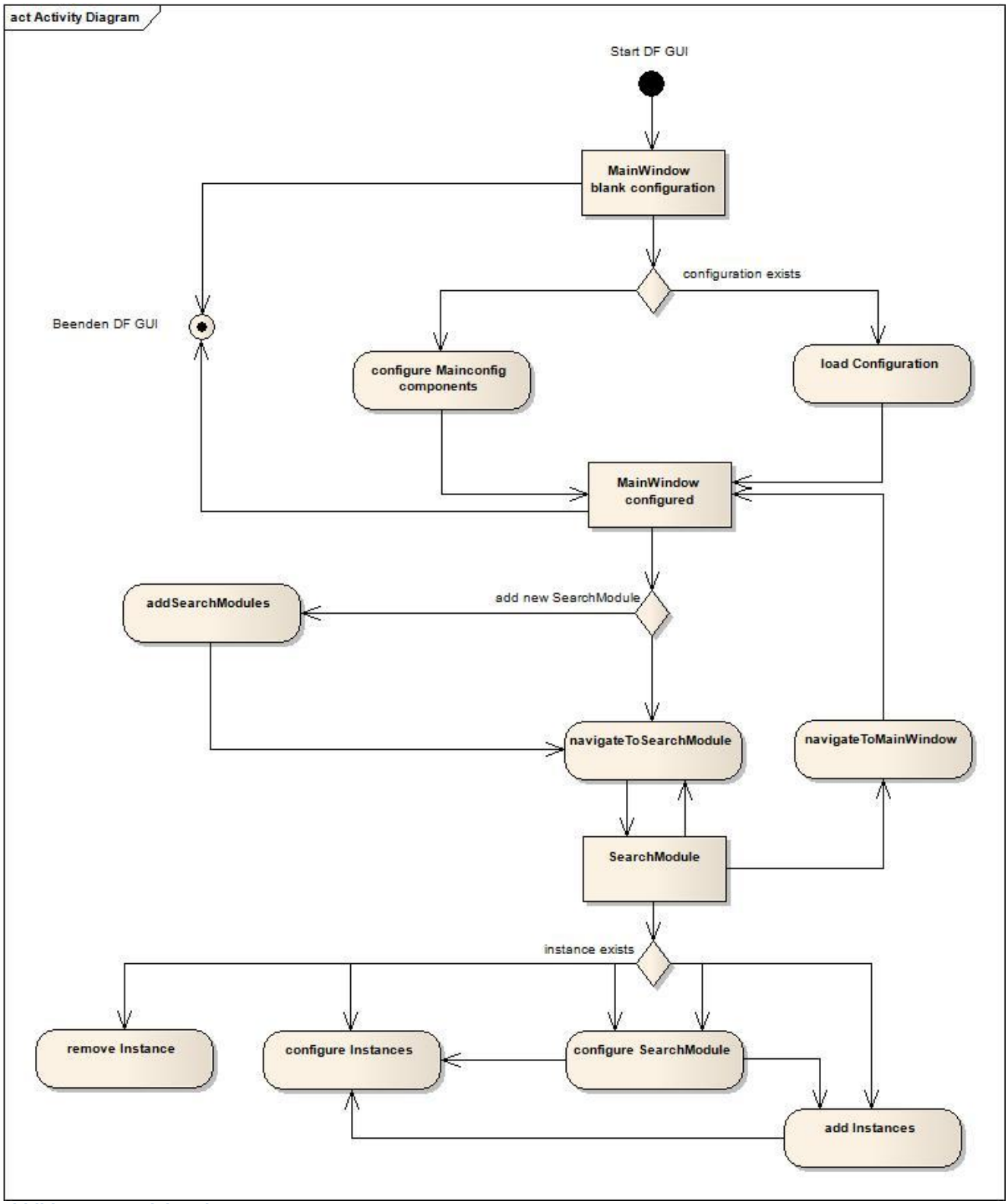


Abbildung 7: Activity Diagram

2.5. Paperprototype

Siehe Anhang A

3. Externes Design

Das externe Design setzt direkt auf dem Paperprototype auf und wurde anhand der Prototypen erstellt. Es wurde bewusst darauf verzichtet, das externe Design mittels eines Zeichnungstools zu gestalten, da dies mit dem GUI Builder oder Expression Blend mittlerweile ebenfalls gut funktioniert und man zusätzlich das GUI für die Implementierung bereits verwenden kann. Deshalb werden im Externen Design Screenshots und Ausschnitte aus dem Visual Studio GUI Builder verwendet.

3.1. GUI Navigation Map

Die GUI Navigation Map fällt in diesem Projekt ziemlich klein aus, da die gesamte Applikation in einem Fenster dargestellt wird. In diesem *MainWindow* wird ein *TabControl* verwendet welches die einzelnen Tabs innerhalb des Fensters steuert. Von jedem sichtbaren Tab kann in jedes andere gewechselt werden. Das *Main config Tab* ist immer zu sehen, es beinhaltet die Mainconfiguration.

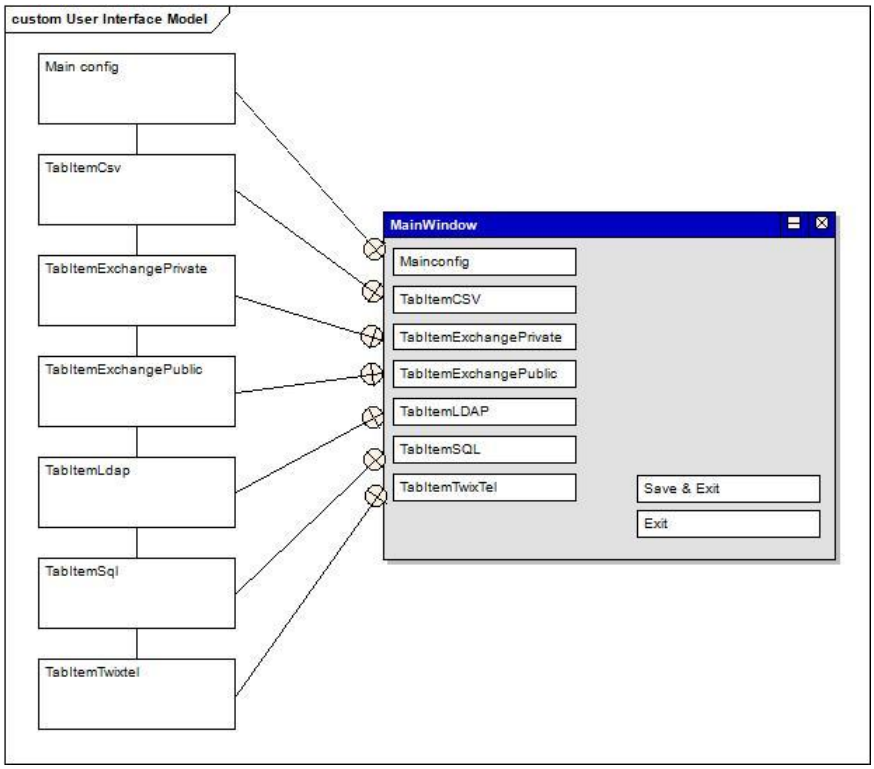


Abbildung 8: GUI Navigation Map

Jedes Tab repräsentiert eine Konfigurationsdatei, wie in der folgenden Tabelle veranschaulicht wird.

Tab	Datei
Main config	DFService.exe.config
Csv	Csv.dll.config
Exchange Public	Exchange.dll.config
Exchange Private	Exchange.dll.config
Ldap	Ldap.dll.config
Sql	Sql.dll.config
Twix Tel	Twixtel.dll.config

Tabelle 21: Tab Repräsentation der Konfigurationsdateien

Die in der Tabelle erwähnten Dateinamen entsprechen der Standardkonfiguration des Directory Framework, diese können selbstverständlich abweichen, wenn der Benutzer die Konfigurationen mit eigens definierten Namen abgespeichert hat. Hier sollte einfach das Mapping der Tabs auf die Konfigurationsdateien illustriert werden.

3.2. Window

Das MainWindow ist das einzige Fenster dieses GUIs. Es enthält das angesprochene TabControl und enthält das *Main config Tab*.
Aus dem *Main config Tab* lassen sich bestehende Konfigurationen einlesen, vorgenommene Konfigurationen speichern und das Directory Framework neu starten. Die Tabs werden nur sichtbar, wenn das dazugehörige SearchModule ausgewählt wurde. Das jeweils selektierte Tab ist weiss eingefärbt und die anderen sind zu 50% durchsichtig.
Das *Main config Tab* setzt sich zusammen aus vier Hauptkomponenten.

1. **Search Module** (*Expander control*)
   In diesem Bereich können die SearchModules ausgewählt werden, die konfiguriert werden sollten. Bei einer Selektion eines SearchModules wird das dazu gehörende Tab eingeblendet und umgekehrt bei einer Deselektion wird es wieder ausgeblendet. Wenn eine Konfiguration geladen wird, werden alle im Konfigurationsfile definierten SearchModules automatisch selektiert und die Tabs werden eingeblendet.

2. **Web Service** (*Expander control*)
   Hier kann die Adresse des Hosts eingestellt werden.

3. **Logging** (*Expander control*)
   Das DF ist mit einem Logger ausgerüstet, der das Verhalten in Logfiles schreibt. Der Logger kann mit sieben verschiedenen Levels arbeiten. (OFF, ALL, INFO, WARNINGS, DEBUG, ERRORS, FATAL)
   In diesem Bereich kann der Level für den Betrieb des DFs, RegularLog und der Level für die Abfragesequenzen (*RequestLog*) definiert werden.

4. **Configuration Settings**
   Dieser Bereich ist als einziger im *Main config Tab* kein Expander control und kann somit nicht minimiert und ausgeblendet werden. Er beinhaltet die Steuerungsfunktionen um Konfigurationen zu laden, zu speichern, das DF neu zu starten oder die Applikation zu beenden.

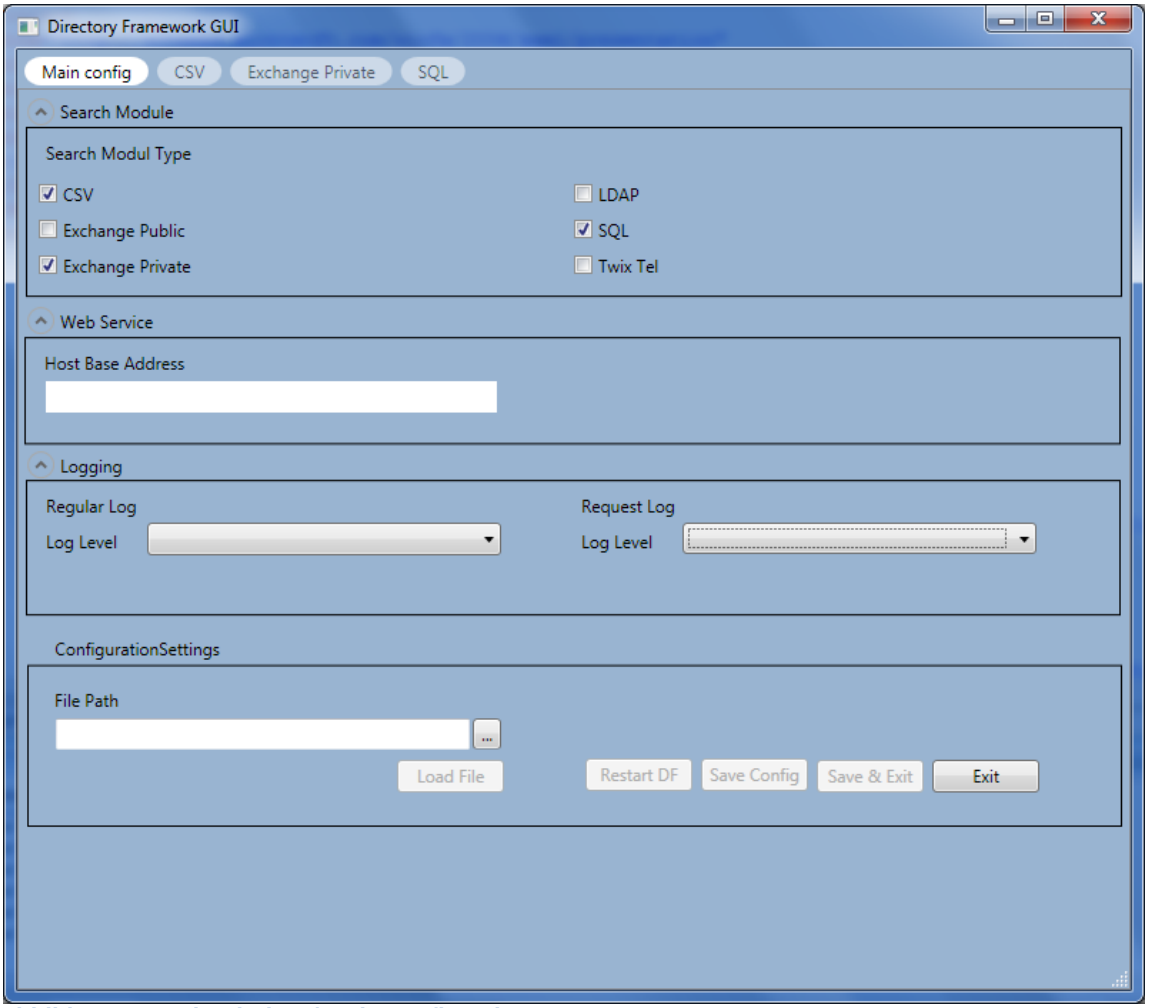


Abbildung 9: MainWindow / Main config Tab

3.3. Tabs (User Controls)

Die sechs verschiedenen Tabs, welche jeweils einem SearchModule entsprechen, sind alle identisch aufgebaut. Sie enthalten drei wichtige Komponenten, diese werden durch schwarze Umrandungen (*Borders*) voneinander getrennt.

Komponentenaufzählung:

1. **Instanz Liste**
   In der Instanz Liste werden alle SearchModules spezifischen Instanzen in einem Container abgelegt. Sobald eine Instanz selektiert ist, werden die Konfigurationsdetails in der Komponente Instanz Konfiguration angezeigt und können in dieser verändert werden.
   Es können eine beliebige Anzahl Instanzen hinzugefügt, einzelne gelöscht oder die gesamte Liste (Container) kann geleert werden.

2. **Instanz Konfiguration**
   In dieser Komponente wird eine selektierte Instanz detailliert angezeigt und kann verändert werden. Die Instanz Konfiguration ist von SearchModule zu SearchModule unterschiedlich und bildet die verschiedenen XML Nodes der Konfigurationsdatei ab.

3. **AdvancedSettings** (SearchModuleConfiguration)
   Die AdvancedSettings sind SearchModule spezifische Informationen, die nur einmal pro SearchModule vorhanden sind, wie zum Beispiel das *Configuration File*, dieses definiert den Namen der Konfigurationsdatei vom aktuell selektierten SearchModule.

Nachfolgend werden alle Tabs des GUIs veranschaulicht, im *Csv Tab* sind die drei Hauptkomponenten markiert.

3.3.1. Csv Tab

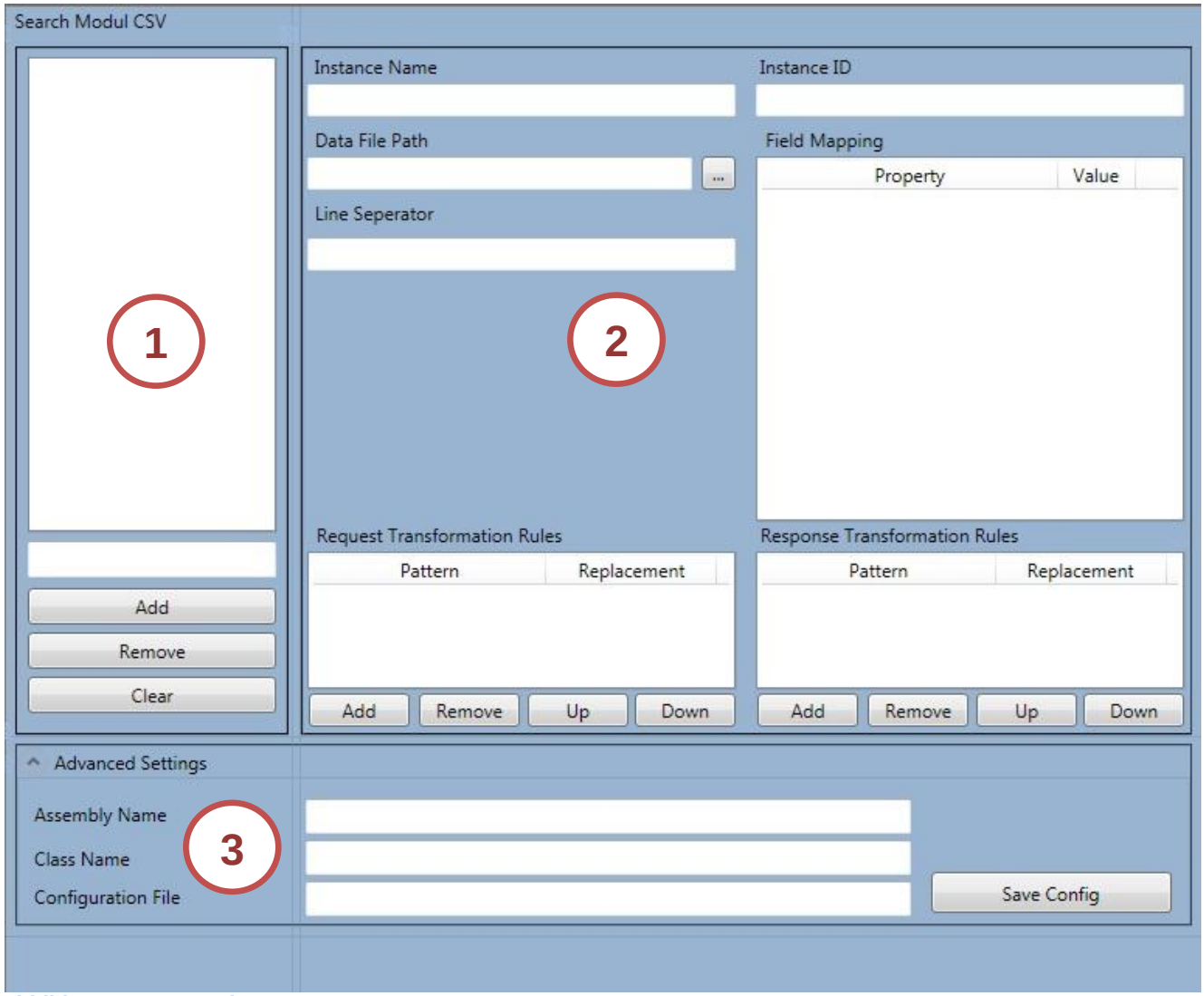


Abbildung 10: Csv Tab

3.3.2. Exchange Public Tab

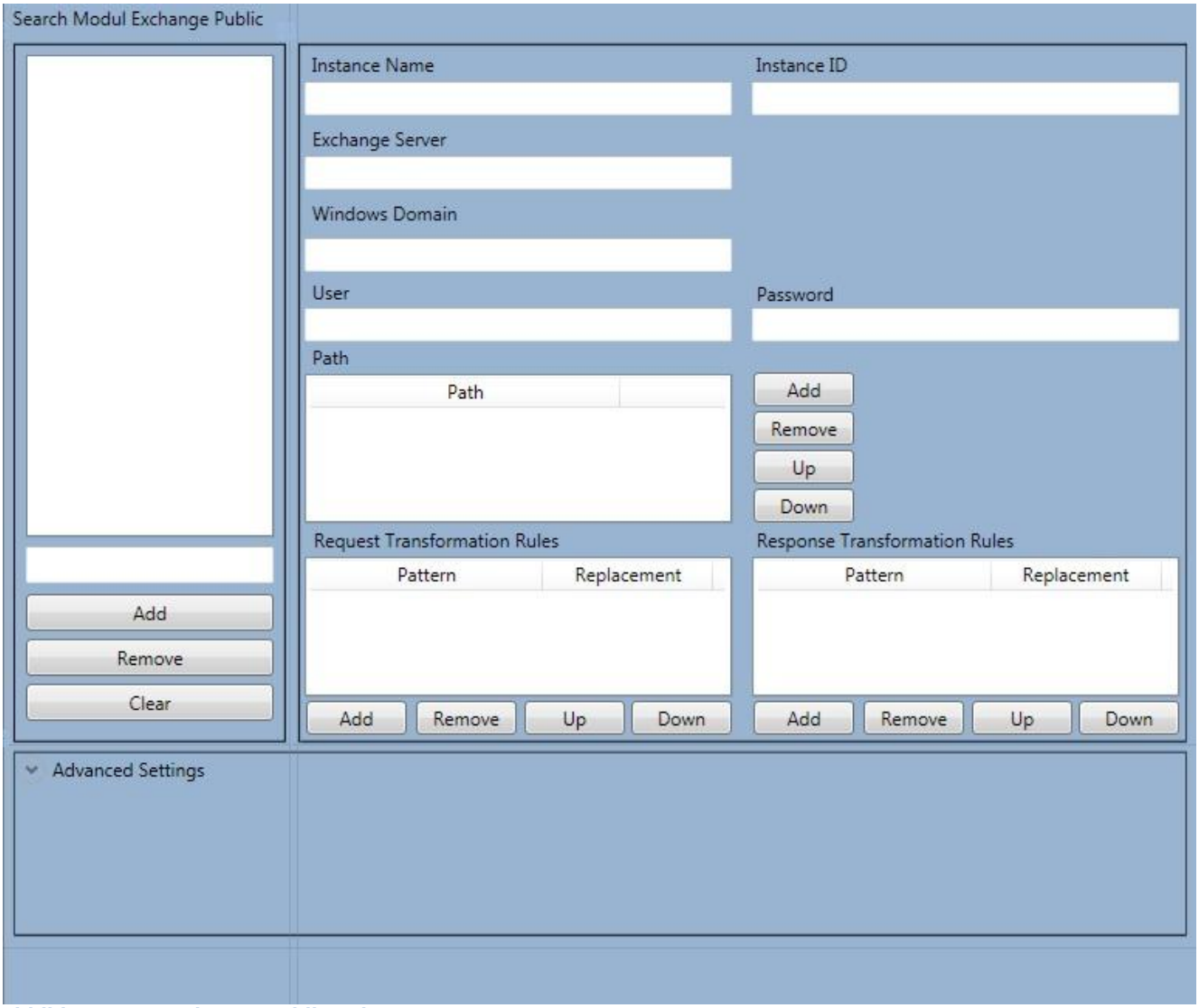


Abbildung 11: Exchange Public Tab

3.3.3. Exchange Private Tab

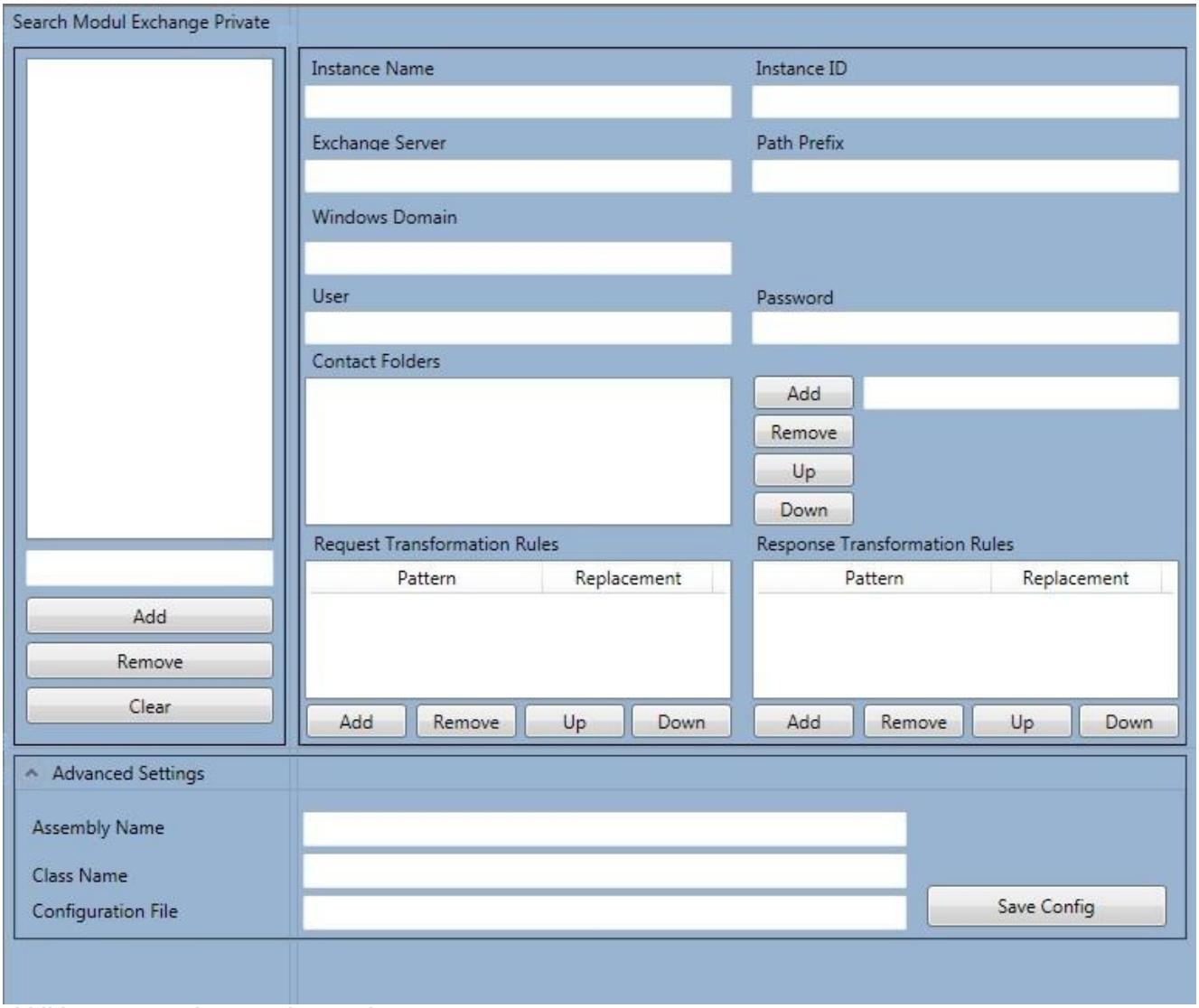


Abbildung 12: Exchange Private Tab

3.3.4. Ldap Tab

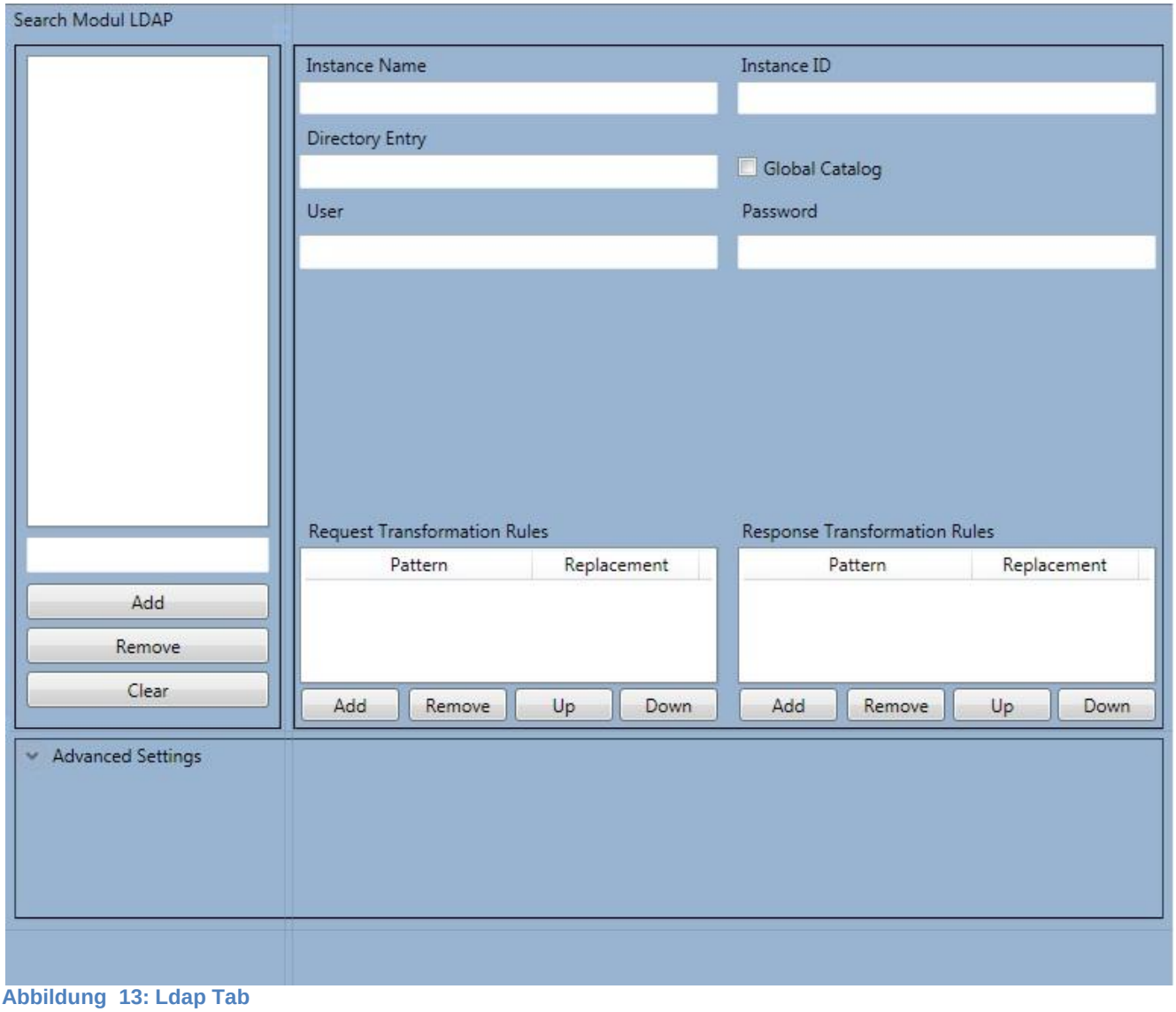


Abbildung 13: Ldap Tab

3.3.5. Sql Tab

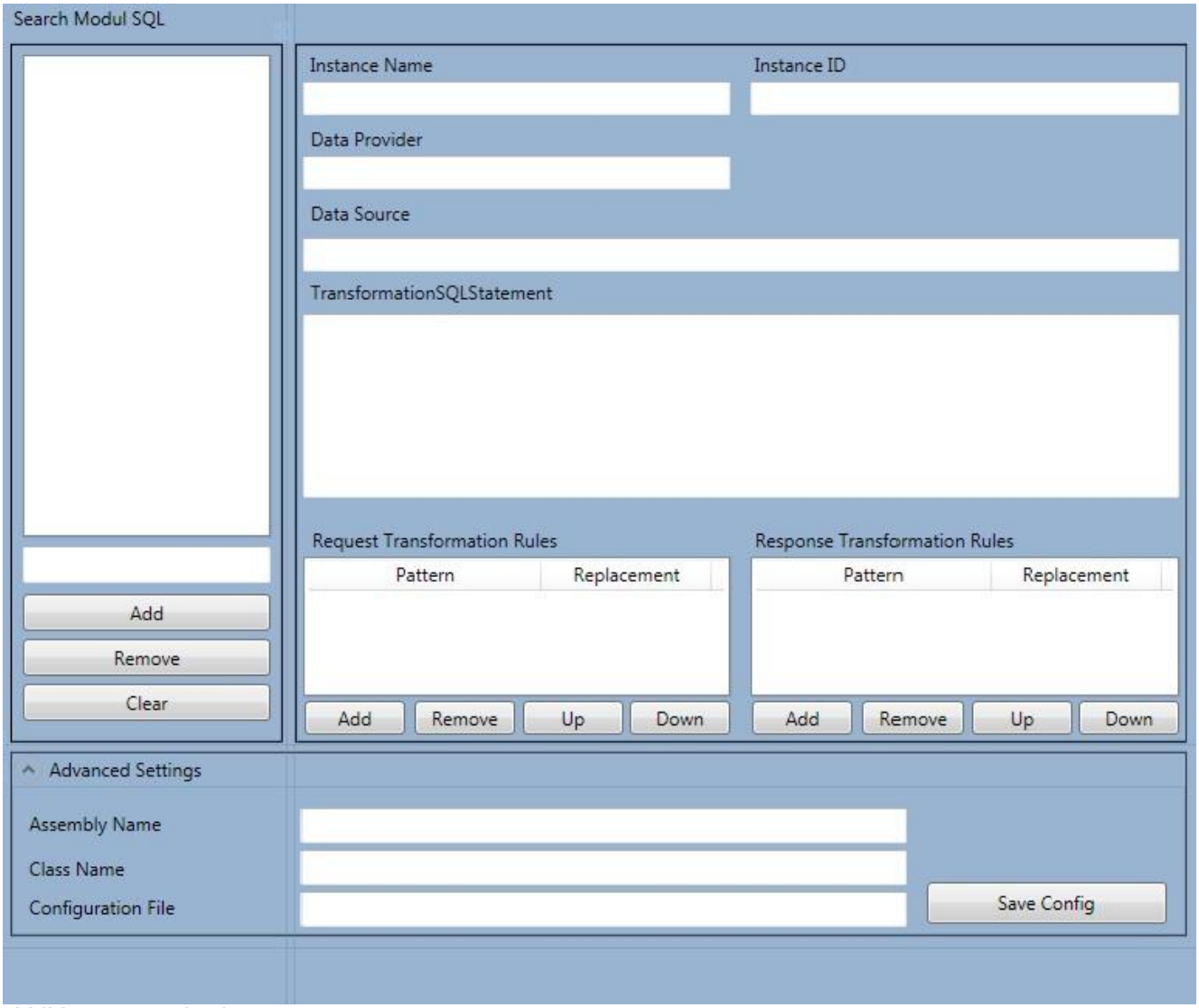


Abbildung 14: Sql Tab

3.3.6. Twixtel Tab

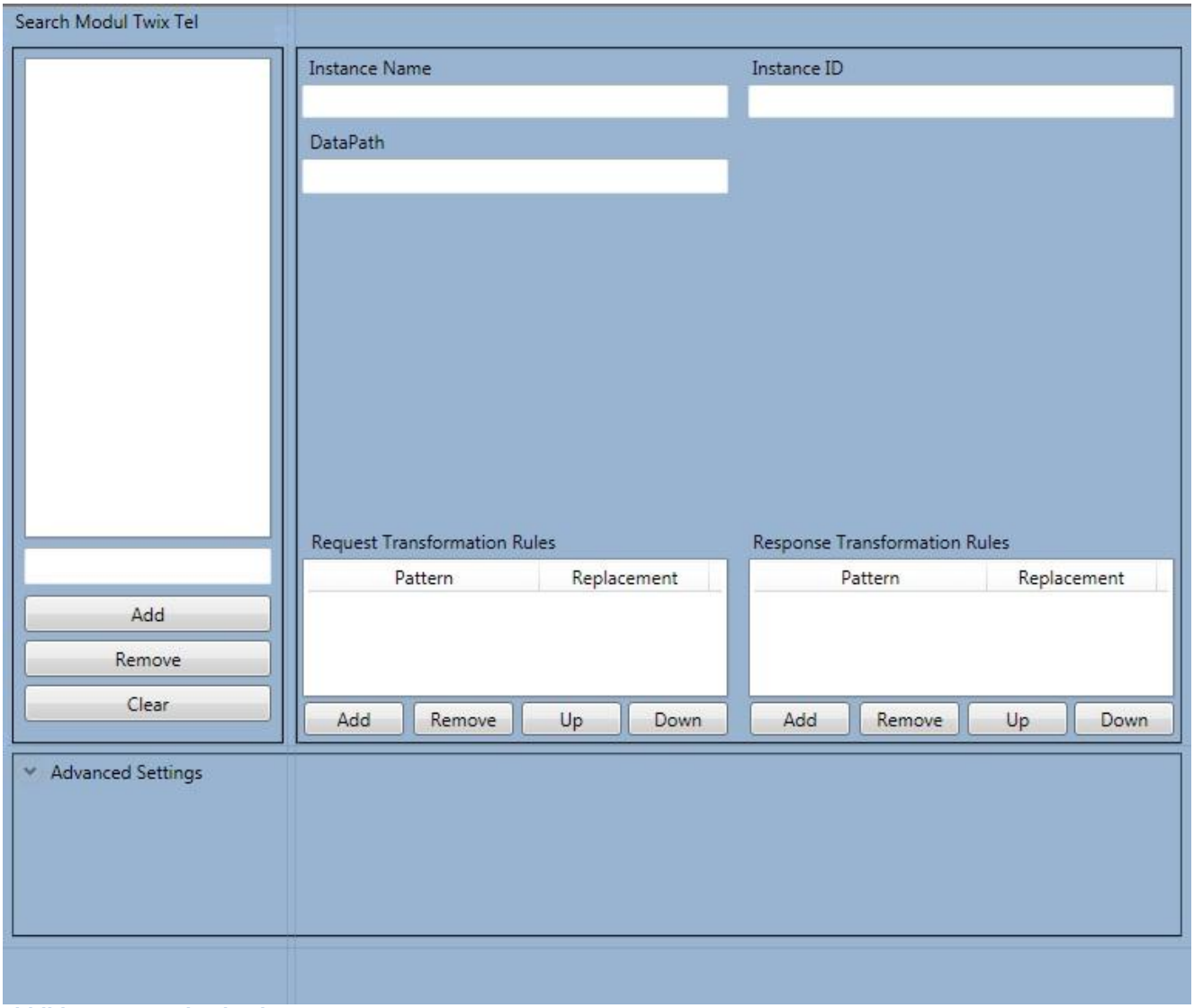


Abbildung 15: Twixtel Tab

4. Software Architektur Dokument

4.1. Umgebung

Das DF GUI wird zusammen mit dem DF ausgeliefert. Es wird nach Beendigung der Semester Arbeit in das bestehende DF integriert.
Die Umgebung des aktuellen Zustandes ist in der nachfolgenden Abbildung dargestellt.

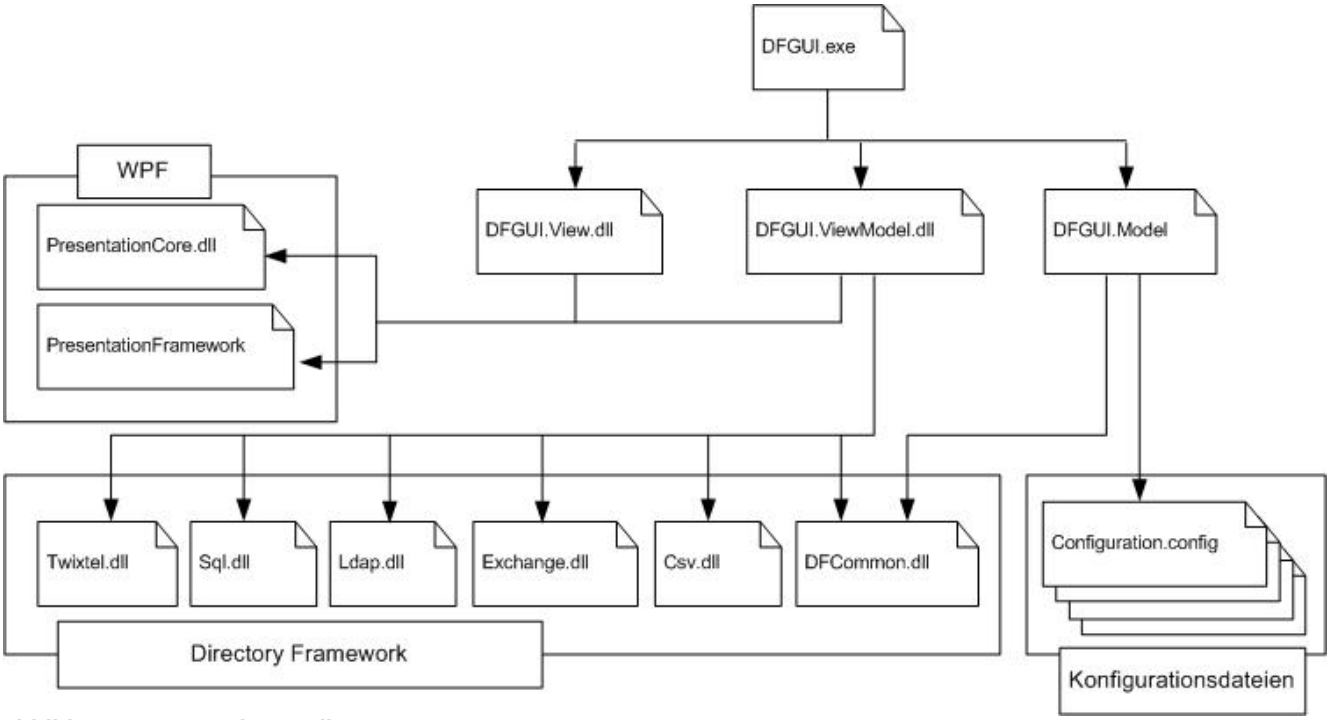


Abbildung 16: Umgebungsdiagramm

4.2. Architektonische Einschränkungen und Ziele

4.2.1. Ziele

- Die GUI Applikation sollte durch Schichten gekapselt werden und die Daten sollten sauber von den Views und UI Komponenten getrennt werden. Zudem sollte die Schichtentrennung eine möglichst grosse Flexibilität erreichen damit man die „oberen“ Schichten unabhängig von den „unteren“ Schichten ersetzen kann.

- Das WPF Project so zu gestalten, dass möglichst wenig Code in den Code Behind Files vorhanden ist, damit die Schichtentrennung möglichst konsequent ist.

- Um die Kompatibilität mit dem DF zu gewährleisten, sollte wo möglich auf Strukturen und Funktionen aus diesem zurückgegriffen werden. Zudem erleichtert dies auch die Wartbarkeit der beiden eng zusammenarbeitenden Applikationen.

- Es muss sichergestellt sein, dass die geschriebenen Konfigurationsdateien vom DF verwendet werden können.

- Es sollte möglich sein, das GUI möglichst einfach mit weiteren SearchModules zu erweitern.

- Interne Fehler der Applikation sollten nicht zu einem Absturz führen, wo möglich sollte der Fehlerfall vermieden werden und eine automatische Fehlerbehandlung durchgeführt werden. Bei Fehlern, wie zum Beispiel korrupte Files, welche das korrekte Einlesen verhindern, sollte der Vorgang abgebrochen werden und dem Benutzer mitgeteilt werden, wo das Problem verursacht wurde.

- Das Aussehen des GUIs, im Sinne von Farben und Formen, sollte einfach verändert werden können um dem Kunden ein auf seine Umgebung abgestimmtes Gesamtbild zu ermöglichen. Zudem sollte auch die optimale Anpassung an das Betriebssystem möglich sein.

- Konstante Default-Werte der Konfigurationen gilt es aus dem Code zu entfernen.

4.2.2. Einschränkungen

- Durch die Verwendung von diversen DF Klassen und Strukturen, musste der Writer gekapselt werden. Die spezifische SearchModuleType Konfiguration ist als Ganzes in einem String gespeichert, somit konnte nicht direkt aus den Objekten ein XML Configurationfile erstellt werden, sondern die XML Abbildung wird zuerst in einen String geschrieben und danach wird dieser String in die Konfigurationsdatei persistiert.

- Der Data Layer besteht durch die Verwendung der DF Klassen, nur aus einer Datenklasse und den Readern und Writern, es gibt keine Kapselung zwischen den verschiedenen SearchModules in Klassen.

- Aus zeitlichen Gründen konnte leider kein Template XAML Control für die verschiedenen SearchModules erstellt werden, welches die Erweiterung mit neuen SearchModules erheblich vereinfachen würde.

4.3. Architektonische Entscheidungen

4.3.1. Schichtenmodel, Model View ViewModel Pattern (MVVM)

Bei einer Entwicklung eines GUIs stellt sich immer die Frage, wie dieses aufgebaut werden soll, wie man eine gute Struktur aufbaut damit man die Daten und die Views sauber kapseln kann. Zudem wird auch immer eine klar ersichtliche Schichtentrennung angestrebt, um eine flexible GUI Applikation zu entwickeln, in der die „obere“ Schicht unabhängig von der „unteren“ Schicht ersetzt werden kann.

4.3.1.1. Faktoren

- Extrahierung der Objekte aus dem DF in die gewünschte Form für die Abbildung in der View.
- Testbarkeit dieser neu generierten Objekte.

4.3.1.2. MVVM – Model View ViewModel[1]

Ein verbreitetes Architekturmuster ist das MVVM Pattern (Model-View-ViewModel), welches als eine Spezialisierung des PM Pattern (Presentation Model) angesehen werden kann und speziell auf die Microsoft Plattformen WPF und Silverlight angepasst wurde. Dieses Pattern fügt eine zusätzliche Abstraktionsebene einer Ansicht ein, das ViewModel, welches den Zustand und das Verhalten dieser Ansicht beinhaltet. Eine View beinhaltet einen Datenkontext, welcher eine ViewModel Klasse darstellt. Die einzelnen Controls der View werden mittels DataBinding an Properties dieser Klasse gebunden und die ganze Darstellungslogik wird im Hintergrund durch dieses Binding gemacht, somit werden Änderungen am ViewModel immer und automatisch in der View dargestellt. Somit ist das ViewModel losgelöst von der View. Das ViewModel führt zusätzlich neben der Datenbereitstellung, auch alle Änderungen an den Modelldaten durch. Die View Klassen sollten keine Kenntnis von den Model Klassen haben und das Model und das ViewModel sollten keine Kenntnisse der Views haben. Durch die Verwendung des MVVM Patterns könnte man auch zu einem gewünschten Zeitpunkt eine Anbindung mittels einer Webserviceschnittstelle und einer Silverlightapplikation in Betracht ziehen und diese mit geringerem Aufwand erzielen, da man auf das Model und das ViewModel der bereits implementierten WPF-Anwendung zurückgreifen könnte.
Weitere Vorteile sind auch die Testbarkeit des GUIs, Tests können auf dem ViewModel implementiert werden und sind vom GUI unabhängig. Es können spezielle ViewModel erstellt werden, die sehr viewnahe sind und nur einen Bruchteil des Models beinhalten.
Auch eine Auswechslung des GUIs ist relativ einfach möglich, da man die Views auf das bestehende ViewModel binden kann.

4.3.1.3. MVP – Model View Presenter[2]

Bei der GUI Entwicklung kann man auf verschiedene und sehr verbreitete und anerkannte Architekturmuster zurückgreifen, welche ihre Funktionsfähigkeiten in vielen Applikationen unter Beweis stellen. Ein weit verbreitetes und beliebtes Architekturmuster ist das MVP Pattern (Model-View-Presenter) welches auf dem MVC (Model-View-Controller) basiert. In diesem Pattern werden die Daten (Model) in der View dargestellt und der Presenter verknüpft diese beiden Schichten. Die Daten welche in der View dargestellt werden, sind das Model und die Fenster für die Benutzerinteraktion sind die Views. Der Presenter ist für die Datenbereitstellung der Views verantwortlich und reagiert auf Benutzerinteraktionen, auch zusätzliche Funktionen wie Eingabeprüfung können vom Presenter übernommen werden.

4.3.1.4. Designentscheid

Die Entscheidung, das Directory Framework GUI mittels des MVVM Pattern zu realisieren ist uns relativ leicht gefallen. Das Directory Framework, für welches unser GUI eine Konfigurationshilfe bieten sollte, basiert auf der .Net Technologie und ist somit an die Microsoft Plattform gebunden. Da Microsoft mit WPF ein starkes Framework zur Verfügung stellt und auch aus dem Projektbeschrieb (Projektantrag) hervorging, dass die Realisation mit aktuellen und verbreiteten Technologien erfolgen sollte, war der Entscheid mit WPF zu entwickeln gefallen.
Die Architektur, bei einer WPF-Applikation, mittels des MVVM Pattern zu realisieren bei ist naheliegend, da dieses Architekturmuster speziell auf WPF und Silverlight zugeschnitten ist. Zudem erreicht man mit diesem eine lose Kopplung zwischen den Layern und man könnte zu einem späteren Zeitpunkt auch eine Portierung vornehmen zu einer Silverlight Lösung über eine Webservice Anbindung.
Das Directory Framework GUI ist eine etwas spezielle Lösung um das Directory Framework zu konfigurieren. Das GUI wird benutzt um Konfigfiles zu bearbeiten und enthält keine Applikation im eigentlichen Sinne im Hintergrund, sondern informiert das Directory Framework nach geänderter Konfiguration, welches sich dann neu startet um die Konfigfiles neu zu laden. Somit kann das GUI je nach Kundenwunsch wiederum anders sein und

[1] http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
[2] http://msdn.microsoft.com/en-us/magazine/cc188690.aspx

man könnte eine kundenbasierte GUI Lösung anbieten. Denn das Model und ViewModel, das auf die im Hintergrund stehenden Konfigfiles angepasst sind, ändern sich nicht, somit kann die View verändert und gebunden werden.

4.3.2. Verwendung von Commands

Um eine optimale Schichtentrennung zu erreichen, sollte möglichst wenig Code in den WPF Code Behind Dateien enthalten sein. Dies kann erreicht werden durch den Verzicht auf User Control Events. Durch diesen Verzicht kann der Code der Logik von der Codebehind Datei in ein ViewModel verschoben werden, dass in einer eigens dafür vorgesehenen Schicht beheimatet ist und trotz der Schichtentrennung mittels WPF auf ein Control der View gebunden werden kann.

4.3.2.1. Faktoren

- Bessere Schichtentrennung
- Verringerung des Codes in der Codebehind Datei
- Verwendung eines Commands auf mehreren Views, Reduktion von doppeltem Code

4.3.2.2. Lösung, Verwendung von Commands

Logikfunktionen welche in allen oder mehreren Tabs benutzt werden, wurden in der abstrakten ViewModel Superklasse *ViewModelBase* implementiert. Commands welche im ViewModel implementiert sind, können im XAML File direkt einem Control zugewiesen werden, sofern der *DataContext* der View auf das ViewModel gesetzt ist. Das setzen des *DataContext* geschieht während der Initialisierung eines Tabs im Konstruktor der Codebehind Datei.

Beispiel einer Codebehind Datei eines SearchModuleType Tab, hier Twixtel:

```
/// <summary>
/// Interaction logic for TabItemTwixTel.xaml
/// </summary>
public partial class TabItemTwixTel : UserControl
{
  private TwixTelViewModel twixTelViewModel;

      public TabItemTwixTel(Mainconfig mainconfig)
      {
            this.InitializeComponent();
      twixTelViewModel = new TwixTelViewModel(mainconfig);
      DataContext = twixTelViewModel;
      }
}
```

Beispiel einer Zuweisung eines Commands auf ein Control, hier in der Twixtel View und auf dem Save Config Button:

```
<Button Command="{Binding Path=SaveConfigCommand}" . . . />
```

Wobei der *SaveConfigCommand* ein implementierter *RelayCommand* im ViewModel ist.

4.3.2.3. Erwogene Alternativen

Die Verwendung von den Control Events, welche einen erheblichen Teil der Logik auf dem UI Layer bedeutet hätte und zusätzlich viel duplicated Code verursacht hätte.

4.3.3. Verwendung von DF Komponenten

Die Funktionalität des Einlesens der Konfigurationsdateien, wird vom DF sowie auch vom GUI benutzt. Die Verwendung derselben Readfunktionen birgt viel Vorteile in sich, verlangt jedoch auch die Verwendung derselben Datenstruktur oder legt die Verwendung dieser Struktur nahe.

4.3.3.1. Faktoren

- Wartbarkeit
- Sicherstellung der Funktionalität

4.3.3.2. Lösung

Die Daten das Data Layers sollten auf den Objekt Klassen des DFs aufbauen und in derselben Struktur nach dem Einlesen oder Neuerstellen abgespeichert werden. Dies war eine Anforderung nach der Analyse der Readfunktionen des DFs und dem Entscheid, diese Funktionen zu verwenden. Durch die Verwendung des DFs für das Lesen der Konfigurationsdateien, kann die Wartbarkeit beider Applikationen erheblich verbessert werden, da der Code nur einmal vorhanden ist. Zudem kann sichergestellt werden, dass die Konfigurationsdateien in beiden Applikationen funktionieren, wenn der Readprozess erfolgreich ausgeführt wurde.

4.3.3.3. Erwogene Alternativen

Die Verwendung desselben Readers, die eingelesenen Daten jedoch in eine eigene Datenstruktur im Data Layer zu speichern. Dies würde allerdings redundante Daten auf demselben Layer bedeuten, was auch nicht gerade erstrebenswert wäre. Eine eigene auf die neuangelegte Datenstruktur abgestimmte Readfunktion zu implementieren, würde allerdings die Funktionsfähigkeit der Konfigurationsdateien in Frage stellen. Es würde die Testbarkeit um einiges erschweren. Ein weiterer Nachteil wäre die Wartbarkeit, wenn das Konfigurationsfile seine Struktur ändern würde, müssten die Readfunktionen des DFs und des GUIs angepasst werden.

4.3.4. Fehlerbehandlung

In dieser GUI Applikation gibt es diverse Möglichkeiten, bei denen ein Fehlerfall (Exception) auftreten kann, so zum Beispiel beim Lesen und Schreiben der Konfigurationsfiles oder bei der Wahl des Speicherpfads oder wenn ein falsches File eingelesen werden möchte.

4.3.4.1. Faktoren

- Stabiles Funktionieren der Applikation
- Automatische Fehlerbehandlung wo möglich

4.3.4.2. Lösung

Exceptions werden wo möglich automatisch behandelt, wie zum Beispiel dass ein File welches nicht vorhanden ist automatisch generiert wird und der Benutzer nichts von diesem Fehlerfall merkt. Wenn der Fehler nicht automatisch behandelt werden kann, sollte eine Benachrichtigung per *MessageBox* dem User den Fehler möglichst informativ beschreiben und die aktuelle Aktion abbrechen, jedoch ohne das die Applikation abstürzt. Es sollte dem Benutzer mitgeteilt werden wo und warum der Fehleraufgetreten ist, zum Beispiel beim Einlesen eines korrupten Files, damit in diesem Beispiel der Fehler ausserhalb der GUI Applikation behoben werden kann.

4.3.4.3. Erwogene Alternativen

Keine erwogenen Alternativen, aber eine Erweiterung der Fehlerbehandlung durch den Einsatz von einem Logger, welcher zusätzlich noch einen Output in ein File generiert.

4.3.5. GUI Farb- und Formanpassung

Da das Layout, die Farb- und Formgestaltung bekanntlich Geschmackssache sind, sollten diese beiden Aspekte veränderbar sein. Zudem kann die Gestaltung des GUIs mit einem aktuellen Betriebssystem (Windows 7 oder Vista) dazu führen, dass das GUI in älteren Betriebssystemen (Windows XP), fast bis zur Unerkenntlichkeit entstellt wird. Windows Vista ist die erste Windowsversion welche auf WPF zugeschnittene Treiber enthält, die ein Anti-Aliasing erlauben.

4.3.5.1. Faktoren

- Sicherstellung der Benutzbarkeit des GUIs auch in älteren Betriebssystemen
- Eventuelle Anpassung nach Kundenwunsch und optimale Einbettung in seine Umgebung

4.3.5.2. Lösung

Die Farbanpassung kann im MainWindow.xaml vorgenommen werden, und zwar kann man die Fensterfarbe im Tag *Window* und die Farbe der Tabs im Tag *TabControl* anpassen, das Attribut heisst in beiden Fällen Background.

Die Form, gemeint sind hier die Tabs in der TabControl, welche abgerundet und weiss hinterlegt sind. Diese Eigenschaften können im TabItemTemplate.xaml geändert und nach Wunsch angepasst werden.

4.3.5.3. Erwogene Alternativen

Keine.

4.3.6. Konstanten

Um gewisse Standardwerte im GUI zu setzen, werden konstante Felder benötigt, das Handling dieser Konstanten sollte mittels einer Verwaltung gesteuert werden.

4.3.6.1. Faktoren

- Einfacher Zugriff
- Zentraler Sammelpunkt aller Konstanten
- Entfernung aus dem Code

4.3.6.2. Lösung

Alle Konstanten werden in einer konstanten Klasse *Constants* abgelegt.
Für gewisse Konstanten wird allerdings auch die *Constants* Klasse aus dem DF verwendet. So zum Beispiel um den Service neu zu starten. Die Konstante wird vom DF wie auch vom GUI verwende. Die *Constants* Klasse vom GUI beinhaltet nur konstante Felder welche nur vom GUI verwendet werden. Dies sind vor allem Standard-Konfigurationswerte.

4.3.6.3. Erwogene Alternativen

Konstanten verteilt im Code zu lassen, was den Komfort der Wartung erheblich vermindern würde.

4.4. Architekturkonzepte

4.4.1. Schichtenmodell

Das DF-GUI ist eine reine GUI-Applikation und ist nach dem MVVM Pattern aufgebaut. Es besteht aus den Schichten View, welche die GUI Views enthält, ViewModel welche die Daten des Models abbildet und den Views zur Verfügung stellt und dem Model, das Datenstruktur bildet. Die einzelnen Layers sind in verschiedene Projekte unterteilt. So ist die View ein WPF Project und das ViewModel und das Model sind Libraries, die eingebunden sind. Die Persistenz bilden die Konfigurationsdateien, welche eine bestimmte XML Struktur beinhalten, die vom Directory Framework vorgegeben ist, welche im Dateisystem abgelegt werden.

4.4.1.1. Übersicht

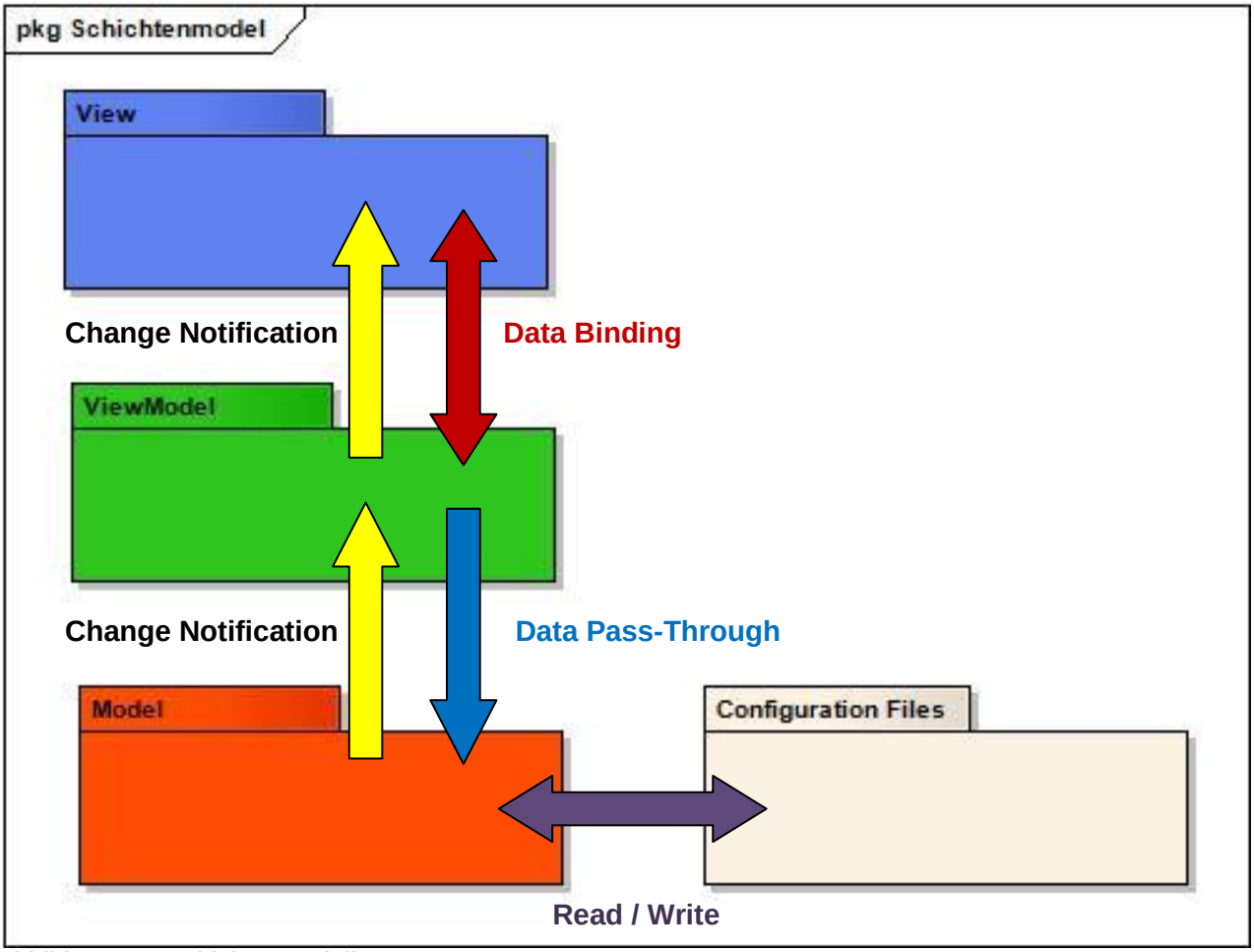


Abbildung 17: Schichtenmodell

4.5. Design Entscheidungen

4.5.1. RelayCommands [3]

Die Verwendung der RelayCommand Logik wird oft zusammen mit dem MVVM Pattern verwendet. Sie erlaubt die Auslagerung von GUI Funktionalitäten von den Codebehind Dateien in die eigenen ViewModels. Ein weiterer Ansatz wäre, dass man das *ICommand* Interface jeweils in den ViewModels implementiert. Dies kann aber zu viel dupliziertem Code führen. Abhilfe bietet hier die RelayCommand Logik, welche das Implementieren eines Commands mittels delegate im Konstruktor erlaubt.
Die *RelayCommand* Klasse implementiert das *ICommand* Interface, welches explizit verlangt, dass man folgende Codefragemente implementiert:

- Prädikat *CanExecute* → bool CanExecute(object parameter)
- EventHandler *CanExecuteChanged* → event EventHandler CanExecuteChanged {add …} {remove …}
- Methode *Execute* → void Execute(object parameter)

Zudem sind zwei Konstruktoren implementiert. Der eine nimmt ein Action delegate Objekt entgegen und sollte verwendet werden, falls ein Command immer ausgeführt werden kann. Intern nutzt dieser den zweiten Konstruktor, welcher als zweiter Parameter noch ein Predicate delegate Objekt entgegen nimmt, welches benützt werden kann, um eine Bedingung zu definieren für die Ausführung des Commands.

Ein Beispiel anhand des *LoadConfigurationCommand* aus dem *MainconfigViewModel*:

```
private ICommand loadConfigurationCommand;
public ICommand LoadConfigurationCommand
{
  get
  {
      if(loadConfigurationCommand == null)
      {
         loadConfigurationCommand = new RelayCommand(
         param => this.LoadConfigurationFromFile(),
         param => this.CanLoadConfigurationFromFile());
      }
      return loadConfigurationCommand;
  }
}
```

Hier ist ein Codeausschnitt der das Instanziieren eines RelayCommand aufzeigt. Mittels delegate und lambda expression werden hier die Parameter ExecuteMethod und CanExecutePredicate mitgegeben.
In der nachfolgenden Abbildung ist das Vorgehen noch bildlich dargestellt.

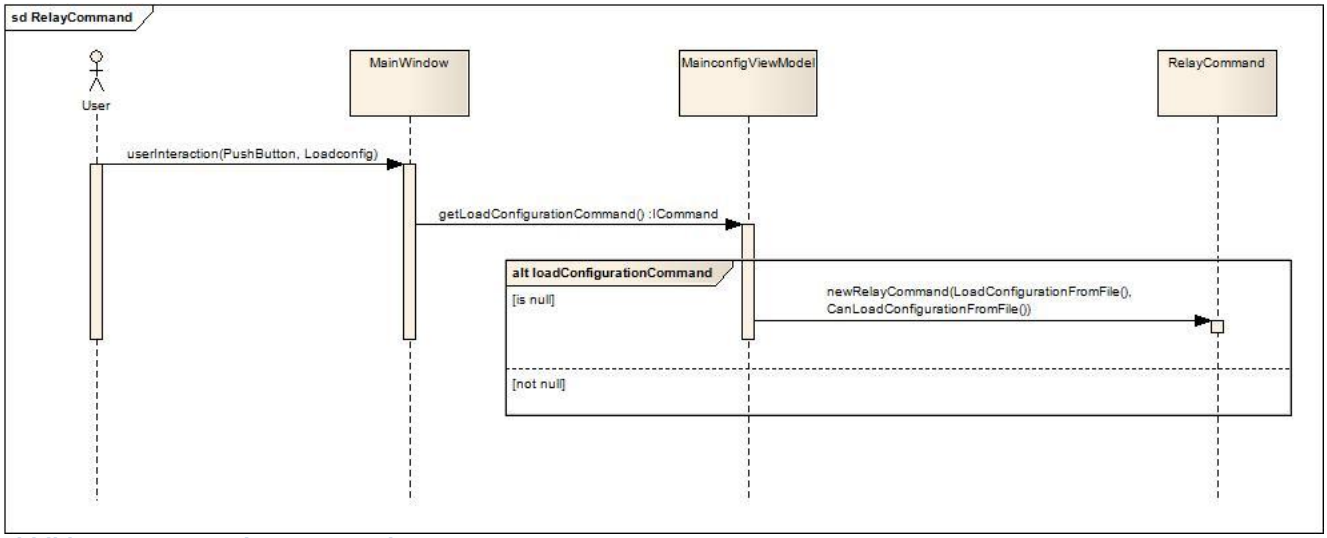


Abbildung 18: SD RelayCommand

[3] http://msdn.microsoft.com/en-us/magazine/dd419663.aspx#id0090030

4.5.2. ConfigurationChangeController

Der *ConfigurationChangeController* dient hauptsächlich der Kommunikation zwischen den verschiedenen ViewModels. Nach dem Laden einer Konfiguration aus dem Mainconfig Tab durch ein Buttonklick „Load Configuration“, werden die Daten mittels der Readerklasse in das Model geladen, danach sollten diese Daten in die ViewModels geladen werden, damit die neu eingelesene Konfiguration auf die Views projeziert werden können. Für die Daten aus dem *MainconfigViewModel* ist dies kein Problem, die anderen ViewModels müssen aber benachrichtigt werden, damit diese auch die aktuelle Konfiguration aus dem Model laden können. Dies geschieht mit dem *ConfigurationChangeController*, der als Singleton implementiert ist und somit für alle zugänglich ist. Beim Applikationsstart werden die Tabs und ihre dazugehörende ViewModels instanziiert, im Konstruktor der einzelnen ViewModels registrieren sie sich beim Event *ConfigurationChanged* mit der Methode *OnConfigurationChanged*, welche beim Auslösen des Events ausgeführt wird. Der *ConfigurationChanged* Event ist im *ConfigurationChangedController* implementiert.

Die untenstehende Abbildung zeigt die Registration an dem Event, Beispiel anhand des LdapVM.

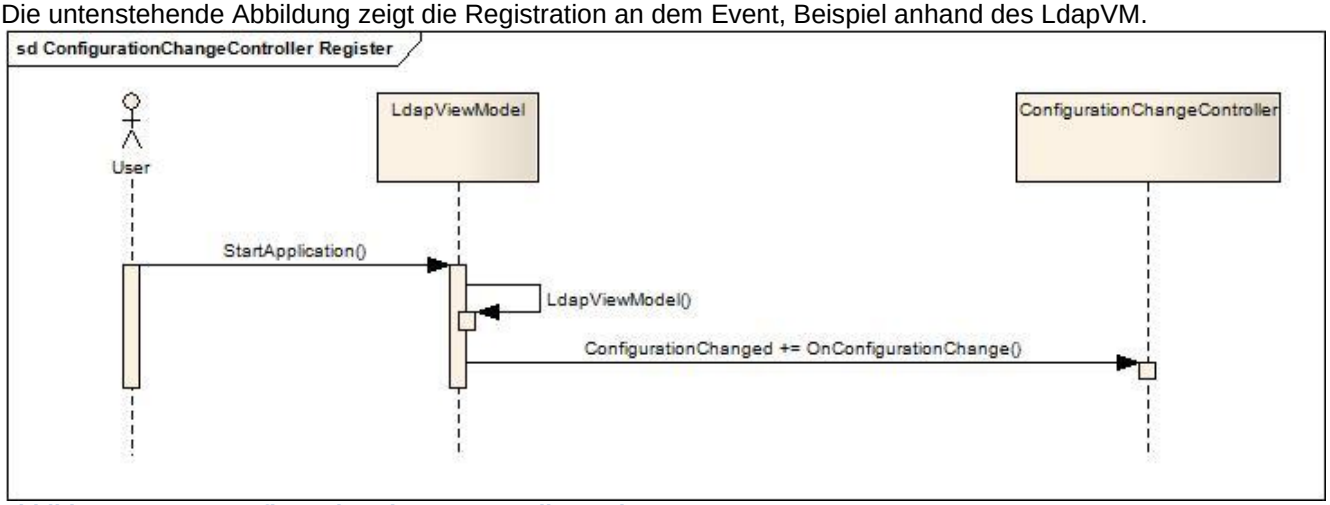


Abbildung 19: SD ConfigurationChangeController register

Wenn nun die ViewModels am Event registriert sind, kann das *MainconfigViewModel* nach dem Lesen aus dem Model die Fire Methode aufrufen, welche ebenfalls auf dem *ConfigurationChangeController* implementiert ist und den Event auslöst.
Jetzt werden alle ViewModels informiert, welche sich registriert haben und führen die registrierte Methode aus. In diesem Beispiel ist das die *ReadFromModel* Methode. Somit werden alle in der Gesamtkonfiguration enthaltenen Daten gelesen und auf die Views projeziert.

Die untenstehende Abbildung zeigt die Auslösung des Events, Beispiel anhand des *LdapViewModels*.

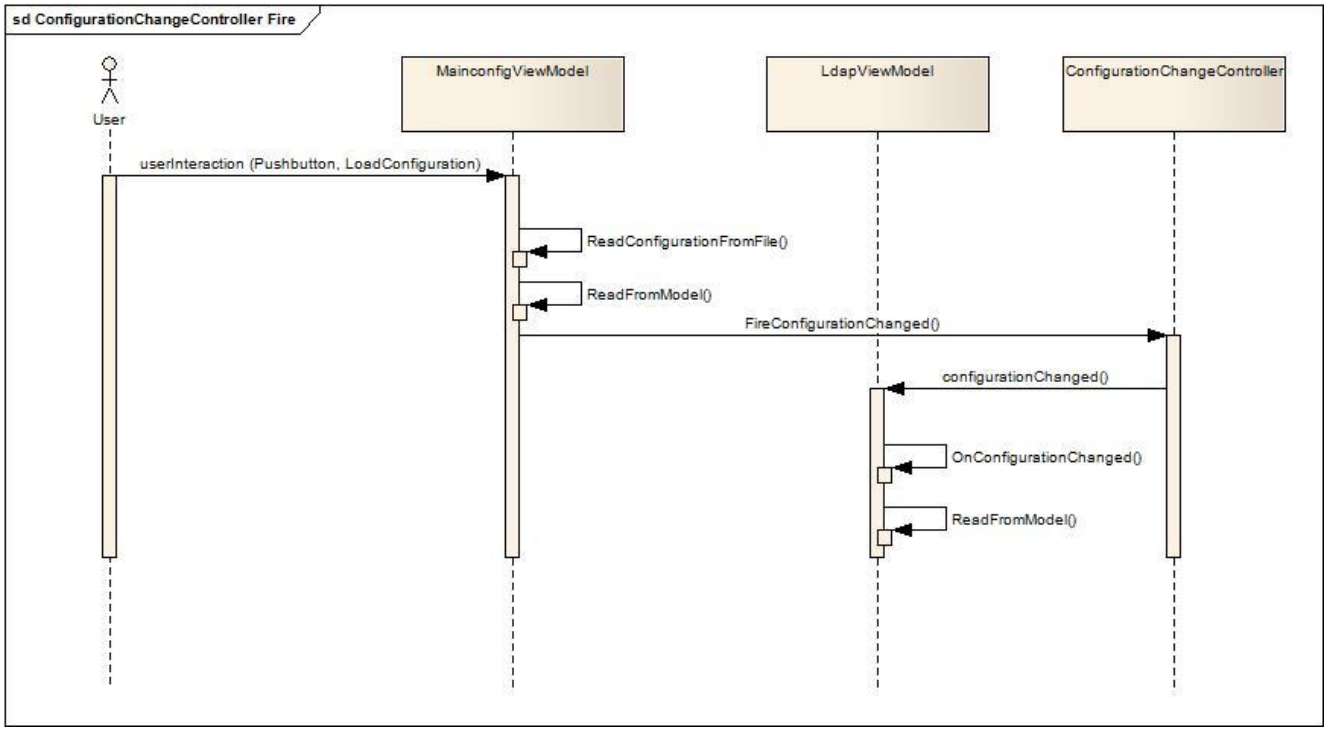


Abbildung 20: SD ConfigurationChangeController fire

Auf dem *ConfigurationChangeController* ist noch ein zweiter Event implementiert, welcher benutzt wird falls der Save Button betätigt wird. Auch bei diesem Event registrieren sich die ViewModels während der Instanziierung. Nun wird der Event gefeuert, wenn die Konfiguration gespeichert werden sollte, allerdings nur wenn die Speicherung aus dem *MainconfigViewModel* ausgelöst wird. Danach läuft es genau gleich ab wie bei oben beschriebenem Beispiel, nur dass die Daten diesmal ins Model und danach in die Konfigurationsfiles geschrieben werden.

4.6. Logische Architektur

Das DF GUI beinhaltet eine Daten- (Model), eine Logik/Daten- (ViewModel) und eine Userinterfaceschicht (View). Diese drei Schichten repräsentieren das klassische MVVM Entwurfsmuster. Die Daten werden mittels Writer Klassen aus dem Model persistiert. Diese Auslagerung der Daten in die Konfigurationsdateien wird je nachdem vom Kunden in das Installationsverzeichnis des DFs oder an einem beliebig selektierten Ort gespeichert.

4.6.1. Übersicht

Die Applikation DF GUI besteht aus der Solution *DirectoryFramework-GUI*, diese ist in drei Hauptprojekte aufgeteilt.

- *DFGUI.View*
- *DFGUI.ViewModel*
- *DFGUI.Model*

Die Projekte wiederum sind durch Namespaces gegliedert. Die einzelnen Klassen befinden sich direkt in den Projects oder innerhalb eines Namespaces. Folgende Abbildung sollte die Struktur und Solutionorganisation veranschaulichen.

4.6.2. Packagestruktur

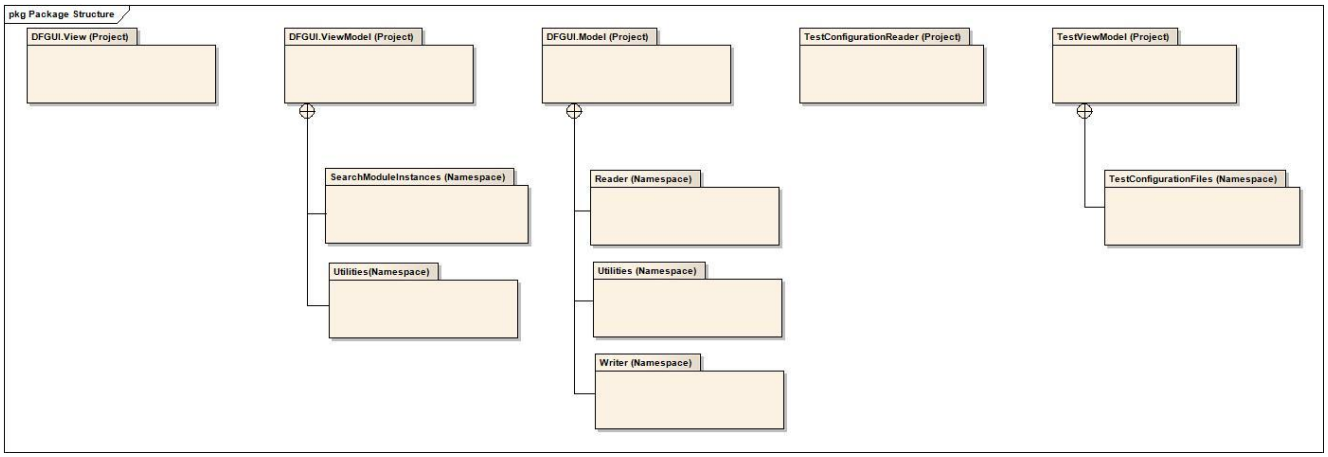


Abbildung 21: Packagestruktur

4.6.3. Solutionübersicht

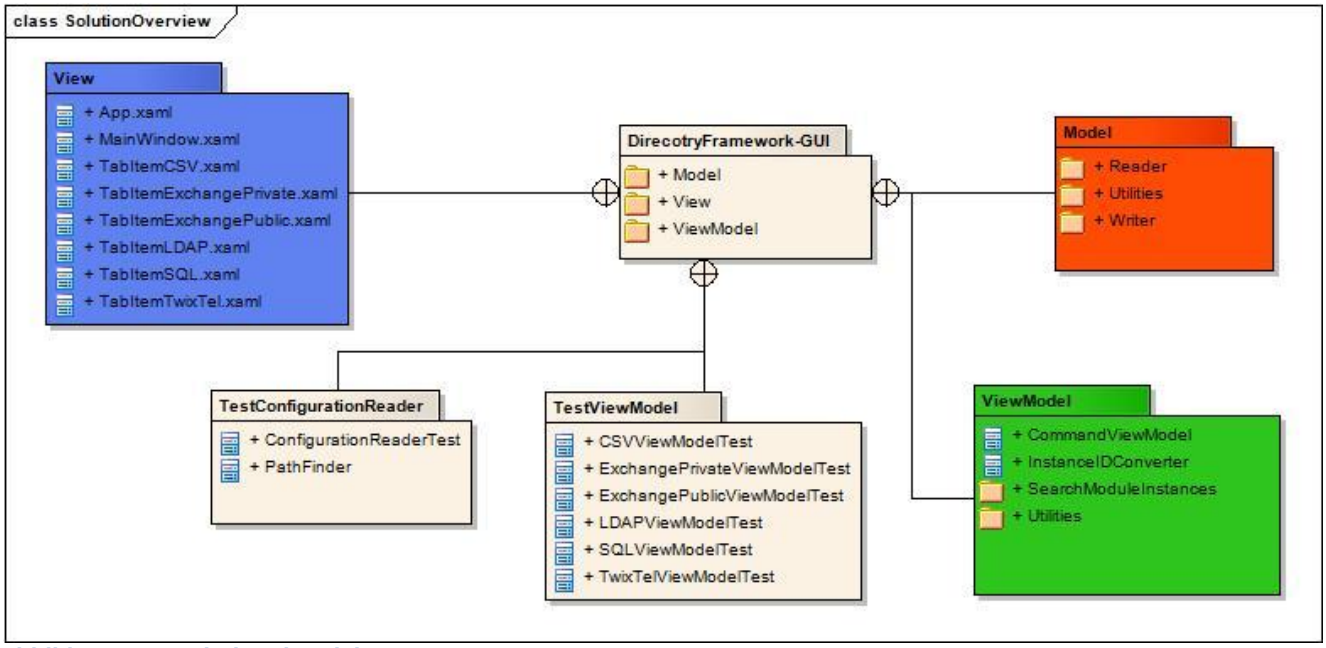


Abbildung 22: Solutionübersicht

4.6.4. View

In diesem Projekt sind die XAML Files enthalten mit den dazugehörigen Codebehind Dateien. Einzig die *Mainconfig* Codebehind Datei erhält mehr Logik als die Definition des DataContext. Sie enthält das Tab Management, das aufgrund der SearchModule Selektion, Tabs ein und ausblendet. Im Project *DFGUI.View* gibt es keine Namespaces welche das Project gliedert.

4.6.5. ViewModel

Im ViewModel wird das Datenabbild aus dem Model auf die Views generiert. Die Daten aus dem Model werden in eine neue Datenstruktur gebracht, die optimal auf die jeweilige View abgestimmt ist. Zudem ist die ganze Benutzerinteraktionslogik in diesem Project in den einzelnen ViewModel Klassen implementiert. Jedes ViewModel wird an eine View gebunden und stellt deren DataContext dar. Die beiden Namespaces in diesem Projekt dienen zum einen um die neue Datenstruktur abzubilden, zum anderen sind es Helper Klassen um die konstanten Felder zur Verfügung zu stellen und um den Speichervorgang aller Konfigurationsdateien zu unterstützen.

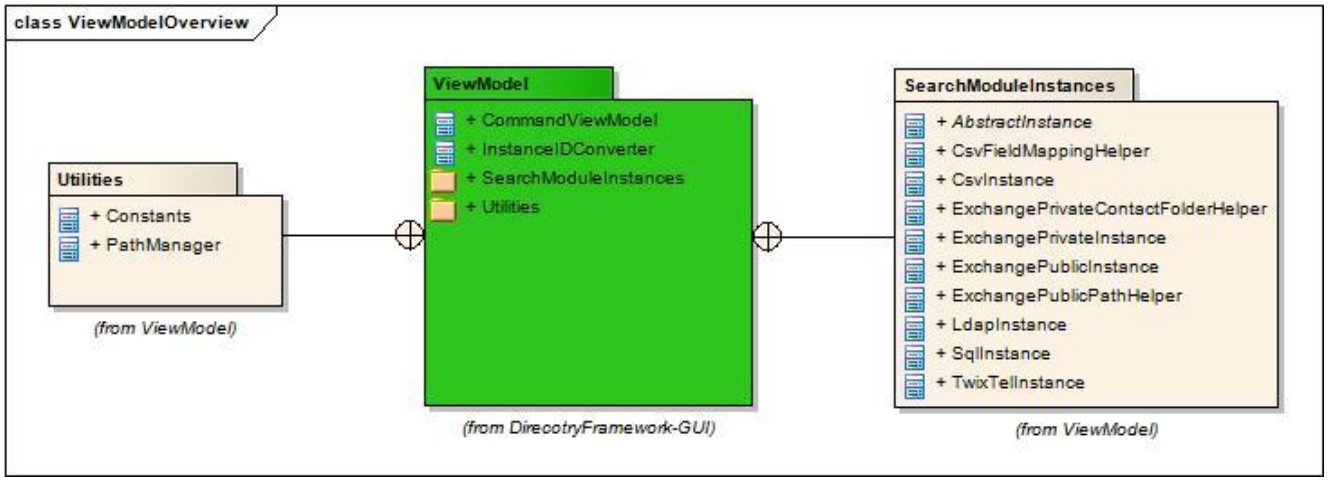


Abbildung 23: ViewModelübersicht

4.6.6. Model

Das Projekt Model repräsentiert den Data Layer und beinhaltet die Klasse *Mainconfig*. Diese Klasse enthält die gesamte Datenstruktur, welche für die Lese- und Schreibfunktionen benötigt werden. Zudem befinden sich die Reader- und Writer Klassen und die *Constants* Klasse für die Konstantenfelder im Data Layer in diesem Projekt.

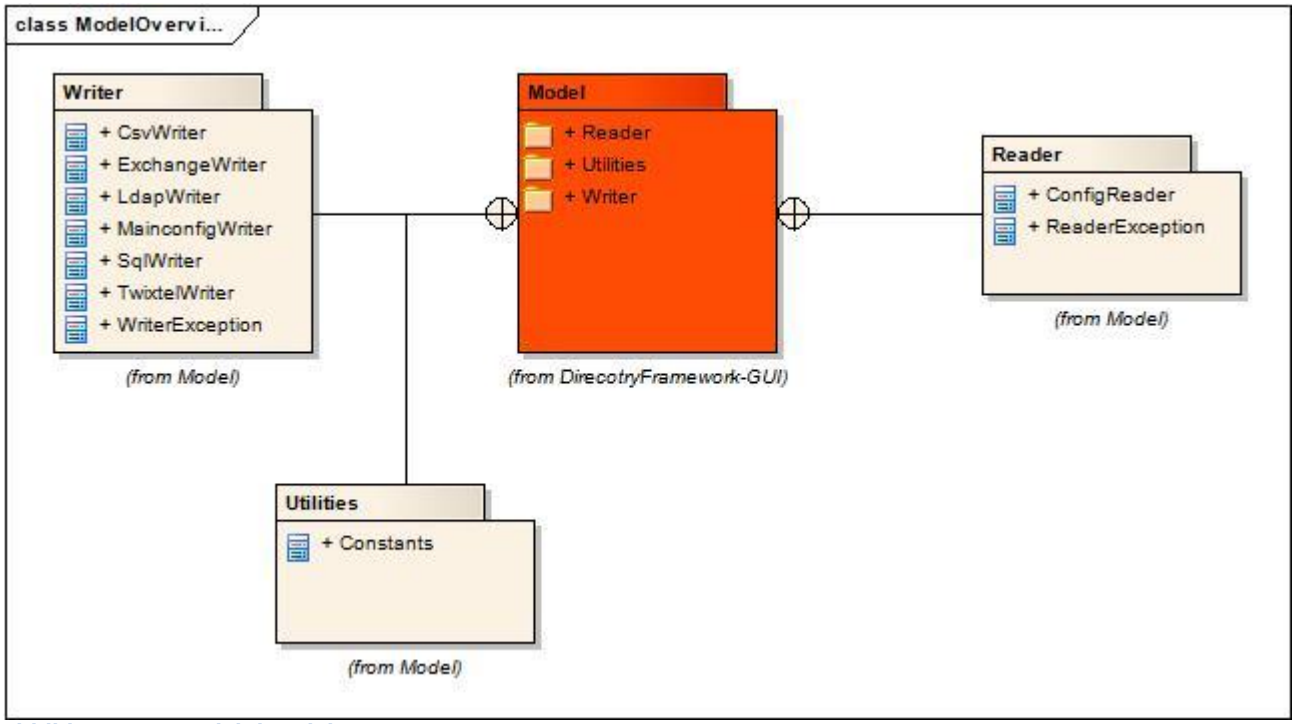


Abbildung 24: Modelübersicht

4.6.7. Persistenz

Die Persistenzschicht wird durch die Konfigurationsfiles dargestellt.
Weitere Informationen sind im Abschnitt *4.9 Datenspeicherung* ersichtlich.

4.7. Design Pakete

4.7.1. Package View

4.7.1.1. Beschreibung des Packages

Die einzelnen Views sind in XAML Files definiert. Das *MainWindow* leitet von *Window* ab und bildet das einzige Fenster des GUIs. Es beinhaltet ein *TabControl*, welches die einzelnen Tabs steuert und die *UserControls* des ersten Tab, welches die Mainconfig abbildet und den *DataContext* des *MainconfigViewModel* erhält. In der Codebehind Datei (MainWindow.xaml.cs) ist die Logik für die Taberzeugung implementiert sowie das Setzen des *DataContext*.

In den einzelnen TabItem XAML Files sind die *UserControls* der SearchModuleTypes definiert. Auch bei diesen wird der Datenkontext in den Codebehind Dateien gesetzt. Zudem wird in allen XAML Files das Binding auf die einzelnen Properties des zugewiesenen ViewModels gemacht.

4.7.1.2. Klassendiagramm

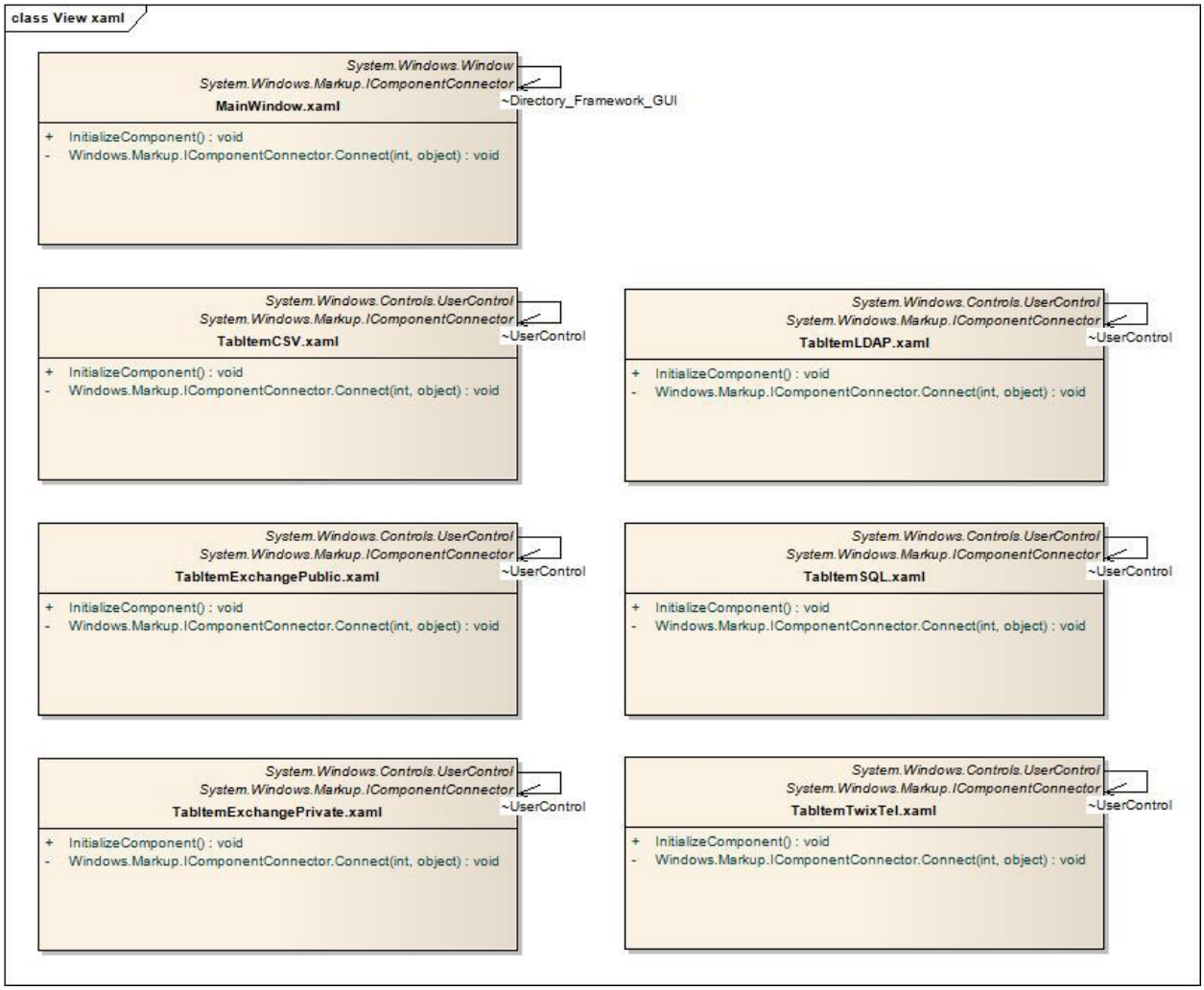


Abbildung 25: Klassendiagramm View XAML Files

Übersichtshalber wurde auf die Attribute im Klassendiagramm verzichtet, da in den XAML Files alle Controls, welche deklarativ definiert sind, als Attribute dargestellt werden. Alle Controls sind mit einem Namen versehen, welcher mit dem Typ beginnt und danach seine Funktionalität beschreibt.

Beispiel eines Buttons:

```
<Button Name="buttonSaveConfig" . . . >
```

Abbildung 26: Klassendiagramm View XAML.CS (Codebehind)

4.7.1.3. Klassenbeschreibung

Klasse	Beschreibung
App	Die *App* Klasse ist die Startup Klasse des GUIs, welche von WPF generiert wurde und die Main Methode beinhaltet.
MainWindow	*MainWindow* erbt von *Window* und ist das einzige Fenster der Applikation. Setzt den *DataContext* des *MainWindow* auf das *MainconfigViewModel* und beinhaltet die Logik der Generierung der TabItems (CreateTabItemX-Methoden) und deren Sichtbarkeit (x_Checked- / x_Unchecked-Methoden).
TabItemCSV	Alle TabItem Klassen erben von *UserControl* und setzen in ihren Konstruktoren den *DataContext* auf das jeweilige ViewModel, welche gerade Initialisiert werden.
TabItemExchangePublic	
TabItemExchangePrivate	
TabItemLDAP	
TabItemSQL	
TabItemTwixTel	

Tabelle 22: Klassenbeschreibung View

4.7.2. Package ViewModel

4.7.2.1. Beschreibung des Packages

In diesem Layer wird eine zusätzliche Abstrahierung eingebaut mit gewollter Redundanz. Im Normalfall lässt sich so eine sehr viewnahe Zwischenschicht einfügen, welche nur ein Bruchteil des gesamten Models repräsentieren kann. In diesem Projekt ist dies nicht der Fall, da das DF-GUI abgekapselt vom eigentlichen Framework ist. In dieser Schicht werden die Daten aus dem Model in eine eigene Struktur gemappt welche der einzelnen Views sehr nahe kommt. Die eingelesenen Daten sind in verschachtelten Objekten abgelegt, im ViewModel werden diese zu einzelnen Instanzen zusammengestellt, damit sich diese wunschgemäss in einer View darstellen lassen. Durch dieses Mapping enthält das ViewModel einige Klassen mehr als das Model, in dem alles zentral in einer Klasse abgelegt ist. Zusätzlich sind auch noch Funktionen und Commands in den einzelnen ViewModel-Klassen implementiert. Die einzelnen ViewModels werden mittels Data Binding an die dazugehörige View gebunden, und sie referenzieren auf die Datenstruktur des Models.

4.7.2.2. Klassendiagramme

4.7.2.2.1. *ViewModel*

Siehe *Anhang A*

4.7.2.2.2. *ViewModel SearchModuleInstance*

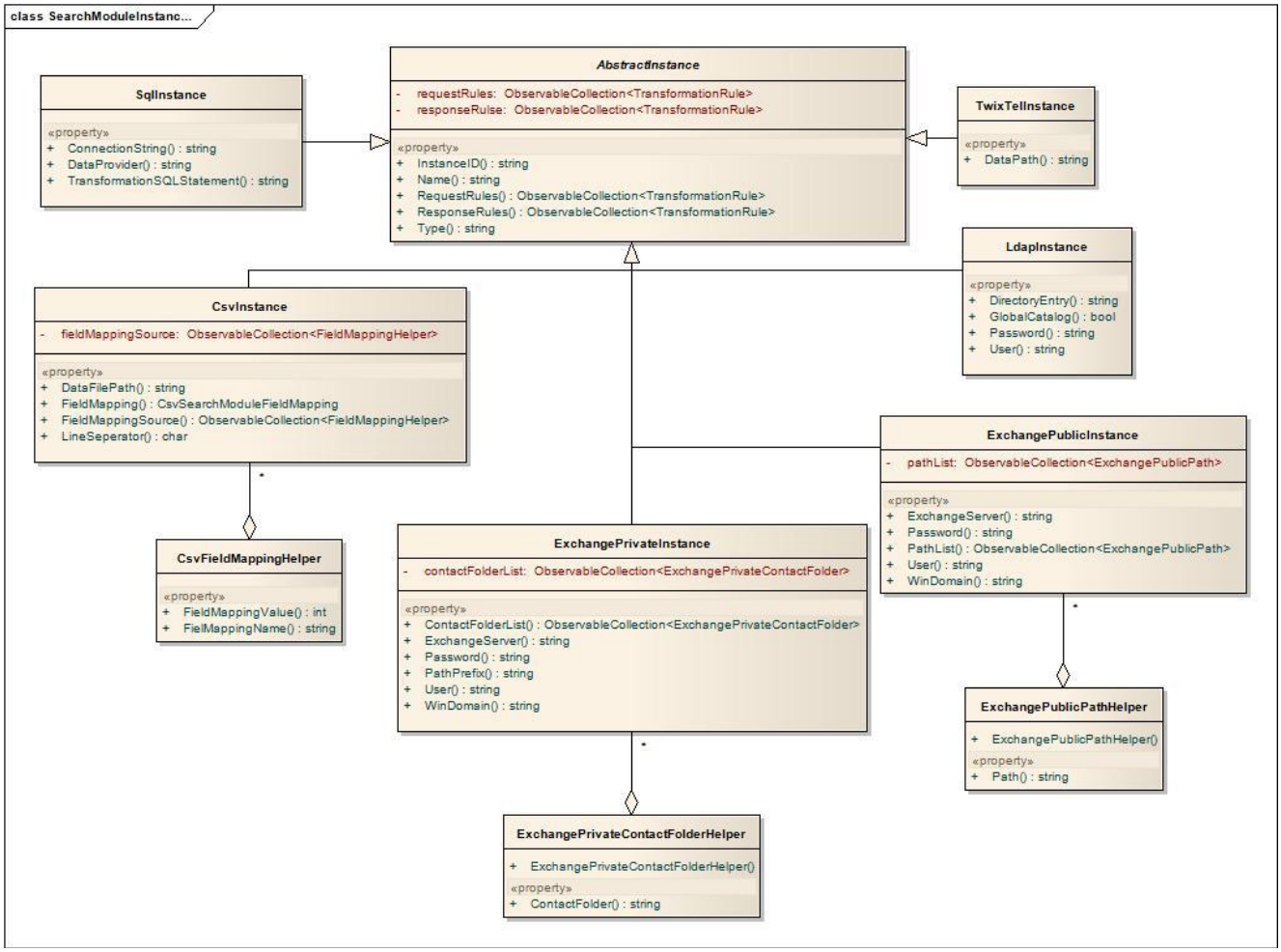


Abbildung 27: Klassendiagramm ViewModel SearchModule Instances

4.7.2.3. Klassenbeschreibung

Klasse	Beschreibung
ViewModelBase	Ist die abstrakte Superklasse aller ViewModel-Klassen. Sie enthält Properties, Commands und dazugehörige Logik- und Prädikatmethoden, allgemeine Konvertermethoden (für Listen und Observablecollections) und

	deklariert Methoden welche die Subklassen implementieren müssen.
ConfigurationChangeController	Ist als Singleton implementiert und beinhaltet public *Events* für das Laden und Speichern der Konfigurationen. Alle ViewModel Klassen, mit Ausnahme des *MainconfigViewModels*, registrieren sich an diesen beiden Events (im Konstruktor). Beim Auslösen des *LoadConfigCommand* oder dem *SaveConfigCommand*, wird der Event gefeuert und alle registrierten Klassen führen die registrierte Methode aus, was zum einen das Laden (*ReadFromModel()*) und zum anderen das Schreiben (*WriteToModel()*) nach sich zieht.
RelayCommand	Die *RelayCommand* Klasse implementiert das Interface *ICommand*. Dieses Interface verlangt die Implementation des Prädikates *CanExecute*, des EventHandlers *CanExecuteChanged* und der Methode *Execute*. Durch die *RelayCommand* Klasse lässt sich die Command Logik mittels delegate im Konstruktor einspeisen, so können die Commands aus dem ViewModel mittels lambda Expression elegant erzeugt werden.
MainconfigViewModel CsvViewModel ExchangePublicViewModel ExchangePrivateViewModel LdapViewModel SqlViewModel TwixtelViewModel	Die ViewModel-Klassen stellen die Daten für die jeweilige View zur Verfügung. Zudem werden Logikelemente welche nur von der eigenen View benötigt wird, in diesen konkreten Klassen implementiert. Die Methoden *ReadFromModel*, *WriteToModel* und *SaveConfigToFile* werden von allen VM Klassen implementiert, diese werden entweder durch einen Command oder durch das Feuern eines Events ausgelöst. Alle SearchModule ViewModels enthalten eine ObservableCollection *InstanceList* mit dem jeweiligen Instancetype. Diese *InstanceList* bildet das zentrale Glied der Datenstruktur eines ViewModels, es binhaltet alle relevanten Konfigurationsobjekte welche beim Schreibvorgang benötigt werden. Beispiele: - *CsvViewModel* enthält ObservableCollection<*CsvInstance*> - *LdapViewModel* enthält ObservableCollection<*LdapInstance*>

AbstractInstance	Ist die abstrakte Superklasse aller Instance Klassen und enthält die gemeinsamen Properties.
CsvInstance ExchangePublicInstance ExchangePrivateInstance LdapInstance SqlInstance TwixtelInstance	Die konkreten Instance Klassen entsprechen der Datenstruktur für das UI. Sie werden aus den Modeldaten, beim Einlesen (*ReadFromModel*() in den ViewModel-Klassen) in diese Struktur abgefüllt. Diese Objekte sind optimal auf die Views angepasst und können mittels UI verändert werden. Beim Zurückschreiben in die Model-Klasse Mainconfig (*WriteToModel*() und *SaveToConfigFile*() beide in den ViewModel-Klassen) werden die Daten wieder in der DF Datenstruktur gespeichert.

Tabelle 23: Klassenbeschreibung ViewModel

4.7.3. Package Model

4.7.3.1. Beschreibung des Packages

Im Model Layer befinden sich die Daten für die Konfigurationsdateien des Directory Framework. Sie werden entweder mittels des *ConfigurationReaders* eingelesen, oder es wird ein leeres Template geladen, um eine neue Konfiguration zu erstellen. Wenn die Neukonfiguration oder die Anpassung im GUI vorgenommen wurde, werden diese Daten zuerst ins Model und danach in die *.config Files geschrieben. Das Model setzt sich aus ein paar wenigen Klassen zusammen und speichert die benötigten Daten in Objekte welche auch im DF vorhanden sind, welche in der Klasse *Mainconfig* verankert sind.
DF Klassen aus dem Namespace *DFCommon.Utilities.Configuration*: *DirectoryServerConfiguration* und *SearchModuleConfiguration* und *SearchModuleInstanceConfiguration*. In der **Fehler! Verweisquelle konnte icht gefunden werden.***Abbildung 30: Object Graph Config Reader* wird dies visuell dargestellt.
In diesen Objekten kann man fast alle benötigten Informationen aus den Konfigurationsdateien speichern, welche dann im ViewModel gemappt werden um sie im GUI in den Views darzustellen. Eine Ausnahme bilden die Informationen aus dem Mainconfiguration File DFService.exe.config. Diese Informationen werden direkt in Properties der Klasse *Mainconfig* geschrieben und nicht in die DF Klassen. Diese verwenden eigene Reader, die nicht vom DF übernommen werden konnten. Das Lesen in die DF Klassen geschieht auch über Reader aus dem DF. Der *PathManager* stellt sicher, dass die Struktur der Ablage der Konfigurationsdateien stimmt.

4.7.3.2. Klassendiagramm

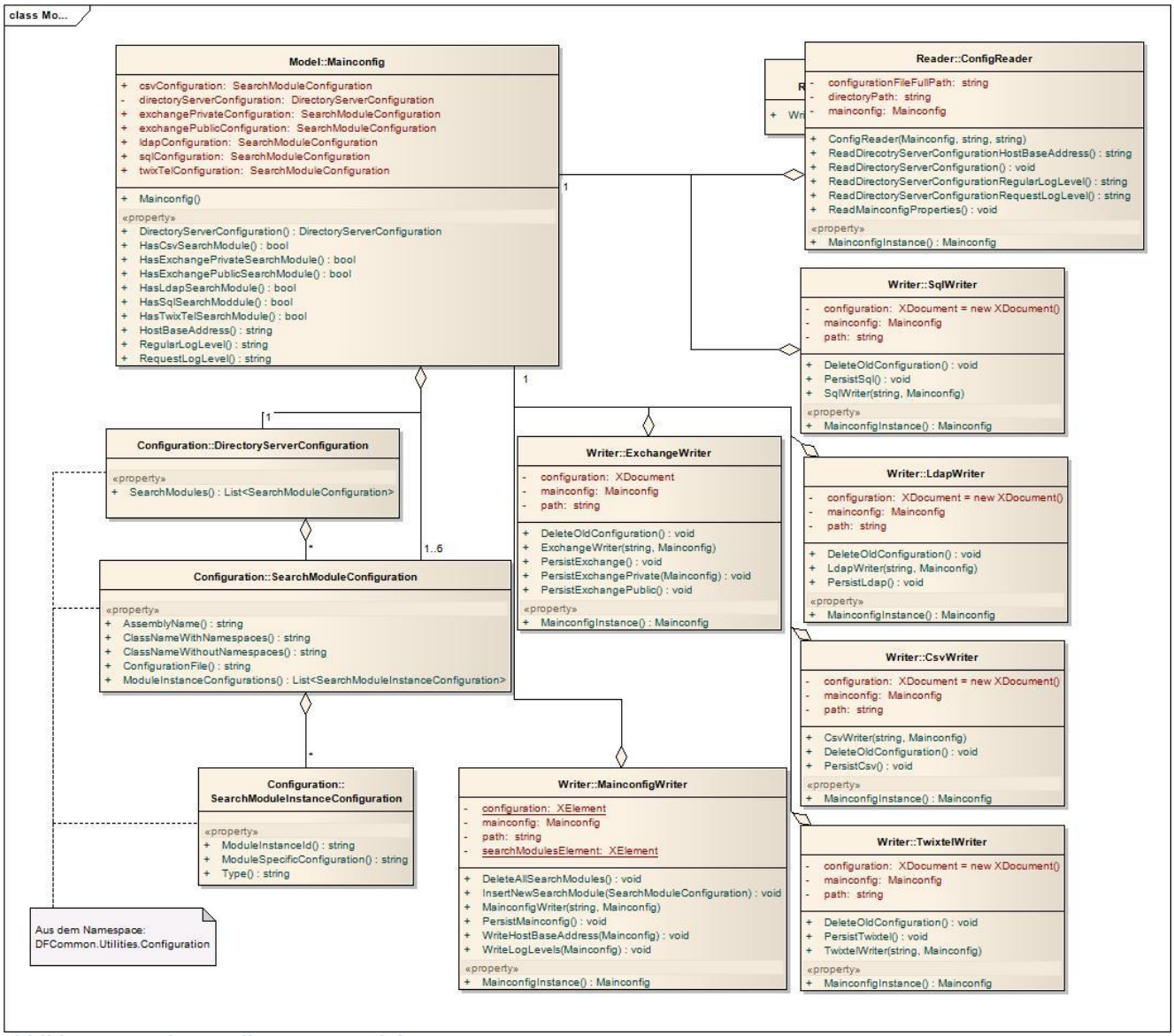


Abbildung 28: Klassendiagramm Model

4.7.3.3. Klassenbeschreibung

Klassen	Beschreibung
Mainconfig	Die *Mainconfig* Klasse ist der zentrale Mittelpunkt, sie enthält die Objekte *DirectoryServerConfiguration* und die Configurations der SearchModules. Diese Objekte stammen aus der DF ProblemDomain. (Namespace *DFCommon.ProblemDomain*) Die Properties der Mainconfiguration des DFs, die Hostadresse des Services und die LogLevels sind ebenfalls in dieser Klasse beheimatet. Zudem existiert pro SearchModuleType eine bool Property, welches definiert ob ein SearchModuleType konfiguriert wurde oder nicht.
ConfigReader	Der *ConfigReader* beinhaltet eine Readmethode für das Einlesen der SearchModuleConfiguration und je eine Readmethode, für die zusätzlich benötigten Properties aus der Mainconfiguration: *HostBaseAddress*, *RegularLogLevel* und *RequestLogLevel*
MainconfigWriter	Der *MainconfigWriter* löscht als erstes alle Childnodes des Elements „SearchModules“ und schreibt anhand der Selektion (bool Properties aus *Mainconfig*) die neuen Childnodes. Zudem gibt es je eine Writemethode für das *HostBaseAddress* Property und die LogLevel Properties.

CsvWriter ExchangeWriter LdapWriter SqlWriter TwixTelWriter	Die SearchModule Writer Klassen sind alle identisch aufgebaut. Sie enthalten eine Methode, welche die alte Konfiguration löscht und eine welche die neue Konfiguration schreibt. Durch die Verwendung der Klassen aus der DF Problemdomain ist die ganze Konfiguration eines SearchModules in einem String verpackt, welcher lediglich noch in ein File geschrieben werden muss. Wichtig ist, dass man diesen String noch in das Tag <configuration> packt, damit das Konfigurationfile wieder die gewünschte Struktur erhält.
ReaderException	Die *ReaderException* wird geworfen falls während des Readvorgang ein Fehler auftritt, durch diese Exception sollte dem Benutzer mitgeteilt werden, wo und weshalb der Fehler aufgetreten ist.
WriterException	Dieser Exceptiontype sollte geworfen werfen, wenn ein Fehler während des Writevorgang auftritt. Auch hier sollte dem Benutzer der Grund des Fehlschlagens mitgeteilt werden. Zum Beispiel falls das File schreibgeschützt ist in welches man schreiben möchte.

Tabelle 24: Klassenbeschreibung Model

4.7.3.4. SD ConfigReader

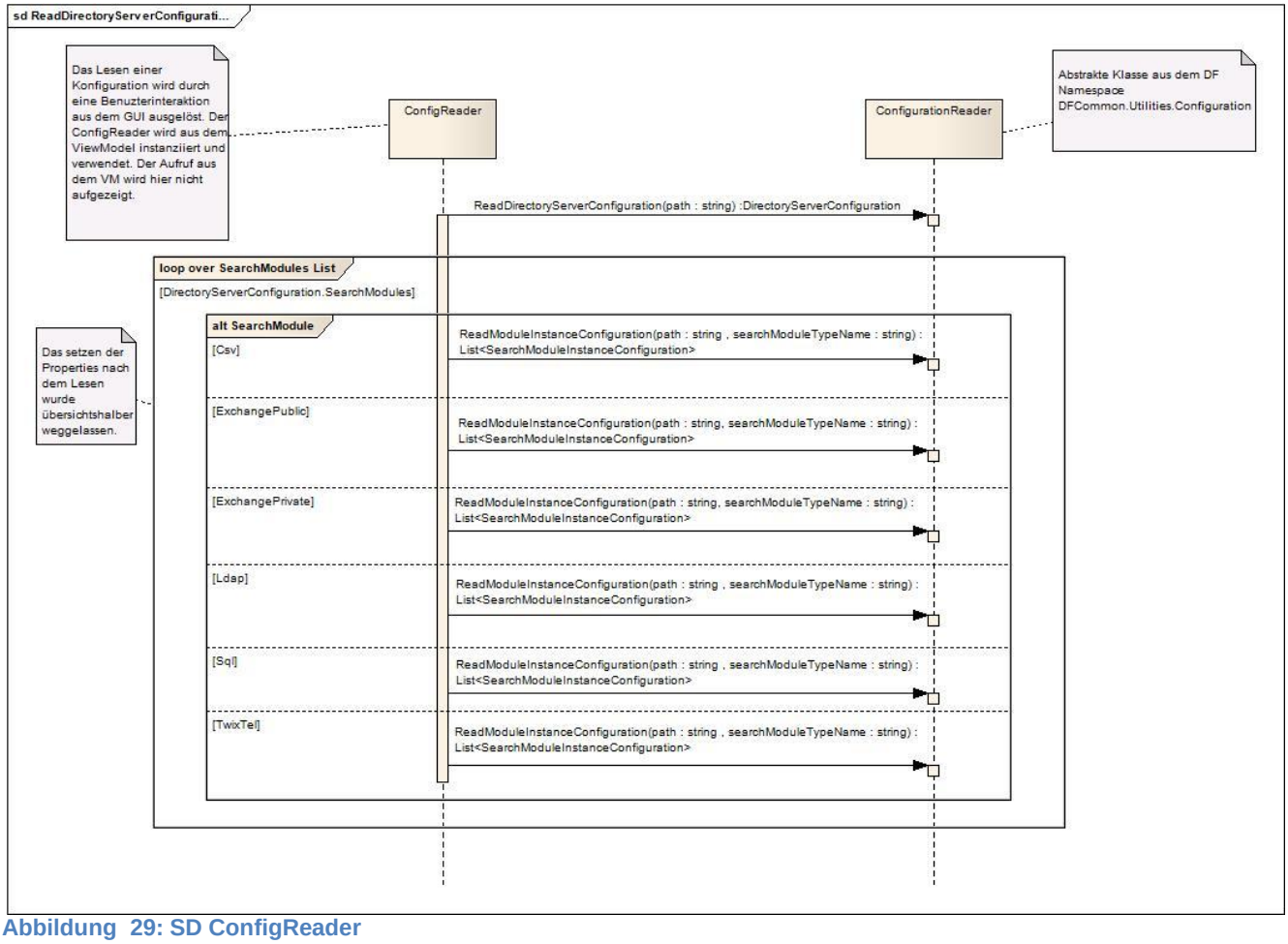


Abbildung 29: SD ConfigReader

Die Umsetzung des Einlesens der Konfiguration ist in der Klasse *ConfigReader* enthalten. Er wird anhand der Benutzerinteraktion in den Views, über das ViewModel instanziiert und verwendet. Der *ConfigReader* erhält durch den Konstruktor den Pfad der auf die Mainkonfigurationsdatei verweist und den Pfad des Verzeichnisses in dem sich diese Datei befindet.
Folgende Abbildung zeigt einen Ausschnitt aus dem DF, dieser sollte Veranschaulichen wie und wo die einzelnen eingelesenen Konfigurationen abgelegt werden.

4.7.3.5. Object Graph Config Reader

Die nachfolgende Abbildung zeigt einen Snapshot der Objekte nach der Ausführung der Operation *ReadDirectoryServerConfiguration*, welch das einlesen der Konfigurationsdateien ist.

In diesem Beispiel wurden zwei Csv Instanzen und je eine Instanz der anderen SearchModules eingelesen. Die Anzahl Instanzen ist jedoch abhäng der vorhanden Konfiguration und kann von System zu System unterschiedlich sein.

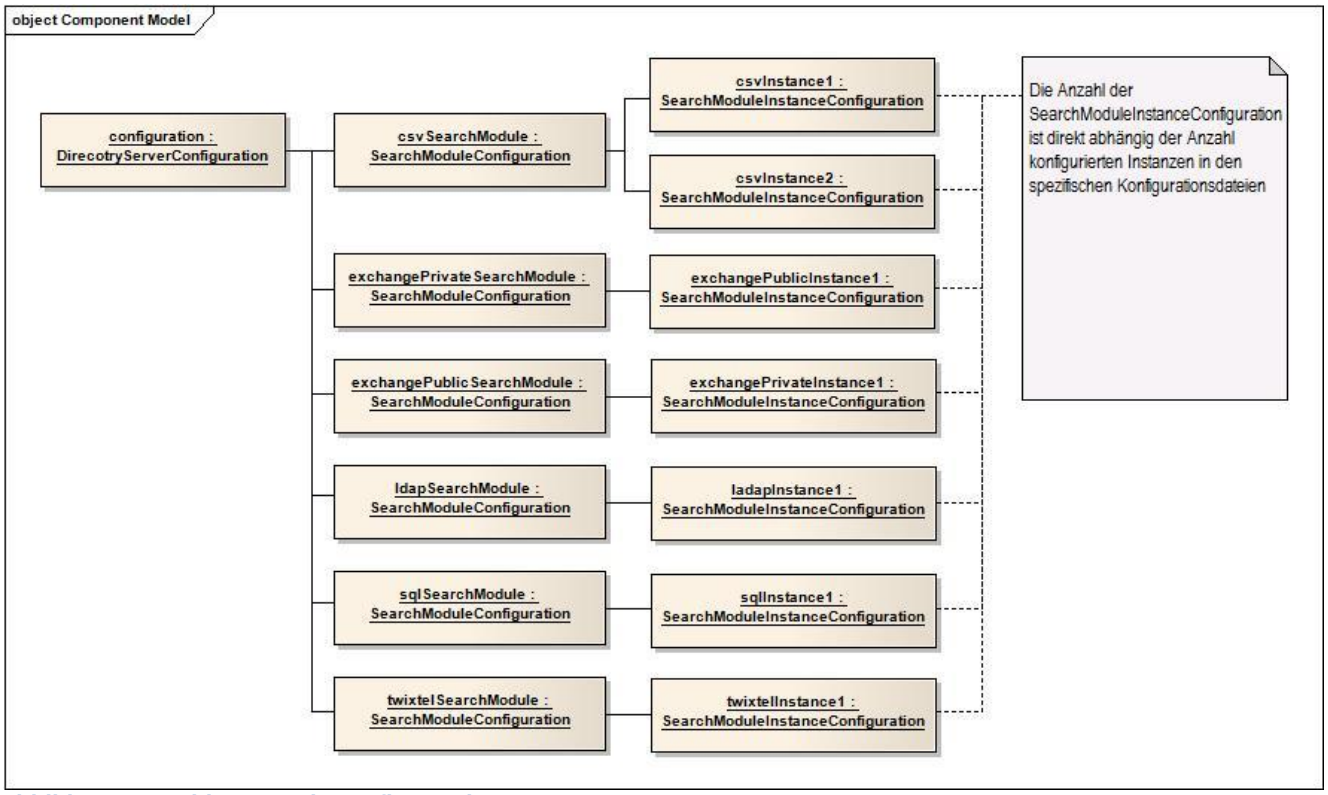


Abbildung 30: Object Graph Config Reader

In den *SearchModuleInstanceConfiguration* Objekten sind der Type, die *ModuleInstanceId* und die *ModuleSpecificConfiguration* als String Properties enthalten. Das *ModuleSpecificConfiguration* Property enthält eine komplette Abbildung einer SearchModule Instanz und ist bereits in der XML Struktur gespeichert, wie sie das DF erwartet.

4.7.3.6. Operationen

- **ReadDirectoryServerConfiguration( )**
  Die Methode *ReadDirectoryServerConfiguration* liest die SearchModule Konfiguration ein. Dazu verwendet diese die gleichnamige statische Methode aus dem DF aus dem Namespace *DFCommon.Utilities.Configuration*, diese liefert eine *DirectoryServerConfiguration* welche in der *Mainconfig* abgelegt wird. In dieser DirectoryServerConfiguration befindet sich eine Liste mit *SearchModuleConfiguration*, durch diese Liste kann identifiziert werden welche SearchModule eingelesen wurden. Anhand dieser Liste werden danach die SearchModule spezifischen konfigurationen einglesen, sofern ein SearchModule vorhanden ist. Dazu wird ebenfalls eine statische Methode aus dem DF verwendet, *ReadModuleInstancesConfiguration*. Diese liefert ein SearchModule abhängige InstanceConfiguration, welche in der *ModuleInstanceConfiguration* des jeweiligen SearchModules abgelegt wird. Diese InstanceConfiguration ist eine Liste von *SearchModuleInstanceConfiguration* Objekten. In den einzelnen ModuleInstanceConfiguration Objekten sind die Informationen einer Instanz des SearchModule gespeichert. Dies sind die Instance Id und der Typ sowie die spezifische Instanz Konfiguration als String, *ModuleSpecificConfiguration*.

- **ReadDirectoryServerConfigurationHostBaseAddress( )**
  Öffnet das Mainconfiguration File und selektiert mittels LinqToXml Query das gewünschten Element und liest den String in das Property.

- **ReadDirectoryServerConfigurationRegularLogLevel( )**
  Öffnet das Mainconfiguration File und selektiert mittlels LinqToXml Query das gewünschte Element und liest den String in das Property.

- **ReadDirectoryServerConfigurationRequestLogLevel( )**
  Öffnet das Mainconfiguration File und selektiert mittlels LinqToXml Query das gewünschte Element und liest den String in das Property.

4.8. Prozesse und Threads

Das Produkt dieses Teilprojekts der SA ist eine reine GUI Applikation welche auf der Windows Presentation Foundation aufbaut, somit wird das Prozess- und Threadmanagement von diesem übernommen. Grundsätzlich wird eine WPF Applikation mit zwei Threads gestartet, einen für das UI Management und einen der das Rendering übernimmt. Es wurden keine Nebenläufigkeiten implementiert und Datenzugriffe müssen nicht besonders abgesichert werden, da alles seriell abläuft.

4.9. Datenspeicherung

Die Daten in diesem Projekt werden in XML Files gespeichert und persistiert. Der Persistenz Layer wird momentan durch die sechs Konfigurationsdateien des Directory Framework dargestellt.

- DFService.exe.config
- Csv.dll.config
- Exchange.dll.config
- Ldap.dll.config
- Sql.dll.config
- Twixtel.dll.config

Diese sind näher beschrieben in der Konfigurationsanleitung des DFs.
…\09_Anhang\DF_Konfigurationsanleitung.docx
Die beiden Exchange SearchModules Public und Private werden in einem Konfigurationsfile abgespeichert und persistiert. Die Konfigurationsdateien können nicht in einer beliebigen XML Struktur abgelegt werden sondern müssen in einer genau definierten Form vorhanden sein, damit sie vom DF auch weiter verwendet werden können.
Durch die Erweiterung des Directory Frameworks durch einen weiteren SearchModuleType würde sich auch die Persistenzschicht um weitere Konfigurationsdateien erweitern. Die momentan vorhandenen Konfigurationsdateien sind im Dokument Konfigurationsanleitung der DF Dokumentation festgehalten.
Nach der Installation des DF sind die Konfigurationsdateien in folgendem Pfad abgelegt:
`„\\%Installationpath%\INS\DirectoryFramework\*.*.config“`

5. Implementierung

In diesem Kapitel werden spezielle Implementierungsdetails beschrieben, welche das Verständnis des DF GUI unterstützen. Es werden nur die wichtigsten Implementierungen beschrieben, welche spezielle Aspekte enthalten.

5.1. Helperklassen bei den SearchModuleInstance Klassen

5.1.1. CsvFieldMappingHelper

Hilfsklasse im Viewmodel im Namespace *DFGUI.ViewModel.SearchModuleInstances*. Wird verwendet um die Properties der DF Klasse *CsvSearchModuleFieldMapping* in einer ObservableCollection abzubilden. Die *CsvFieldMappingHelper* Klasse beinhaltet ein string Property für den Namen der FieldMapping Properties und ein int Property für den Value derselben. Eine Objekt der *CsvInstance* Klasse stellt eine ObservableCollection<*CsvFieldMappingHelper*> zur Verfügung, in der die 44 FieldMapping Properties enthalten sind, und binded diese an das Control *ListView* aus dem *CsvTab*. Durch das Benutzen dieser Hilfsklasse werden alle Werte die im GUI verändert werden wieder in die ObservableCollection zurück geschrieben mittels dem WPF DataBinding. Die 44 Properties werden per Reflection in die ObservableCollection abgefüllt.

5.1.2. ExchangePublicPathHelper

Hilfsklasse im ViewModel im Namespace *DFGUI.ViewModel.SearchModuleInstances*. Diese Hilfsklasse wird benötigt weil die Verwendung einer ObservableCollection<*string*> keine direkte Veränderungen aktualisiert. Die Verwendung einer ObservableCollection<*ExchangePublicPathHelper*> löst dieses Problem und die getätigten Änderungen, die im GUI vorgenommen werden, werden stets in der Collection aktualisiert. Die Klasse *ExchangePublicPathHelper* beinhaltet ein string Property *Path*.

5.1.3. ExchangePrivateContactFolderHelper

Selbe Verwendung und Implementation wie beim vorangegangenen Abschnitt *5.1.2 ExchangePublicPathHelper.*

5.2. Pathmanager

Der *Pathmanager* befindet sich im Namespace *DFGUI.ViewModel.Utilities* und wird verwendet um die Pfadangaben zu separieren. Wenn eine Datei mit dem Win32 Dialog selektiert wird, erhält man den gesamten Pfad inklusive dem Filenamen. Damit man diesen Pfad weiter verwenden kann, zum Beispiel damit die spezifischen Konfigurationsdateien im selben Pfad gespeichert werden, muss dieser Pfad aufgetrennt werden, in Pfad und Datei. Wenn die Konvention der Speicherorte geändert wird, kann dies im Pfadmanager angepasst werden.

5.3. Constants Klassen

In den *Constants* Klassen sind alle konstanten Felder definiert welche benutzt werden. Alle diese Felder befinden sich in der *Constants* Klasse im Namespace *DFGUI.ViewModel.Utilities* und werden verwendet um default Werte der Konfiguration zu definieren.

6. Ausblick

Durch diverse Probleme, welche im Verlauf der Arbeit immer wieder auftauchten, mussten zum Teil gewünschte Features weggelassen werden, oder Kompromisslösungen verwendet werden. Grundsätzlich betrifft dies die Punkte, Erweiterbarkeit des GUIs mit weiteren SearchModules, das Schreiben der Konfigurationsdateien und die Vereinfachung und Aufspaltung von gewissen Einstellungskomponenten im GUI.

6.1. Verwendung von Templates

Durch die Verwendung von Templates kann die Erweiterbarkeit um einiges vereinfacht werden. Erweiterbarkeit im Sinne von hinzufügen neuer Tabs für neue SearchModules. Man könnte das Grundgerüst der Tabs zur Verfügung stellen und müsste dann nur noch die spezifischen Teile des neuen SearchModule einfügen. Wünschenswert in diesem Template wären alle Komponenten, welche in allen Tabs kongruent sind. Dies wären auf jeden Fall die Advanced Settings sowie die Instanz Liste, dazu kommen könnten auch noch die Elemente *InstanceId* und *InstanceName* von der Instanz Konfiguration.
Die drei genannten GUI Komponenten sind im Abschnitt *3.3 Tabs (User Controls)* genauer beschrieben und anhand einer Abbildung visuell dargestellt. *Abbildung 10: Csv Tab*

6.2. Write Funktionalität

Die Writer Klassen der SearchModules sind sehr einfach aufgebaut, denn die SearchModule Konfiguration liegt bereits vollumfänglich im richtigen Format in einem String pro Instanz vor. Es müssen somit nur noch die einzelnen Strings nacheinander in das File geschrieben werden. Die Writer Klassen setzen sich zusammen aus einem *FileStream* und einem *StreamWriter*. Diese beiden Objekte führen den Schreibvorgang aus, welcher zuerst ein XML Node *configuration* erstellt und danach die einzelnen spezifischen Konfigurationen hinzufügt, abgeschlossen wird der Vorgang durch das Anfügen des EndTag des Nodes *configuration*.

6.3. Vereinfachung von Konfigurationskomponenten

In der Datenstruktur des DFs sind Properties vorhanden, welche für die Konfiguration im GUI noch weiter aufgespalten werden könnten um die Konfiguration noch zu vereinfachen. Die Trennung und Zusammensetzung dieser Properties, welche für die Abbildung in die Konfigurationsdateien wieder notwendig ist, würde im jeweiligen ViewModel implementiert werden.

Beispiele für diese Vereinfachung von Konfigurationskomponenten wären:

ViewModel	Property	Type	Value
LDAPViewModel	DirectoryEntry	String	`LDAP://ad.ocs.lab.local/DC=ocs,DC=lab, DC=local`
SQLViewModel	ConnectionString	String	`Data Source=sql.ocs.lab.local\OCSR2;Initial Catalog=DirectoryFramework;User Id=DirectoryFramework;Password=ins@lab;`

Tabelle 25: Beispiele Vereinfachung von Konfigurationskomponenten

6.4. Pfadmanagement und Verwendung von Template Konfigurationsdateien

Nach der Selektion eines Pfades wird dieser im *ConfigurationChangeController* gespeichert. Es wird kein Unterschied gemacht ob die Selektion durch Laden oder Speichern einer Konfigurationsdatei erfolgt ist. Dieser Pfad wird für die weitere Verwendung des Konfigurationsvorgangs behalten. Für den Normalfall, dass nach der Installation des DFs und des GUIs, die Konfiguration vorgenommen wird und die Konfigurationsdateien aus dem Installationsverzeichnis verwendet, werden ist dies kein Problem. Wenn jedoch eine alternative Konfiguration erstellt werden soll, können Probleme entstehen. Zum einen wäre da das Problem, dass bei einer Speicherung an einem neuen Ort, ein Template Konfigurationsfile geladen wird, welches mit ziemlich grosser Wahrscheinlichkeit den falschen Registrierungsschlüssel enthält. Wenn der Registrierungsschlüssel nicht korrekt ist, hat dies zur Folge, dass der DF Service INS-DirectoryFramework nicht mehr korrekt startet. Wenn ein solches Problem auftritt, wird dies im LogFile RegularLog im Installationsverzeichnis im Ordner Log festgehalten. Die Problemlösung wäre, dass das Template File nach der Installation und der Eingabe des Registrierungsschlüssels, das File DFService.exe.config aus dem Installationsverzeichnis ist.

7. Literaturverzeichnis

Dokumentationsliteratur
Craig Larman, UML 2 und Patterns angewendet – Objektorientierte Softwareentwicklung, 1. Auflage 2005
http://de.wikipedia.org
http://msdn.microsoft.com/en-us/magazine/cc188690.aspx
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

Entwicklungsliteratur
Jesse Liberty und Donald Xie, Programmieren mit C# 3.0, 3. Auflage 2008
Thomas Claudius Huber, Windows Presentation Foundation Das umfassende Handbuch, 2008
http://msdn.microsoft.com/en-us/magazine/cc188690.aspx
http://msdn.microsoft.com
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
http://social.msdn.microsoft.com/forums
https://wiki.ins/index.php/.NET_Programming
Diverse Foren

8. Abbildungsverzeichnis

Abbildung 1: Use Case Model 8
Abbildung 2: Strukturdiagramm 14
Abbildung 3: SSD UC 1 Directory Framework neu konfigurieren 15
Abbildung 4: SSD UC 2 Directory Framework Konfiguration erweitern 16
Abbildung 5: SSD UC 3 Directory Framework Konfiguration ändern 17
Abbildung 6: SSD UC 4 Directory Framework Konfiguration verringern 18
Abbildung 7: Activity Diagram 21
Abbildung 8: GUI Navigation Map 22
Abbildung 9: MainWindow / Main config Tab 23
Abbildung 10: Csv Tab 25
Abbildung 11: Exchange Public Tab 26
Abbildung 12: Exchange Private Tab 27
Abbildung 13: Ldap Tab 28
Abbildung 14: Sql Tab 29
Abbildung 15: Twixtel Tab 30
Abbildung 16: Umgebungsdiagramm 31
Abbildung 17: Schichtenmodell 37
Abbildung 18: SD RelayCommand 38
Abbildung 19: SD ConfigurationChangeController register 39
Abbildung 20: SD ConfigurationChangeController fire 40
Abbildung 21: Packagestruktur 41
Abbildung 22: Solutionübersicht 41
Abbildung 23: ViewModelübersicht 42
Abbildung 24: Modelübersicht 42
Abbildung 25: Klassendiagramm View XAML Files 43
Abbildung 26: Klassendiagramm View XAML.CS (Codebehind) 44
Abbildung 27: Klassendiagramm ViewModel SearchModule Instances 45
Abbildung 28: Klassendiagramm Model 47
Abbildung 29: SD ConfigReader 48
Abbildung 30: Object Graph Config Reader 49

9. Tabellenverzeichnis

Tabelle 1: UC 1 Directory Framework neu konfigurieren 9
Tabelle 2: UC2 Directory Framework Konfiguration erweitern 10
Tabelle 3: UC 3 Directory Framework Konfiguration ändern 10
Tabelle 4: UC 4 Directory Framework Konfiguration verringern 11
Tabelle 5: SUC 1 Mainconfig erstellen 11
Tabelle 6: SUC 2 SearchModule Instance erstellen / hinzufügen 12
Tabelle 7: SUC 3 Mainconfig ändern / erweitern / verringern 12
Tabelle 8: SUC 4 SearchModule Instance ändern 13
Tabelle 9: SUC 5 SearchModule Instance löschen / entfernen 13
Tabelle 10: Konzeptbeschreibung 15
Tabelle 11: CO 1 configureMainconfig 18
Tabelle 12: CO2 addSearchModuleType 19
Tabelle 13: CO3 configureSearchModuleType 19
Tabelle 14: CO 4 addInstances 19
Tabelle 15: CO 5 configureInstances 19
Tabelle 16: CO 6 saveConfiguration 20
Tabelle 17: CO 7 loadConfiguration 20
Tabelle 18: CO 8 saveConfiguration 20
Tabelle 19: CO 9 removeSearchModuleType 20
Tabelle 20: CO 10 removeInstance 21
Tabelle 21: Tab Repräsentation der Konfigurationsdateien 22
Tabelle 22: Klassenbeschreibung View 45
Tabelle 23: Klassenbeschreibung ViewModel 46
Tabelle 24: Klassenbeschreibung Model 48
Tabelle 25: Beispiele Vereinfachung von Konfigurationskomponenten 51

Directory Framework GUI

Testdokumentation

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
01.12.2009	1.0	erstellt	Müller	
	1.0	Systemtest 1	Müller	
11.12.2009	1.1	Systemtest 2	Müller	
14.12.2009	1.1	UnitTests	Müller	
16.12.2009	1.2	Systemtest 3	Müller	
	1.3	Reader / Writer Tests	Müller	
17.12.2009	1.4	Directory Server Restart Test	Müller	Dietrich
17.12.2009	2.0	Final Version	Müller	

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. Systemtest 4
1.1. Voraussetzungen 4
1.2. Vorbereitungen 4
1.3. Systemtest 1 Sprint 6 4
1.3.1. Systemtest UC 1: Directory Framework neu konfigurieren 4
1.3.2. Systemtest UC 2: Directory Framework Konfiguration erweitern 4
1.3.3. Systemtest UC 3: Directory Framework Konfiguration ändern 4
1.3.4. Directory Framework Konfiguration verringern 4
1.3.5. Systemtest Zusammenfassung 4
1.4. Systemtest 2 Sprint 7 5
1.4.1. Systemtest UC 1: Directory Framework neu konfigurieren 5
1.4.2. Systemtest UC 2: Directory Framework Konfiguration erweitern 5
1.4.3. Systemtest UC 3: Directory Framework Konfiguration ändern 5
1.4.4. Directory Framework Konfiguration verringern 5
1.4.5. Systemtest Zusammenfassung 6
1.5. Systemtest 3 Sprint 7 6
1.5.1. Systemtest UC 1: Directory Framework neu konfigurieren 6
1.5.2. Systemtest UC 2: Directory Framework Konfiguration erweitern 6
1.5.3. Systemtest UC 3: Directory Framework Konfiguration ändern 6
1.5.4. Directory Framework Konfiguration verringern 6
1.5.5. Systemtest Zusammenfassung 7
2. Reader / Writer Tests 7
3. Directory Framework restart Test 8
4. UnitTests 9
4.1. ConfigurationReaderTest 9
4.2. ViewModelTest 10
4.2.1. CSVViewModelTest 10
4.2.2. ExchangePublicViewModelTest 11
4.2.3. ExchangePrivateViewModelTest 11
4.2.4. LDAPViewModelTest 11
4.2.5. SQLViewModelTest 12
4.2.6. TwixTelViewModel 12
5. Tabellenverzeichnis 12

1. Systemtest

1.1. Voraussetzungen

Die nachfolgenden Systemtests wurden auf Rechnern mit Microsoft Windows getestet. Da das DF GUI mit C# und WPF entwickelt wurde, war die Plattformunabhängigkeit auch nie ein Kriterium.

1.2. Vorbereitungen

Um die Tests auszuführen wurden verschiedene Konfigurationsdateien generiert um verschiedene Testfälle abzudecken. Aufgedeckte Fehler werden in der Bug List festgehalten. *…\06_Testdokumentationen\Bug List.xlsx*

1.3. Systemtest 1 Sprint 6

Der nachfolgende Test wurde auf der bestehenden und noch nicht fertig entwickelten Version des DF GUIs gemacht, die Test sollten die grundlegenden Funktionalitäten der Applikation testen und allfällige Probleme aufzeigen. Der Test baut auf den Use Cases auf.

1.3.1. Systemtest UC 1: Directory Framework neu konfigurieren

- Das Hinzufügen von neuen SearchModulen funktioniert einwandfrei und die zusätzlichen Tabs werden generiert und angezeigt.
- Die Mainconfig Einstellungen HostbaseAddress, Regular- und RequestLogLevel können vorgenommen werden.
- Das Hinzufügen von neuen Instanzen in den SearchModule spezifischen Konfigurationen sowie die detaillierte Instance Konfiguration funktionieren.
- Das Speichern ist noch nicht möglich, da die Write Funktion noch nicht implementiert ist.

1.3.2. Systemtest UC 2: Directory Framework Konfiguration erweitern

- Das Einlesen einer bestehenden Konfiguration funktioniert, wirft allerdings eine Exception wenn das File korrupt ist.
- Das Hinzufügen von neuen SearchModules und deren neuen Instanzen funktioniert.
- Das Speichern ist noch nicht möglich, da die Write Funktion noch nicht implementiert ist.

1.3.3. Systemtest UC 3: Directory Framework Konfiguration ändern

- Das Einlesen einer bestehenden Konfiguration funktioniert, wirft allerdings eine Exception wenn das File korrupt ist.
- Das Ändern der vorhandenen Konfiguration funktioniert, gespeichert werden kann die Konfiguration noch nicht, da die Write Funktion noch nicht implementiert ist und somit kann dieser Test noch nicht als gelungen betrachtet werden.

1.3.4. Directory Framework Konfiguration verringern

- Das Einlesen einer bestehenden Konfiguration funktioniert, wirft bei korrupten Files allerdings eine Exception.
- Das Löschen von einzelnen Instanzen, sowie das Entfernen eines SearchModules funktioniert.
- Das Speichern ist allerdings noch nicht möglich.

1.3.5. Systemtest Zusammenfassung

Use Case	Implementation erfolgt	Fehler / Unschönheiten	Status
UC 1: DF neu konfigurieren	teilweise	Die Write Funktion ist noch nicht implementiert.	in Bearbeitung
UC 2: DF Konfiguration erweitern	teilweise	Wirft Exception welche noch nicht behandelt wird. Die Write Funktion ist noch nicht implementiert.	in Bearbeitung
UC 3: DF Konfiguration ändern	teilweise		in Bearbeitung
UC 4: DF Konfiguration verringern	teilweise		in Bearbeitung

Tabelle 1: Systemtest 1 Zusammenfassung

1.4. Systemtest 2 Sprint 7

Der nachfolgende Test wurde auf der bestehenden Version des DF GUIs gemacht, welche die Grundfunktionalitäten abdecken sollte. Der Test sollte die grundlegenden Funktionalitäten der Applikation testen und allfällige Probleme aufzeigen. Der Test baut auf den Use Cases auf.

1.4.1. Systemtest UC 1: Directory Framework neu konfigurieren

- Das Hinzufügen von neuen SearchModulen funktioniert einwandfrei und die zusätzlichen Tabs werden generiert und angezeigt.
- Die Mainconfig Einstellungen *HostbaseAddress*, *Regular*- und *RequestLogLevel* können vorgenommen werden.
- Das Hinzufügen von neuen Instanzen in den SearchModule spezifischen Konfigurationen sowie die detaillierte Instance Konfiguration funktionieren.
  Allerdings gibt es noch Probleme bei den SerchModules Csv, ExchangePublic, ExchangePrivate. Alle drei SearchModules enthalten eine *ObservableCollection* welche vorgenommene Einstellungen nicht übernimmt und speichert. Zu erwähnen ist jedoch, dass in allen SearchModules schon solche Collections in Verwendung sind und funktionieren.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Jedoch wird eine Exception geworfen, wenn die Datei welche man ändern möchte schreibgeschützt ist. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.4.2. Systemtest UC 2: Directory Framework Konfiguration erweitern

- Das Einlesen einer bestehenden Konfiguration funktioniert. Die Exception wird gefangen und dem Benutzer wird mitgeteilt wo und warum der Fehler aufgetreten ist.
- Das Hinzufügen von neuen SearchModules und deren neuen Instanzen funktioniert.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Jedoch wird eine Exception geworfen, wenn die Datei, welche man ändern möchte schreibgeschützt ist. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.4.3. Systemtest UC 3: Directory Framework Konfiguration ändern

- Das Einlesen einer bestehenden Konfiguration funktioniert. Die Exception wird gefangen und dem Benutzer wird mitgeteilt wo und warum der Fehler aufgetreten ist.
- Das Ändern der vorhandenen Konfiguration funktioniert, ausser in drei Fällen, im Csv SearchModule werden Änderungen welche im *Fieldmapping* gemacht wurden nicht übernommen von der *ObservableCollection* im ViewModel. Dasselbe Problem tritt in den SearchModules ExchangePublic und ExchangePrivate auf, bei der Liste der *Paths* und der Liste der *ContactFolders*. Es sind allerdings in allen ViewModels ObservableCollections im Einsatz, diese funktionieren einwandfrei.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Jedoch wird eine Exception geworfen, wenn die Datei, welche man ändern möchte schreibgeschützt ist. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.4.4. Directory Framework Konfiguration verringern

- Das Einlesen einer bestehenden Konfiguration funktioniert. Die Exception wird gefangen und dem Benutzer wird mitgeteilt, wo und warum der Fehler aufgetreten ist.
- Das Löschen von einzelnen Instanzen, sowie das Entfernen eines SearchModules funktioniert.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Exception wird geworfen wenn die Datei, welche man ändern möchte schreibgeschützt ist. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.4.5. Systemtest Zusammenfassung

Use Case	Implementation erfolgt	Fehler / Unschönheiten	Status
UC 1: DF neu konfigurieren	Ja	Unbehandelte Exception bei der Write Funktion, Updatemechanismus in drei ObservableCollection funktioniert nicht.	in Bearbeitung
UC 2: DF Konfiguration erweitern	Ja		in Bearbeitung
UC 3: DF Konfiguration ändern	Ja		in Bearbeitung
UC 4: DF Konfiguration verringern	Ja	Unbehandelte Exception bei der Write Funktion.	in Bearbeitung

Tabelle 2: Systemtest 2 Zusammenfassung

1.5. Systemtest 3 Sprint 7

Der nachfolgende Test wurde auf der bestehenden Version des DF GUIs gemacht, welche am Feature stopp angelangt ist. Der Test sollte die grundlegenden Funktionalitäten der Applikation testen und allfällige Probleme aufzeigen. Der Test baut auf den Use Cases auf.

1.5.1. Systemtest UC 1: Directory Framework neu konfigurieren

- Das Hinzufügen von neuen SearchModulen funktioniert einwandfrei und die zusätzlichen Tabs werden generiert und angezeigt.
- Die Mainconfig Einstellungen *HostbaseAddress*, *Regular*- und *RequestLogLevel* können vorgenommen werden.
- Das Hinzufügen von neuen Instanzen in den SearchModule spezifischen Konfigurationen sowie die detaillierte Instance Konfiguration funktionieren. Die Probleme mit den *ObservableCollections*, welche die Änderungen nicht übernahmen und nicht speicherten, wurden mittels Helperklassen behoben.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Die geworfene Exception die im letzten Test erwähnt wurde, wird nun behandelt und der User wird informiert. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.5.2. Systemtest UC 2: Directory Framework Konfiguration erweitern

- Das Einlesen einer bestehenden Konfiguration funktioniert. Die Exception wird gefangen und dem Benutzer wird mitgeteilt wo und warum der Fehler aufgetreten ist.
- Das Hinzufügen von neuen SearchModules und deren neuen Instanzen funktioniert.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Die beim letzten Test festgestellte Exception welche beim Schreibprozess geworfen wird, ist nun abgefangen und wird behandelt. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.5.3. Systemtest UC 3: Directory Framework Konfiguration ändern

- Das Einlesen einer bestehenden Konfiguration funktioniert. Die Exception wird gefangen und dem Benutzer wird mitgeteilt wo und warum der Fehler aufgetreten ist.
- Das Ändern der vorhandenen Konfiguration funktioniert, alle Konfigurationselemente können geändert werden und die Informationen werden übernommen.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben inklusive Exception handling. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert.

1.5.4. Directory Framework Konfiguration verringern

- Das Einlesen einer bestehenden Konfiguration funktioniert. Die Exception wird gefangen und dem Benutzer wird mitgeteilt wo und warum der Fehler aufgetreten ist.
- Das Löschen von einzelnen Instanzen, sowie das Entfernen eines SearchModules funktioniert.
- Das Speichern funktioniert und die Konfigurationsdateien werden neu geschrieben. Wenn ein File nicht vorhanden ist, in das geschrieben werden möchte, wird eines generiert und die Konfiguration darin gespeichert. Die Writer Exception, welche durch den Zugriff auf ein schreibgeschütztes File verursacht wurde, wird nun behandelt.

1.5.5. Systemtest Zusammenfassung

Use Case	Implementation erfolgt	Fehler / Unschönheiten	Status
UC 1: DF neu konfigurieren	Ja	Unschönheiten sind noch vorhanden, die Grundfunktionalitäten sind jedoch verfügbar.	Erledigt
UC 2: DF Konfiguration erweitern	Ja		Erledigt
UC 3: DF Konfiguration ändern	Ja		Erledigt
UC 4: DF Konfiguration verringern	Ja		Erledigt

Tabelle 3: Systemtest 3 Zusammenfassung

2. Reader / Writer Tests

Zusätzlich zu den Systemtests wurden noch erweiterte Test ausgeführt um bestimmte Komponenten zu testen, so zum Beispiel für den Reader und den Writer. Folgende Testfälle wurden behandelt.

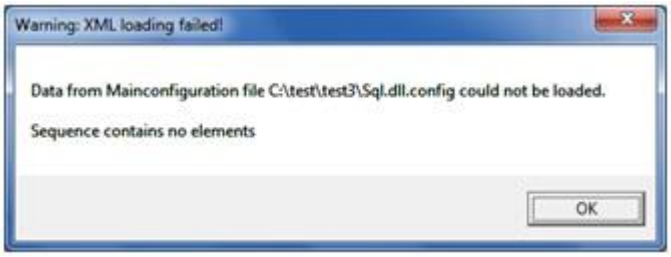
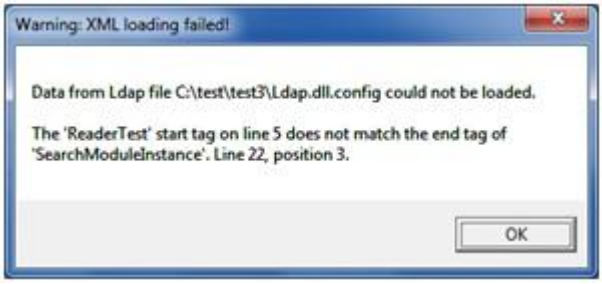
Reader Test	Resultat	Bemerkung
Einlesen einer korrekten Mainconfig Datei.	OK	Konfiguration wird eingelesen, Daten werden dargestellt und die zusätzlichen SearchModule Tabs wurden generiert und eingelesen.
Einlesen der Template Files aus dem DF.	OK	Die gesamte Konfiguration wurde eingelesen und richtig dargestellt inklusive aller Instanzen.
Einlesen einer Mainconfig Datei, in der das SearchModule Ldap konfiguriert ist, jedoch ist diese Konfigurationsdatei nicht vorhanden im Dateisystem.	OK	Die gesamte Konfiguration wurde eingelesen inklusive dem Ldap SearchModule, dieses ist allerdings leer, was in diesem Fall ok ist.
Anstelle eines Mainconfig Files wird eine SearchModule spezifische Konfigurationsdatei geladen.	OK	
In der Ldap Konfigurationsdatei wurde ein zusätzliches Tag <Reader Test> eingefügt in Zeile Nummer 5.	OK	

Tabelle 4: Reader Tests

Writer Test	Resultat	Bemerkung
Speichern einer kompletten Konfiguration, welche geladen wurde.	OK	Die Konfiguration wurde erfolgreich in alle Dateien geschrieben.
Speichern einer kompletten Konfiguration, welche neu erstellt wurde.	OK Warning	Das Speichern an einen neuen Ort und das Erzeugen der Dateien funktioniert. Jedoch muss beachtet werden, dass für das Mainconfig File ein Template File zur Erzeugung genutzt wurde und somit mit grosser Wahrscheinlichkeit der Registrierungsschlüssel falsch ist. Dies führt dazu, dass der Restart des DFs fehlschlägt. Der Benutzer wird über dieses Problem informiert.
Speichern einer kompletten Konfiguration, welche geladen wurde. Die spezifischen Konfigurationen wurden jedoch mit einem neuen Dateinamen versehen und mussten somit neu generiert werden.	OK	Die Konfiguration wurde erfolgreich in alle Dateien geschrieben. Die spezifischen Konfigurationsdateien wurden erstellt und die Konfiguration wurde darin gespeichert.

Tabelle 5: Writer Tests

3. Directory Framework restart Test

Nach der vorgenommenen Konfiguration des Directory Frameworks, kann dieses neu gestartet werden damit die Konfiguration neu geladen wird. Der Neustart des Directory Framework wird geloggt im RegularLog. Die Logdatei befindet sich im Installationsverzeichnis des Directory Framework im Unterordner Log.

Service Restart Test	Resultat	Bemerkung
Test 1: Nach getätigter Konfiguration wird der Neustart des Directory Frameworks erzwungen.	Fehlgeschlagen	Der Service wird beendet, das Starten funktioniert allerdings nicht. Durch einen asynchronen Callback wird eine Exception geworfen. Comment: Object reference not set to an instance of an object. Source: DFCommon StackTrace: at DFCommon.Application.CommunicationServer.EndShutdown(IAsyncResult result)\r\n at System.ServiceModel.AsyncResult.Complete(Boolean completedSynchronously) Das Regular Log des DFs loggt den Eintrag: *INFO DFService.Service.Service [(null)] - DFService: Started successfully.*
Test 2: Nach getätigter Konfiguration wird der Neustart des Directory Frameworks erzwungen.	OK	Der Service wird neu gestartet. Problem sollte behoben sein, es konnte mittels dem Test aus dem DF GUI nicht mehr reproduziert werden.

Tabelle 6: Restart Test

4. UnitTests

4.1. ConfigurationReaderTest

Die nachfolgenden Tests prüfen den ConfigurationReader, der sich im Model Layer befindet. Um das Einlesen der Readmethoden zu testen, wurde in diesem Projekt ein Ordner angelegt mit den zu testenden Konfigurationsdateien.
Die Tests sind beschränkt und testen nicht den gesamten Umfang der Readmethoden ab. Zum Beispiel wird der String der spezifischen Instanzkonfiguration nicht überprüft, in diesem befindet sich der gesamte Inhalt der XML Datei.

ReadCsvConfigurationTest

Prüft folgende Properties aus dem MainconfigFile „DFService.exe.config“:
- AssemblyName
- ClassName
- ConfigurationFile

Prüft die Anzahl der spezifischen Csv Konfigurationsinstanzen.

ReadDirectoryServerConfigurationHostBaseAddressTest

Prüft das Property HostBaseAddress aus dem MainconfigFile.

ReadDirectoryServerConfigurationRegularLogLevelTest

Prüft das Property RegularLogLevel aus dem MainconfigFile.

ReadDirectoryServerConfigurationRequestLogLevelTest

Prüft das Property RequestLogLevel aus dem MainconfigFile.

ReadDirecotryServerConfigurationTest

Prüft die bool Properties, ob ein SearchModule vorhanden ist oder nicht.
- HasCsvSearchModule
- HasExchangePublicSearchModule
- HasExchangePrivateSearchModule
- HasLdapSearchModule
- HasSqlSearchModule
- HasTwixtelSearchModule

ReadLdapConfigurationTest

Prüft folgende Properties aus dem MainconfigFile „DFService.exe.config“:
- AssemblyName
- ClassName
- ConfigurationFile

Prüft die Anzahl der spezifischen Ldap Konfigurationsinstanzen.

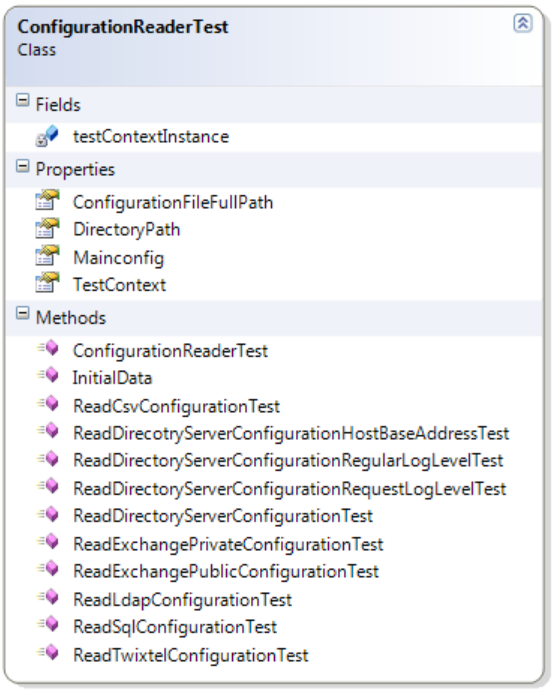
ReadExchangePrivateConfigurationTest

Prüft folgende Properties aus dem MainconfigFile „DFService.exe.config“:
- AssemblyName
- ClassName
- ConfigurationFile

Prüft die Anzahl der spezifischen ExchangePrivate Konfigurationsinstanzen.

ReadExchangePublicConfigurationTest

Prüft folgende Properties aus dem MainconfigFile



„DFService.exe.config“: - AssemblyName - ClassName - ConfigurationFile Prüft die Anzahl der spezifischen ExchangePublic Konfigurationsinstanzen.	
ReadSqlConfigurationTest	
Prüft folgende Properties aus dem MainconfigFile „DFService.exe.config“: - AssemblyName - ClassName - ConfigurationFile Prüft die Anzahl der spezifischen Sql Konfigurationsinstanzen.	
ReadTwixtelConfigurationTest	
Prüft folgende Properties aus dem MainconfigFile „DFService.exe.config“: - AssemblyName - ClassName - ConfigurationFile Prüft die Anzahl der spezifischen Twixtel Konfigurationsinstanzen.	

Tabelle 7: ConfigurationReaderTest

4.2. ViewModelTest

Die Testklassen im Testprojekt ViewModelTest sind alle identisch aufgebaut, da die grundlegende Funktionalität der ViewModels identisch ist. Nachfolgend werden die „kritischen“ Methoden und Funktionalitäten getestet, welche für den Lese- und Schreibvorgang essentiell sind. Die Create Methoden mit der Parameterliste (CreateX(Parameter)), lesen mittels der jeweiligen statischen Readmethode ProcessModuleInstanceConfigurationSection aus dem DF, die SearchModule spezifische Konfiguration aus und bildet diese in die ViewModel SearchModule Struktur ab. Die GenerateModuleInstanceConfigurationString Methode generiert wieder um den String aus der oben genannten Struktur. Zudem wird die Generierung neuer Instanzen aus dem GUI geprüft. Diese drei Tests werden auf allen ViewModels ausgeführt. Zudem werden auf dem *LDAPViewModel* noch zusätzliche Operationen, welche die Instanz Liste verändert, getestet. Diese jeweiligen Methoden sind in allen ViewModels identisch, darum werden sie nur einmal geprüft.

4.2.1. CSVViewModelTest

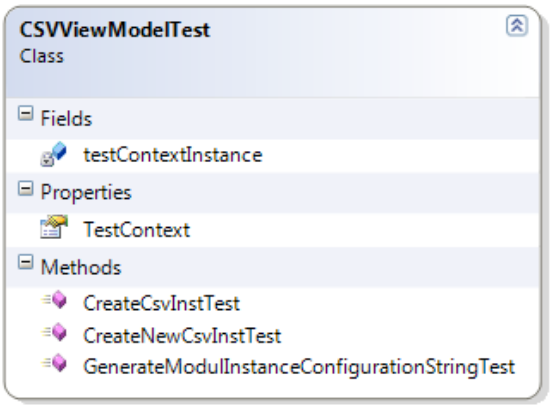
CreateNewCsvTest	
Prüft die CreateCsv() Methode, welche benutzt wird, wenn neue Instanzen im GUI hinzugefügt werden.	
CreateCsvTest	
Prüft die CreateCsv(Parameter) Methode, welche benutzt wird beim Einlesen einer bestehenden Konfiguration.	
GenerateModuleInstanceConfigurationStringTest	
Prüft die Generierung des ConfigurationString aller Csv-Instanzen. Nach diesem Methodendurchlauf ist die gesamte spezifische Csv-Konfiguration in einem String enthalten, dieser wird beim Schreibvorgang in die Konfigurationsdatei verwendet.	

Tabelle 8: CSVViewModelTes

4.2.2. ExchangePublicViewModelTest

CreateNewExchangePublicTest
Prüft die CreateExchangePublic() Methode, welche benutzt wird ,wenn neue Instanzen im GUI hinzugefügt werden.
CreateExchangePublicTest
Prüft die CreateExchangePublic(Parameter) Methode, welche benutzt wird beim Einlesen einer bestehenden Konfiguration.
GenerateModuleInstanceConfigurationStringTest
Prüft die Generierung des ConfigurationString aller ExchangePublic-Instanzen. Nach diesem Methodendurchlauf ist die gesamte spezifische ExchangePublic-Konfiguration in einem String enthalten, dieser wird beim Schreibvorgang in die Konfigurationsdatei verwendet.

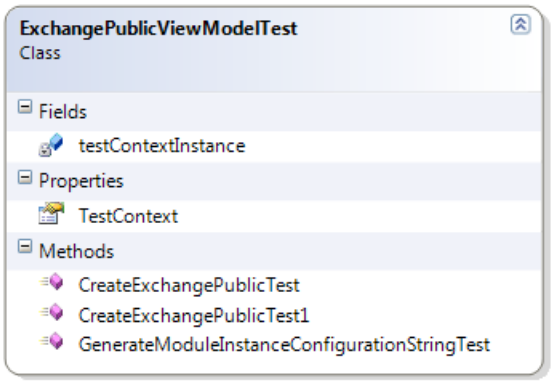


Tabelle 9: ExchangePublicViewModelTest

4.2.3. ExchangePrivateViewModelTest

CreateNewExchangePrivateTest
Prüft die CreateExchangePrivate() Methode, welche benutzt wird, wenn neue Instanzen im GUI hinzugefügt werden.
CreateExchangePrivateTest
Prüft die CreateExchangePrivate(Parameter) Methode, welche benutzt wird beim Einlesen einer bestehenden Konfiguration.
GenerateModuleInstanceConfigurationStringTest
Prüft die Generierung des ConfigurationString aller ExchangePrivate Instanzen. Nach diesem Methodendurchlauf ist die gesamte spezifische ExchangePrivate Konfiguration in einem String enthalten, dieser wird beim Schreibvorgang in die Konfigurationsdatei verwendet.

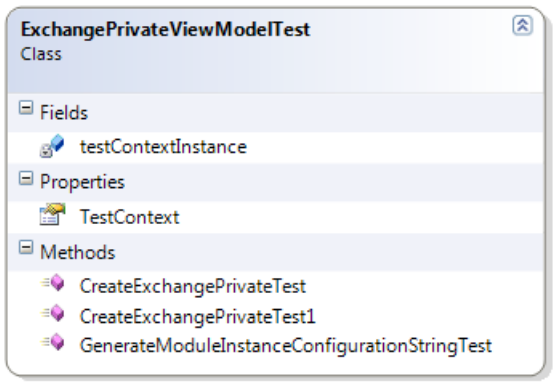


Tabelle 10: ExchangePrivateViewModelTest

4.2.4. LDAPViewModelTest

AddInstanceTest
Prüft die Existenz von Ldap-Instanzen nach dem Hinzufügen in der Instanzliste.
ClearInstanceListTest
Prüft das Löschen aller vorhandenen Ldap-Instanzen in der Instanzliste.
CreateLdapTest
Prüft die CreateLdap() Methode, welche benutzt wird, wenn neue Instanzen im GUI hinzugefügt werden.
CreateNewLdapTest
Prüft die CreateLdap(Parameter) Methode, welche benutzt wird beim Einlesen einer bestehenden Konfiguration.
GenerateModuleInstanceConfigurationStringTest
Prüft die Generierung des ConfigurationString aller Ldap-Instanzen. Nach diesem Methodendurchlauf ist die gesamte spezifische Ldap-Konfiguration in einem String enthalten, dieser wird beim Schreibvorgang in die Konfigurationsdatei verwendet.
RemoveInstanceTest
Prüft das Entfernen einer selektierten Ldap-Instanz aus der Instanzliste.

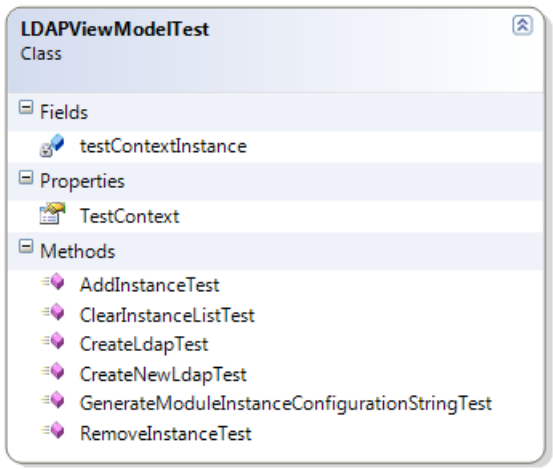


Tabelle 11: LDAPViewModelTest

4.2.5. SQLViewModelTest

CreateNewSqlTest
Prüft die CreateSql() Methode, welche benutzt wird, wenn neue Instanzen im GUI hinzugefügt werden.
CreateSqlTest
Prüft die CreateSql(Parameter) Methode, welche benutzt wird beim Einlesen einer bestehenden Konfiguration.
GenerateModuleInstanceConfigurationStrinTest
Prüft die Generierung des ConfigurationString aller Sql-Instanzen. Nach diesem Methodendurchlauf ist die gesamte spezifische SqlPrivate-Konfiguration in einem String enthalten, dieser wird beim Schreibvorgang in die Konfigurationsdatei verwendet.

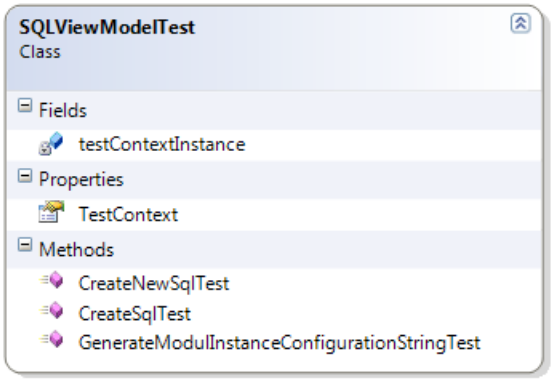


Tabelle 12: SQLViewModelTest

4.2.6. TwixTelViewModel

CreateNewTwixTelTest
Prüft die CreateTwixtel() Methode, welche benutzt wird, wenn neue Instanzen im GUI hinzugefügt werden.
CreateTwixTelTest
Prüft die CreateTwixtel(Parameter) Methode, welche benutzt wird beim Einlesen einer bestehenden Konfiguration.
GenerateModuleInstanceConfigurationStringTest
Prüft die Generierung des ConfigurationString aller Twixtel-Instanzen. Nach diesem Methodendurchlauf ist die gesamte spezifische TwixtelPrivate-Konfiguration in einem String enthalten, dieser wird beim Schreibvorgang in die Konfigurationsdatei verwendet.

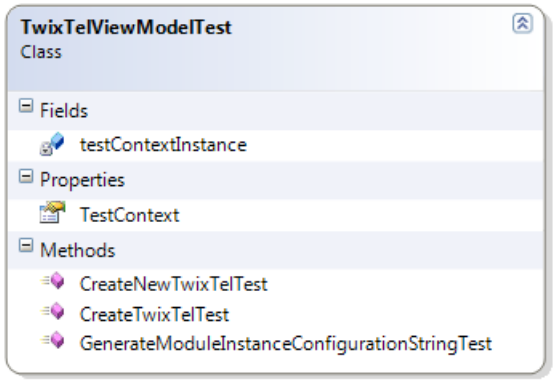


Tabelle 13: TwixTelViewModel

5. Tabellenverzeichnis

Tabelle 1: Systemtest 1 Zusammenfassung 4
Tabelle 2: Systemtest 2 Zusammenfassung 6
Tabelle 3: Systemtest 3 Zusammenfassung 7
Tabelle 4: Reader Tests 7
Tabelle 5: Writer Tests 8
Tabelle 6: Restart Test 8
Tabelle 7: ConfigurationReaderTest 10
Tabelle 8: CSVViewModelTes 10
Tabelle 9: ExchangePublicViewModelTest 11
Tabelle 10: ExchangePrivateViewModelTest 11
Tabelle 11: LDAPViewModelTest 11
Tabelle 12: SQLViewModelTest 12
Tabelle 13: TwixTelViewModel 12

Directory Framework GUI

Benutzerhandbuch

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
16.12.2009	1.0	erstellt	Müller	
	1.1	Ergänzungen	Müller	Dietrich
17.12.2009	2.0	Final Version	Müller	

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. Benutzerhandbuch 4
1.1. Installationsvoraussetzungen 4
1.2. Installationsanleitung 4
1.3. Konfigurationsanleitung 4
1.3.1. Main config Tab 4
1.3.1.1. Main config spezifische Konfigurationselemente 5
1.3.1.2. Aktivitätsbedingungen der Buttons 5
1.3.2. Csv Tab 5
1.3.2.1. Csv spezifische Konfigurationselemente 6
1.3.3. Exchange Public Tab 6
1.3.3.1. Exchange Public spezifische Konfigurationselemente 7
1.3.3.2. Aktivitätsbedingungen der Buttons 7
1.3.4. Exchange Private Tab 7
1.3.4.1. Exchange Private spezifische Konfigurationselemente 7
1.3.4.2. Aktivitätsbedingungen der Buttons 8
1.3.5. Ldap Tab 8
1.3.6. SQL Tab 9
1.3.6.1. Sql spezifische Konfigurationselemente 9
1.3.7. Twix Tel Tab 10
1.3.8. Instanz Liste 11
1.3.8.1. Aktivitätsbedingungen der Buttons 11
1.3.9. Instanz Konfiguration 11
1.3.9.1. Aktivitätsbedingungen der Buttons 11
1.3.10. Advanced Settings 11
1.3.10.1. Aktivitätsbedingung des Button 11
1.3.11. Tabellarische Auflistung der Konfigurationselemente 11
1.3.11.1. Main konfiguration 11
1.3.11.2. Csv Konfiguration 12
1.3.11.3. ExchangPublic Konfiguration 13
1.3.11.4. ExchangePrivate Konfiguration 13
1.3.11.5. Ldap Konfiguration 13
1.3.11.6. Sql Konfiguration 14
1.3.11.7. Twixtel Konfiguration 14

1. Benutzerhandbuch

1.1. Installationsvoraussetzungen

Software	
Betriebssystem	Microsoft Windows Server 2003 / 2008
sonstiges	.NET Framework 3.5 SP1

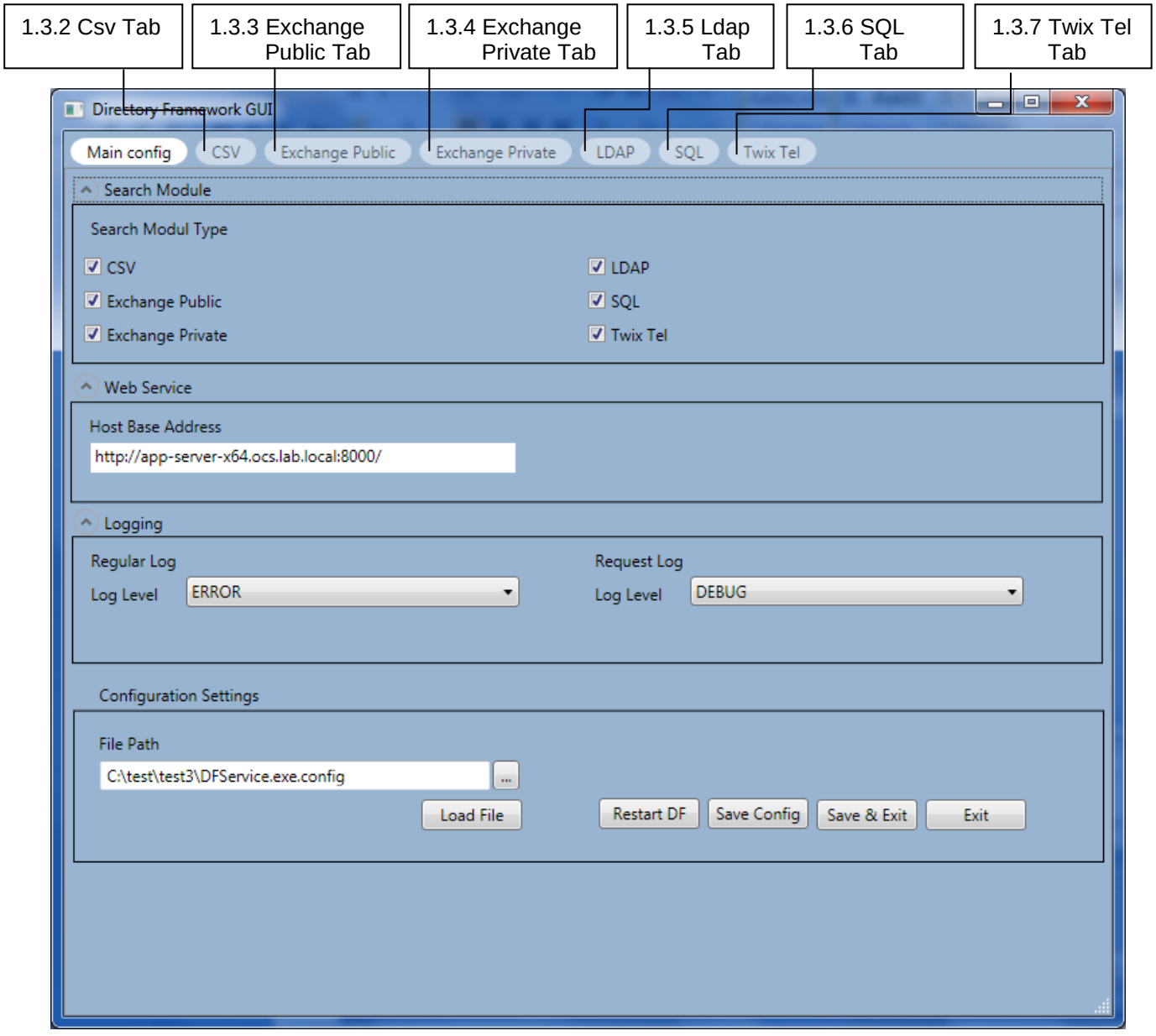
1.2. Installationsanleitung

Entspricht der Anleitung des Directory Framework. Dieses Dokument befindet sich im *Anhang A*.

1.3. Konfigurationsanleitung

Das Directory Framework GUI dient zur Konfiguration desselben. Über die grafische Oberfläche lassen sich die gesamten SearchModule Konfigurationen vornehmen, sowie ein Teil der Main Konfiguration. Nachfolgend werden alle Tabs bildlich dargestellt und die einzelnen Konfigurationselemente erklärt.

1.3.1. Main config Tab



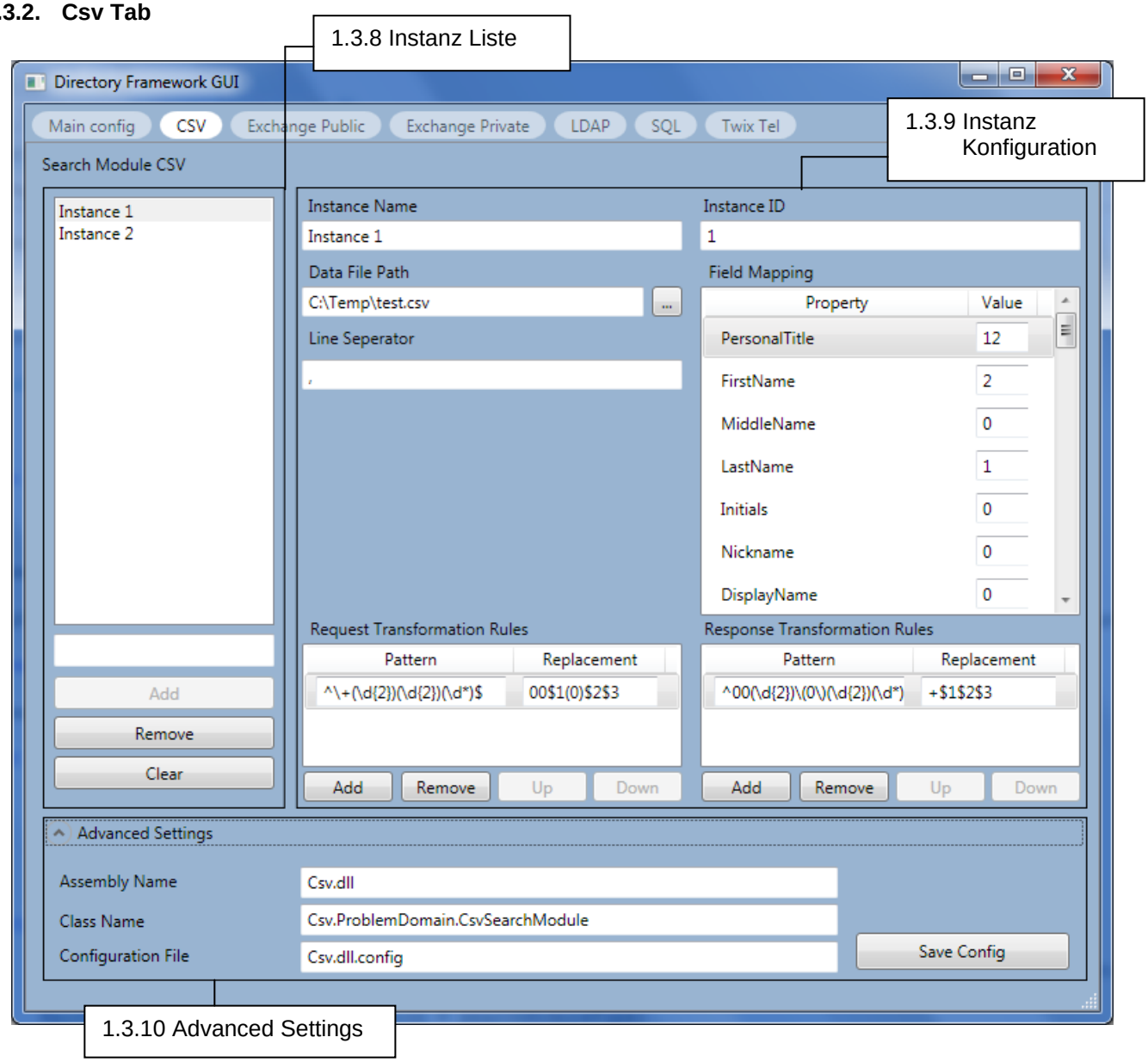
1.3.1.1. Main config spezifische Konfigurationselemente

Die Konfigurationselemente *HostBaseAddress*, *RegularLogLevel* und *RequestLogLevel* müssen zwingend konfiguriert werden, damit die Konfiguration gespeichert werden kann. SearchModule hingegen müssen nicht zwingend ausgewählt werden um einen Speichervorgang auszulösen.

1.3.1.2. Aktivitätsbedingungen der Buttons

Button	Bedingung für die Aktivierung
Load File	Der *Load File* Button wird aktiv, wenn eine Datei selektiert wurde, oder wenn der Pfad manuell eingegeben wurde. Bei letzterem wird allerdings überprüft ob das File auch vorhanden ist. Wenn ein manuell eingegebener Pfad auf ein File verweist, welches im Dateisystem nicht vorhanden ist bleibt der Button inaktiv.
Restart DF	Der *Restart DF* Button kann erst nach einem erfolgreichen Speichervorgang verwendet werden und wird auch erst dann aktiv.
Save Config	Der *Save Config* Button wird aktiviert wenn die Konfigurationselemente *HostBaseAddress*, *RegularLogLevel* und *RequestLogLevel* konfiguriert wurden.
Save & Exit	Exakt gleiche Bedingung wie beim *Save Config* Button.
Exit	Der *Exit Button* ist immer aktiv und kann jeder Zeit verwendet werden.

1.3.2. Csv Tab



1.3.2.1. Csv spezifische Konfigurationselemente

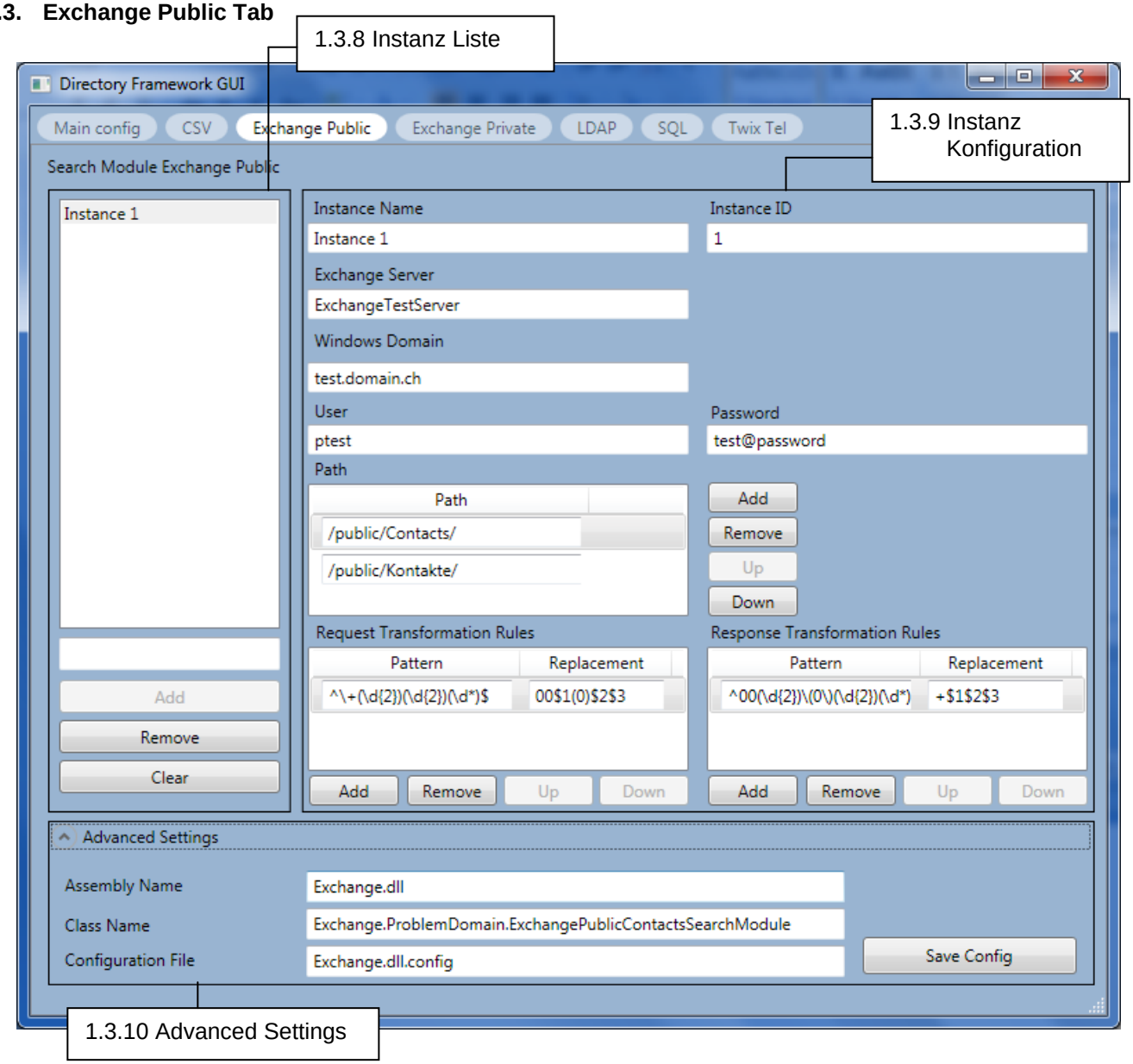
Die *FieldMapping* Liste enthält 44 vorgefertigte Elemente. Diese Elemente sollten mit der Nummer der Spaltenüberschrift aus dem Csv File übereinstimmen. Wenn ein Element nicht in der Csv Datei vorhanden ist, wird dieses mit dem Wert 0 konfiguriert.

Zur Veranschaulichung ein Beispiel:

A	B	C	D	E	F	G	H	I	J	K	L
Name	Vorname	…	…	…	…	…	…	…	…	…	Anrede

Csv Datei		Directory Framework Konfiguration	
Spaltenüberschrift	Spalte	Property	Value
Name	A	LastName	1
Vorname	B	FirstName	2
…	…	…	…
Anrede	L	PersonalTitle	12

1.3.3. Exchange Public Tab



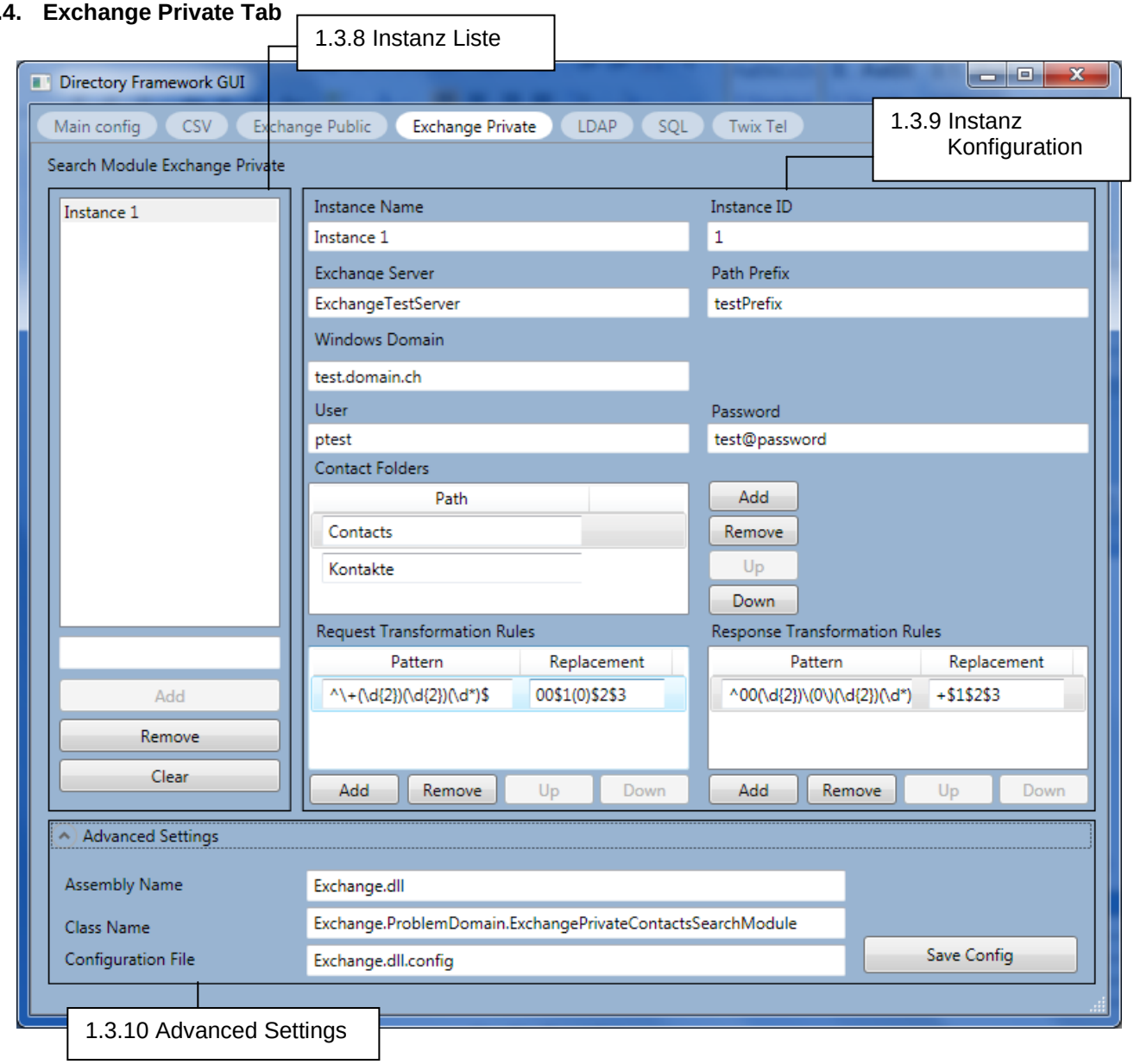
1.3.3.1. Exchange Public spezifische Konfigurationselemente

Im *Exchange Public Tab* gibt es eine zusätzliche Liste, bei welcher die Listenelemente die Public Contacts *Paths* definieren. Das Directory Framework berücksichtigt die Reihenfolge, deshalb ist auch diese Liste mit den Buttons *Up* and *Down* ausgestattet, sowie natürlich mit den *Add* und *Remove* Buttons um Elemente hinzuzufügen und zu entfernen.

1.3.3.2. Aktivitätsbedingungen der Buttons

Button	Bedingung für die Aktivierung
Add	Der *Add* Button wird aktiviert sobald eine Instanz selektiert wurde aus der Instanz Liste.
Remove	Der *Remove* Button ist aktiv, wenn sich ein Element in der Liste befindet.
Up	Der *Up* Button ist aktiv, wenn die Liste nicht leer ist und nicht das erste Element selektiert ist.
Down	Der *Down* Button ist aktiv, solange Elemente in der Liste vorhanden sind und nicht das letzte Element der Liste selektiert ist.

1.3.4. Exchange Private Tab



1.3.4.1. Exchange Private spezifische Konfigurationselemente

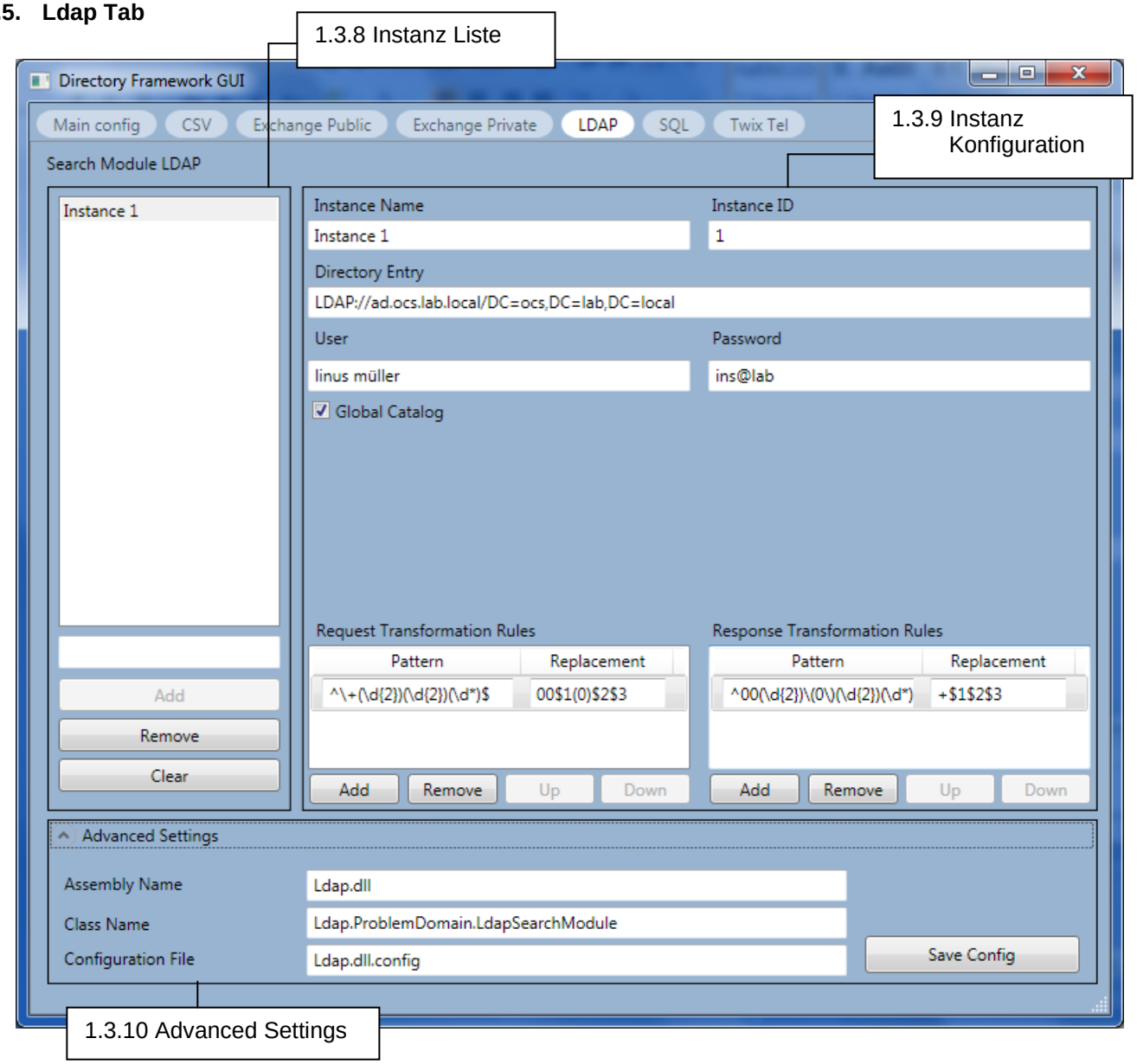
Im *Exchange Private Tab* gibt es eine zusätzliche Liste, bei welcher die Listenelemente die *Contact Folders* definieren. Das Directory Framework berücksichtigt die Reihenfolge, deshalb ist auch diese Liste mit den Buttons

Up and *Down* ausgestattet, sowie natürlich mit den *Add* und *Remove* Buttons um Elemente hinzuzufügen und zu entfernen.

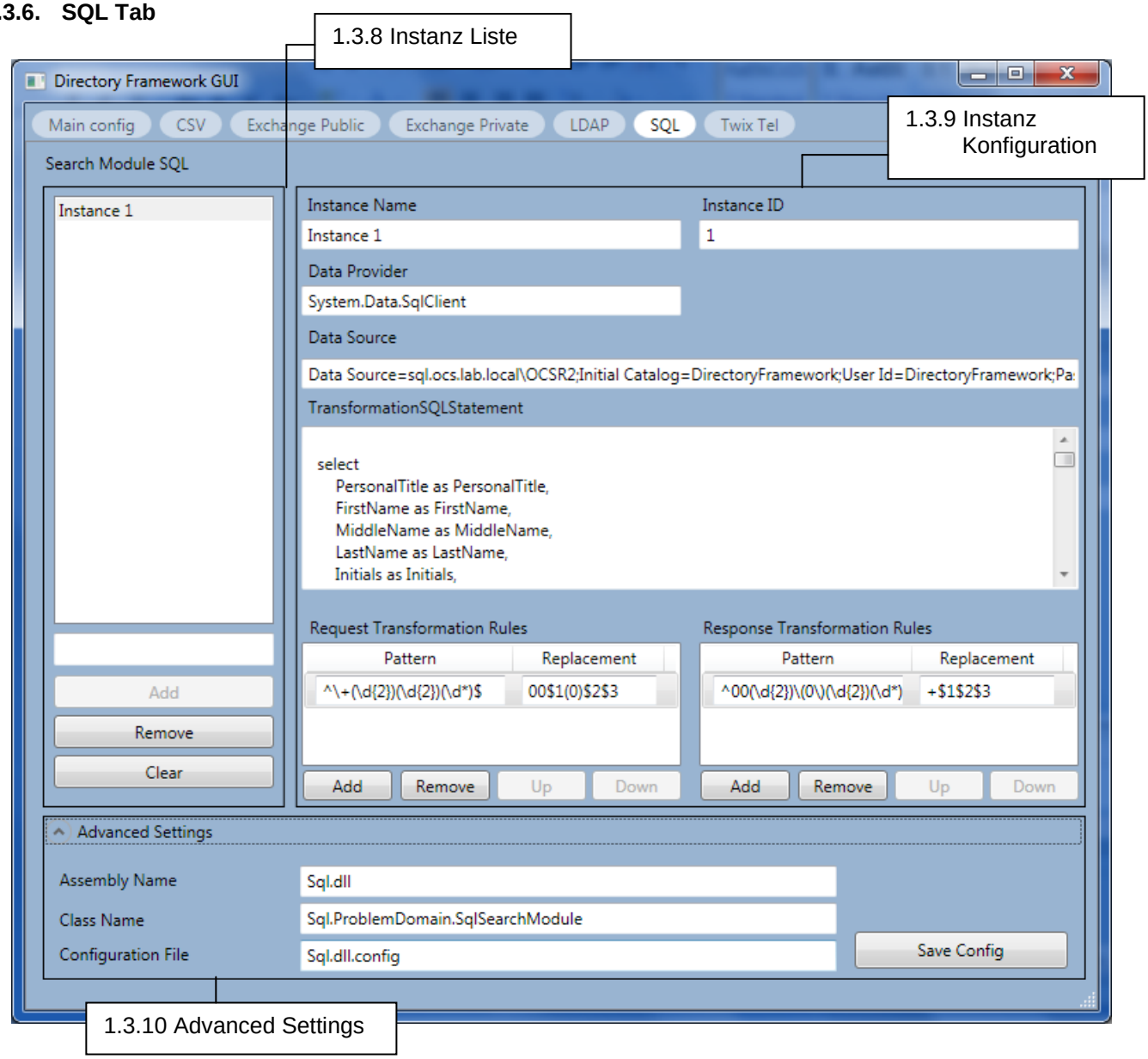
1.3.4.2. Aktivitätsbedingungen der Buttons

Button	Bedingung für die Aktivierung
Add	Der *Add* Button wird aktiviert sobald eine Instanz selektiert wurde aus der Instanz Liste.
Remove	Der *Remove* Button ist aktiv, wenn sich ein Element in der Liste befindet.
Up	Der *Up* Button ist aktiv, wenn die Liste nicht leer ist und nicht das erste Element selektiert ist.
Down	Der *Down* Button ist aktiv, solange Elemente in der Liste vorhanden sind und nicht das letzte Element der Liste selektiert ist.

1.3.5. Ldap Tab



1.3.6. SQL Tab



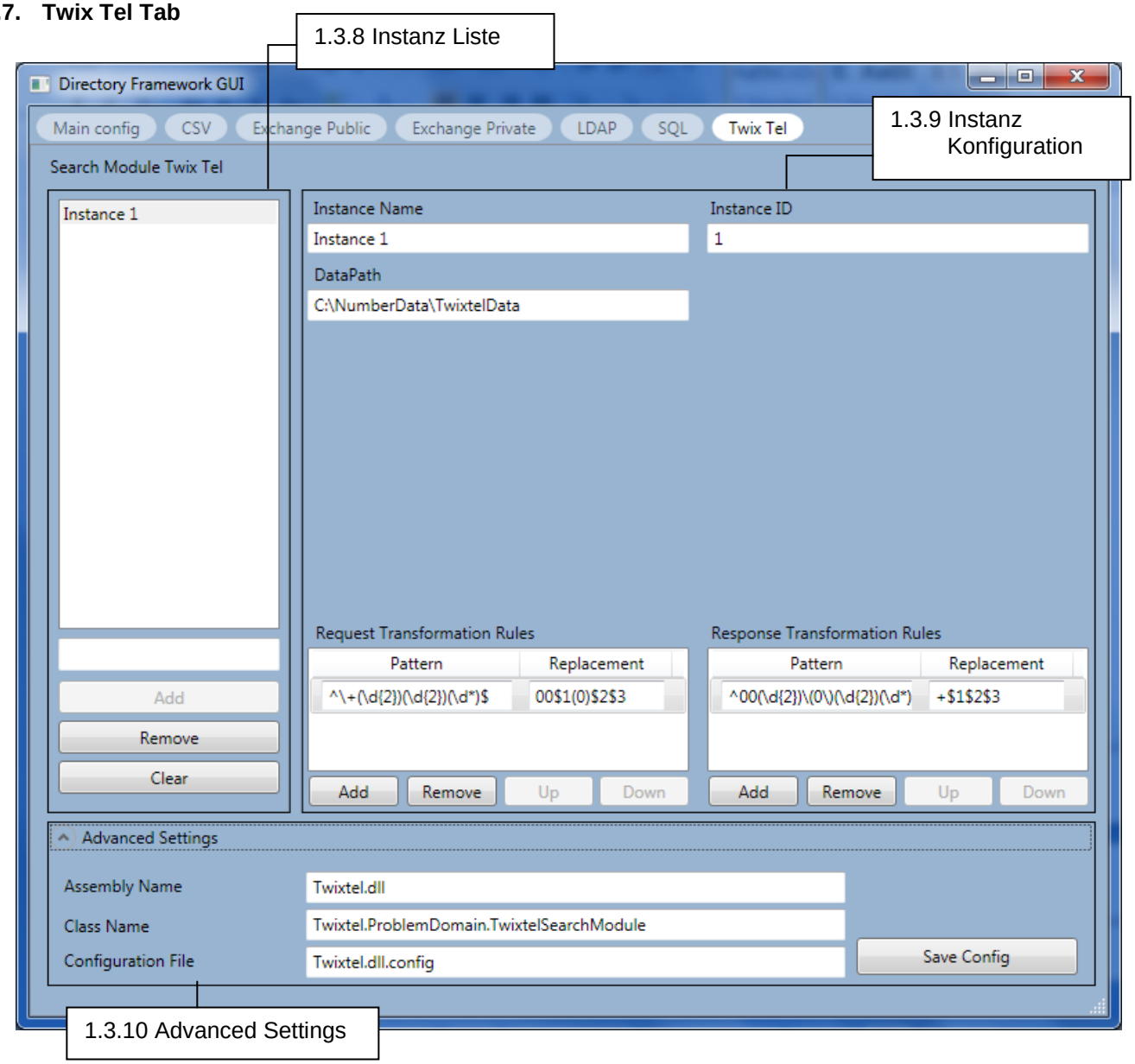
1.3.6.1. Sql spezifische Konfigurationselemente

Der *Connection String* setzt sich aus mehreren Angaben zusammen welche mit einem `;` getrennt werden. Die einzelnen Unterelemente des *Connection String* sind folgende:

- Data Source
- Initial Catalog
- User Id
- Password

Das *SQL Transformation Statement* ist die SQL Query welche als String abgespeichert wird.

1.3.7. Twix Tel Tab



1.3.8. Instanz Liste

In der Instanz Liste lassen sich SearchModule Instanzen hinzufügen, selektierte entfernen oder die gesamte Liste löschen. In der TextBox unterhalb der Liste kann man die Anzahl der neu gewünschten Instanzen definieren, sobald der Fokus dieser TextBox verloren geht, wird der *Add* Button aktiv und kann betätigt werden. Der *Remove* Button ist aktiv sobald eine Instanz in der Liste selektiert ist. Die gesamte Liste löschen mit dem *Clear* Button ist möglich solange die Liste nicht leer ist.
Bei einer Änderung der Selektion wird immer die Instanz Konfiguration mit den Werten der aktuell selektierten Instanz aktualisiert.

1.3.8.1. Aktivitätsbedingungen der Buttons

Button Name	Bedingung für die Aktivierung
Add	Der *Add* Button wird aktiviert sobald in der TextBox ein Wert steht und der Fokus nicht mehr auf der TextBox liegt. (TextBox oberhalb des Add Buttons)
Remove	Der *Remove* Button wird aktiv gesetzt wenn eine Instanz aus der Instanz Liste selektiert ist.
Clear	Der *Clear* Button ist aktiv, solange die Instanz Liste nicht leer ist.

1.3.9. Instanz Konfiguration

Zeigt die Detailkonfiguration der selektierten Instanz an und alle Werte können verändert werden. Die Elemente der beiden Transformation Rules Listen können innerhalb dieser verändert werden. Mit den darunter liegenden Buttons können neue Rules hinzugefügt, selektierte entfernte und die Reihenfolge der vorhandenen Elemente geändert werden. Letzter Punkt der Reihenfolge ist wichtig, weil diese vom Directory Framework berücksichtigt wird. Die Buttons werden aktiv, wenn die Funktionalität die sich dahinter verbirgt, auch möglich ist.

1.3.9.1. Aktivitätsbedingungen der Buttons

Button	Bedingung für die Aktivierung
Add	Der *Add* Button wird aktiviert sobald eine Instanz selektiert wurde aus der Instanz Liste.
Remove	Der *Remove* Button ist aktiv, wenn sich ein Element in der Liste befindet.
Up	Der *Up* Button ist aktiv, wenn die Liste nicht leer ist und nicht das erste Element selektiert ist.
Down	Der *Down* Button ist aktiv, solange Elemente in der Liste vorhanden sind und nicht das letzte Element der Liste selektiert ist.

1.3.10. Advanced Settings

In den Advanced Settings können Einstellungen getätigt werden, welche nur einmal pro SearchModule vorkommen und für alle Instanzen gelten. Die Felder *Assembly Name* und *Class Name* sind direkt vom Directory Framework abhängig und sollten im Normallfall nicht verändert werden, solange die Implementation des Directory Frameworks nicht verändert wird. Das Feld *Configuration File* definiert den Namen der Konfigurationsdatei, welcher beim Speichern verwendet wird. Der Save *Config Button* ist aktiv, solange die Instanz Liste Elemente enthält. Wichtig zu beachten ist, dass dieser *Save Config* Button nur das aktuelle SearchModule speichert.

1.3.10.1. Aktivitätsbedingung des Button

Button	Bedingung für die Aktivierung
Save Config	Der Button wird aktiviert sobald die Instanz Liste ein Element beinhaltet.

1.3.11. Tabellarische Auflistung der Konfigurationselemente

1.3.11.1. Main konfiguration

Die Hauptkonfiguration beinhaltet die Einstellungen der Konfigurationsdatei DFService.exe.config.

Einstellungskomponenten
Auswahl der SearchModules
HostBaseAddress
RegularLogLevel
RequestLogLevel

1.3.11.2. Csv Konfiguration

Die Csv Konfiguration beinhaltet die Einstellungen der Konfigurationsdatei Csv.dll.config.

Einstellungstyp	Einstellungskomponenten
SearchModule Einstellungen	Assembly Name
	Class Name
	Configuration File
SearchModule Instanz Einstellungen	Instance Name
	Instance ID
	Data File Path
	Line Separator
	Field Mapping - Personal Title - First Name - Middle Name - Last Name - Initials - Nickname - Display Name - Title - Profession - User ID - Employee Number - Manager - Organization - Department - Room Number - Phone Business - Phone Mobile - Phone Home - Phone Ip Phone - Fax - Pager - Street Office - Street No Office - City Office - Zip Code Office - Country Office - Street Home - Street No Home - City Home - Zip Code Home - Country Home - Street Mailing - Street No Mailing - Zip Code Mailing - Country Mailing - Street Other - Street No Other - City Other - Zip Code Other - Country Other - Email Address one - Email Address two - Email Address three
	Request Transformation Rules - Pattern

	- Replacement
	Response Transformation Rules - Pattern - Replacement

1.3.11.3. ExchangPublic Konfiguration

Die ExchangePublic Konfiguration beinhaltet die Einstellungen der Konfigurationsdatei Exchange.dll.config.

Einstellungstyp	Einstellungskomponenten
SearchModule Einstellungen	Assembly Name
	Class Name
	Configuration File
SearchModule Instanz Einstellungen	Instance Name
	Instance ID
	Exchange Server
	Windows Domain
	User
	Password
	Path
	Request Transformation Rules - Pattern - Replacement
	Response Transformation Rules - Pattern - Replacement

1.3.11.4. ExchangePrivate Konfiguration

Die ExchangePrivate Konfiguration beinhaltet die Einstellungen der Konfigurationsdatei Exchange.dll.config.

Einstellungstyp	Einstellungskomponenten
SearchModule Einstellungen	Assembly Name
	Class Name
	Configuration File
SearchModule Instanz Einstellungen	Instance Name
	Instance ID
	Exchange Server
	Path Prefix
	Windows Domain
	User
	Password
	Contact Folders
	Request Transformation Rules - Pattern - Replacement
	Response Transformation Rules - Pattern - Replacement

1.3.11.5. Ldap Konfiguration

Die Ldap Konfiguration beinhaltet die Einstellungen der Konfigurationsdatei Ldap.dll.config.

Einstellungstyp	Einstellungskomponenten
SearchModule Einstellungen	Assembly Name
	Class Name
	Configuration File
SearchModule Instanz Einstellungen	Instance Name
	Instance ID

	Directory Entry
	Global Catalog
	User
	Password
	Request Transformation Rules - Pattern - Replacement
	Response Transformation Rules - Pattern - Replacement

1.3.11.6. Sql Konfiguration

Die Sql Konfiguration beinhaltet die Einstellungen der Konfigurationsdatei Sql.dll.config.

Einstellungstyp	Einstellungskomponenten
SearchModule Einstellungen	Assembly Name
	Class Name
	Configuration File
SearchModule Instanz Einstellungen	Instance Name
	Instance ID
	Data Provider
	Data Source
	Transformation Sql Statement
	Request Transformation Rules - Pattern - Replacement
	Response Transformation Rules - Pattern - Replacement

1.3.11.7. Twixtel Konfiguration

Die Twixtel Konfiguration beinhaltet die Einstellungen der Konfigurationsdatei Twixtel.dll.config.

Einstellungstyp	Einstellungskomponenten
SearchModule Einstellungen	Assembly Name
	Class Name
	Configuration File
SearchModule Instanz Einstellungen	Instance Name
	Instance ID
	Data Path
	Request Transformation Rules - Pattern - Replacement
	Response Transformation Rules - Pattern - Replacement