Diplomarbeit

Entwicklung einer Kampagnenmanagementlösung für KMUs als Office Business Application

ausgeführt zum Zweck der Erlangung des akademischen Grades eines
„Diplom-Ingenieurs für technisch-wissenschaftliche Berufe“
am Masterstudiengang Telekommunikation und Medien
der Fachhochschule St. Pölten

ausgeführt von
Andreas Riegler, B.Sc.
tm0710262055

unter der Erstbetreuung von
Dipl. Ing. (FH) Fritz Grabo

Zweitbegutachtung von
Dipl. Ing. Grischa Schmiedl

Ort, Datum Unterschrift

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

Kurzfassung

Diese Arbeit beschäftigt sich mit der Entwicklung einer Customer Relationship Management (CRM) Software für Klein- und Mittelunternehmen (KMU). CRM-Systeme sind oftmals gerade für KMU nicht rentabel, da sie dem Unternehmen keinen direkten Gewinn bieten, die Anschaffung aber meist mit erheblichen Kosten verbunden ist.

Es soll daher untersucht werden, wie mit den ausgewählten Technologien, Windows SharePoint Services 3.0, VSTO und Silverlight 2.0, ein CRM-System entwickelt werden kann, welches auf vorhandene Komponenten eines Unternehmens aufsetzt und dadurch Kosten spart. Das zu entwickelnde CRM-System soll dabei in Microsoft Outlook 2007 integriert werden, wodurch der Benutzer kein neues, eigenständiges Programm erlernen muss, sondern seine vorhandenen Erfahrungen mit Outlook 2007 einsetzen kann.

Um die genannten Punkte zu erfüllen, wurden zunächst alle Technologien einzeln betrachtet und Vor- sowie Nachteile dieser Technologien im Hinblick auf die Aufgabenstellung bewertet. Danach wurde ein Prototyp des CRM-Systems entwickelt, um die Kombination der einzelnen Technologien aufzuzeigen. Als letzter Schritt wurde eine Bewertung der Technologien durchgeführt und ein Ausblick auf weitere Möglichkeiten des CRM-Systems wie Silverlight mobile oder Offlinefähigkeit des Systems gegeben.

Abstract

This masther thesis is about the analysis and development of an Customer Relationship Managment (CRM) software for small and medium sized companies. CRM systems aren't very cost-effective for these companies, because most of the time they won't produce any direct profit, but cause extensive costs in purchasing.

Thus it shall be investigated if a CRM system based on the selected technologies, Windows SharePoint Services, VSTO, Silverlight and existing enterprise componente can be developed while reducing costs. The CRM system which

should be developed, should be integrated into Microsoft Office 2007, to reuse the existing experience of the users.

To fulfill these points, the technologies have been examined separately and advantages and disadvantages of these technologies have been evaluated. After this evaluation, a prototype of the CRM system has been developed to show the combination of the chosen technologies. As a last step, a conclusion of the used technologies was made and an outlook of additional possibilities concerning the CRM system, like Silverlight mobile or offline mode was given.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Problembenennung und Motivation

Oftmals sind für Klein- und Mittelunternehmen (KMU) bestehende Customer Relationship Management (CRM)-Lösungen nicht leistbar oder in einer billigeren Variante vom Funktionsumfang so minimiert, dass die noch vorhandenen Features nicht ausreichen.
(vgl. [Gre04], S. 319)

Das Microsoft Dynamics CRM Systems[1] kostet 1.246 € (Stand 02/2009) pro Benutzerlizenz. Da es sich hierbei um Einzelbenutzerlizenzen handelt, also diese Kosten pro Benutzer entstehen, müsste ein Unternehmen mit 10 Mitarbeitern 12.460€ allein für Lizenzgebühren zahlen. Die Anschaffung eines CRM-Systems wird somit erschwert und stellt für manche Firmen sogar eine unüberwindbare Hürde dar, da die Verwendung eines CRM-Systems keine direkten Auswirkungen auf den finanziellen Gewinn eines Unternehmens hat. Eine mögliche Alternative wäre die Verwendung von Open-Source CRM-Systemen. Allerdings weisen auch diese oft einen eingeschränkten Funktionsumfang auf und lassen sich nicht in vorhandene Applikationen wie Microsoft Outlook[2] integrieren, wodurch eventuell vorhandene Benutzererfahrung nicht genützt werden kann.

[1] http://www.microsoft.com/dynamics/crm/default.mspx
[2] http://www.microsoft.com/outlook

Aufgrund dieser Probleme soll erforscht werden, ob mithilfe von Visual Studio Tools for Office Applications (VSTO)[3] auf Basis von Microsoft Outlook 2007 sowie Windows SharePoint Services (WSS)[4] eine kostengünstige Lösung, welche einen passenden Funktionsumfang für KMU bietet, entwickelt werden kann.

1.2 Ziele und Vorgehensweise

Um die oben genannten Probleme zu lösen, liegt es nahe, Technologien zu wählen, welche Integrationsmöglichkeiten in eine vorhandene Applikation ermöglichen und vorhandene Lizenzen wiederverwenden um etwaige Anschaffungskosten gering zu halten. Durch die Integration in eine vorhandene Applikation können vorhandene Benutzererfahrungen genützt werden, welche gerade den Einstieg in das CRM-System erleichtern. Zu diesem Zwecke sollen die in Kapitel 2.1 definierten Technologien eingesetzt werden.

Diese Arbeit zeigt die mögliche Implementierung eines CRM-Systems und die Verwendung der einzelnen Komponenten im Detail auf und bewertet diese auf Basis der vorhandenen Funktionalitäten und den vorhandenen Schnittstellen zu den anderen verwendeten Technologien. Dazu wird zunächst recherchiert, welche Möglichkeiten Windows SharePoint Services bieten, um die in Kapitel 1.1 angesprochenen Problemfelder beseitigen zu können. In den darauffolgenden Kapiteln wird dies auch für VSTO und Silverlight 2.0[5] durchgeführt, wobei bei VSTO die Einbindung in Microsoft Outlook 2007 und bei Silverlight 2.0 die grafische Aufbereitung der Daten in Outlook 2007 untersucht wird.

Im Anschluss an die durchgeführten Recherchen sowie einer Bewertung jeder Komponente, wird aufgrund der gesammelten Erkenntnisse eine Lösung mit Hilfe von C#, VSTO und Windows SharePoint Services entwickelt und im Zusammenhang mit der Aufgabenstellung relevante Funktionen im Detail besprochen.

[3] http://msdn.microsoft.com/en-us/office/aa905533.aspx
[4] http://office.microsoft.com/de-de/sharepointtechnology/FX011204871031.aspx
[5] http://silverlight.net

Kapitel 2

Rahmenbedingungen und Begriffsdefinitionen

2.1 Rahmenbedingungen

Im nachfolgenden werden die Rahmenbedingungen, welche dem CRM-System zu Grunde liegen definiert.
Das zu entwickelnde CRM-System, sollte keine zusätzlichen Kosten für die KMUs verursachen, da diesen Unternehmen meist kein Budget für die Anschaffung eines CRM-Systems zur Verfügung steht und die Verwendung keinen direkten Gewinn mit sich bringt. Deswegen sollen oftmals vorhandene Lizenzen und Softwareprodukte, wie Microsoft Windows Server und SQL Server, wiederverwendet werden um Lizenz- sowie Softwarekosten möglichst gering zu halten. Das CRM-System soll dabei als Office Business Application in Outlook 2007 integriert werden. Microsoft Outlook wird von Microsoft als Teil der Office Produktreihe als zentrales Kommunikationsmittel innerhalb von Firmen positioniert und weist einen Marktanteil von ca. 90% auf.
(vgl. [Mey09])

Deshalb bietet es sich an, das CRM-System direkt in Outlook 2007 zu integrieren, da somit auch vorhandene Benutzererfahrungen mit Outlook genutzt werden und Benutzer sich nicht erst in ein neues Umfeld einarbeiten und gegebenenfalls Schulungen für das Programm besuchen müssen.

Aufgrund dieser Anforderungen wurden folgende Komponenten für die Umsetzung gewählt:

- Windows SharePoint Services
- Visual Studio Tools for Office Applications
- Silverlight 2.0

Das folgende Kapitel beschreibt kurz die verwendeten Technologien und ihren Zweck innerhalb des CRM-Systems. Die einzelnen Technologien werden danach in den Kapiteln 3-5 im Detail besprochen.

2.2 Technologien

Windows SharePoint Services
Windows SharePoint Services 3.0 (WSS) ist ein kostenloses Produkt der Firma Microsoft, welches umfangreiche Funktionalitäten wie Dokumentenmanagement, Wikis und Benachrichtigungssysteme innerhalb einer Webseite zur Verfügung stellt. WSS stellt die serverseitige Komponente des CRM-Systems dar und dient damit auch als Basis des Systems. Dabei dient WSS nicht nur als Dokumentenverwaltung, sondern erledigt eine Vielzahl von anderen Aufgaben wie Ausführung und Steuerung von Workflows und Möglichkeiten die Applikation skalierbar zu gestalten. Die benötigte WSS Funktionalität wird durch WebServices, welche bereits in großer Menge vorhanden sind, von VSTO und Silverlight konsumiert.

Aufgrund des großen Funktionsumfanges von WSS erleichtert und beschleunigt sich die Entwicklung erheblich, da viele Funktionen nicht selbst entwickelt werden müssen, jedoch muss eine Einarbeitungszeit in die vorhandenen Funktionalitäten von WSS eingeplant werden. Aufgrund dieser Vorteile wurde WSS als Basis für das CRM-System ausgewählt und wird in Kapitel 3 behandelt werden.

Visual Studio Tools for Office Applications 2008
Wie bereits im Kapitel 2.1 erklärt, setzt das CRM-System auf Outlook 2007 auf. Durch die Visual Studio Tools for Office Applications (VSTO) ist die

Erstellung sogenannter Office Business Applications (OBA) möglich. Die gesamte CRM Funktionalität ist über eigens entwickelte Modifizierungen von Outlook 2007 verwendbar. Dadurch bleibt der Benutzer in einer bekannten Umgebung und ein hohes Maß an Bedienbarkeit wird erhalten.

VSTO ist ein Toolset für Visual Studio 2005/2008 und ermöglicht die Erstellung von Programmfunktionalitäten für Produkte der Microsoft Office Produktfamilie. Es dient also als Schnittstelle zwischen Outlook und der grafischen Darstellung der Daten durch Silverlight. Das Erstellen von Erweiterungen für Office Produkte ist auch mittels Visual Basic for Applications (VBA) möglich, allerdings bietet VSTO eine Vielzahl von Vorteilen gegenüber VBA. Der größte Vorteil von VSTO ist die Unterstützung des kompletten .NET Frameworks, welches komplexere Funktionalitäten ermöglicht. Weiters können mittels VSTO WebServices angesprochen werden und eigene Objekte, im Sinne der objektorientierten Entwicklung, wiederverwendet werden. Ein Vorteil der zwar nur Entwickler betrifft, aber auch nicht ignoriert werden darf ist die Entwicklungsumgebung. VSTO wird in der Entwicklungsumgebung Visual Studio entwickelt, welches im Vergleich zum VB Editor in Office Produkten Features wie Debugging, Zugriff auf alle .NET Basisklassen und umfangreiche Deploymentmöglichkeiten bietet.

Silverlight 2.0

Die grafische Aufbereitung der in WSS gespeicherten Daten und Objekte soll mittels Silverlight 2.0 durchgeführt werden. Silverlight ist eine neue Technologie von Microsoft und basiert auf der Windows Presentation Foundation (WPF) welche in .NET 3.0 eingeführt wurde. Für die Entwicklung wird die Version 2.0 verwendet, da 3.0 zum Zeitpunkt der Erstellung noch nicht öffentlich verfügbar war.

Die zur Darstellung benötigten Daten werden mittels WebServices von WSS abgerufen und je nach Art in Outlook dargestellt. Silverlight bietet dazu derzeit 30 vorhandene „Widgets“ um Daten auf unterschiedliche Weise darstellen zu können. Zusätzlich existieren noch weitere Open-Source Widgets. Weiters besitzt Silverlight derzeit fünf verschiedene Diagrammtypen, welche sich vor allem für Auswertungen im CRM-System eignen.

Da noch nicht viele Referenzimplementierungen von Silverlight als Office Business Application existieren, soll diese Implementierung aufzeigen, ob sich Silverlight zur Darstellung in Microsoft Office Produkten eignet.

2.3 Begriffsdefinitionen

Line of Business Systeme
Line of Business Systeme sind kritische Applikationen, welche einen zentralen Punkt in Unternehmen darstellen und dieses in einem oder mehreren Punkten wie Ressourcen Planung oder Account Management unterstützen. LOB Applikationen sind großteils Programm-Suiten, welche eine Vielzahl an unterschiedlichen Tools für Unternehmen anbieten. Beispiele für bekannte LOB Applikationen sind SAP[1] und Siebel[2].
(vgl. [Leo07], S. 762)

Office Business Application
Office Business Applications sind Applikationserweiterungen die dazu dienen, dokumentenbasierte oder LOB Prozesse in vorhandene Microsoft Office Produkte zu integrieren. OBA bieten somit eine Schnittstelle zwischen oft verwendeten Microsoft Produkten wie Outlook oder Word und verknüpfen diese mit vorhandenen LOB Daten. Somit können oftmals nicht einfach zugängliche Daten für eine größere Anzahl an Benutzern bereitgestellt werden. Endnutzer haben den Vorteil, dass diese über ein bereits bekanntes Programm auf Daten von Drittsystemen zugreifen können und dadurch kein neues Programm „erlernen“ müssen.

Rich Internet Application
Der Ausdruck Rich Internet Application (RIA) wurde das erste Mal 2002 in einem Whitepaper von Jeremy Allaire erwähnt (vgl. [All02], S. 1f). Es handelt sich dabei um webbasierte, möglichst plattformunabhängige Applikationen, welche denselben Funktionsumfang wie herkömmliche Desktopbasierte Applikationen aufweisen. Diese Applikationen sollen neben dem Funktionsumfang

[1] http://www.sap.com
[2] www.oracle.com/applications/crm/siebel/index.html

auch bestmögliche Antwortzeiten aufweisen. Deshalb wird ein Teil der Logik bereits am Client ausgeführt und der serverseitige Aufwand auf Datenhaltung und -bereitstellung reduziert. Da es sich bei Silverlightapplikationen um RIA handelt, sollen diese Anforderungen auch von Silverlight erfüllt werden.

Eweek untersuchte im April 2008 die Fähigkeiten von Silverlight bezüglich RIAs und kam zu dem Ergebnis, dass Silverlight für Microsoft Entwickler eine gute Wahl für die Entwicklung von RIAs werden könnte.
(vgl. [Ewe09])

Kapitel 3

Serverseitige Speicherung der Daten mittels WSS 3.0

3.1 Allgemeines

Das Unternehmen Microsoft selbst beschreibt Windows SharePoint Services 3.0 wie folgt:
„Windows SharePoint Services is a versatile technology included in Microsoft Windows Server 2003 that enables organizations and business units of all sizes to increase the efficiency of business processes and improve team productivity. With tools for collaboration that help people stay connected across organizational and geographic boundaries, Windows SharePoint Services gives people access to documents and information they need.[...]“ (vgl. [Mic08b])

Windows SharePoint Services 3.0, bildet das kostenlose Grundgerüst für Microsoft Office SharePoint Server 2007 (MOSS 2007). Die Kernfunktionalitäten sind dabei bereits durch WSS implementiert und werden durch zusätzliche Features von SharePoint Server 2007 erweitert. Für das zu entwickelnde CRM-System ist es deshalb ausreichend, auf die Grundfunktionalitäten von WSS zurückzugreifen und damit Kosten bei Lizenzen zu sparen. Besitzt man eine Windows Server 2003 Lizenz, hat man dadurch automatisch die Berechtigung WSS zu installieren und zu verwenden.

Wie in Abbildung 3.1 zu erkennen ist, baut WSS auf vorhandenen Technologien wie C#, ASP.NET und auf die Windows Workflow Foundation (WF) auf. Es ist daher für Entwickler einfach, eine Applikation auf Basis von WSS

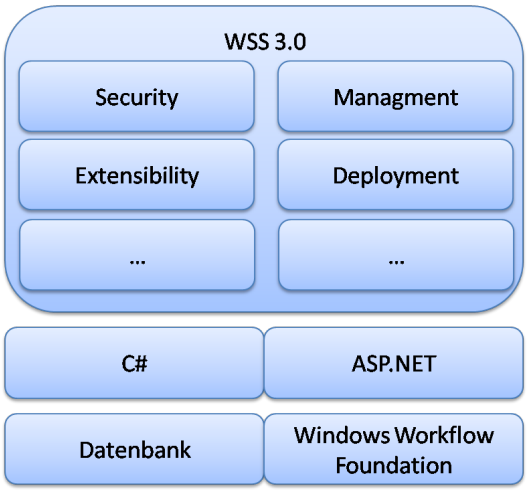

Abbildung 3.1: Aufbau WSS

zu entwickeln, da bereits ein umfangreiches Grundgerüst vorhanden ist und keine vollständige Applikation entwickelt werden muss. Ein Beispiel für die Verwendung von WSS ist MOSS 2007, welcher eine Applikation auf Basis von WSS ist.

Wie viele andere web-basierte Applikationen ist auch WSS eine Drei-Schichten-Applikation, die je nach den Bedürfnissen des entsprechenden Einsatzszenarios skaliert und erweitert werden kann. Die Schichten werden wie folgt aufgeteilt:

1. Die oberste Schicht wird oft als „Web Front End (WFE)“-Schicht bezeichnet und dient dazu, einem Benutzer Daten zur Verfügung zu stellen, Anfragen eines Benutzers entgegen zu nehmen und die Antwort wieder darzustellen. Die Darstellung der Daten geschieht im Falle von WSS meist mittels spezieller Web Parts, welche kleine Webseitenkomponenten darstellen die auf einer SharePoint Seite platziert werden können, oder dem Download eines Objektes aus der Dokumentenbibliothek.

2. Die mittlere Schicht beherbergt die Applikationslogik und ist dazu da, Anfragen zu verarbeiten und dynamischen Inhalt für die Webseite zu erzeugen. Beispiele für diese Schicht sind ein Such-Service oder die E-Mail Benachrichtigung bei Änderungen.

3. Die Datenschicht ist für die Speicherung und Verwaltung der eigentlichen Daten und Konfigurationsparameter zuständig. Alle Objekte innerhalb einer WSS Webseite, wie zB Dokumente, Kontakte oder Aufgaben werden durch diese Schicht in einer definierten Datenbank gespeichert.

(vgl. [Laa07], S. 14)

Um WSS verwenden zu können, müssen eine Reihe von Voraussetzungen erfüllt werden. Als Betriebssystem muss mindestens ein Windows Server 2003 mit SP1 (799,89€ Stand 02/2009 inklusive 10 Clientlizenzen) vorhanden sein. Eine Installation auf neueren Betriebssystemen wie Windows Server 2008 (1.352,29€ Stand 02/2009 inklusive 10 Clientlizenzen) ist ohne Einschränkungen möglich. Es ist allerdings zu beachten, dass bei Verwendung von Windows Server 2003 Web Edition nur eine Basis Installation von WSS möglich ist und kein SQL-Server als Datenbank verwendet werden kann.

Bei einer Basisinstallation von WSS wird automatisch eine „Windows-Internal-Database“ angelegt, welche aber oft nicht ausreichend ist. Es sollte deshalb bereits vor der Installation von WSS ein SQL Server installiert und konfiguriert werden. WSS unterstützt dabei folgende Datenbanken:

- SQL Server 2000 mit dem letzten SP (2.999€ Stand 02/2009)
- SQL Server 2005 mit SP 1 (5.799€ Stand 02/2009)
- SQL Server 2008 (6.284€ Stand 02/2009)

Da sowohl Konfiguration als auch Zugriff auf WSS großteils über einen Browser geschieht, muss am Server die Internet Information Services (IIS) Applikation ab Version 6.0 installiert werden. Falls man IIS 7.0 verwenden will, benötigt man dafür das SP1 für WSS. Will man E-Mail Funktionalitäten von WSS verwenden, muss zusätzlich SMTP am IIS konfiguriert werden.

Während der Installation von WSS werden vier Datenbanken angelegt, die unterschiedliche Daten der WSS-Installation speichern. Die wichtigste Datenbank ist die Content Datenbank welche mit dem Namen „WSS_CONTENT“ erzeugt wird. Diese enthält den gesamten Content aller Webseiten die in dieser WSS-Installation angelegt wurden. Auch hochgeladene Dateien werden hier abgelegt. Außer dem Content werden weiters noch definierte Benutzer,

Gruppen, Rechte und Listen abgelegt. Die anderen Datenbanken speichern vorgenommene Konfigurationen und Informationen die für das Such-Service benötigt werden.

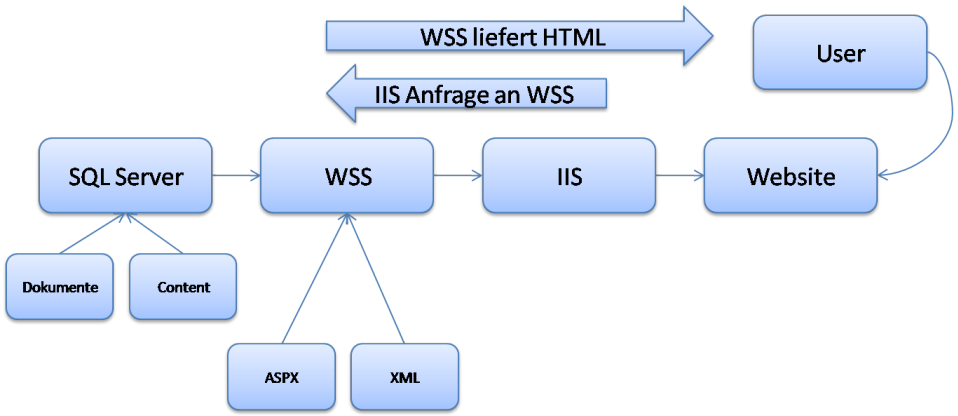

Abbildung 3.2: Zusammenspiel der einzelnen Komponenten

Abbildung 3.2 zeigt das Zusammenspiel der einzelnen Komponenten. Durch IIS werden Anfragen eines Benutzers an WSS weitergeleitet, der die benötigten Daten vom SQL Server erhält und das Ergebnis als HTML-Seite an den Benutzer zurückliefert.

Wie bereits erwähnt, enthält WSS eine Vielzahl an Features. Die nachfolgende Auflistung beschreibt kurz einige Kernfunktionalitäten, welche die Basis vieler WSS Webseiten bilden.

- Benachrichtigungen
  - Detaillierte Information über Änderungen: Wenn ein „überwachtes“ Objekt verändert wird, erhält man eine entsprechende Nachricht auf der Startseite der Applikation, mit der Möglichkeit genauere Informationen anzusehen (Änderungszeitpunkt, was wurde geändert, etc.).
  - E-Mail Benachrichtigungen: Zusätzlich zur normalen Benachrichtigung auf der Startseite kann man Benachrichtigungen so konfigurieren, dass man bei jeder Änderung eine E-Mail mit den entsprechenden Änderungen erhält.

  - RSS Feeds: Eine weitere Möglichkeit der Benachrichtigung bieten RSS Feeds. Man kann über einen Browser oder einen RSS Feed Reader jederzeit Informationen über neue oder veränderte Dokumente ansehen.

- Integration in Microsoft Office
  - Integration in Office 2007: Mittels Office 2007 ist es möglich, direkt von den einzelnen Komponenten der Office Produktreihe auf WSS zuzugreifen. Dateien können in WSS gespeichert / geladen, Workflows durchgeführt und Tasks zugewiesen werden.

- User Interface
  - Web Part Oberfläche: Die gesamte WSS Oberfläche basiert auf sogenannten Web Parts. Bei diesen handelt es sich um erweiterbare Sammlungen verschiedener Objekte wie Listen, Menüs, Datenquellen, welche zur Darstellung und Aufbereitung von Daten verwendet werden. Abbildung 3.3 zeigt eine WSS Startseite inklusive einiger eingefügter WebParts.

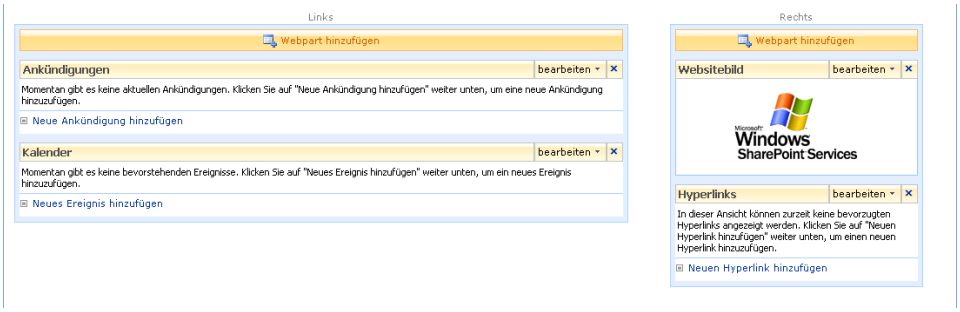

Abbildung 3.3: WSS Startseite mit WebParts

  - verschiedene Navigationsmenüs: WSS bietet eine Vielzahl vordefinierter Navigationsmenüs, um die Webseite individuell gestalten zu können.

- Speicherung und Security

- Dokumentenbibliotheken: Dokumentenbibliotheken dienen dazu, Objekte, wie zB Dokumente, Dateien oder Kontakte, in WSS zu speichern und zu verwalten. Dadurch wird es möglich, Dokumente „auszuchecken“ und zu sperren, um sicher zu gehen, dass keine Person das Dokument verändern kann.
- Verwaltung von Metadaten: Zu einem Objekt in der Dokumentenbibliothek lassen sich zusätzliche Metadaten definieren, welche Informationen über das Objekt wie Autor, Datum der letzten Änderung, Anschrift, etc. beinhalten können. WSS liefert eine Vielzahl bereits definierter Metadaten mit.
- Policies bis auf Dateiebene: Policies können in WSS bis auf Dateiebene definiert werden. Somit kann für jedes Element definiert werden, wer welche Rechte für dieses Objekt besitzt.
- Versionierung von Objekten: Seit WSS 3.0 ist es möglich, jedes Objekt in SharePoint zu versionieren. In vorherigen Versionen war dies nur für Objekte in einer Dokumentenbibliothek möglich. Nun hat man allerdings auch die Möglichkeit einen Task oder eine Diskussion zu versionieren um Änderungen zu verfolgen.

(vgl. [Mic08c])

3.2 CRM-relevante Features von WSS

3.2.1 Skalierbarkeit von WSS

Wie in Kapitel 2 erwähnt wurde, ist die Skalierbarkeit des CRM-Systems einer der Faktoren für die Bewertung von WSS. Die Erweiterung einer bestehenden Applikationsumgebung ist oft sehr umständlich, da zu klein dimensionierte WebServer oder Begrenzungen bei der Datenbank einen Flaschenhals in der Applikation darstellen können. Unter WSS 2.0 war mit dem Hinzufügen eines zusätzlichen WebServers ein erheblicher Mehraufwand verbunden, da alle Server einzeln administriert und konfiguriert werden mussten.

Mit WSS 3.0 führte Microsoft die sogenannte *Central Administration* ein, welche eine zentrale Konfigurationen der gesamten SharePoint Komponenten ermöglicht. Wird eine Änderung mittels dieser Administrationsseite durchgeführt, werden die Änderungen an alle konfigurierten SharePoint Server propagiert. Bei dieser Verknüpfung mehrere Web-Frontend-Server (WFE) spricht man von einer WFE-Farm, welche beliebig viele WFE beinhalten kann. Um dieses automatische verteilen von Änderungen einfach zu halten, speichert Microsoft alle WebServer-Einstellungen in den Datenbanken, von denen die WFE ihre Daten und Einstellungen beziehen. Durch diese Kollaboration haben alle Server der Farm dieselben Objekte der Applikation, ohne dass ein zusätzlicher Wartungsaufwand entsteht. Dies geschieht durch ein Service innerhalb der Central Administration, welches periodisch die Datenbanken auf Änderungen prüft und diese, falls vorhanden, an alle restlichen Server aussendet.

Um diese Aussagen zu verifizieren, wurde eine Teststellung aufgebaut, bei der zunächst nur ein WFE vorhanden war und insgesamt zwei weitere WFE mittels Central Administration hinzugefügt wurden. Nach dem Hinzufügen der zusätzlichen WFE, wurden einige Einstellungsänderungen vorgenommen und kontrolliert ob sich diese auf alle WFE verteilen. Dies gestaltete sich zunächst etwas schwierig, da das Verwenden der Central Administration anfänglich einige Zeit in Anspruch nimmt.

Die Speicherung von Nutzdaten wurde unter WSS 3.0 sehr großzügig dimensioniert. WSS legt diese Nutzdaten in der bereits erwähnten Content Datenbank ab. Da es sich hierbei um eine SQL-Server Datenbank handelt, sind der Dimensionierung dieser Datenbank nur physische Grenzen gesetzt. Mit WSS ist es möglich, mehrere hundert GB an Nutzdaten pro Server und innerhalb einer gesamten SharePoint Topologie mehrere TB an Daten zu verwalten.
(vgl. [O'C07], S. 80-81)

Pav Cherny rät in einem Artikel des Technet Magazins zu folgendem Grundaufbau, um ein skalierbares WSS-System zu erreichen:

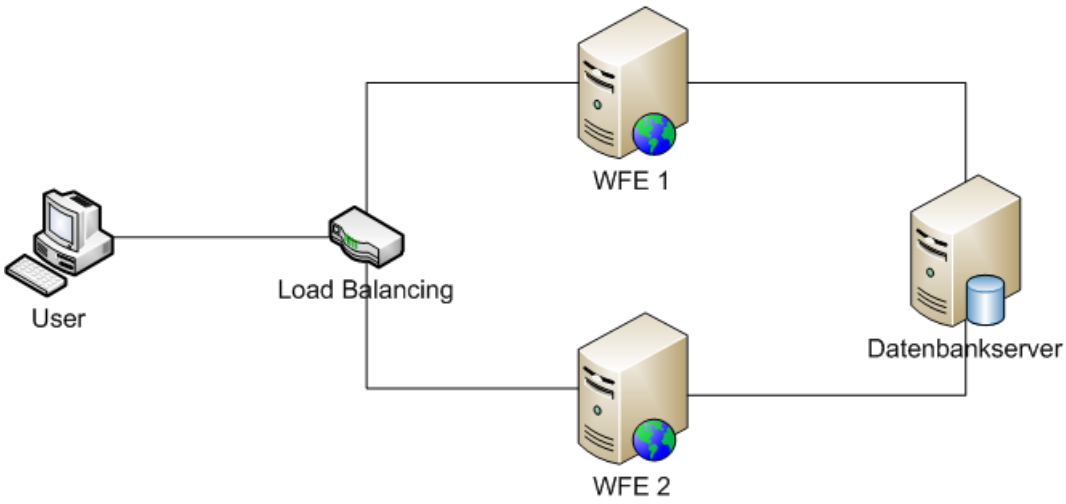

Abbildung 3.4: Grundaufbau eines skalierbaren WSS-Systems

Dabei werden zwei WFE Server eingerichtet, auf denen mittels Load Balancing die Anfragen auf die vorhandenen Webseiten aufgeteilt werden. Die Datenbank wird auf einen eigenen Server ausgelagert, um einen höheren Grad an Skalierbarkeit aus Datenbanksicht zu gewährleisten, da neue Datenbankserver einfach hinzufügt werden können. Zusätzlich wird die Auslastung der WFE reduziert, wodurch Anfragen schneller abgearbeitet werden können. Mit diesem Grundaufbau können sowohl WebServer als auch Datenbankserver leicht erweitert werden.
(vgl. [Mic08a])

3.2.2 Workflows

Bei Workflows handelt es sich um Business Prozesse die an ein beliebiges Objekt in WSS angefügt werden. Diese Business Prozesse können je nach Komplexität der Anforderung sehr simpel sein oder komplexe Abläufe im Hintergrund erledigen. Handelt es sich um komplexe Abläufe, ist es oft besser einen Workflow in mehrere aufzuteilen, anstatt einen einzelnen großen Workflow zu

entwickeln. Dadurch wird einerseits die Fehlersuche erleichtert, als auch das Umändern oder Austauschen bestimmter Workflowkomponenten.

Wie bereits in Kapitel 3.1 zu WSS erwähnt, baut WSS unter anderem auf die Windows Workflow Foundation (WF) auf. Dadurch ist es in WSS möglich, Dokumenten oder Listen einen oder mehrere Workflows zu hinterlegen, die automatisch bei eintreten eines bestimmten Events oder von einem Benutzer gestartet werden, falls dieser die benötigten WSS-Rechte (Edit-Permission) besitzt. Dabei können mehrere Workflows, die dasselbe Objekt betreffen, parallel ablaufen. Es ist allerdings nicht möglich, denselben Workflow mehrere Male an das gleiche Objekt zu binden. Während der Abarbeitung eines Workflows ist es möglich, dass dieser zusätzliche Informationen oder Genehmigungen bei anderen Benutzern einholt und diese Informationen den Workflowablauf beeinflussen.

Workflows sind ein mächtiges Instrument und erleichtern den Alltag eines Benutzers erheblich. Oft muss ein Dokument von mehreren Personen nacheinander bearbeitet oder von gewissen Personen genehmigt werden. Durch die Verwendung von Workflows werden diese Aufgaben vereinfacht und auf Prozessebene normalisiert.

Als Nachteil von WSS gegenüber MOSS 2007 kann die Anzahl der im Produktumfang enthaltenen Workflows gesehen werden. Während bei der Installation von MOSS 2007 eine Vielzahl oft verwendeter Workflows bereits mitinstalliert werden, enthält WSS nur einen einzigen vorgefertigten Workflow. Dabei handelt es sich um den sogenannten „Issue Tracking Workflow“, welcher zur Erfassung von Problemen benutzt werden kann. Jedoch gibt es aufgrund der großen WSS Community eine Vielzahl von Open-Source Workflows, wovon viele auf http://www.codeplex.com zum Download angeboten werden.

3.2.2.1 Windows Workflow Foundation

Die Windows Workflow Foundation ist die Workflow Plattform von Microsoft und bietet eine Workflow Laufzeitumgebung, sowie eine Vielzahl an Tools um Workflows zu entwickeln. Die WF ist Teil des .NET Frameworks 3.0 und

zusätzlich in den Betriebssystemen Windows Vista und Windows Server 2008 fix integriert.

Teil der WF ist die Base Activity Library (BAL), welche eine Sammlung verschiedener Activities ist, die in einem Workflow verwendet werden können. Darunter sind auch gängige Programmierfunktionen wie Schleifen und Bedingungsoperatoren um mit Workflows auch komplexere Geschäftsprozesse abbilden zu können. Falls eine Activity benötigt wird die nicht in der WF enthalten ist, kann man diese auch selbst entwickeln.

Die folgende Aufzählung gibt einen Überblick über die wichtigsten Activites der BAL:

- IfElseActivity: Bei dieser Activity handelt es sich um eine If-Else-Condition um eine Entscheidung aufgrund der vorhergegangen Activity oder eines definierten Wertes zu treffen.
- WhileActivity: Eine Activity um ein oder mehrere Activities solange auszuführen, bis eine bestimmte Kondition eintritt. Diese Activity wird oft bei Genehmigungsworkflows verwendet und läuft solange, bis die Empfängerperson das Dokument genehmigt.
- TerminateActivity: Diese Activity beendet den aktuellen Workflow sofort und wird meist verwendet, wenn der Workflow einen ungültigen Status erlangt hat.
- CodeActivity: Mithilfe der CodeActivity kann selbstgeschriebener C#-Code während eines Workflows ausgeführt werden.
- SequenceActivity und ParallelActivity: Bei der SequenceActivity handelt es sich um eine Zusammenfassung mehrere anderer Activites, welche nacheinander ausgeführt werden. Die ParallelActivity erlaubt es, mehrere SequenceActivities gleichzeitig nebeneinander abzuarbeiten.

Diese Activities bieten ein gutes Grundgerüst um einfache aber auch komplexe Workflows zu modellieren. Zusätzlich zu den oben genannten existieren noch einige weitere Activities, welche aber nicht näher besprochen werden, da sie für die Erstellung des CRM-Systems nicht relevant sind.

Entwickelt man Workflows mit Visual Studio enthält die Activity-Toolbox auch einige Activities die in der BAL vorhanden sind, aber in WSS nicht unterstützt werden. (vgl. [Bru07], S. 179-182)

Abschließend zeigt Abbildung 3.5 die Verbindung zwischen der WF und einem Workflow.

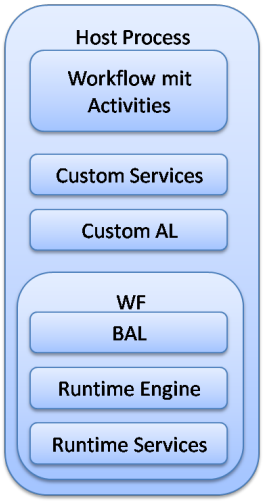

Abbildung 3.5: Verbindung der WF mit einem Workflow

3.2.2.2 Arten von Workflows

Die WF unterstützt zwei unterschiedliche Arten von Workflows, den sequentiellen Workflow und die State Machine. Bei einem sequentiellen Workflow handelt es sich um einen Workflow der einen definierten Start und ein definiertes Ende besitzt. Während der Abarbeitung eines sequentiellen Workflows können zwar ParallelActivities und IfElseActivies auftreten, jedoch müssen diese am Ende zu einem Endpunkt zusammengeführt werden. Es ist nicht möglich, mehrere Endpunkte zu definieren. Ein typisches Beispiel eines sequentiellen Workflows ist die Genehmigung eines Dokumentes durch einen Vorgesetzten.

Abbildung 3.6 zeigt die exemplarische Umsetzung eines Genehmigungsworkflows. Die Umsetzung des Workflows zeigt allerdings nur eine einfache Version dieses Workflows. Bei der IfElseActivity „Vorgesetzter_sieht_sich_Dokument_an“

wartet der Workflow solange, bis sich der entsprechende Benutzer das Dokument angesehen und eine Entscheidung getroffen hat. Man spricht hierbei von einer User Action während alle anderen Activities als System Action bezeichnet werden.

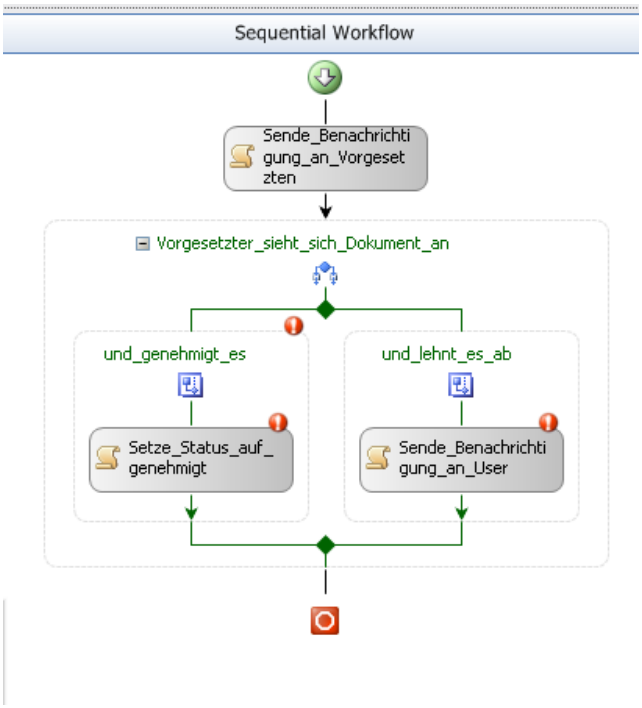

Abbildung 3.6: Beispiel eines sequentiellen Workflows

Im Gegensatz zu den sequentiellen Workflows hat ein State Machine Workflow keine bestimmte Abarbeitungsfolge und muss auch kein definiertes Ende aufweisen. Ein State Machine Workflow verhält sich wie eine herkömmliche State Machine. Er beinhaltet definierte Zustände, wobei der erste Zustand immer der Start-Zustand ist. Diese Zustände können auf gewisse Ereignisse reagieren und aufgrund eines solchen Ereignisses in einen anderen definierten Zustand wechseln. Hat man ein Ende definiert, endet der Workflow bei Erreichen des Ende- Zustandes. Ist kein Ende definiert, läuft der Workflow ewig und erhält in WSS auch nie den Status beendet. Ein Beispiel für einen State Machine Workflow ist ein Shop mit Warenkorb. Da jeder Benutzer eine Webseite unterschiedlich benutzt, muss man hier einen State Machine Workflow verwenden, um auf bestimmte Ereignisse, wie das Absenden einer Bestellung reagieren zu können. Abbildung 3.7 zeigt die exemplarische Umsetzung eines State Ma-

chine Workflows. Wie zu erkennen ist, sind State Machine Workflows meist um ein vielfaches komplexer als sequentielle Workflows und benötigen in der Planung und Entwicklung mehr Zeit.

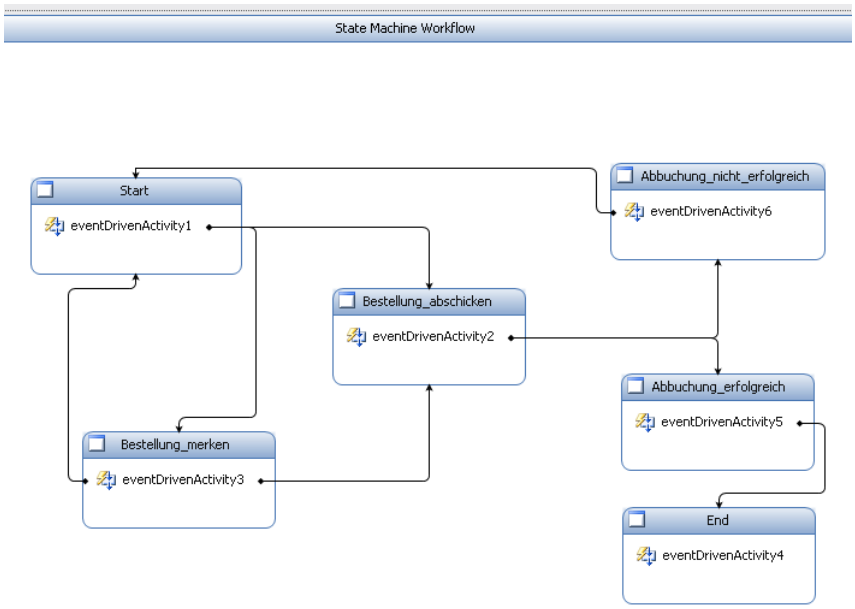

Abbildung 3.7: Beispiel eines State Machine Workflows

3.2.3 Bereitstellung von Daten mittels WebServices

Da im Falle des CRM-Systems Benutzer nicht direkt auf eine Webseite zugreifen, sondern Daten über Outlook abrufen und ändern, müssen diese von WSS bereitgestellt werden. Da es mittels Silverlight möglich ist Daten per WebService abzurufen, werden die Daten per WebService von WSS zur Verfügung gestellt.

Dabei müssen nicht alle WebServices selbst geschrieben werden. Während der WSS-Installation werden bereits 20 WebServices installiert, die viele Standardfälle wie das Downloaden oder Aktualisieren eines Objektes abdecken. Alle WebServices werden unter
C:\Program Files\Common Files\MicrosoftShared\

web server extensions\12\ISAPI abgelegt. Lediglich das Adminstration WebService wird unter ADMISAPI gespeichert. MOSS 2007 bietet zwar erneut eine umfangreichere Sammlung an WebServices, welche allerdings nur für MOSS-spezifische Features verwendet werden können. Es entsteht daher kein Nachteil oder Mehraufwand wenn man nur WSS einsetzt.

Request- und Responseobjekte eines WebServices enthalten oft sogenannte Globally Unique Identifiers (GUIDs). Dabei handelt es sich um einen eindeutigen 32 Zeichen langen Schlüssel, der ein Objekt innerhalb von WSS definiert. Man kann zwar auch den intern definierten Namen eines Objektes verwenden, jedoch empfiehlt es sich immer mit GUIDs zu arbeiten, da sich diese nicht ändern, während der Name jederzeit umgeändert werden kann.

Da manche vorhandenen oder selbst programmierten WebServices komplexe Request und Response Formate aufweisen, werden im nachfolgenden einige Elemente, welche in den meisten WebServices vorhanden sind, beschrieben.

Request Parameter

- *ListName*: Der Listenname, von welchem Informationen abgerufen werden sollen. Dabei kann entweder eine GUID übergeben werden, oder der interne Name der Liste.
- *ViewFields*: Mittels ViewFields kann eine Einschränkung der zurückgegebenen Felder vorgenommen werden, um nur die Informationen zu erhalten, die man benötigt. Folgendes Beispiel liefert als Response nur die Id und die Telefonnummer von gefundenen Einträgen zurück.

```
1  <ViewFields>
2    <FieldRef Name="ID">
3    <FieldRef Name="Phone">
4  </ViewFields>
```

Listing 3.1: Beispiel ViewFields

- *Query*: Im Query Parameter kann definiert werden, welche Suchquery an das WebService abgesendet wird. Dabei wird das sogenannte CAML (Collaborative Application Markup Language) Format verwendet. Bei CAML handelt es sich um eine XML Abfragesprache, die in WSS und

MOSS für verschiedene Zwecke benutzt wird. Mithilfe von CAML ist es möglich, komplexe Suchqueries an das WebService abzusenden.

```
<Query>
  <Where>
    <And>
      <Lt>
        <FieldRef Name="Lieferdatum" />
        <Value Type="DateTime"><Today /></Value>
      </Lt>
      <IsNull>
        <FieldRef Name="Erledigt" />
      </IsNull>
    </And>
  </Where>
</Query>
```

Listing 3.2: Beispielquery mit CAML

Listing 3.2 zeigt eine einfache Abfrage in CAML-Syntax. Es werden alle Objekte gesucht, bei denen das Lieferdatum kleiner als das Aktuelle ist und das Feld „Erledigt“ noch nicht gesetzt wurde.

Response Parameter

- *Xml*: Alle gefundenen Objekte werden als Child-Objekte von Xml zurückgegeben. Die vorhandenen WebServices geben dabei keine typischen SharePoint Objekte, wie zB ein Listenelement, zurück sondern speichern die einzelnen Eigenschaften des Objektes in den Properties eines <row> Elements. Es ist also notwendig, die benötigten Felder programmatisch in die entsprechenden Objektklassen zu mappen. Die Typen der einzelnen Properties werden über das Property XmlDataSchema definiert.

(vgl. [Pyl07], S. 647-648)

Da es während der Erstellung des CRM-Systems zu Implementierungsfällen kommen kann, an denen mit den Standardwebservices keine Lösung gefunden werden kann, wird im folgenden kurz auf die Erstellung eigener WSS WebServices eingegangen. Um ein WebService in WSS verfügbar zu machen, sind

nach der Erstellung der gewünschten Funktionalität noch einige zusätzliche Schritte durchzuführen. Da WSS seine URLs virtualisiert, können die automatisch generierten Disco[1] und WSDL[2] Dateien, welche für Beschreibungen und Auffinden von WebServices verantwortlich sind, nicht verwendet werden. Um passende Dateien zu erstellen, kann das WebService temporär gehostet und dann, dem in ASP.NET enthaltenen Programm *disco.exe* die URL des WebServices übergeben werden. Im Anschluss müssen noch einige Anpassungen in den erzeugten Dateien vorgenommen werden, damit diese WSS kompatibel werden. Zunächst muss in beiden erstellten Dateien der Anfangs-XML-Tag durch folgende Zeilen ersetzt werden, um die notwendigen SharePoint DLLs zu registrieren:

```
1 <%@ Page Language="C#" Inherits="System.Web.UI.Page" %>
2 <%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture
      =neutral, PublicKeyToken=71e9bce111e9429c" %>
3 <%@ Import Namespace="Microsoft.SharePoint.Utilities" %>
4 <%@ Import Namespace="Microsoft.SharePoint" %>
5 <% Response.ContentType = "text/xml"; %>
```

Listing 3.3: Registrierung der benötigten SharePoint DLLs

Danach müssen beide Dateinamen auf folgendes Format umgeändert werden: *name*disco.aspx und *name*wsdl.aspx, damit diese von WSS erkannt werden. *Name* muss dabei mit dem Namen des WebServices ersetzt werden. Zuletzt müssen die beiden aspx und die asmx Datei in das Verzeichnis
C:\Program Files\Common Files\MicrosoftShared\
web server extensions\12\ISAPI kopiert werden. Durch diesen Kopiervorgang wird das entwickelte WebService aktiviert.

3.3 Bewertung

WSS 3.0 bildet mit seinem umfangreichen Funktionsumfang eine gute Basis für die Entwicklung eines CRM-Systems. Nutzdaten wie Kunden oder Supportdokumente können in WSS abgelegt (oder aus anderen Datenquellen integriert),

[1] Discovery of Web Services ist ein von IBM entwickeltes Protokoll zur Bekanntmachung von WebServices
[2] Web Service Definition Language ist eine Beschreibungssprache für WebServices

verwaltet und versioniert werden. Dadurch haben sämtliche Benutzer zu jeder Zeit einen identischen Blick auf die vorhandenen Daten. Durch die Integration von WebServices ist es für andere Applikationen sehr einfach, die gespeicherten Daten abzufragen, bestehende Daten zu ändern oder neue Daten in WSS abzulegen.
Die Verwendung von Workflows ermöglicht es der CRM-Applikation, komplexe Geschäftsfälle und -prozesse dem Benutzer in einer einfachen Form zur Verfügung zu stellen und durch den Ablauf zu leiten.
Auch die Anforderung der Skalierbarkeit erfüllt WSS. Die seit 3.0 vorhandene Zentraladministration erleichtert die Verwaltung einer WFE-Farm erheblich und bringt Zeitvorteile gegenüber anderen Systemen.

Diesen Vorteilen steht ein hoher Initialaufwand gegenüber. Da es sich bei WSS um eine Applikationssuite mit vielen verschiedenen Features handelt, ist es notwendig zunächst einen Überblick über die verschiedenen Einstellungsmöglichkeiten zu erlangen und diese in Teststellungen auszuprobieren. Nachdem diese Einarbeitungsphase abgeschlossen wurde, sind Konfigurationsänderungen sowie Neuinstallationen schnell durchführbar. Ein weiterer Nachteil entsteht durch die Voraussetzungen, die für eine Installation von WSS notwendig sind. Dadurch können, je nach fehlenden Komponenten, Kosten im Bereich von ca. 10.000€ entstehen, welche laut den definierten Rahmenbedingungen möglichst gering gehalten werden sollen.
Da alle anderen Rahmenbedingungen jedoch erfüllt und in manchen Punkten sogar übertroffen werden, ist WSS 3.0 als Serverkomponente geeignet.

Kapitel 4

Clientseitige Anbindung an Outlook durch VSTO 2008

4.1 Allgemeines

VSTO (Visual Studio Tools for Office Systems 2008) ist ein Toolset für Visual Studio, welches die Entwicklung von Office-Erweiterungen mit VB.NET und C# ermöglicht. Hervorzuheben ist, dass im Vergleich zu seinem inoffiziellen Vorgänger VBA (Visual Basic for Applications), welcher früher und auch heute noch zur Entwicklung von Office-Erweiterungen verwendet wird, VSTO auf dem .NET Framework basiert und somit alle Features des Frameworks verwendet werden können. VBA hingegen weist nur ein begrenztes Set an Funktionalitäten auf und seit Veröffentlichung von Office XP im Jahre 2001 wurden keine Erweiterungen zu VBA mehr veröffentlicht. VSTO 2008 baut weiters auf seinem Vorgänger VSTO 2005 auf, wodurch die Office 2003 Unterstützung auch in VSTO 2008 noch vorhanden ist, die neuen Features allerdings ausschließlich in Office 2007 verwendbar sind.

Um das Objekt Modell von VSTO 2008, welches in Kapitel 4.3 beschrieben wird erklären zu können, ist es zunächst notwendig einige allgemeine Features zu beschreiben, auf denen VSTO 2008 aufbaut. Es werden allerdings nur Features beschrieben, welche direkte Auswirkungen auf die Programmierung von Outlook PlugIns haben, da andere Office Produkte nicht Teil dieser Arbeit sind.

Application-Level AddIns

Unter Application-Level AddIns versteht man Klassenbibliotheken (Assemblies) welche mit einer Office Applikation wie Outlook verbunden werden. Diese Bibliothek läuft dabei im Kontext der Office Applikation und ist nicht an ein einzelnes Dokument (z.B. eine E-Mail Nachricht) gebunden. Innerhalb dieser AddIns hat man Zugriff auf das komplette .NET Framework als auch auf die zusätzlichen Office Komponenten welche durch VSTO bereitgestellt werden. Nachdem ein installiertes AddIn geladen wurde (dies geschieht bei **händischen** Start der Applikation), reagiert es je nach Funktion auf auftretende Events oder erweitert die Applikation um neue Funktionalitäten. Im Falle des CRM-Systems wäre die Darstellung von Kunden, welche in WSS gespeichert werden, eine solche Erweiterung.
(vgl. [Mic09b])

Registry Entries

Neben der Klassenbibliothek selbst sind Registry Entries notwendig, um einem Office Produkt mitzuteilen, welche AddIns geladen werden sollen. Dazu wird bei der Installation ein Registry Key unter
HKEY_CURRENT_USER\Software\Microsoft\Office*applicationname*\Addins*AddInID*
angelegt welcher 4 Werte beinhaltet, die das Verhalten des AddIns regeln. Die Eigenschaften der einzelnen Werte können unter der unten angeführten URL nachgelesen werden.
(vgl. [Mic09d])

Application & Deployment Manifest

Diese beiden Manifestdateien dienen dazu, die dazugehörigen Klassenbibliothek und den richtigen Einstiegspunkt in das AddIn zu finden. Das Deployment Manifest referenziert das Application Manifest der derzeit aktuellen Version und das Application Manifest verweist wiederum auf den Einstiegspunkt des AddIns.
Abschließend zeigt Bild 4.1 die Verbindung der einzelnen Komponenten.

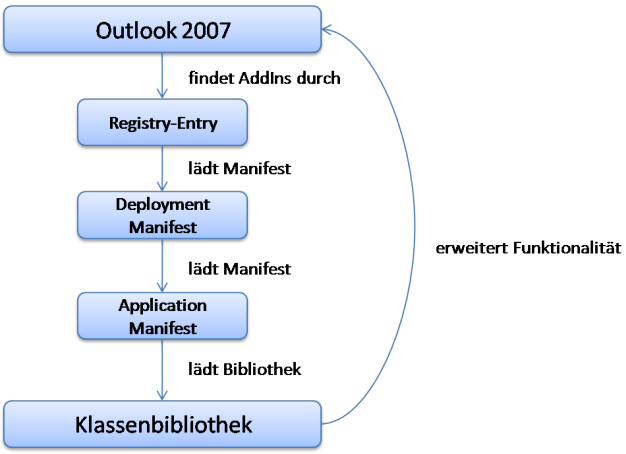

Abbildung 4.1: Verbindung der VSTO Komponenten

Outlook Form Regions

Unter Outlook Form Regions versteht man selbst entwickelte Form Regionen, welche dazu dienen, zusätzliche Funktionalität in bereits vorhandene Regionen einzubetten oder vorhandene Regionen zu ersetzen. Diese Form Regionen können in jede Komponente des Outlook Object Models (siehe Kapitel 4.3) eingefügt werden. Es gibt allerdings eine Beschränkung von 30 Regionen die zu jeder Outlook Form hinzugefügt werden können. Dies ist für die Anforderungen des CRM-Systems ausreichend, da versucht werden soll auf Basisfunktionalitäten von Outlook 2007 zurückzugreifen und nur dann zusätzliche Funktionalität implementiert wird wenn die Basisfunktionalitäten nicht ausreichend sind. Das Ersetzen vorhandener Regionen ist erst seit VSTO 2008 Bestandteil von VSTO.[1]
(vgl. [And08], S. 4-5)

[1] Zuvor war dies nur durch umständliche Methoden möglich, da alle Funktionalitäten nachgebildet werden mussten

4.2 Microsoft Outlook 2007 als Grundlage des CRM-Systems

Outlook 2007 soll bei der Erstellung des CRM-Systems als Grundgerüst für den Client dienen. Wie bereits in Kapitel 2.1 beschrieben, wird VSTO verwendet, um Outlook um die benötigten Schnittstellen zu erweitern. Aufgrund der Wiederverwendung von Outlook 2007 kann auch auf bereits vorhandene Benutzererfahrung zurückgegriffen werden, was einen zusätzlichen Vorteil gegenüber eigenständigen CRM-Produkten bietet.

Um das CRM-System anzusprechen, soll ein zusätzlicher Ordner, welcher sich auf der selben Ebene wie Inbox, Sent Items, etc. befindet, eingefügt werden. Der Screenshot in Abbildung 4.2 illustriert dies.

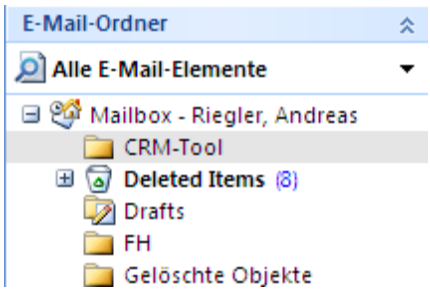

Abbildung 4.2: Ordner für Zugriff auf CRM-System

Bei Auswahl des Ordners „CRM-Tool“ soll im mittleren Bereich von Outlook die Startseite des CRM-Systems angezeigt werden und somit dem Benutzer eine Navigation durch das CRM-System innerhalb des Outlook-Clients ermöglichen.

Zusätzlich zu diesem Ordner sollen Informationen über Kontakte direkt über den Contacts-Reiter in Outlook abgefragt werden können. Dazu wird eine neue Schaltfläche im FluentUI (siehe Kapitel 4.3.1) eingefügt, welcher in der Detailansicht eines Kontaktes hinzugefügt wird. Bei Klick auf diese Schaltfläche sollen die entsprechenden Informationen aus dem CRM-System in einem neuen Fenster angezeigt werden. Dadurch kann ein Benutzer direkt in die Subseite des CRM-Systems einsteigen, was eine Steigerung der Usability bewirkt.

4.3 Das Outlook Object Model

Das Outlook Object Model (OOM) unterscheidet sich von anderen VSTO Object Modellen, da Daten in einer Hierarchie von Ordnern gespeichert werden (siehe Abbildung 4.3), welche unterschiedliche Objekte beinhalten. Da diese Struktur stellenweise sehr undurchsichtig ist und das Ändern eines Objektes oft durch „try and error“ erreicht wird, werden in den folgenden Kapiteln die einzelnen Objekte beschrieben.

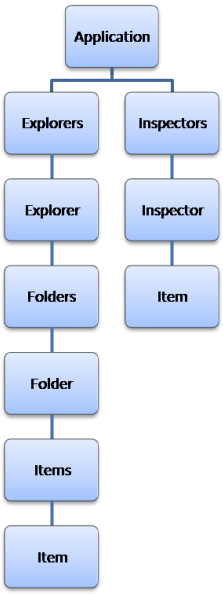

Abbildung 4.3: Das Outlook Object Model

Abbildung 4.3 zeigt schematisch den Aufbau und Beziehungen innerhalb des OOM. Bei den Elementen Explorers, Folders, Items und Inspectors handelt es sich um Collections, die die Gesamtheit aller zugehörigen Objekte einer Outlook-Instanz beinhalten und verwalten. Abbildung 4.4 zeigt, welchen Verwendungszweck die einzelnen Komponenten des OOM in Outlook 2007 haben.

4.3.1 Komponenten des OOM

Fluent & Ribbon

Mit Office 2007 führte Microsoft das sogenannte Fluent UI (siehe Abbildung

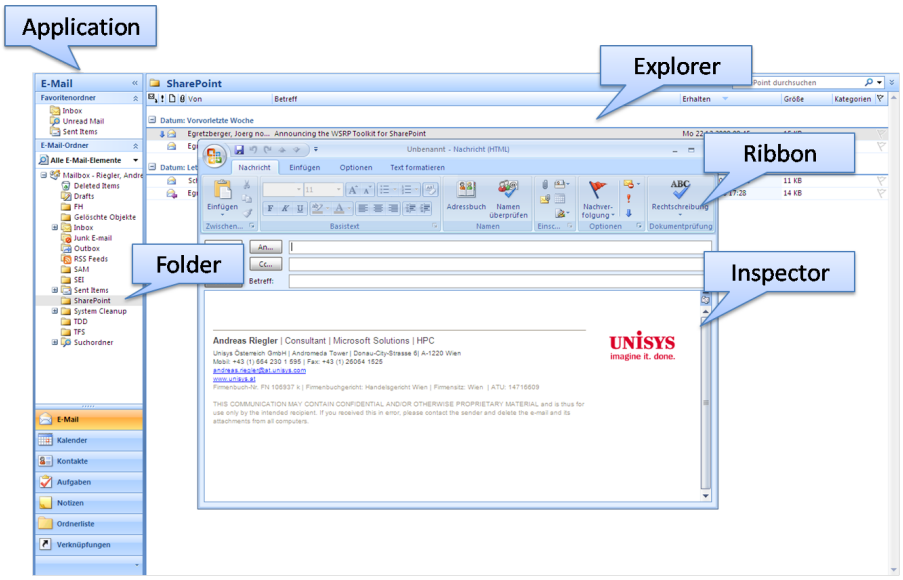

Abbildung 4.4: Verwendungszweck der OOM - Komponenten

4.5) in viele Office Komponenten ein. Dabei handelt es sich um eine Neuentwicklung bzw. Neuanordnung der bisherigen Menüleisten. Eine Studie der Forrester Group, welche im Auftrag von Microsoft durchgeführt wurde zeigte, dass sich die Effektivität der Benutzer durch Verwendung des neuen FluentUI erhöht hatte. Allerdings wird in dieser Studie auch angemerkt, dass die Umstellung des User Interfaces eine erhebliche Änderung der Benutzererfahrung mit sich brachte, da sich das Fluent UI vom bisherigen Menü sehr stark unterscheidet. (vgl. [Gro07], S.15)

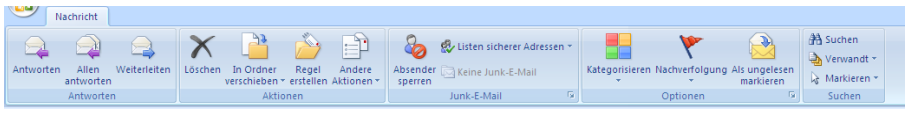

Abbildung 4.5: Fluent UI aus Outlook 2007

Application Object

Das Application Object stellt das Grundobjekt einer Outlook 2007 Applikation dar. Mithilfe dieses Objektes ist es möglich, sämtliche Inhalte der Applikation zu verändern oder auszutauschen. Es ist möglich auf Explorer und Inspector Objekte zuzugreifen und diese in einem Application Kontext zu erzeugen. Weiters wird das Application Object dazu verwendet, neue Objekte wie E-Mails oder Kontakte innerhalb einer laufenden Outlook Instanz zu erzeugen ohne

den gesamten Objektpfad (wie in Abbildung 4.3) abarbeiten zu müssen. Das Application Object dient somit zur zentrale Verwaltung und hat keine spezifischen Aufgaben wie andere Objekte des OOM. Listing 4.1 zeigt die Erzeugung eines Application Objects und das Anlegen eines Kontaktes durch das Application Object. Somit können neue Kontakte innerhalb des CRM-Systems ohne Aufwand für den Benutzer erstellt werden.

```
1 Outlook.Application app = Globals.ThisAddIn.Application;
2 Outlook.ContactItem newContact = app.CreateItem(Outlook.OlItemType
      .olContactItem) as Outlook.ContactItem;
3 newContact.NickName = "Mit VSTO erstellter Kontakt";
4 newContact.Save();
```

Listing 4.1: Erzeugen eines Application Objects

In Zeile 1 wird auf das Objekt *Globals.ThisAddIn* zugegriffen, welches das laufende AddIn repräsentiert. Es handelt sich dabei um eine automatisch angelegte globale Variable, welche sich in der statischen Klasse *Globals* befindet. Somit kann man von jeder Stelle des AddIns auf diese Variable zugreifen und diese verwenden. Danach wird mittels *CreateItem* ein Kontaktobjekt angelegt. Dazu wird der Funktion ein Wert des Enums *OlItemType* übergeben, um VSTO mitzuteilen um welche Objektart es sich handelt. Damit dieses Objekt dauerhaft in Outlook gespeichert wird, muss ein Aufruf der *Save*-Funktion durchgeführt werden.
(vgl. [And08], S. 28-30)

Folder Object

Ein Folder Object ist vom Inhalt mit einem Windows Dateiordner zu vergleichen. Innerhalb eines solchen Objektes können sich Outlook Dateien sowie weitere Folder Objekte befinden. Mithilfe des Folder Objektes kann auf diese Inhalte zugegriffen werden und Funktionen wie das Kopieren oder Löschen einer Datei / eines Ordners durchgeführt werden.

Outlook beinhaltet mehrere Default Folder, welche für jeden Benutzer angelegt werden und in der *Folders* Collection beinhaltet sind. Zu diesen Default Foldern gehören Ordner wie *Inbox*, *Sent Items* oder *Drafts*. Bei allen Default Foldern handelt es sich um Instanzen der bereits angesprochenen Folder Klasse. Der Zugriff auf diese Ordner erfolgt über *Application.Session.GetDefaultFolder*

und wird im folgenden Beispiel veranschaulicht indem zunächst der Posteingang-Ordner geöffnet wird und die Anzahl der ungelesenen Nachrichten per MessageBox ausgegeben werden.

```
1 Outlook.Folder folder = app.Session.GetDefaultFolder(
2       Outlook.OlDefaultFolders.olFolderInbox) as Outlook.Folder;
3 MessageBox.Show(folder.UnReadItemCount.ToString());
```

Listing 4.2: Erzeugen und Verwenden eines Folder Objects

(vgl. [And08], S. 94-98)

Explorer Object

Das Explorer Object definiert den „zentralen“ Bereich von Outlook, in dem die Übersicht der enthaltenen Objekte eines Folder Objekts dargestellt werden. Outlook kann dabei mehrere Explorer Objekte gleichzeitig verwalten, wobei immer nur eines aktiv angezeigt wird, während die anderen bis zu ihrer Löschung (durch entfernen des Objektes) im Speicher existieren oder aktiviert werden. Dieses Objekt besitzt Funktionen um den gegenwärtigen Folder , die Ansicht oder den Filter zu ändern. Es wird also gesteuert, welche Inhalte und Informationen der Inhalte dargestellt werden.

```
1 Outlook.Explorer explorer = app.ActiveExplorer();
2 explorer.CurrentFolder = folder;
3 explorer.ShowPane( Outlook.OlPane.olPreview, false );
```

Listing 4.3: Aktives Explorer Objekt modifizieren

Listing 4.3 zeigt eine exemplarische Verwendung des Explorer Objects. Dabei wird auf Variablen der vorherigen Listings zugegriffen um zunächst das aktive Explorer Objekt zu erhalten und den angezeigten Folder auf das Objekt der InBox umzusetzen. In Zeile 3 wird das Vorschau-Fenster von Nachrichten oder Aufgaben ausgeblendet. Diese Funktion wird innerhalb des CRM-Systems des öfteren angewendet werden, um Inhalte nicht dauerhaft anzuzeigen, sondern diese nach Eintreten eines Events anzuzeigen oder wieder zu schließen.
(vgl. [And08], S. 83-87)

Inspector Object

Ein einzelnes Objekt innerhalb von Outlook (zB ein Kontakt, eine Aufgabe

oder eine E-Mail) wird durch das Inspector Object repräsentiert. Dabei handelt es sich um ein eigenständiges Fenster, welches innerhalb der Applikation geöffnet wird und Detailinformationen zu diesem Objekt anzeigt. Da es möglich ist, mehrere Inspector Objekte gleichzeitig anzuzeigen, werden diese über die Inspectors Collection verwaltet, um auf alle Objekte einen gezielten Zugriff durchführen zu können. Das Inspector Object kann sowohl die Darstellung der Daten verändern als auch das zugrunde liegende Objekt selbst. Um eine Referenz eines vorhandenen Inspector Objektes zu erhalten, kann einerseits das Application Object verwendet werden oder über das anzuzeigende Objekt mittels *GetInspector*-Methode eine Referenz erzeugt werden. Da die Verwendung ähnlich der eines Explorer Objektes ist, wird an dieser Stelle auf ein Code-Beispiel verzichtet.
Für die Implementierung des CRM-Systems wird das Inspector Objekt zur genauen Darstellung und Manipulation der Daten mittels Silverlight verwendet werden.
(vgl. [And08], S. 89-91)

Namespace Object
Das Namespace Objekt beinhaltet Methoden und Eigenschaften um auf das Messaging Application Programming Interface (MAPI) zuzugreifen. MAPI ist die einzige Möglichkeit auf Daten, welche in Outlook verwaltet werden, einzuwirken. Dieses Objekt bietet eine Vielzahl an Funktionen und Eigenschaften, jedoch werden nur einige wenige für die Entwicklung des CRM-Prototypen benötigt.

Bei den benötigten Funktionen handelt es sich einerseits um die Abfrage des angemeldeten Outlook Benutzers, um diese Informationen auf die entsprechenden SharePoint Berechtigungen anwenden zu können. Dadurch wird gewährleistet, dass ein Benutzer nur die Funktionalitäten benutzen kann, für die er auch die entsprechenden Benutzerrechte besitzt. Da sowohl WSS als auch Outlook auf dieselben Active Directory Benutzer zugreifen, ist kein zusätzlicher Verwaltungs- bzw. Synchronisationsaufwand notwendig.

Die zweite benötigte Funktionalität ist die Überprüfung des Verbindungsstatus von Outlook mittels *Offline()* Funktion oder der *ExchangeConnectionMode* Eigenschaft des Namespace Objektes. Ersteres liefert bei einer aktiven Ver-

bindung *true* und *false* falls keine Verbindung vorhanden ist. Benötigt man genauere Informationen über den Status der Verbindung, sollte man die *ExchangeConnectionMode* Eigenschaft abfragen, welche folgende Stati besitzen kann:

- *olConnected*: Outlook ist mit dem Exchange Server verbunden, aber es wurde der Offline-Betrieb in Outlook aktiviert.
- *olDisconnected*: Outlook ist nicht mit dem Exchange Server verbunden, aber der Online-Betrieb aktiviert.
- *olOnline*: Outlook ist mit dem Exchange Server verbunden und der Online-Betrieb aktiviert.
- *olOffline*: Outlook ist nicht mit dem Exchange Server verbunden und der Offline-Betrieb aktiviert.

(vgl. [And08], S. 80)

Das CRM-System soll nur ausgeführt werden können, wenn der Status *olOnline* als Rückgabewert erhalten wurde. Dadurch wird gewährleistet, dass der Benutzer erfolgreich gegenüber dem Exchange Server authentifiziert wurde und keine weiteren Authentifizierungsmaßnahmen getroffen werden müssen. Der Status *olConnected* ist nicht ausreichend, da sich Outlook bei diesem Status im Offline-Betrieb befindet und ein solcher Betrieb für das CRM-System in einer ersten Version nicht vorgesehen ist.

4.3.2 Einbettung von Silverlight in Outlook

Die Einbettung von Silverlight in Outlook 2007 gestaltet sich durch die umfangreichen Funktionalitäten von VSTO sehr einfach. Um eine Webapplikation wie das in Silverlight entwickelte CRM-System in Outlook darzustellen, muss ein neues Folder Object angelegt werden und zwei Eigenschaften dieses Objektes gesetzt werden. Danach wird bei Auswählen des entsprechenden Ordners die definierte Webapplikation im Explorer Object geladen.

```
Outlook.Folder inbox = app.Session.GetDefaultFolder(
    Outlook.OlDefaultFolders.olFolderInbox ) as Outlook.Folder;
Outlook.Folder parent = null;
if( inbox != null && inbox.Parent is Outlook.Folder )
{
    parent = inbox.Parent as Outlook.Folder;
    if( parent != null )
    {
        Outlook.Folder crm = parent.Folders.Add( "CRM-Tool",
            System.Reflection.Missing.Value )
                            as Outlook.Folder;
        crm.WebViewURL = "http://silverlight";
        crm.WebViewOn = true;
    }
}
```

Listing 4.4: Darstellung einer Webapplikation in Outlook

Da nicht direkt auf den Hauptordner zugegriffen werden kann, muss zunächst ein vorhandener DefaultOrdner und im Anschluss dessen Parent Objekt instanziert werden. Mittels *Folders.Add()* kann danach ein neuer Folder hinzugefügt werden. Der Konstruktor erwartet dabei den Namen sowie den Typ des Folders. Um den Default-Wert auszuwählen muss *System.Reflection.Missing.Value* verwendet werden, da es sich bei dieser Methode um einen Wrapper eines COM-Aufrufes handelt, welcher *null* als Parameter nicht akzeptiert.

Um die oben angesprochene Einschränkung (Anzeige nur bei Status olOnline) zu erfüllen, muss ein Event-Receiver programmiert werden, welcher auf einen Folderwechsel reagiert und eine Überprüfung durchführt, ob der Zielfolder „CRM-Tool“ ist.

```
Outlook.Explorer explorer = app.ActiveExplorer();
explorer.BeforeFolderSwitch += new ExplorerEvents_10_
    BeforeFolderSwitchEventHandler( BeforeFolderSwitch );

private void BeforeFolderSwitch( object newFolder, ref bool cancel
     )
{
    Outlook.Folder folder = newFolder as Outlook.Folder;
    if( folder.Name.Equals( "CRM-Tool" ) )
    {
```

```
9         if( app.Session.ExchangeConnectionMode != Outlook.
              OlExchangeConnectionMode.olOnline )
10        {
11            MessageBox.Show( "Outlook befindet sich nicht im
                  Online-Modus. CRM-System kann nicht angezeigt
                  werden!",
12                             "CRM-System Fehler" );
13            folder.WebViewOn = false;
14        }
15        else
16            folder.WebViewOn = true;
17    }
18 }
```

Listing 4.5: EventReceiver in VSTO

Zunächst muss der Eventhandler am *BeforeFolderSwitch*-Event registriert werden (Zeile 1 & 2). Wird nun der Folder „CRM-Tool“ ausgewählt, wird überprüft ob Outlook den Status olOnline besitzt. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben und die Webapplikation nicht angezeigt. Bei obigen Code ist zu beachten, dass es sich hierbei nur um einen Prototypen handelt und für die CRM-Implementierung im else-Zweig des Codes mehr Aktionen getätigt werden müssten.

4.4 Deployment von VSTO-Applikationen mittels ClickOnce

4.4.1 Allgemeines

ClickOnce ist eine Microsoft Technologie die mit .NET 2.0 eingeführt wurde und den Deploymentvorgang von auf Windows basierten Programmen vereinfacht. Durch ClickOnce wurden einige Probleme des herkömmlichen Microsoft Windows Installers beseitigt. Dieser benötigte zur Installation Administratorrechte, welche gerade in Firmenumgebungen sehr selten an jeden Benutzer vergeben wurden. Bei einer ClickOnce Installation werden diese Rechte nicht mehr benötigt und die Applikation erhält nur die Rechte, die sie benötigt.

Dabei kann eine Applikation über mehrere Wege von einem Benutzer installiert werden, welche im Kapitel 4.4.2 genauer erörtert werden. Weiters wurde das Problem der Abhängigkeiten von Applikationen behoben. Das „Installationspaket“ enthält alle benötigten Referenzen und beeinflusst durch seine Installation keine bereits vorhandene Software. Auch der Updatemechanismus wurde erheblich vereinfacht und bietet nun die Möglichkeit, bei fehlerhaft eingespielten Updates ein Rollback auf eine frührere Version durchzuführen oder gewisse Updates als zwingend zu markieren. VSTO unterstützt die volle Funktionalität von ClickOnce und Microsoft empfiehlt die Verwendung von ClickOnce bei einem Deployment von VSTO Software.

4.4.2 Publishing und Installation einer Applikation

Wie bereits erwähnt, bietet ClickOnce mehrere Möglichkeiten die Applikation den Benutzern zur Verfügung zu stellen. Ein Installationspaket kann dabei durch folgende Möglichkeiten installiert werden:

- Webseite
- freigegebenes Netzlaufwerk
- CD / DVD

Um eine ClickOnce Installation vorzubereiten, muss ein „Deployment Package“ in Visual Studio erzeugt werden, welches das zu installierende Programm beinhaltet. Dazu kann entweder der Publish Wizard verwendet werden, oder die entsprechenden Optionen direkt am Projekt eingestellt werden. Die zweite Möglichkeit erlaubt die Konfiguration von mehr Optionen als der Publish Wizard. Im nachfolgenden wird die Erstellung eines Deployment Packages über die Projektoptionen erklärt, da der Publish Wizard sehr einfach gehalten wurde und keiner zusätzlichen Erklärung bedarf.

Abbildung 4.6 zeigt die Publishing Maske einer fertigen VSTO Applikation. Der Publishing Folder gibt an, an welchen Ort das Package kopiert werden soll. In diesem Falle wurde ein Netzwerkshare gewählt. Soll die Applikation mittels einer Webseite verfügbar gemacht werden, ist hier die entsprechendeURL einzutragen.

Um mögliche Abhängigkeiten automatisch mitzuinstallieren, kann mittels der Prerequisites Schaltfläche ausgewählt werden, welche Abhängigkeiten für dieses Projekt existieren. Eine weitere wichtige Option ist die „Publish Version“ welche angibt, um welche Version es sich bei diesem Publish-Vorgang handelt. Dies ist vor allem für Rollbacks notwendig, da dort eine Versionsnummer angegeben werden muss. Nachdem das Deployment Package erzeugt wurde, kann

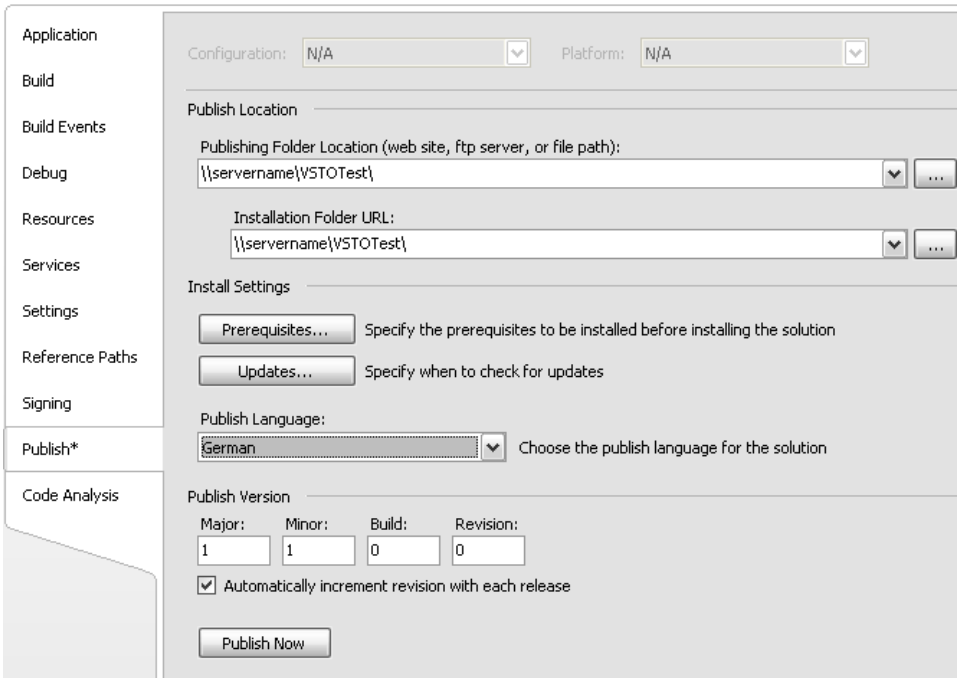

Abbildung 4.6: Publishing Maske einer VSTO Applikation

die Applikation durch ausführen der erzeugten *setup.exe* von jedem Benutzer der Zugriff auf diese Datei hat installiert werden.

4.4.3 Update einer bestehenden Applikation

Bei einer herkömmlichen Installation mittels Windows Installer musste für ein Update oft die komplette Applikation neu installiert werden und dem Benutzer händisch mitgeteilt werden, dass ein Update seiner Applikation vorhanden ist. Deshalb wurde bei ClickOnce darauf geachtet, automatische Updates einzuführen, welche keine Neuinstallation zur Folge haben. Dabei werden auch nur die Dateien vom Installationsordner heruntergeladen, die auch wirklich geändert

wurden, was eine Reduktion der benötigten Datenmenge bedeutet. Bei Erstellung des Deployment Packages, kann das Updateverhalten der Applikation angegeben werden. Es ist möglich während dem Start der Applikation, nach dem Start der Applikation oder mittels User Interface eine Updateüberprüfung durchzuführen. Weiters kann bei den ersten beiden Varianten festgelegt werden, in welchem Intervall eine Überprüfung stattfinden soll, um nicht bei jedem Start auf eine neue Version zu überprüfen. Falls nach Start der Applikation ein Update gefunden wurde, wird das Update erst beim nächsten Start der Applikation heruntergeladen und installiert.

Zur Überprüfung auf ein neues Update wird die sogenannte Deployment Manifest Datei überprüft. Liegt eine neuere Version der Datei vor, wird das entsprechende Update oder Rollback durchgeführt. Sollte das Update ein höheres Security-Level als die Vorgängerversion benötigen, wird aus Gründen der Sicherheit der Benutzer gefragt, ob die zusätzlichen Rechte gewährt werden sollen. Lehnt der Benutzer dies ab, wird die alte Version weiterverwendet. Handelt es sich jedoch um ein zwingendes Update, kann die Applikation nicht mehr ausgeführt werden, bis der Benutzer die benötigten Rechte akzeptiert.

4.5 Bewertung

VSTO 2008 eignet sich aufgrund seiner einfachen Integration in Outlook 2007 sehr gut zur Erstellung eines Wrappers für das CRM-System. Zusätzlich zur einfachen und vor allem schnellen Entwicklung (es entsteht kaum zusätzlicher Mehraufwand bei der Erlernung von VSTO) profitieren die Benutzer von vorhandener Benutzererfahrung. Dadurch fällt die Lernphase des CRM-Systems kürzer aus, als wenn sich der Benutzer zuerst in ein neues Programm einarbeiten müsste.

Durch die Verwendung von VSTO kann weiters der aktuelle Benutzer und der Verbindungsstatus ausgelesen werden. Somit ist keine zusätzliche Authentifizierung gegenüber dem CRM-System notwendig, da Benutzerinformationen und der Verbindungsstatus direkt aus Outlook ausgelesen werden und somit die passenden Informationen sofort angezeigt werden.

Die Verteilung der erstellten Applikation gestaltet sich durch die Verwendung von ClickOnce sehr einfach, da bei Aufrufen der Applikation automatisch nach Updates gesucht wird und die Applikation gegebenenfalls aktualisiert wird. Dadurch entsteht weder für den Administrator noch für den Benutzer ein zusätzlicher Aufwand und es wird sichergestellt, dass alle Benutzer mit derselben aktuellen Version arbeiten.

Kapitel 5

Clientseitige Visualisierung der Daten mittels Silverlight

5.1 Allgemeines

Zur Visualisierung der Daten innerhalb des CRM-Systems wird Version 2.0 von Silverlight verwendet, da der Funktionsumfang von Version 1.0 nicht ausreichend ist und Version 3.0 noch nicht genug Stabilität für eine Implementierung aufweist. Silverlight ist eine Microsoft Technologie, die 2007 in der Version 1.0 veröffentlicht wurde. Es handelt sich dabei um ein Framework, welches zur Entwicklung von Rich-Internet-Applications (RIA), welche sowohl Browser- als auch Plattformunabhängigkeit aufweisen, entworfen wurde. Dazu ist ein Browser-PlugIn notwendig, welches bereits für Windows und Mac OS X verfügbar ist. An einer Portierung für Linux wird bereits durch Novell[1] gearbeitet. Durch dieses PlugIn ist es möglich, auf einen größeren Umfang von Funktionen zuzugreifen, als es bisher mit Mitteln wie HTML und JavaScript (JS) möglich ist.

Im Oktober 2008 wurde von Microsoft Silverlight 2.0 veröffentlicht, welche eine Vielzahl an neuen Features zur Verfügung stellte. Viele dieser neuen Features erleichtern die Erstellung des CRM-Systems erheblich und steigern die Sicherheit der Applikation durch Einführung von Securityfunktionalitäten. Zusätzlich wurde in Version 2.0 die Verwendung einer Untermenge des .NET Frameworks und eine Untermenge von WPF zur UI-Modellierung ermöglicht.

[1] http://www.mono-project.com/Moonlight

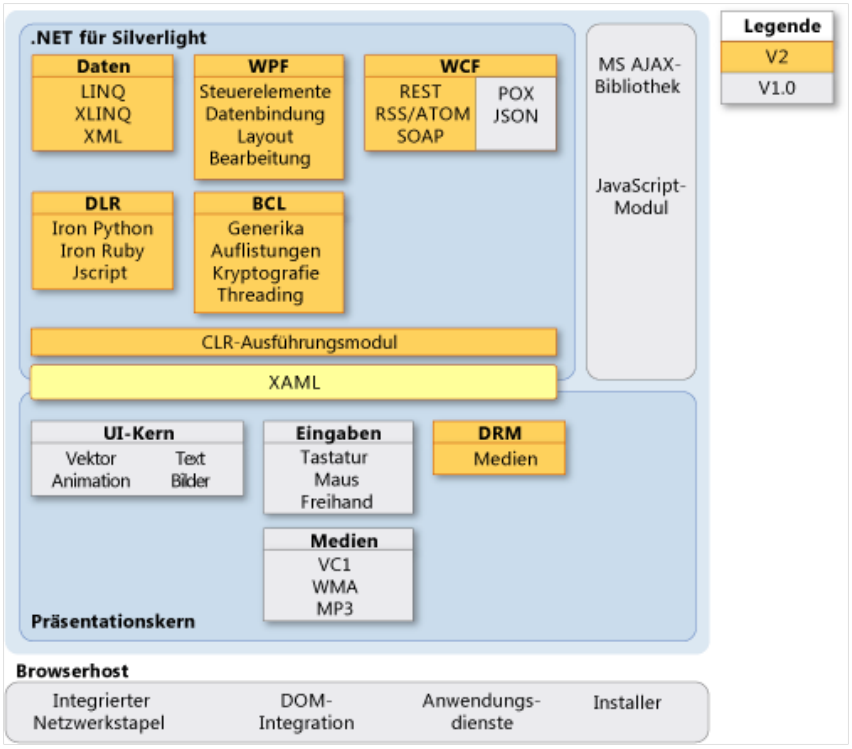

Abbildung 5.1: Unterschiede Silverlight 1.0 und 2.0
http://msdn.microsoft.com/en-us/library/bb404713(VS.95).aspx

(vgl. [Mac08], S. XXV ff.)
Abbildung 5.1 zeigt die Unterschiede zwischen Version 1.0 und 2.0.

Durch die enge Verbindung von Silverlight mit dem .NET-Framework ist es möglich, umfangreiche browserbasierte Applikationen zu erstellen. Jedoch ergeben sich dadurch auch einige Beschränkungen, wovon die wichtigsten im Folgenden kurz besprochen werden.

Cross-Domain Connectivity: Um die Sicherheit von Silverlight- und anderen Webapplikationen zu gewährleisten, ist es standardmäßig nicht möglich, Cross-Domain Verbindungen (Zugriff auf Web-Ressourcen einer anderen Domäne) von einer Silverlightapplikation zu initiieren. Sollte die Silverlightkomponente des CRM-Systems also auf einem anderen WebServer als WSS installiert werden, wäre ein Zugriff auf WSS standardmäßig nicht möglich. Da ein generelles Verbot Silverlight beim Abruf von Informationen jedoch stark

einschränken würde, wurde zusätzlich ein Sicherheitskonzept ähnlich dem von Adobe Flash entwickelt. Dazu werden sogenannte Policy Dateien verwendet. Die Verwendung und Konfiguration einer solchen Policy Datei wird im Kapitel 5.3 besprochen.

Drucken von Inhalten: Silverlight 2.0 ermöglicht es noch nicht, Seiten direkt an einen lokalen oder Netzwerkdrucker zu senden. Da die Entwicklung von v3.0 zum Zeitpunkt der Erstellung dieser Arbeit noch nicht weit genug fortgeschritten war, um eine Evaluierung durchzuführen, kann keine Aussage getroffen werden, ob diese Beschränkung in v3.0 bereits aufgehoben wurde. Diese Beschränkung hat auch direkte Auswirkungen auf das CRM-System. Oftmals werden abgerufene Informationen vom Benutzer ausgedruckt, um sie bei Terminen und Besprechungen vorweisen zu können. Es besteht also derzeit keine Möglichkeit, die mit Silverlight abgerufenen Informationen auszudrucken außer händisch einen Screenshot der Silverlight Daten zu machen und diesen mit einem Drittprogramm zuzuschneiden und auszudrucken.

Zugriff auf lokale Speichermedien: Da eine Silverlightapplikation innerhalb des Browsers in einer Sandbox, d.h. einem abgesicherten Ausführungsbereich, läuft kann unter anderem nicht ohne zusätzlichen Aufwand auf lokale Speichermedien wie Festplatten oder einen USB-Stick zugegriffen (lesend sowie schreibend) werden. Silverlight stellt jedoch den sogenannten *Isolated Storage* zur Verfügung, um Daten permanent auf einem physischen Medium zu speichern. Der Isolated Storage belegt bei Anforderung standardmäßig 1MB Speicherplatz auf der Festplatte, welcher per Request noch erweitert werden kann. Abgespeicherte Daten sind jedoch nur von der Silverlightapplikation abrufbar und können nicht von anderen Applikationen oder dem Benutzer ausgelesen werden. Der Zugriff auf den Isolated Storage erfolgt übr die *IsolatedStorageFile* Klasse, welche Methoden zum Erzeugen, Löschen und Ändern von Dateien und Ordnern beinhaltet. Weiters können im Isolated Storage Konfigurationsdaten der Applikation abgelegt werden und diese mittels Hilfsklasse gespeichert und gelesen werden. Damit ist es aber noch nicht möglich, bereits vorhandene Dateien mittels Silverlight zu verarbeiten, da der Isolated Storage keinen Zugriff auf das Dateisystem ermöglicht. Da Silverlight jedoch ein Subset des .NET Frameworks beinhaltet, ist es durch die Verwendung der *OpenFileDialog* Klas-

se möglich, lesend auf das Dateisystem zuzugreifen. Dazu muss der Benutzer die zu öffnende Datei per Windows File Dialog auswählen und bestätigen. (vgl. [Mac08], S. 493 ff.)

Wie in Kapitel 2.2 bereits angesprochen, wird Silverlight zur grafischen Darstellung der in WSS gespeicherten Informationen verwendet. Deshalb werden in den folgenden Unterkapiteln diejenigen Komponenten behandelt, die für die Erstellung der Silverlightapplikation notwendig sind.

5.2 XAML

Für die Entwicklung von Silverlightapplikationen ist es notwendig, die Extensible Application Markup Language (XAML) zu verstehen. Es ist zwar möglich, ein Silverlightprogramm nur mit der Entwicklungsumgebung Expression Blend 2.0[2] zu schreiben und somit keinen eigenen XAML-Code erstellen zu müssen, jedoch kann man dadurch nicht alle Möglichkeiten und Feinheiten von Silverlight ausnützen.

Unter XAML versteht man eine Markup Sprache die XML-basiert ist und zur Erstellung von grafischen Oberflächen in Silverlight verwendet wird. Dazu werden alle vorhandenen Silverlightobjekte als XAML-Elemente (mit identischem Namen) und den entsprechenden Attributen zur Verfügung gestellt, was das Erstellen einer grafischen Oberfläche stark vereinfacht und auch lesbarer macht. Weiters besitzt jedes XAML-Dokument zusätzlich eine Code Behind Klasse, um zusätzliche Funktionalitäten auszuprogrammieren und verwenden zu können.

Um einen Einblick in XAML zu erhalten, zeigt Listing 5.1 eine leere Silverlightseite welche mit Visual Studio erzeugt wurde.

```
1 <UserControl x:Class="SilverlightApplication1.Page"
2            xmlns="http://schemas.microsoft.com/client/2007"
3              xmlns:x="http://schemas.microsoft.com/winfx/2006/
                   xaml"
4              Width="400" Height="300">
```

[2] http://www.microsoft.com/expression/products/Overview.aspx?key=blend

```
5
6     <Grid x:Name="LayoutRoot" Background="White"/>
7  </UserControl>
```

Listing 5.1: XAML-Aufbau

Das erste Element eines XAML-Dokumentes ist immer *UserControl* und definiert die verwendeten Namespaces, die zugehörige C#-Klasse sowie Höhe und Breite der Silverlightapplikation. Weiters dient das *UserControl*-Element als Container für alle weiteren Elemente. Durch die Verwendung eines *Grid*- oder *Canvas*-Elements können Elemente innerhalb einer Seite je nach gewähltem Element positioniert werden. Erst durch diese Einbindung und die Verwendung zusätzlicher Elemente wie Textboxen und Diagrammen wird eine grafische Visualisierung von Daten durchgeführt.

Wie in Listing 5.1 ersichtlich ist, werden von Visual Studio zwei Namespaces im *UserControl*-Element definiert. Der erste Namespace „http://schemas.microsoft.com/client/2007“ ist der Default-Namespace und muss von jeder Silverlightapplikation eingebunden werden. Er beinhaltet alle weiteren Elemente wie das verwendete *Grid*-Element. Der zweite Namespace deklariert XAML spezifische Erweiterungen, die als Elemente oder Attribute verwendet werden können. In Zeile 6 des Listings wird das Attribut *x:Name* verwendet, welches einem Element einen eindeutigen Namen zuweist. Unter diesem Namen ist das Element von der zugehörigen C#-Klasse ansprechbar. (vgl. [Sca08], S. 17f)

Funktionalitäten oder eigene Silverlightelemente werden oft in Assemblies ausgelagert. Um diese Funktionalitäten verwenden zu können, muss die entsprechende Assembly-Datei im *UserControl*-Element deklariert werden. Dies geschieht über das Attribut xmlns:*Bezeichner*, der folgenden Wert erhält: xmlns:*Bezeichner*="clr-namespace:*Namespacename*;assembly=*Assemblyname*" Danach sind die Funktionalitäten über *Bezeichner*:*Funktionsbezeichnung* verwendbar.
Um mit einer Silverlightapplikation funktionale Operationen durchführen zu können, ist eine XAML-Datei nicht ausreichend. Wie bereits erwähnt, besitzt jede XAML-Datei eine Code-Behind C#-Klasse, welche zur Implementierung der gewünschten Funktionalität verwendet wird. Für das CRM-System wird

die Code-Behind Klasse Funktionen beinhalten, welche Web-Services ansprechen um Kundendaten zu erhalten und diese mittels Datenbindung (siehe Kapitel 5.4) darzustellen. Des weiteren übernimmt die Code-Behind Klasse das Handling und die Steuerung von Events, um auf den Input des Benutzers reagieren zu können. Das Handling von Events wird in diesem Kapitel nicht behandelt, da es dem Handling normaler .NET Applikationen gleicht und bis auf spezielle Events keine Unterschiede vorhanden sind.

5.3 Security Model

5.3.1 Zugriff auf Ressourcen

Normale .NET Applikationen unterliegen der sogenannten Code Access Security (CAS), welche die Modifikation lokaler Ressourcen und Kommunikation über einen Netzwerkkanal regelt, um Applikationen keinen uneingeschränkten Zugriff auf diese Ressourcen zu gewähren. Dazu muss die Common Language Runtime (CLR) zunächst überprüfen, ob ein Zugriff auf die benötigte Ressource möglich ist. Falls dies nicht der Fall ist, darf der entsprechende Code nicht ausgeführt werden.
Für Silverlightapplikationen wurde ein anderes Security Modell gewählt, welches restriktiver ist als die CAS. In Silverlight ist der gesamte vom Entwickler geschriebene Code als „security transparent“ eingestuft. Das bedeutet, dass diesem Code aus Sicht der CLR nicht vertraut wird. Um Zugriff auf benötigte Ressourcen zu erlangen, wurde ein dreistufiges Konzept entwickelt.

1. Der entwickelte Code sowie Assemblies von Drittanbietern, welche im Code verwendet werden, haben selbst keinen Zugriff auf Ressourcen die der ausführende Host zur Verfügung stellt. Um Zugriff zu erlangen, muss die Anfrage an die zweite Ebene weitergeleitet werden, welche zur Erfüllung des Security Modells eingeführt wurde. Ein Beispiel für diese Ebene wäre eine Funktion, welche Daten in den Isolated Storage speichern möchte und dazu die passende Funktion aus der folgenden Schicht aufrufen muss.

2. Der Silverlight Plattformcode, welche die mittlere Schicht bildet, dient zur Weiterleitung der Codeanfragen an den Plattformcode der Hostapplikation. Somit wird sichergestellt, dass Silverlightapplikationen keinen direkten Zugriff auf diesen Plattformcode haben. Diese Schicht stellt unter anderem Funktionalitäten wie den Isolated Storage der Applikation zur Verfügung und weist bereits einen „full trust“ auf. Diese Schicht würde im Falle des Zugriffs auf den Isolated Storage eine Funktion der untersten Schicht aufrufen, welche den tatsächlichen Speichervorgang durchführt. Die entsprechende Funktion ist von der darüberliegenden Ebene nicht erreichbar, da diese einen zu geringen Securitystatus aufweist.

3. Der Plattformcode bildet die unterste Ebene im dreistufigen Konzept. Hier wird der tatsächliche Zugriff auf die entsprechenden Ressourcen des Hosts durchgeführt. Welche Ressourcen angesprochen werden und die dazugehörigen Operationen werden dem Plattformcode von der zweiten Schicht übergeben. In dieser Schicht würde der tatsächliche Dateizugriff durchgeführt werden und die Datei mit dem übergebenen Inhalt erzeugt werden.

(vgl. [Sca08], S. 229 ff.)

5.3.2 Verschlüsselung der Applikationsdaten

Die Kommunikation zwischen dem ausführenden Hostsystem und der Silverlightapplikation wird über einen sicheren Kanal durchgeführt. Daten die temporär im Isolated Storage abgelegt werden, können jedoch theoretisch von jedem Benutzer ausgelesen und verändert werden. Es muss also sichergestellt werden, dass die lokal gespeicherten Daten vor unauthorisiertem Zugriff geschützt werden und die Datenintegrität dieser Daten zu jedem Zeitpunkt gewährleistet wird.
Um diese Punkte erfüllen zu können, können die abgelegten Daten einerseits mittels Encryption verschlüsselt und durch Verwendung eines Hash-based message authentication codes (HMAC) auf Gültigkeit verifiziert werden. Silverlight 2.0 integriert dabei einige bekannte Verfahren, welche in Version 1.0 noch gänzlich fehlten.

Um bei der Verschlüsselung der Daten keinen Algorithmus zu verwenden, der mit heutigen Mitteln sehr schnell entschlüsselt werden kann, stellt Silverlight nur den *Advanced Encryption Standard (AES)*[3] zur Verfügung, welcher zur Gruppe der symmetrischen Verschlüsselungsmethoden gehört und den Data Encryption Standard (DES) als Standardverschlüsselungsmethode ablöste. Für eine Implementierung werden die Klassen *AesManaged*, welche für die Definition der benötigten Parameter benutzt wird, und *CryptoStream* um die gewünschten Daten zu verschlüsseln, verwendet.
Falls Kundendaten vom Benutzer lokal abgespeichert werden, ist es sehr wichtig die Datenintegrität zu gewährleisten, da manipulierte Daten ansonsten an den Server gesendet werden könnten. In Silverlight sind dazu die gängigen Hashfunktionen SHA-1 und SHA-256 implementiert. Beide Versionen erwarten einen geheimen Schlüssel um aus Kombination mit den eigentlichen Daten eine eindeutige Prüfsumme berechnen zu können. Durch jede Änderung an den Daten würde eine andere Prüfsumme errechnet werden. Durch den Einsatz einer Hashfunktion kann die Korruption von Daten also gänzlich ausgeschlossen werden.

5.3.3 Policy File

Um mit einer Silverlightapplikation keine Denial Of Service (DoS) oder Cross-Side Scripting Attacken durchführen zu können, wurde mit Silverlight 2.0 ein Sicherheitssystem eingeführt, welches den Zugriff auf fremde Domänen verhindert. Da es für diese Fremddomänen aber möglich sein sollte, Silverlightapplikationen den Zugriff explizit zu gewähren, wurde das Policy File System entwickelt. Damit kann festgelegt werden, ob domänenfremde Applikationen Zugriff auf angebotene Dienste wie WebServices erhalten.
Zur Steuerung des Zugriffes wird eine *clientaccesspolicy.xml* Datei am Zielsystem benötigt. Bevor ein tatsächlicher Zugriff von Silverlight auf den entsprechenden Server durchgeführt wird, versucht Silverlight diese Datei vom Server zu laden. Kann diese Datei nicht gefunden werden, verbietet Silverlight automatisch jegliche Kommunikation mit diesem Server, bis die Applikation geschlossen und neu gestartet wird. Konnte die Datei vom Server geladen

[3] AES unterstützt nur Schlüssel mit einer Länge von 128, 192 oder 256 Bit

werden, wird überprüft, ob der Inhalt valide ist und die entsprechenden Eigenschaften für einen Zugriff gesetzt wurden. Ist dies der Fall, können ab diesem Zeitpunkt Anfragen an den Server gestellt werden und neue Anfragen an den Subdomänen abgesetzt werden, falls diese im Policy File spezifiziert wurden. Im folgenden wird ein exemplarischer Aufbau der *clientaccesspolicy.xml* Datei gezeigt, welche den Zugriff auf Ressourcen des Systems ermöglicht.

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="http://mysilverlighturi"/>
      </allow-from>
      <grant-to>
        <resource path="/cms" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Listing 5.2: Beispiel einer clientaccesspolicy.xml Datei

Das Beispiel aus Listing 5.2 ermöglicht jeder Silverlightapplikation welche unter
http://mysilverlighturi gehostet wird, den Zugriff auf das virtuelle Verzeichnis /cms und alle Unterordner dieses Verzeichnisses. Ein Zugriff auf einen anderen Ordner wird jedoch nicht gestattet.
Mit Hilfe einer clientaccesspolicy Datei können Freigaben auf Ressourcen sehr genau und einfach spezifiert werden und es entfällt die Überprüfung innerhalb der angesprochenen Ressource.
(vgl. [Mic09c])

5.4 Datenanbindung

5.4.1 Ansprechen von WebServices

Ein wichtiger Punkt bei der Erstellung des CMS-Systems ist der Zugriff auf serverseitig gespeicherte Daten. Wie bereits in Kapitel 3 erwähnt, stellt WSS 3.0 eine Vielzahl an WebServices zur Verfügung, welche Zugriff auf unterschiedliche Komponenten von WSS ermöglichen. Deshalb ist es naheliegend, innerhalb der Silverlightapplikation die entsprechenden WebServices anzusprechen um die gespeicherten Daten grafisch darstellen zu können.
Das Ansprechen von vorhandenen WebServices gestaltet sich in Silverlight sehr einfach. Zunächst muss in Visual Studio eine Web Reference auf das gewünschte WebService angelegt werden. Dadurch erstellt Visual Studio automatisch eine passende Proxy Klasse, welche den Code für das Ansprechen der einzelnen Funktionen des WebServices beinhaltet. Der Klassenname entspricht dabei dem WebServicenamen, um bei mehreren referenzierten WebServices das richtige Objekt ansprechen zu können. Während der Definition der Web Reference ist darauf zu achten, dass Silverlight nur asynchrone Aufrufe unterstützt, damit keine Verzögerungen bei der Ausführung der Applikation auftreten.
Nachdem die Referenz angelegt wurde, kann innerhalb der Applikation ein Objekt der Proxy Klasse angelegt und das entsprechende Service aufgerufen werden. Weiters muss ein Eventhandler registriert werden, welcher die Antwort des WebServices abarbeitet und das Ergebnis an die entsprechenden Funktion zur Visualisierung der erhaltenen Daten weiterleitet. Listing 5.3 zeigt den Aufruf eines WebServices und die Registrierung eines neuen Eventhandlers.
(vgl. [Mac08], S. 427 f.)

```
1 private void GetListItem(Guid id)
2 {
3   ListItemClient proxy = new ListItemClient();
4   proxy.GetListItemCompleted += new
5     EventHandler<GetListItemCompletedEventArgs>(
          GetListItemCompleted);
6
7   proxy.GetListItem(id);
```

```
8  }
9
10 private void GetListItemCompleted(object sender,
11   GetListItemCompletedEventArgs args)
12 {
13   ...
14 }
```

Listing 5.3: Ansprechen eines WebServices mit Silverlight

5.4.2 Koppelung der Daten

Silverlight bietet verschiedene Möglichkeiten Daten an das User Interface (UI) zu koppeln. Das sogenannte Databinding unterscheidet dabei folgende Arten:

- **OneTime**: Bei OneTime Binding werden Nutzdaten nur einmal an die entsprechenden Komponenten gebunden. Updates der Nutzdaten werden nicht automatisch an das UI repliziert.
- **OneWay**: OneWay Binding projiziert jede Änderung der Nutzdaten auf das UI. Der Benutzer hat also stets eine aktuelle Sicht auf die Daten und kann somit schnell auf Änderungen reagieren.
- **TwoWay**: TwoWay Databinding ergänzt das OneWay Bindung um die Eigenschaft, dass Änderungen die der Benutzer an einem Objekt durchführt, automatisch an das gekoppelte Nutzdatenobjekt übertragen werden. Somit muss bei der Entwicklung der Applikation nicht darauf geachtet werden, dass alle Änderungen des Benutzers auf das Nutzdatenobjekt übertragen werden.

Um die Daten an die entsprechenden Komponenten des UI zu binden, müssen die Variablen des Nutzdatenobjektes mittels Markup Extension im XAML-Code deklariert werden. Mit dieser Kopplung werden Änderungen automatisch in beide Richtungen übertragen. Listing 5.4 zeigt die Koppelung von Variablen an ein XAML-Objekt.

```
1 <UserControl ... DataContext="Datenklasse">
2   ...
3   <TextBlock Text="{Binding Variablenname}"/>
4   ...
5 </UserControl>
```

Listing 5.4: XAML Datenbindung

Damit Silverlight die Zuordnung einer Variable zu einem Objekt herstellen kann, muss der Klassenname mittels DataContext, wie in Zeile 1 des Listings, angegeben werden. Danach kann mittels Binding Extension der Wert einer Variable jeder passenden Eigenschaft zugewiesen werden.

5.5 Deployment und Wartung

5.5.1 Silverlight Deployment Package

Silverlightapplikationen werden in einer XAP Datei packetiert und von jedem aufrufenden Client auf den Client geladen. Dabei handelt es sich um eine kompilierte Version der Applikation welche in einem speziellen Zip-Format mit allen benötigten Inhalten vorliegt. Abbildung 5.2 zeigt den Inhalt einer einfachen XAP-Datei.

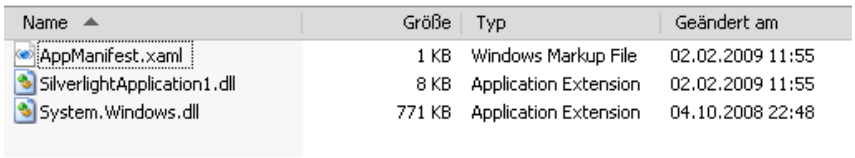

Abbildung 5.2: Aufbau einer einfachen XAP-Datei

Die wichtigste Datei innerhalb der XAP-Datei ist neben der kompilierten Applikation die *AppManifest.xaml* Datei. Sie beinhaltet alle von der Applikation benötigten Assemblies, sowie einen Verweis auf die Applikations-Dll. Falls gewünscht, können Assemblies welche nicht automatisch bereitgestellt werden sollen, aus dieser Datei entfernt werden. Wird die Assembly nun bei Aufruf das erste Mal benötigt, wird diese nachgeladen. Somit kann die eigentliche Applikationsdatei klein gehalten werden und zusätzliche Assemblies zeitversetzt

nachgeladen werden.
Neben der Manifestdatei existieren in der XAP-Datei mehrere Assemblies, welche einerseits die Applikation selbst beinhalten, als auch alle referenzierten Assemblies und verwendete Ressourcen wie Bilder.
(vgl. [Mic09a])

5.5.2 Hosting einer Silverlightapplikation

Um eine Silverlightapplikation zum Download anzubieten, sind folgende Schritte notwendig:

- Erzeugung der XAP Datei
- Erzeugung einer HTML-Seite für das Bereitstellen der Applikation
- Bereitstellen der Applikation durch IIS

Die XAP-Datei kann einerseits durch Visual Studio 2008 oder durch das in Silverlight mitgelieferte Tool *Chiron.exe* erstellt werden. Damit Clients wie Outlook 2007 die Silverlightapplikation verwenden können, ist es notwendig diese durch eine HTML-Seite bereitzustellen. Dazu wird, ähnlich dem einbinden einer Flash-Applikation, ein *<object>*-Tag verwendet, welches die Eigenschaften wie visuelle Größe, verwendete Applikationsversion und die XAP-Datei selbst beschreibt. Da Fehlermeldungen von Silverlight mittels JavaScript abgehandelt werden, kann eine Silverlightapplikation nur dann vom Client abgerufen werden, wenn im entsprechenden Browser JavaScript aktiviert wurde.
Bei der Bereitstellung durch die Internet Information Services (IIS) muss darauf geachtet werden, dass bei Verwendung einer Version kleiner als 7.0 das Abrufen von XAP-Dateien erst durch Verbindung der .xap Extension und dem dazugehörigen MIME-Type möglich ist.

Um eine neuere Version der Applikation zur Verfügung zu stellen, genügt es die aktualisierte XAP-Datei über die bereits vorhandene Version zu kopieren. Da innerhalb der XAP-Datei die Versionsnummer verwaltet wird, erkennt ein Client die neue Version und aktualisiert, falls vorhanden, seine lokale Version.

5.6 Darstellung von Benutzerdaten

Nachdem in diesem Kapitel bereits viele Aspekte von Silverlight aufgezeigt wurden, soll nun mittels XAML ein Maskenprototyp entwickelt werden. Als Prototyp wurde die Detailansicht eines Benutzers gewählt, da hier viele unterschiedliche Komponenten von Silverlight eingesetzt werden können. Der Prototyp selbst besitzt keine Business Logik und damit auch keine Code-Behind Funktionen.
Zur Erstellung der Maske wurde neben Visual Studio 2008 auch Microsoft Expression Blend 2 verwendet, welches eine Entwicklungsumgebung für das Design von Silverlight Masken ist. Während der Erstellung zeigte sich, dass sich das Design mit Expression Blend einfacher gestaltet als mit Visual Studio, da Visual Studio oftmals Probleme mit der Anzeige diverser Komponenten aufweist und die Positionierung nur über die Eigenschaften der Objekte möglich ist. In Expression Blend hingegen können die Elemente direkt über den Designer positioniert werden und die entsprechenden Eigenschaften werden automatisch bestmöglich gesetzt. Man sollte diese Eigenschaften jedoch nachbearbeiten, da Expression Blend bei der Positionierung sehr genau arbeitet und der XAML Code durch zu genaue Positionierung oft unleserlich wird. Abbildung 5.3 zeigt den erstellten Prototypen, welcher im nachfolgenden besprochen wird.

Die Positionierung der Elemente erfolgte durch die Verwendung eines *Grid*-Objektes als Grundlage und mehrerer *StackPanel*-Objekte um innerhalb der einzelnen Zeilen und Spalten die Objekte besser positionieren zu können. Silverlight würde zusätzlich noch eine freie Positionierung der Objekte durch x- und y-Koordinaten ermöglichen, jedoch wurde auf diese Methode aus Gründen der Wartbarkeit verzichtet.
Um eine Grid-Komponente verwenden zu können, müssen zunächst die Anzahl der verwendeten Reihen und Spalten definiert werden. Listing 5.5 zeigt, wie die Definition für den erstellten Prototypen programmiert wurde.

```
1 <Grid x:Name="LayoutRoot" Background="White">
2   <Grid.RowDefinitions>
3     <RowDefinition Height="50" />
4     <RowDefinition Height="600"/>
```

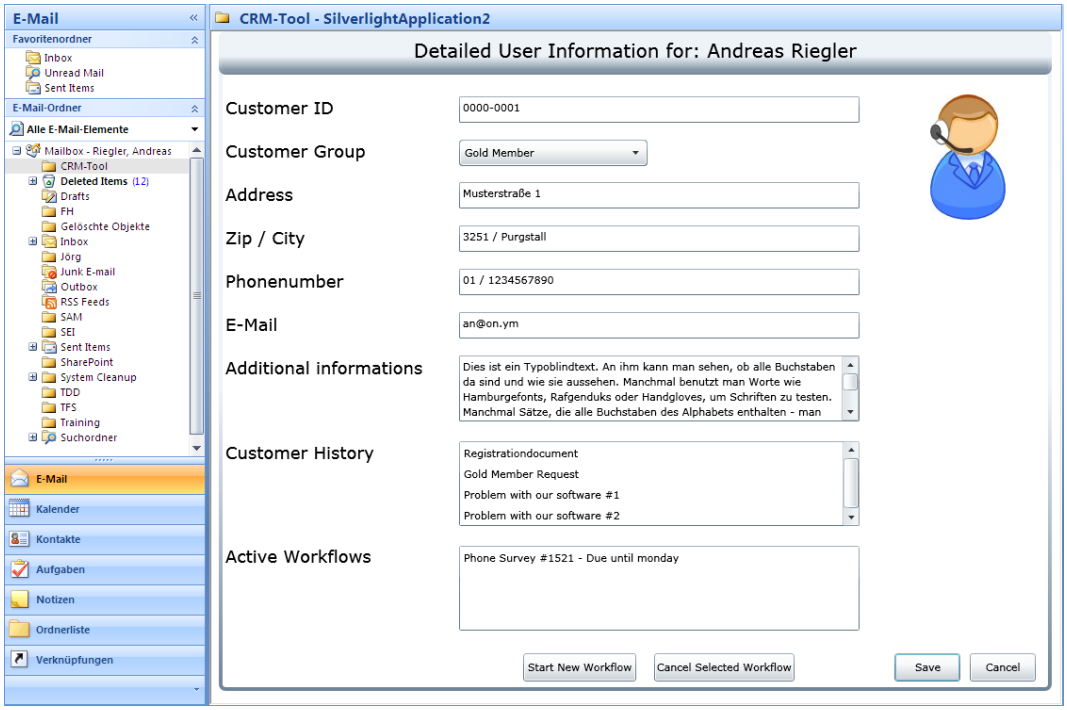

Abbildung 5.3: Prototyp einer Silverlightmaske

```
5     <RowDefinition Height="50"/>
6   </Grid.RowDefinitions>
7   <Grid.ColumnDefinitions>
8     <ColumnDefinition Width="250"/>
9     <ColumnDefinition />
10    <ColumnDefinition Width="200"/>
11  </Grid.ColumnDefinitions>
12  ...
```

Listing 5.5: Definition von Reihen und Spalten innerhalb eines Grid-Objektes

Nachdem das Grundgerüst konfiguriert wurde, können Objekte mittels der Eigenschaften *Grid.Column=„Spaltennummer“* und *Grid.Row=„Zeilennummer“* in einem Sektor positioniert werden. Da sich meist mehrere Objekte innerhalb eines Sektors befinden, wurde die Zuweisung im StackPanel-Objekt durchgeführt. Innerhalb des StackPanels können sich weitere Objekte befinden, welche dann je nach Konfiguration des StackPanel-Objektes positioniert werden. Da für alle Sektoren des Prototypen StackPanel-Objekte zur Darstellung verwendet wurden, wird in Listing 5.6 die Verwendung dieses StackPanels einmalig

gezeigt.

```
<StackPanel Grid.Column="2" Grid.Row="2" HorizontalAlignment="
    Right"
 Orientation="Horizontal"  Margin="0,10,25,10"  >
  <Button Width="70" Margin="0,0,10,0" Content="Save" />
  <Button Content="Cancel" Width="70"/>
</StackPanel>
```

Listing 5.6: Definition/Konfiguration eines StackPanels

Dieses StackPanel fügt dem letzten Sektor des Prototypen zwei Schaltflächen hinzu, welche Horizontal ausgerichtet wurden. Weiters wurde mittels Margin ein Abstand definiert, da die Schaltflächen ansonsten den gesamten Sektor ausfüllen würden.

5.7 Bewertung

Für die Darstellung der gespeicherten Nutzdaten bietet das .NET Umfeld eine Vielzahl von Möglichkeiten. In diesem Kapitel wurden die Fähigkeiten von Silverlight 2.0 für die Darstellung von Benutzerdaten in einem CRM-System ermittelt und die Verwendungsmöglichkeiten aufgezeigt.
Daten können mittels WebServices abgerufen aber auch verändert werden und benötigen in Silverlight, im Vergleich zu anderen .NET Technologien keinen zusätzlichen Entwicklungsaufwand. Es können allerdings nur asynchrone WebServices aufgerufen werden, da bei synchronen WebServiceaufrufen, das UI solange pausiert werden würde, bis Silverlight eine Antwort vom Server erhalten würde.
Silverlight integriert auch einige Möglichkeiten, um Daten zu verschlüsseln und die Integrität der Daten zu gewährleisten. Somit wird die Gefahr der Datenspionage oder Datenmanipulation minimiert.
Falls noch nicht mit Silverlight oder XAML gearbeitet wurde, ist der anfängliche Lernaufwand sehr hoch und es dauert einige Zeit, bis man User Interfaces

entwickeln kann, welche dem Look and Feel von Office Applikationen entsprechen.
Abschließend kann man sagen, dass sich Silverlight für die Umsetzung eines CRM-Systems durchaus eignet, da keine Einschränkung bei der Erstellung des CRM-Systems vorhanden ist. Diese Aufgabe könnte jedoch auch von anderen Technologien wie WPF oder einer normalen Windows Forms Applikation bewerkstelligt werden.

Kapitel 6

Exemplarische Umsetzung des CRM-Systems

6.1 Serverseitige Konfiguration

Um die Daten des CRM-Systems verwalten zu können, müssen in WSS einige Konfigurationen getroffen werden, um Daten einerseits speichern zu können, diese aber mittels Silverlight auch einfach abrufbar zu machen. Zu diesem Zweck wurden zwei unterschiedliche Listen innerhalb von WSS definiert, um Daten bzw. Dokumente verwalten zu können.

Um generelle Daten von Kunden zu speichern, wurde eine Benutzerdefinierte Liste angelegt, in der die Eigenschaften der Kunden erfasst werden. Weiters wurde bei den Berechtigungen der Liste eine Gruppe hinzugefügt, welche alle Benutzer beinhaltet, die das CRM-System benutzen werden. Abbildung 6.1 zeigt die definierten Spalten der Liste.

Da durch das CRM-System auch Kundenanfragen oder -probleme erfasst werden können, sollen diese ebenso in WSS verwaltet werden. Da Dokumente allerdings nicht innerhalb der bereits angelegten Benutzerdefinierten Liste abgelegt werden können, wird zu diesem Zweck eine Dokumentenbibliothek erstellt. Diese kann Dokumente jeder Art beinhalten und bietet zusätzliche Funktionalitäten wie Versionierung. Weiters wird der in Kapitel 6.3 entwickelte Workflow an diese Liste gekoppelt, um bei Hinzufügen eines neuen Dokumentes automatisch gestartet zu werden.

Spalten

In einer Spalte werden Informationen zu den einzelnen Elementen in der Liste gespeichert. Da diese Liste mehrere Inhaltstypen zulässt, werden einig erforderlich oder optional sind) jetzt durch den Inhaltstyp des Elements angegeben. Die folgenden Spalten sind zurzeit in dieser Liste verfügbar:

Spalte (Klicken Sie hier zum Bearbeiten)	Typ	Verwendet in
Adresse	Mehrere Textzeilen	Customer
CustomerDokumente	Nachschlagen	
CustomerID	Eine Textzeile	Customer
E-Mail	Eine Textzeile	Customer
Gruppe	Auswahl	Customer
Land/Region	Eine Textzeile	Customer
Ort	Eine Textzeile	Customer
PLZ	Eine Textzeile	Customer
Telefon (privat)	Eine Textzeile	Customer
Titel	Eine Textzeile	Customer
Vorname	Eine Textzeile	Customer
ZusaetzlicheInformationen	Mehrere Textzeilen	Customer
Erstellt von	Person oder Gruppe	
Geändert von	Person oder Gruppe	

Abbildung 6.1: Benutzerdefinierte Liste zum speichern der Kundendaten

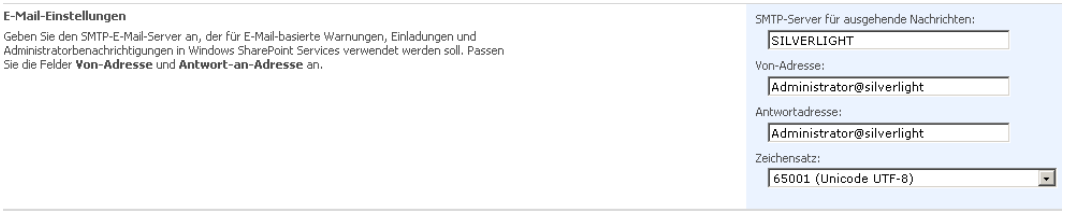

Abbildung 6.2: E-Mail Konfiguration in WSS

Da innerhalb des CRM-Systems automatisch generierte E-Mails versendet werden sollen (siehe Kapitel 6.3), müssen in WSS einige Einstellungen getroffen werden, damit diese Tätigkeit automatisch durchgeführt werden kann. WSS benötigt zum Versenden von E-Mails einerseits eine SMTP-Serveradresse, als auch Informationen, von welchem Konto die E-Mails gesendet werden sollen. Diese Einstellungen werden über die Zentraladministration vorgenommen. Abbildung 6.2 zeigt die entsprechende Konfiguration innerhalb des Prototypen.

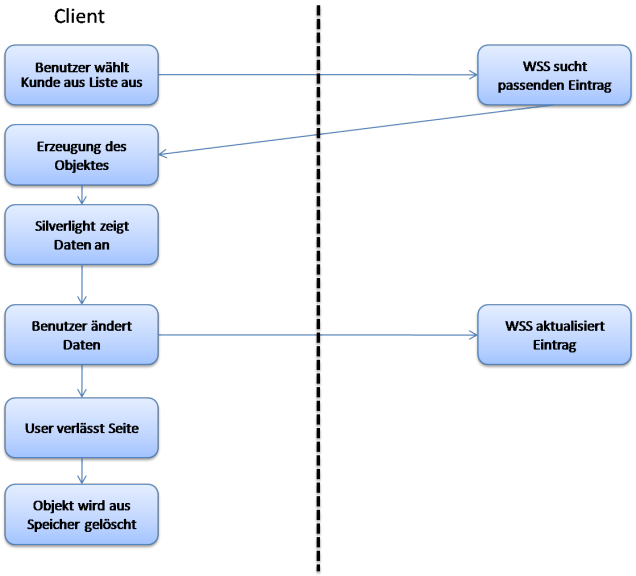

Abbildung 6.3: Lebenszyklus eines Kundenobjektes

6.2 Demonstration des Lebenszyklus eines Objektes anhand eines Kundenobjektes

Nachdem im vorherigen Kapitel allgemeine Einstellungen und die Verwaltung des CRM-Systems gezeigt wurden, wird in diesem Kapitel der Lebenszyklus eines Kundenobjektes im Detail erklärt. Zur clientseitigen Visualisierung wird dazu das UI aus Kapitel 5.6 verwendet. Abbildung 6.3 zeigt den schematischen Lebenszyklus eines solchen Objektes. Auf diese Weise können auch neue Kunden im System hinzugefügt werden.

Wie in Abbildung 6.3 zu erkennen ist, existiert ein Kundenobjekt nur clientseitig. In WSS werden diese Daten innerhalb einer Liste (siehe Kapitel 6.1) gespeichert. Das Kundenobjekt existiert solange, bis der Benutzer die Webseite mit den Informationen des Kunden verlässt.

6.2.1 Speicherung der Kundendaten in C#

Um den Zugriff auf Kundendaten zu vereinheitlichen, wurde eine C#-Klasse entwickelt, welche die einzelnen Daten in Eigenschaften der Klasse speichert

und Silverlight zur Verfügung stellt. Weiters enthält diese Klasse Funktionen, um Daten die von WSS übermittelt worden sind, automatisch in die entsprechenden Properties zu mappen und Daten in eine für WSS lesbare Form zurückzugeben.

```
...
namespace CRM-Tool
{
    public class Customer : INotifyPropertyChanged
    {
        private string Id
        {
            get;
            set;
        }

        public event PropertyChangedEventHandler PropertyChanged;

        private string _CustomerId;
        public string CustomerId
        {
            get { return _CustomerId; }
            set
            {
                _CustomerId = value;
                this.NotifyPropertyChanged( "CustomerId" );
            }
        }
...
```

Listing 6.1: Definition der Klasse Customer

Listing 6.1 zeigt die Definition des erwähnten Kundenobjektes. Aus Gründen der Lesbarkeit wird nur eine Eigenschaft des Kundenobjektes aufgezeigt, da die Deklaration für alle weiteren Dateneigenschaften ähnlich ist. Die Klasse wird vom Interface *INotifyPropertyChanged* abgeleitet, da mit Hilfe dieses Interfaces die automatische Aktualisierung des UI von Silverlight durchgeführt wird. Zusätzlich zu den, in der UI angezeigten Daten, existiert noch eine weitere private Eigenschaft innerhalb des Customer-Objektes. Die Eigenschaft *Id* beinhaltet die GUID des Kundenobjektes, um Updates nach WSS durchführen

zu können, da hier die GUID zur Zuordnung verwendet wird.

Da Änderungen an Kundendaten direkt in das UI übertragen werden sollen, muss jede Eigenschaft der Klasse innerhalb der *set*-Funktion die Funktion *NotifyPropertyChanged()* aufrufen, welcher der Spaltenname des zugehörigen Objektes im UI übergeben wird. Dadurch kann ein automatisches Update durchgeführt werden. Listing 6.2 zeigt die zugehörige Funktion.

```
312 public void NotifyPropertyChanged( string propertyName )
313 {
314     if( PropertyChanged != null )
315     {
316         PropertyChanged( this, new PropertyChangedEventArgs(
                propertyName ) );
317     }
318 }
```

Listing 6.2: NotifyPropertyChanged Funktion

Damit die Daten durch Aufruf der *NotifyPropertyChanged* Funktion in das UI übertragen werden, muss dort ein zusätzliches Mapping eingeführt werden. Dies geschieht mittels Markup Extension welche in Kapitel 5.4.2 bereits erklärt wurde. Weiters erhält jedes XAML-Element eine Eigenschaft *x:Name*, welche das Element eindeutig identifziert und die Verbindung zur Customer-Klasse herstellt (Übergabe des XAML Namens in *NotifyPropertyChanged* Funktion). Somit müssen keine zusätzlichen Funktionalitäten zur Synchronisation der Daten an das UI und in das Kundenobjekt entwickelt werden, was den Programmieraufwand erheblich minimiert.

6.2.2 Verwendung des Lists WebServices

Die Kommunikation zwischen Silverlight und WSS läuft ausschließlich über WebServices ab, da ein Großteil der benötigten Funktionalität bereits durch die vorhandenen WebServices von WSS abgedeckt wird. Silverlight unterstützt dabei nur asynchrone WebServices, da die Applikation nicht auf das Ergebnis eines Aufrufes warten darf und das UI während dieser Zeit nicht bedienbar wäre. Erstellt man eine Verbindung zu einem WebService in Visual Studio

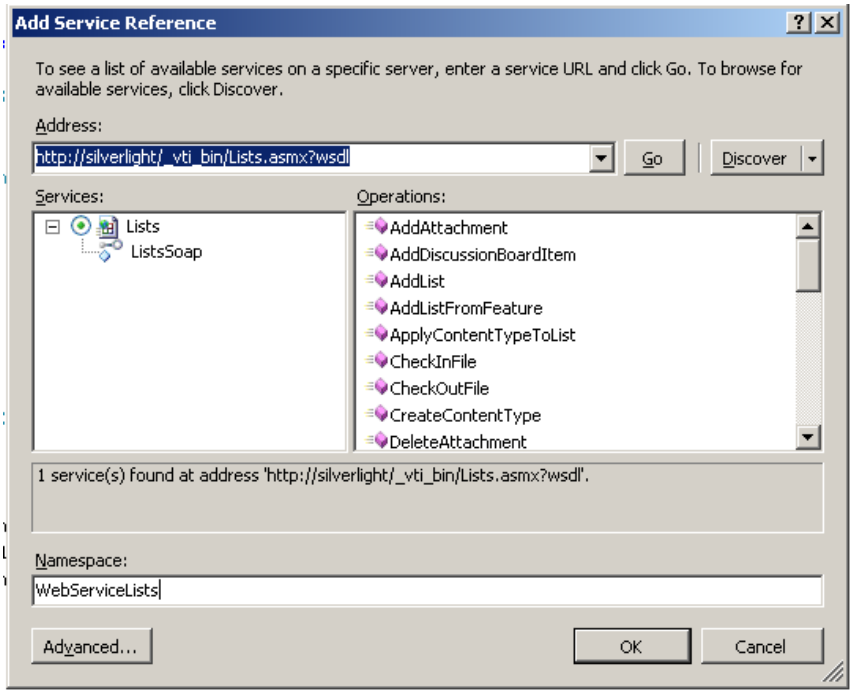

Abbildung 6.4: Referenz auf das Lists WebService erzeugen

2008, werden automatisch die asynchronen Funktionen erzeugt und die Parameter für Silverlight entsprechend vorkonfiguriert. Da Silverlight nur über eine Teilmenge des .NET Frameworks verfügt, können einige Klassen wie *XmlElement*, welche vor allem für Abfragen nach WSS verwendet werden, nicht verwendet werden. Silverlight stellt dafür Klassen aus dem Namespace *System.Xml.Linq* zur Verfügung. Dadurch wird es auch möglich, Linq to XML Abfragen auf Antworten von WSS durchzuführen oder Abfragen nach WSS in diesen Elementen zu kapseln.

Wie in Kapitel 6.1 erwähnt, werden Kundenobjekte innerhalb einer Benutzerdefinierten Liste abgelegt. Um auf diese Elemente Zugriff zu haben, wird innerhalb des Silverlightprojektes in Visual Studio eine Referenz auf das WebService *Lists.asmx* hergestellt. Visual Studio erstellt dadurch alle benötigten Klassen, um auf die einzelnen Funktionen des WebServices asynchron zugreifen zu können. Abbildung 6.4 zeigt die Erstellung einer solchen Referenz.

Dieses WebService wird sowohl für das Laden von Kundendaten aus WSS, als auch zur Speicherung von Kundendaten verwendet. Nachdem das WebService

referenziert wurde, kann mittels *WebServiceLists.ListsSoapClient* eine Instanz dieser Klasse erzeugt werden und die entsprechenden Funktionen aufgerufen werden. Um die Daten eines Kunden abzurufen, wird dazu die Funktion *GetListItemsAsync* verwendet. Diese erwartet unter anderem den Listennamen, sowie eine CAML-Query, welche das Suchergebnis einschränkt. Wird keine Query übergeben, werden alle Elemente der Liste geladen. Listing 6.3 zeigt die Funktion, welche zur Abfrage eines Kunden verwendet wird.

```
35 public void LoadCustomer(string id)
36 {
37     WebServiceLists.ListsSoapClient listclient = new
           WebServiceLists.ListsSoapClient();
38
39     XElement query = XElement.Parse(@"<Query>
40             <Where>
41                 <Eq>
42                     <FieldRef ID='991c005c-a7b4-4553-858c-66d8cf0
                           ad411' />
43                     <Value Type='Text'>"+id+@"</Value>
44                 </Eq>
45             </Where>
46         </Query>");
47
48     XElement queryOpt = XElement.Parse(@"<QueryOptions>
49             <IncludeAttachmentUrls>TRUE</IncludeAttachmentUrls>
50         </QueryOptions>");
51
52     listclient.GetListItemsCompleted += new EventHandler<
           SilverlightApplication2.WebServiceLists.
           GetListItemsCompletedEventArgs>
53                     ( HandleListItemsCompleted );
54     listclient.GetListItemsAsync( "{42CDF289-937C-413B-950B-00734D
           1E11EC}", string.Empty, query, null,
55                                   string.Empty, queryOpt, string.
                                         Empty );
56 }
```

Listing 6.3: LoadCustomer Funktion zum Abfragen eines Kunden

Das *query* Objekt in Zeile 5 beinhaltet die Abfrage eines bestimmten Kunden. Um die Daten eines Kunden abzufragen, wird definiert, dass nur der

Eintrag zurückgegeben wird, dessen eindeutige ID der übergebenen ID entspricht. Dazu wird das interne ID Feld der Liste abgefragt. Zusätzlich wird eine QueryOption festgelegt, um die URLs der Attachments (Dokumente die einem Kunden hinzugefügt wurden) zu erhalten. Somit können diese Dokumente von Silverlight geöffnet werden.

Da es sich bei allen Aufrufen um asynchrone Methoden handelt, muss vor dem Aufruf des WebServices eine Methode definiert werden, welche bei einer Antwort vom Server aufgerufen wird. Dazu wird ein neuer EventHandler (siehe Zeile 18) für die aufzurufende Methode definiert. Im Anschluss an diese Definition können die Benutzerdaten mittels *GetListItemsAsync* Funktion abgefragt werden. Dazu wird dem WebService der Listenname (Parameter 1), die Abfragequery (Parameter 3), sowie die Queryoptionen (Parameter 6) übergeben. Alle anderen Parameter werden mit Standardwerten initialisiert.

Um die Antwort des WebServices abarbeiten zu können muss noch die bereits registrierte Funktion wie in Listing 6.4 ausprogrammiert werden.

```
64 protected void HandleListItemsCompleted(
65  object sender, SilverlightApplication2.WebServiceLists.
        GetListItemsCompletedEventArgs args )
66 {
67     customer.ParseXml( ( XNode ) args.Result );
68 }
```

Listing 6.4: Funktion zum Abarbeiten der WebService-Antwort

Die Funktion ruft bei Aufruf eine Funktion des Kundenobjektes auf, welches das Ergebnis des WebServices, wie in Abbildung 6.5 gezeigt, den entsprechenden Kundenfeldern zuweist. Dazu wird innerhalb der *ParseXml* Methode die *XNode* Variable in ein *XElement* Objekt umgewandelt, um die Attribute des Elementes auf die einzelnen Eigenschaften des Customer Objektes zuweisen zu können. Durch das TwoWay-Binding, werden diese Änderungen auch sofort auf dem UI dargestellt.

Um geänderte Daten an WSS zurückzusenden, muss die WebService-Methode *UpdateListItemsAsync* verwendet werden. Diese erwartet neben dem Listennamen folgendes XML-Konstrukt.

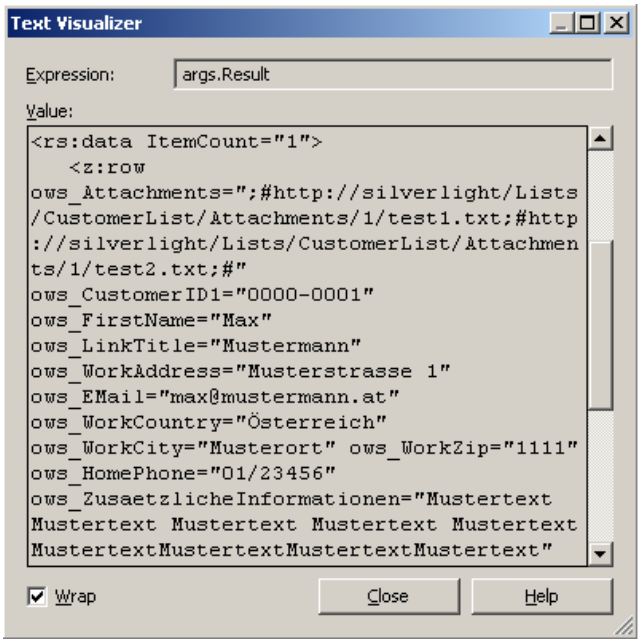

Abbildung 6.5: Antwort des WebServices

```
1 <Batch OnError='Continue'>
2     <Method ID='1' Cmd='Update'>
3         <Field Name='ID'>Interne ID</Field>
4         <Field Name='Feldname'>neuer Wert</Field>
5         ...
6     </Method>
7 </Batch>
```

Listing 6.5: XML-Query zum Speichern von Änderungen

Die Eigenschaft *Cmd* definiert dabei die Art der durchzuführenden Aktion, da mit dieser WebService-Methode auch ein neues Objekt hinzugefügt werden kann. Ein Field-Objekt, welches die interne ID des Objektes beinhaltet, ist bei einem Update zwingend notwendig, da WSS so die Verbindung zu dem entsprechenden Objekt herstellen kann. Im Anschluss an die interne ID werden alle zu ändernden Felder wie in Zeile 4 angegeben. Für die Umsetzung des CRM-Systems werden bei einem Update immer alle Felder übergeben, da durch das TwoWay-Binding nicht ohne zusätzlichen Aufwand festgestellt werden kann, welches Feld sich seit dem letzten Update geändert hat. Bei Antwort des WebServices muss der übergebene ErrorCode überprüft werden, um das

erfolgreiche Update verifizieren zu können. Ein Errorcode der von *0x000000* abweicht, weist auf einen Fehler während des Speichervorgangs hin. Listing 6.6 zeigt die entsprechende Umsetzung innerhalb des CRM-Systems. Da XElement das Ergebnis nicht in eine passende Struktur bringen konnte, wurde der ErrorCode mittels Funktionen der String Klasse ausgelesen und das Ergebnis in der Variable *errorText* abgelegt. Bei der Variable *errorText* handelt es sich um ein TextBox Element, welches in der Silverlight Seite hinzugefügt wurde, um eine entsprechende Statusmeldung ausgeben zu können.

```
1 protected void HandleUpdateListItemsCompleted( object sender,
      SilverlightApplication2.WebServiceLists.
      UpdateListItemsCompletedEventArgs args )
2 {
3     XElement response = ( XElement ) args.Result;
4     bool noErrorFound = response.ToString().Contains(@"<ErrorCode
          >0x000000<");
5
6     errorText.Text = noErrorFound ? "Update successfull." : "Error
           during Update.";
7 }
```

Listing 6.6: Auswertung des ErrorCodes

6.3 Entwicklung eines Workflows am Beispiel der Abwicklung eines Kundenrequests

Ein Szenario, welches in CRM-Systemen sehr oft auftritt ist das Entgegennehmen und Weiterleiten von Kundenproblemen. Dabei müssen Benutzer oft viele Schritte durchführen, um Dokumente an die entsprechenden Personen weiterzuleiten und diese über das Problem zu informieren. Um dieses Szenario zu vereinfachen, kann aber auch ein Workflow verwendet werden. Der Benutzer lädt dazu nur das Dokument mit der Beschreibung des Problems in das CRM-System hoch und alle weiteren Schritte werden automatisch vom Workflow durchgeführt.

Dieses Kapitel soll aufzeigen, wie ein solcher Workflow zu implementieren ist und welche zusätzlichen Schritte notwendig sind, um den Workflow mit dem

CRM-System zu verbinden. Dazu wird zunächst der Workflow mittels Visual Studio 2008 implementiert und danach das CRM-System um die benötigten Funktionalitäten erweitert.

6.3.1 Erstellen des Kundenrequest Workflows

Der zu entwickelnde Workflow soll folgende Funktionalitäten aufweisen:

1. Wurde ein neues Problem hochgeladen, wird einem anderen Benutzer eine neue Aufgabe zugewiesen
2. Der Benutzer erhält eine E-Mail mit der Aufforderung die Aufgabe zu bearbeiten
3. Der Benutzer kann in einem Formular Lösungsvorschläge und Kommentare eingeben und die Aufgabe abschließen
4. Der ursprüngliche Benutzer wird über die eingetragenen Lösungsvorschläge und Kommentare via E-Mail informiert

Um den Workflow zu erstellen, wird in Visual Studio zunächst ein SharePoint 2007 Sequential Workflow angelegt und dieser mittels Konfigurationsmaske an die in Kapitel 6.1 beschriebene Dokumentenbibliothek gebunden. Weiters wird festgelegt, dass der Workflow bei Hinzufügen eines Dokumentes automatisch gestartet wird.
Abbildung 6.6 zeigt die grafische Übersicht über alle verwendeten Workflowkomponenten und ihre Beziehung zueinander.

Das erste Element innerhalb des Workflows dient ausschließlich dazu, Eigenschaften über den Workflow in einer Variable vom Typ *SPWorkflowActivationProperties* zu speichern. Diese Eigenschaften beinhalten unter anderem die ID des Statusfeldes, sowie sämtliche IDs von Elementen die mit dem Workflow in Verbindung stehen. Das Element *createTaskWithContentType1* erstellt danach die Aufgabe, welche einem bestimmten Benutzer zugewiesen wird. Dazu müssen in der Code-Behind Datei des Workflows einige Eigenschaften dieses Elements gesetzt werden, welche in Listing 6.7 gezeigt werden.

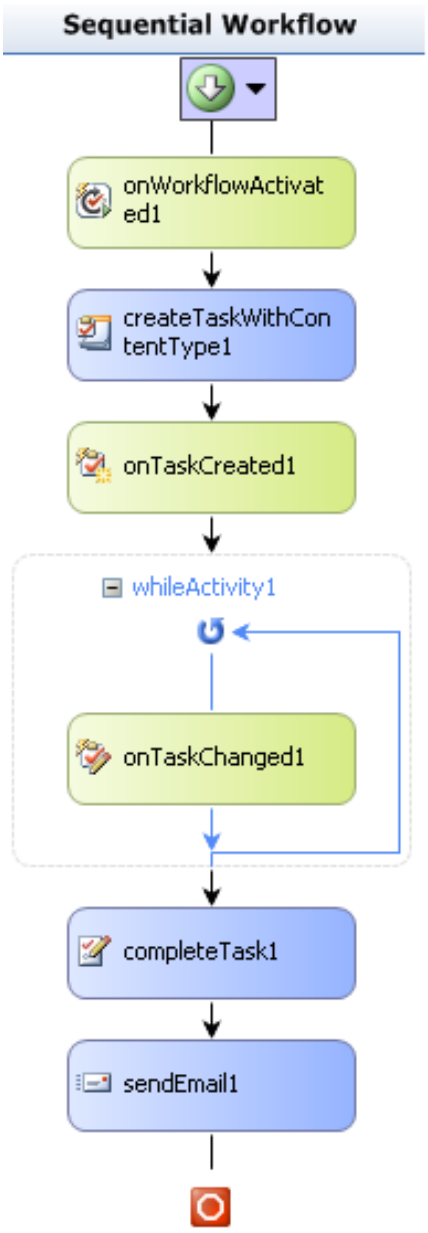

Abbildung 6.6: Workflow zur Abarbeitung eines Kundenrequests

```
1 private void createTaskWithContentType1_MethodInvoking_1(object
      sender, EventArgs e)
2 {
3     createTaskWithContentType1.ContentTypeId = "0x010801001A9FA727
          B671DE42A57CF99BD27D4BD3";
4     createTaskWithContentType1.TaskProperties.AssignedTo = @"
          silverlight\problemsolver1";
5     createTaskWithContentType1.TaskProperties.DueDate = DateTime.
          Now.AddDays(2);
6     createTaskWithContentType1.TaskProperties.Title = "Please
          resolve the assigned problem";
7     createTaskWithContentType1.TaskProperties.EmailBody = "Please
          resolve the assigned problem";
8     createTaskWithContentType1.TaskProperties.
          SendEmailNotification = true;
9 }
```

Listing 6.7: Erzeugung einer Aufgabe mittels Workflow

Zuerst wird der Aufgabe die Id einer aspx Seite hinzugefügt. Diese wurde im Vorfeld mit Hilfe von Sharepoint Designer 2007 erstellt und in SharePoint integriert. Diese Seite beinhaltet das Feld Lösungsvorschlag sowie das Kommentarfeld. Die restlichen Eigenschaften definieren den Zielbenutzer, die Zeit welche er bis zur Erfüllung der Aufgabe hat und Inhalt des automatisch generierten E-Mails, wenn die Aufgabe in WSS hinzugefügt wurde.

Das nächste wichtige Element innerhalb des Workflows ist *onTaskChanged1* welches innerhalb einer While Bedingung so oft ausgeführt wird, bis der Status der Aufgabe auf „Completed“ geändert wurde. Dazu wird bei jeder Änderung der Aufgabe die Aufgabeneigenschaft *Status* abgefragt. Wenn diese Bedingung erfüllt wurde, werden Variablen befüllt, welche für das letzte Element in diesem Workflow benötigt werden. Listing 6.8 zeigt die Implementierung dieses Elements.

```
1 private void onTaskChanged1_Invoked_1(object sender,
      ExternalDataEventArgs e)
2 {
3     Guid id = workflowProperties.TaskList.Fields["Status"].Id;
4     string status = onTaskChanged1_AfterProperties1.
          ExtendedProperties[id].ToString();
```

```
5
6       if (status != null)
7       {
8           if (status.Equals("Completed"))
9           {
10              _taskComplete = true;
11              Guid solutionId = workflowProperties.TaskList.Fields["
                    Solution"].Id;
12              Guid commentsId = workflowProperties.TaskList.Fields["
                    Comments"].Id;
13              string solution = onTaskChanged1_AfterProperties1.
                    ExtendedProperties[solutionId].ToString();
14              string comments = onTaskChanged1_AfterProperties1.
                    ExtendedProperties[commentsId].ToString();
15              sendEmail1_Subject1 = "Problem resolved";
16              sendEmail1_Body1 = "Your reported problem was resolved
                    .\n\nThe provided solution is:\n\n" +
17                                  solution + "\nProblemSolver also
                                        provided the following comments
                                        :\n" +
18                                  comments;
19              sendEmail1_To1 = workflowProperties.OriginatorEmail;
20          }
21      }
22  }
```

Listing 6.8: Erzeugung einer Aufgabe mittels Workflow

Der letzte Schritt innerhalb des Workflows ist die Versendung einer E-Mail an den Initiator des Workflows. Die benötigten Parameter können hier direkt über die Designansicht festgelegt werden, wie Abbildung 6.7 zeigt. Dazu wird bei den entsprechenden Eigenschaften des Elements auf Variablen referenziert, die in Listing 6.8 definiert wurden.

6.3.2 Erstellen des benötigten WebServices

Während der Entwicklung des Prototypen zeigte sich, dass mit den vorhandenen WebServices kein Dokument in eine Dokumentenbibliothek von WSS geladen werden kann. Da Silverlight nur die Verwendung von „GET“ und

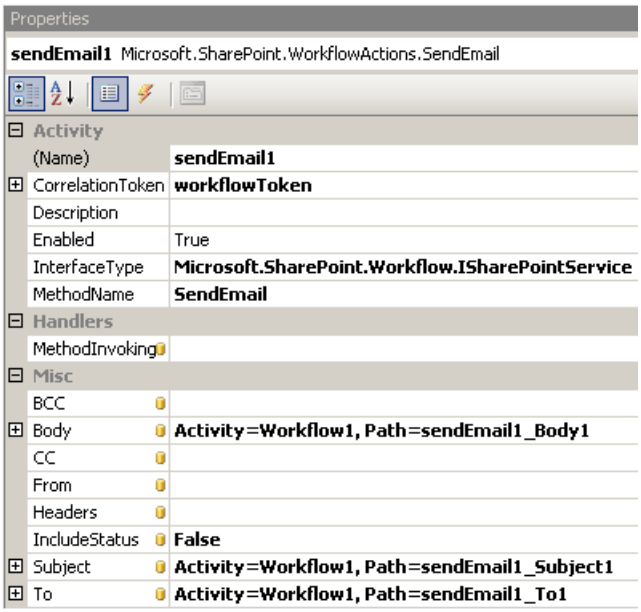

Abbildung 6.7: Eigenschaften des SendEmail Elementes

„POST“ bei HTTP Anfragen zulässt, kann ein Dokument auch nicht durch Verwendung der „PUT“ Option (welche ein Workaroung für den Upload von Dokumenten ist) von HTTP nach WSS geladen werden. Deshalb musste ein eigenes WebService entwickelt werden, welches diese Funktionalität bereitstellt. Listing 6.9 zeigt die Implementierung des benötigten WebServices.

```
1  [WebMethod]
2  public void UploadDocument(string fileName, byte[] fileContents)
3  {
4      using (SPSite siteCollection = new SPSite("http://silverlight
           /"))
5      {
6          using (SPWeb spWeb = siteCollection.OpenWeb())
7          {
8              SPList spList = spWeb.GetList("Customer_Documents");
9
10             spList.RootFolder.Files.Add(spList.RootFolder.Url +
                   "/" + fileName, fileContents, true);
11             spList.Update();
12         }
13     }
```

```
14 }
```

Listing 6.9: Implementierung des UploadDocument WebServices

Die Funktion *UploadDocument* erhält als Übergabeparameter den Dateinamen, welcher in WSS verwendet werden soll, sowie ein Bytearray, welches den Inhalt des Dokumentes beinhaltet. Um das Dokument in der entsprechenden Liste zu erzeugen, muss zunächst eine Instanz der Liste erzeugt werden (siehe Zeile 8). Das Hinzufügen eines Dokumentes zu einer Liste erfolgt danach über die Funktion *Add* des instanzierten Listenobjektes. Um das Dokument permanent zu speichern, muss als letzte Operation ein Update für die entsprechende Liste durchgeführt werden. Durch Aufruf der Updatefunktion wird gleichzeitig, der zuvor entwickelte Workflow gestartet.

6.3.3 Integration in Silverlight

Nachdem alle serverseitigen Implementierungen durchgeführt wurden, muss als letzter Schritt das UI in Silverlight erweitert werden, um die entsprechenden Funktionalitäten dem Benutzer zur Verfügung zu stellen. Dazu wurde zunächst eine neue Silverlight Seite entwickelt, welche zum Upload eines Dokumentes verwendet wird. Diese Seite soll als PopUp bei Drücken der „Start New Workflow“ Schaltfläche angezeigt werden. Da Standardbenutzer nur diesen Workflow starten können, ist es nicht notwendig eine Auswahlmaske der möglichen Workflows anzuzeigen. Abbildung 6.8 zeigt das Aussehen des erstellten PopUps.

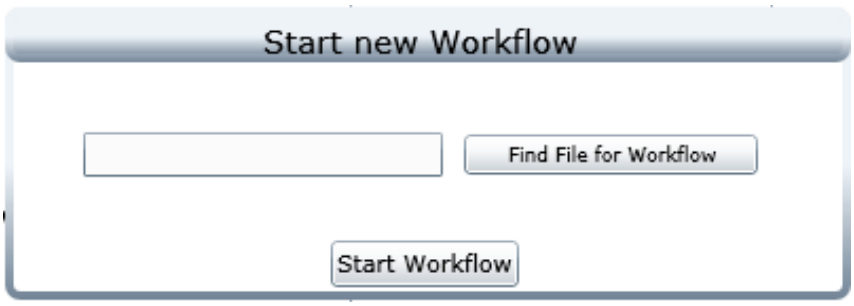

Abbildung 6.8: Starten eines neuen Workflows durch Silverlight

Da Silverlight keinen direkten Zugriff auf das Dateisystem besitzt, muss diese Auswahl durch die Klasse *OpenFileDialog* durchgeführt werden. Der Benutzer erhält dadurch die Standardmaske zum Auswählen der gewünschten Datei. Mittels „Start Workflow“ Schaltfläche, wird die zuvor ausgewählte Datei geöffnet und der Inhalt ausgelesen. Danach muss das WebService aus Kapitel 6.3.2 instanziert werden und die Methode *UploadDocumentAsync* aufgerufen werden. Dadurch wird das Dokument in die vorgesehene Dokumentenbibliothek übertragen und der zugehörige Workflow gestartet. Listing 6.10 zeigt das Auslesen des Dokumentes und die dazugehörige UploadDocument Methode.

```
1  private void StartWorkflow_Click( object sender, RoutedEventArgs e
       )
2  {
3    //fi (FileInfo) beinhaltet Details zur selektierten Datei
4      if (fi != null)
5      {
6          string buf;
7          using (StreamReader reader = fi.OpenText())
8              buf = reader.ReadToEnd();
9
10     //Konvertieren der Variable buf in ein byte Array
11         byte[] bufArray = (new UnicodeEncoding()).GetBytes(buf);
12
13         UploadDocument.ServiceSoapClient uploadClient = new
               UploadDocument.ServiceSoapClient();
14
15     //Dokument in die Bibliothek laden
16         uploadClient.UploadDocumentAsync(fi.Name, bufArray);
17
18         //schliessen des PopUps
19         pop.IsOpen = false;
20     }
21 }
```

Listing 6.10: Speicherung eines Dokumentes in eine Dokumentenbibliothek

Kapitel 7

Abschließende Bewertung und Ausblick

7.1 Abschließende Bewertung

Diese Arbeit beschäftigte sich mit den Technologien WSS, VSTO sowie Silverlight, welche zusammen die Entwicklung eines CRM-Systems ermöglichen sollen. Die Kombination der einzelnen Technologien ist ohne größeren Aufwand möglich, da jede Technologie bereits passende Schnittstellen zu den anderen Technologien besitzt und nur an einigen Stellen zusätzliche Funktionalitäten erstellt werden mussten, um eine Verbindung der Komponenten zu ermöglichen.

WSS bietet alle benötigten Funktionalitäten, um als serverseitige Komponente des CRM-Systems eingesetzt zu werden. Durch die Verwendung von WSS werden jedoch einige Betriebssystemkomponenten als auch Software vorausgesetzt. Dies kann zu Kosten führen, welche laut Zielvorgabe jedoch möglichst gering gehalten werden sollen. Da es sich hierbei allerdings um Standard Microsoftprodukte handelt, sind diese bereits in vielen Unternehmen vorhanden und es müssen gegebenenfalls nur einzelne Softwarekomponenten zugekauft werden. Ein weiterer Nachteil zeigte sich erst während der Entwicklung des Prototypen. WSS bietet in Kombination mit Silverlight keinerlei Möglichkeit, Dokumente in eine entsprechende Dokumentenbibliothek hochzuladen. Diese Schnittstelle musste erst als WebService entwickelt werden, damit diese Aufgabe durchgeführt werden konnte. Wäre MOSS 2007 als Serverkomponente eingesetzt worden, wäre kein zusätzlicher Aufwand entstanden, da diese Funktionalität mit MOSS 2007 mitgeliefert wird.

VSTO eignet sich als Wrapper für Microsoft Outlook 2007 sehr gut da eine einfache Integration des CRM-Systems in Outlook möglich ist. Es werden alle benötigten Funktionalitäten bereitgestellt und die Implementierung gestaltet sich durch viele vorhandene Hilfsklassen sehr einfach. Zusätzlich profitiert das CRM-System dadurch von vorhandener Benutzererfahrung mit Outlook, wodurch das CRM-System einfacher zu bedienen ist und der benötigte Lernaufwand zur Verwendung des CRM-Systems sehr gering ausfällt.

Für die clientseitige Visualisierung der Daten wurde Silverlight v2.0 untersucht. Es stellte sich heraus, dass die Kombination von WSS und Silverlight einige Schwachstellen birgt und diese erst durch zusätzliche Implementierungen beseitigt werden konnten. Die Darstellung der Daten gestaltet sich mit Silverlight allerdings einfach, da vorhandene .NET Möglichkeiten sowie XAML zur Visualisierung verwendet werden können. Die Visualisierung der Daten könnte allerdings auch von jeder anderen .NET Technologie übernommen werden, da Silverlight keine Funktionen bereitstellt, die Silverlight von anderen Technologien abhebt.

7.2 Ausblick

Das CRM-System bietet eine Vielzahl an Möglichkeiten um zusätzliche Funktionalitäten erweitert zu werden. Derzeit ist es nur möglich, Daten zu bearbeiten, wenn eine Onlineverbindung besteht. Daten könnten jedoch vorab Offline verfügbar gemacht werden und lokal auf den Rechner des Benutzers geladen werden. Somit könnten diese auch Offline bearbeitet werden und bei der nächsten Verbindung mit dem CRM-System synchronisiert werden. Dadurch würde eine größere Flexibilität bei der Benutzung des CRM-Systems entstehen, da Mitarbeiter nicht immer einen Zugang zum System benötigen würden und trotzdem in der Lage wären, Kundendaten zu editieren.

Eine weitere Erweiterungsmöglichkeit des CRM-Systems wäre die Verfügbarkeit mittels Mobiltelefon. Microsoft arbeitet derzeit an einer Implementierung von „Silverlight Mobile“[1], welche ohne großen Mehraufwand, vorhandene Sil-

[1] http://silverlight.net/learn/mobile.aspx

verlightapplikationen auf Mobiltelefonen verfügbar machen soll. Damit wäre ein Benutzer nicht mehr an einen Rechner oder Laptop gebunden, sondern könnte direkt über das Mobiltelefon Informationen zu Kunden abrufen, ändern oder neue Kunden anlegen.

Literaturverzeichnis

[All02] Jeremy Allaire. *Macromedia Flash MX - A next-generation rich client.* Macromedia, 2002.

[And08] Ty Anderson. *Pro Office 2007 Development with VSTO.* Apress, 2008.

[Bru07] Margriet Bruggeman. *Pro SharePoint 2007 Development Techniques.* Apress, 2007.

[Ewe09] Eweek. Ria war is brewing. http://etech.eweek.com/content/application_development/ria_war_is_brewing.html, 01.02. 2009.

[Gre04] Paul Greenberg. *CRM at the Speed of Light - Essential Customer Strategies for the 21st Century.* McGraw-Hill Osborne Media, 2004.

[Gro07] Forrester Group. The fluent user interface: It decision-maker perception of productivity, training, and support requirements. Technical report, Forrester Group, 2007.

[Laa07] Kevin Laahs. *Microsoft SharePoint 2007 Technologies: Planning, Design and Implementation.* Digital Press, 2007.

[Leo07] Wynne Leon. *Microsoft SharePoint Server 2007 Bible.* John Wiley & Sons, 2007.

[Mac08] Matthew MacDonald. *Pro Silverlight 2 in C# 2008.* Apress, 2008.

[Mey09] Meyers. Microsoft office - wissen. http://lexikon.meyers.de/wissen/Microsoft+Office, 01.02. 2009.

[Mic08a] Microsoft. Building your sharepoint infrastructure. http://technet.microsoft.com/en-us/magazine/2008.05.insidesharepoint.aspx, 05. 2008.

[Mic08b] Microsoft. Windows sharepoint services 3.0. http://technet.microsoft.com/de-at/windowsserver/sharepoint/bb684455(en-us).aspx, 29.12. 2008.

[Mic08c] Microsoft. Windows sharepoint services 3.0 evaluation guide. http://technet.microsoft.com/en-us/windowsserver/sharepoint/bb400753.aspx, 29.12. 2008.

[Mic09a] Microsoft. Application structure silverlight. http://msdn.microsoft.com/en-gb/library/cc838120(VS.95).aspx, 03.02. 2009.

[Mic09b] Microsoft. Architecture of application-level add-ins. http://msdn.microsoft.com/en-us/library/bb386298.aspx, 08.01. 2009.

[Mic09c] Microsoft. Network security access restrictions in silverlight 2. http://msdn.microsoft.com/en-us/library/cc645032(VS.95).aspx, 04.02. 2009.

[Mic09d] Microsoft. Registry entries for application-level add-ins. http://msdn.microsoft.com/en-us/library/bb386106.aspx, 08.01. 2009.

[O'C07] Errin O'Connor. *Windows SharePoint Services 3.0 Inside Out*. Microsoft Press, 2007.

[Pyl07] James Pyles. *SharePoint 2007 The definitive Guide*. O'Reilly, 2007.

[Sca08] Jeff Scanlon. *Accelerated Silverlight 2*. Apress, 2008.

Abbildungsverzeichnis

Verzeichnis der Listings