Xml-Export-Tool

Projektdokumentation

Autor
Thomas Bosch

Ausbildungsberuf
Fachinformatiker
Anwendungs-
entwicklung

Datum
26.04.2004

Version
1.0

cellent.

Inhalt

Inhalt

1. Ausgangssituation

Eine namhafte deutsche Privatbank verwendet *‚SAP/VM'*, ein Kursplanungstool, welches den Teamleitern und Vorgesetzten ermöglicht, Seminardaten und -beschreibungen von verschiedenen Dienstleistungsanbietern einzusehen und ihre Mitarbeiter direkt auf diese Seminarangebote zu buchen.

Aus datenschutztechnischen Gründen kann der Name der deutschen Privatbank in dieser Projektdokumentation nicht genannt werden.

Am 4.November des Jahres 2003 bekamen wir von Herrn Raupach, der für die Findung neuer Weiterbildungsmöglichkeiten seiner Mitarbeiter der deutschen Privatbank zuständig ist, eine Anfrage. Da die Firma Cellent AG auch Seminare anbietet, möchten sich die Teamleiter und Vorgesetzte der deutschen Privatbank über diese informieren können und ihre Mitarbeiter gegebenenfalls in die Teilnehmerliste einiger Seminare einschreiben.

Herr Raupach schickte der Cellent AG alle nötigen Informationen, wie die Seminardaten aufzubereiten sind, damit sie in dem Kurplanungstool der deutschen Privatbank verwendet werden können.

Die Kosten und Abrechnung für diesen Auftrag sind nicht Bestandteil dieses Projekts und der Dokumentation, da ausschließlich die Geschäftsleitung solche Vorgänge abwickelt.

2. Projektumfeld

Ansprechpartner:

Herr Raupach	**deutsche Privatbank**
Herr Leis	**Cellent AG, Hüttlingen**
Herr Gronbach	**Cellent AG, Hüttlingen**
Herr Stegmaier	**Cellent AG, Hüttlingen**
Herr Huber	**Cellent AG, Hüttlingen**
Herr Kiendl	**Cellent AG, Hüttlingen**
Frau Gudynas	**Cellent AG, Hüttlingen**

Herr Raupach:

Mein Ansprechpartner für dieses Projekt war Herr Raupach, ein Mitarbeiter der deutschen Privatbank, welcher die Seminardaten des Unternehmens Cellent AG angefordert hat und in seinem Betrieb für die Weiterbildung seiner Mitarbeiter zuständig ist.

Herr Raupach ist in seiner Firma dafür verantwortlich, dass die Teamleiter und Vorgesetzten die Möglichkeit haben, zwischen verschiedenen Seminarangeboten von unterschiedlichen Kursanbietern zu wählen und ihre Mitarbeiter auf diese Seminare zu buchen.

Herr Christoph Gronbach:

Die Projektverantwortlichkeit wurde dem Teamleiter Christoph Gronbach des Unternehmens Cellent AG auferlegt.

An ihn konnte man sich wenden, wenn Probleme auftraten, die dieses Projekt betrafen.

Herr Alwin Stegmaier, Herr Heiko Huber, Herr Florian Kiendl:

Bei projektspezifischen Angelegenheiten konnten viele Mitarbeiter der Softwareentwicklungsabteilung der Cellent AG um Rat gebeten werden.

Darunter sind vor allem Heiko Huber, Alwin Stegmaier und Florian Kiendl zu erwähnen.

Frau Sabrina Gudynas:

Bei noch offenen und zu klärenden Fragen, die die eigentlichen Seminare betreffen, stand die Seminarbeauftragte der Firma Cellent AG, Sabrina Gudynas, zur Verfügung.

Projektumgebung:

Alle Tätigkeiten während des Projekts wurden in den Räumen der Softwareentwicklungsabteilung, der Cellent AG, durchgeführt.

Hardware:

- *‚Dell Latitude'*-Notebook
- Prozessor: 1 GHz
- Arbeitsspeicher: 512 MB
- Festplatte: 45 GB

Software:

- Betriebssystem: *‚Microsoft Windows XP Professional'*
- Entwicklungsumgebung: *‚Microsoft Visual Studio .NET 2003'*:

Erstellung des C#-Quellcodes

- *‚XMLSPY 5 Professional Edition'*:

  Betrachtung der Quelltexte der Xml-Dateien und des Xml-Schemas

- *‚SQL-Server 2000 Developer Edition'*:

  Anzeige der Seminardaten aus der Seminartabelle

3. Projektbeschreibung

Universelles Xml-Datenaustauschformat:

Das Seminarangebot der Firma Cellent AG soll in ein einheitliches Xml-Datenaustauschformat übermittelt werden, damit die Seminare in die Stammdaten des Planungstools eingestellt werden können.

Xml-Schema und Xml-Beispieldateien:

Das Xml-Schema, das die Regeln für die Struktur, die Anordnung, die möglichen Elemente, Attribute und Werte der Xml-Dateien der Seminarangebote definiert und die Xml-Beispieldateien wurden separat von Herrn Raupach zugeschickt.

Validator:

Außerdem schickte Herr Raupach der Cellent AG einen Validator, mit dessen Hilfe die erstellten Xml-Dateien auf formale Richtigkeit und Gültigkeit hin überprüfen werden konnten und die Fehlersuche in beträchtlichem Maße erleichtert wurde

Speicherort der Seminardaten:.

Die Seminardaten und -beschreibungen befinden sich in einer Tabelle einer Datenbank, die auf einem SQL-Server liegt.

Diese Daten können nicht eins-zu-eins übernommen werden, da viele Spalteninhalte entweder überhaupt keine Verwendung finden, dem Verwendungszweck entsprechend konvertiert werden müssen, oder internen Weiterverarbeitungszwecken dienen, die nach außer hin nicht publik gemacht werden dürfen.

Darüber hinaus sind die Daten, die Bestandteil der Xml-Dateien werden sollen, in dem mitgelieferten Xml-Schema definiert.

4. Projektziele

Anforderungen der deutschen Privatbank:

Das angestrebte Ziel dieses Projektes sollte primär sein, dass auch die deutsche Privatbank die Chance hat, sich für die angebotenen Seminare der Cellent AG zu interessieren, die Seminarbeschreibungen zu studieren und diese Seminare natürlich auch für ihre Mitarbeiter zu buchen.

Anforderungen der Cellent AG:

Das Unternehmen Cellent AG möchte, dass seine Mitarbeiter noch mehr Seminare abhalten und dadurch auch dessen Umsatz steigern.

5. Ist-Analyse

Im Rahmen einer Ist-Analyse werden alle für das Projekt relevanten Informationen ermittelt, um einen Überblick über die aktuelle Gesamtsituation des Projekts zu erhalten.

Teilnehmer der Ist-Analyse:

Herr Bosch	**Cellent AG, Hüttlingen**
Herr Leis	**Cellent AG, Hüttlingen**
Herr Gronbach	**Cellent AG, Hüttlingen**
Herr Stegmaier	**Cellent AG, Hüttlingen**

Speicherort der Seminardaten:

Die Seminardaten des Unternehmens Cellent AG befinden sich in nur einer Tabelle, die vom Entwickler der Datenbank, Jürgen Leis, mit *‚VER'* (für Veranstaltungen) benannt wurde.

Da es sich um nur eine Tabelle und nicht um mehrere Tabellen handelt, muss auf keine Beziehungen, die zwischen Tabellen bestehen können, geachtet werden.

Diese Tabelle befindet sich auf der Datenbank *‚S24_INT'* auf dem SQL-Server *‚ZEUS'*.

Suche nach den benötigten Seminardaten:

Die Seminarbeauftrage der Firma Cellent AG, Sabrina Gudynas, stellte eine Spalteninhaltsbeschreibung der Seminartabelle zur Verfügung.

So konnte die Zuordnung der bereits vorhandenen Seminardaten zu den, von der deutschen Privatbank erforderlichen Seminardaten vorgenommen werden.

Sind nützliche Referenzen oder vorhandenes Wissen über diesen Themenkomplex bereits vorhanden?

Da die Seminardaten in der Firma Cellent AG zuvor noch nie in ein einheitliches Xml-Datenaustauschformat gebracht wurden, konnte nicht auf bereits vorhandene Referenzen zurückgegriffen werden. Die für dieses Projekt notwendigen Prozessschritte mussten alle von Grund auf erarbeitet werden.

6. Soll-Analyse

Die Soll-Analyse beinhaltet alle Informationen, die zur erfolgreichen Realisierung des Projekts benötigt werden:

- Welche genauen Vorstellungen hat der Kunde?
- In welchem Zeitraum soll das Projekt durchgeführt werden?

Die Ist-Analyse ist dabei zumindest teilweise die Grundlage.

Teilnehmer der Soll-Analyse:

Herr Bosch	**Cellent AG, Hüttlingen**
Herr Leis	**Cellent AG, Hüttlingen**
Herr Gronbach	**Cellent AG, Hüttlingen**
Herr Stegmaier	**Cellent AG, Hüttlingen**

einzelne Prozessschritte des Projekts:

Das vollständige Seminarangebot des Unternehmens Cellent AG soll in ein einheitliches Xml-Datenaustauschformat konvertiert werden. Die Seminardaten befinden sich in einer Tabelle einer Datenbank, die sich auf einem SQL-Server befindet.

Aus dieser Tabelle müssen entsprechend dem mitgelieferten Xml-Schema, Daten selektiert und gegebenenfalls umgewandelt werden.

Schnittstelle:

Dazu wird eine Schnittstelle benötigt, die diese Selektionen und Konvertierungen programmatisch vornimmt.

Als Schnittstelle wurde gemeinsam von den Teilnehmern der

Soll-Analyse die Microsoft Visual Studio .Net-Entwicklungsumgebung in Verbindung mit der Programmiersprache C# festgelegt.

grafische Benutzeroberfläche:

Ob die grafische Benutzeroberfläche nun als Windows-Applikation oder als Web-Anwendung realisiert wird, sollte im Laufe der Entwicklung entschieden werden bzw. im Laufe der fortgeschrittenen Planung des Projektes.

Xml-Dateien:

Im Zuge der Anforderungen der deutschen Privatbank sollen zwei Arten von Xml-Dateien generiert werden.

Die einen Xml-Dateien sollen die Seminarbeschreibungen und die anderen sollen letztendlich Informationen über die einzelnen Veranstaltungen eines jeden Seminars enthalten.

Die Seminarbeschreibungen sind in dem Verzeichnis *‚records'* und die Veranstaltungen in dem Verzeichnis *‚events'* aufzulisten.

Diese Angaben wurden alle von Herrn Raupach von der deutschen Privatbank zugeschickt und sind dementsprechend als Auflagen zu betrachten.

Sind alle Xml-Dateien den Vorlagen entsprechend erstellt, sollen sie der namhaften deutschen Privatbank entweder per Diskette, CD oder über den E-Mail-Versand gebracht werden, nachdem sie von dem Validierungsprogramm, dem so genannten Validator, auf formale Richtigkeit und Gültigkeit hin überprüft wurden.

Mit Hilfe des proprietären Validierungsprogramms soll darüber hinaus auch die anschließende Fehlersuche erleichtert werden.

Ziele des Projekts:

Die Xml-Dateien des Seminarangebots der Cellent AG können daraufhin in die Stammdaten des Kursplanungstools der deutschen Privatbank eingestellt werden.

Ist das geschehen, haben die Teamleiter und Vorgesetzten der deutschen Privatbank die Möglichkeit, die Seminardaten und -beschreibungen einzusehen und ihre Mitarbeiter auf diese Seminarangebote zu buchen.

zukünftige Folgeprojekte:

Ein weiterer Agendapunkt der Soll-Analyse war auch noch die zukünftige Verwendbarkeit und Skalierbarkeit dieses Projektes.

Dieses Projekt soll als Referenz und auch als Basis für zukünftige Seminardatenbereitstellungen verwendet werden.

Es soll auch anderen Unternehmen, außer der deutschen Privatbank, möglich sein, die Seminarinhalte der Cellent AG zu betrachten und daraus ein gehobenes Interesse zu entwickeln, das die Grundlage potentieller Seminarbuchungen darstellt.

7. Entwurf

7.1 Anforderungen

7.1.1 Anforderungsunterlagen und -beschreibungen

Die deutsche Privatbank schichte folgende gepackte Zip-Dateien an das Unternehmen Cellent AG:

- *‚anbieterPack.zip'*:
  - *‚Anbieterhandbuch.pdf'*:

    LOMeX-Konverter-Anbieterhandbuch
  - *‚JAVA-E.xml'*:

    Beispiel-Xml-Datei, die die einzelnen Veranstaltungen des in der Beispiel-Xml-Datei *‚JAVA-R.xml'* beschriebenen Seminars zusammenfast.
  - *‚JAVA-R.xml'*:

    In dieser Beispiel-Xml-Datei wird ein einzelnes Seminar beschrieben.
  - *‚LOMeX_seminar.dtd'*:

    DTD der Xml-Dateien
  - *‚LOMeX_seminar.xsd'*:

    Xml-Schema der Xml-Dateien
- *‚validator.zip'*:

  Das Validierungsprogramm, der Validator, der die Xml-Dateien mit Hilfe des Xml-Schemas auf ihre Gültigkeit hin überprüft. Die Fehlersuche in den Xml-Dateien wird durch dieses Hilfsprogramm stark vereinfacht.

7.1.1.1 LOMeX-Konverter

- Beschreibung:

  Dieses Programm soll die zuvor erstellten Xml-Dateien der Seminardaten externer Dienstleistungsanbieter zur Weiterverwendung in dem SAP-System der deutschen

Privatbank verarbeiten.

- Ziele:

  Dieses Programm integriert Seminarstammdaten von externen Seminaranbietern in das SAP-System der deutschen Privatbank.

- Architektur:

  Das Programm LOMeX bekommt von Seminaranbietern Beschreibungen der extern bei der deutschen Privatbank angebotenen Seminare als Eingabe. Diese Beschreibungen werden in mehreren Xml-Dateien geliefert. In einem weiteren Verarbeitungsschritt werden die eingelesenen Xml-Daten verarbeitet und Regeln unterworfen. Anschließend werden die Seminardaten in das SAP-System geladen.

7.1.2 Forderungen an die Struktur der Xml-Dateien

zwei Arten von Xml-Dateien:

Es sollen zwei verschiedene Arten von Xml-Dateien erstellt werden.

- Seminarbeschreibungen:

  Xml-Dateien, die die Beschreibung eines bestimmten Seminars beinhalten. Jedes Seminar soll in einer separaten Xml-Datei beschrieben werden. Die Notation für den Xml-Dateinamen ist*: ‚Seminarkürzel-R.xml'*. Alle Xml-Dateien, die die Beschreibungen der einzelnen Seminare enthalten, sollen in dem Verzeichnis *‚records'* aufgelistet werden.

- Veranstaltungsbeschreibungen:

  Xml-Dateien, die die verschiedenen Veranstaltungen eines bestimmten Seminars beinhalten. Ein Seminar kann öfters als ein Mal pro Jahr stattfinden. Deshalb ist es nur logisch, wenn diese Daten getrennt aufgenommen und behandelt werden. In einer solchen Xml-Datei sollen alle zu einem Seminar gehörenden Veranstaltungen aufgelistet werden. Solche Daten sind z.B. der Veranstaltungebeginn, das Veranstaltungsende und die Dauer der Veranstaltung. Die Notation für den Xml-Dateinamen wurde wie folgt definiert: *‚Seminarkürzel-E.xml'*. Alle Xml-Dateien, die die Beschreibungen der einzelnen Veranstaltungen eines bestimmten Seminars enthalten, sollen in dem Verzeichnis *‚events'* gespeichert werden.

Xml-Schema *‚LOMeX_seminar.xsd':*

Das mitgelieferte Xml-Schema *‚LOMeX_seminar.xsd'* beinhaltet

die in den Xml-Dateien erforderlichen Daten, wie sie in der Gesamtstruktur anzuordnen sind, welche Reichenfolge einzuhalten ist und welche gültigen Werte erlaubt sind. Dieses Xml-Schema muss auch jeweils in den Verzeichnissen *‚records'* und *‚events'*, in denen sich auch die Xml-Dateien befinden, vorhanden sein, damit eine Gültigkeitsprüfung vorgenommen werden kann.

7.2 Bestehende und erforderliche Datenbankinhalte

Speicherort der Seminardaten:

- SQL-Server:

  ‚ZEUS'

- Datenbank:

  ‚S24_INT'

- Tabelle:

  ‚VER'

  → Die kompletten Seminardaten befinden sich in nur einer Tabelle.

  → Da die gesamten Seminardaten und -beschreibungen in nur einer Tabelle zusammengefasst sind, entfällt die Komplexität der Beziehungsdefinitionen und -berücksichtigungen.

  → Dadurch befinden sich diese Daten aber nicht in der 3. Normalform, was aber für dieses Projekt nicht von Belang ist.

Spalteninhaltsbeschreibungen der Tabelle *‚VER':*

Die Beschreibungen der Inhalte der einzelnen Spalten konnten von der Seminarverantwortlichen der Cellent AG, Sabrina Gudynas, zur Verfügung gestellt werden. Sie nutzt ein Programm, mit dessen Hilfe sie die Erklärungen der Inhalte der einzelnen Spalten der Seminartabelle bekommt und diese so schnell auswerten kann.

erforderliche Datenbankinhalte:

Die aus der Seminartabelle erforderlichen Seminardaten werden in dem, Xml-Schema beschrieben, das die Cellent AG von der deutschen Privatbank erhalten hat.

7.3 Benutzeroberfläche

7.3.1 Auswahl der Benutzeroberfläche

Die grundlegende Entscheidung, bevor in *‚Microsoft Visual Studio.NET'* begonnen wird zu programmieren, sollte immer die Auswahl der am besten für die Problemstellung beeignete Benutzeroberfläche sein. Es bestehen zwei Möglichkeiten, die Benutzeroberfläche in *‚MS Visual Studio .NET'* zu gestalten:

1. Windows-Applikation:

   Eine Windows-Applikation ist

   - einfacher zu realisieren,
   - weniger komplex und mit
   - bedeutend weniger Aufwand umzusetzen

   als eine Web-Anwendung.

2. Web-Applikation:

   Hier wird mit *‚ASP.NET'*, einem herausragenden Teil der *‚.NET-Technologie'* entwickelt und man kann sich dessen erweiterter Funktionalität bedienen.

Vorteile einer Web-Anwendung:

- Eine Web-Anwendung kann mit einem Browser von jedem Computer aus geöffnet werden.
- Um eine Web-Applikation zu verwenden, muss das Programm nicht wie bei der Windows-Anwendung extra installiert werden.
- Der Web-Server, auf dem sich das Programm befindet, wird zentral verwaltet. Kommt es zu Änderungen am Programm, müssen diese Änderungen nur ein Mal zentral auf dem Web-Server vorgenommen werden und nicht auf jedem einzelnen Client-Rechner, auf dem die Windows-Applikation installiert ist.
- Eine Web-Anwendung ist unabhängig vom verwendeten Betriebssystem und der vorhandenen Hardware aufrufbar. Dieser Sachverhalt wird auch *‚Portabilität'* genannt.

Aufgrund dieser Vorteile, habe ich mich für eine Web-Applikation als Benutzerschnittstelle entschieden.

7.3.2 Gestaltung der Benutzeroberfläche

- Interaktion des Benutzers:

  Nun war noch die Frage zu klären, in wie weit der Benutzer

dieses Programms interagieren und festlegen kann, was genau geschehen soll. Welche Möglichkeiten sollen dem Anwender gegeben werden?

Der Benutzer oder Anwender dieses Programms ist derjenige, der die Xml-Dateien erstellen soll.

Der Anwender soll eine Benutzeroberfläche präsentiert bekommen, die auf dem ersten Blick erkennen lässt, welche Funktion das vorliegende Programm hat und wie man es intuitiv und ohne vorhergehende Erklärung bedienen kann.

- Speicherort der Tabelle:

  Der Anwender soll die Möglichkeit bekommen, den Speicherort der Tabelle anzugeben, in der sich die Seminardaten befinden, also den Namen des SQL-Servers, der Datenbank und der Tabelle. Zu Beginn, wenn das Programm gestartet wird, sollen bereits Standardwerte in den Eingabefeldern stehen, falls der Anwender diese Angaben nicht kennt oder keine Änderungen vornehmen möchte. Ändert sich z.B. der Name des SQL-Servers, dann kann der Benutzer einen anderen Namen eingeben, oder es wird der Standard-SQL-Server-Name auf den neuen Wert geändert.

- Speicherorte der Xml-Dateien:

  Zusätzlich soll der Anwender die Speicherorte der Xml-Dateien, einerseits für die Seminarbeschreibungen und andererseits für die Veranstaltungen der einzelnen Seminare, bestimmen können. Auch diese Eingabefelder sollen zu Beginn mit Standardwerten gefüllt sein.

- Steuerelemente:

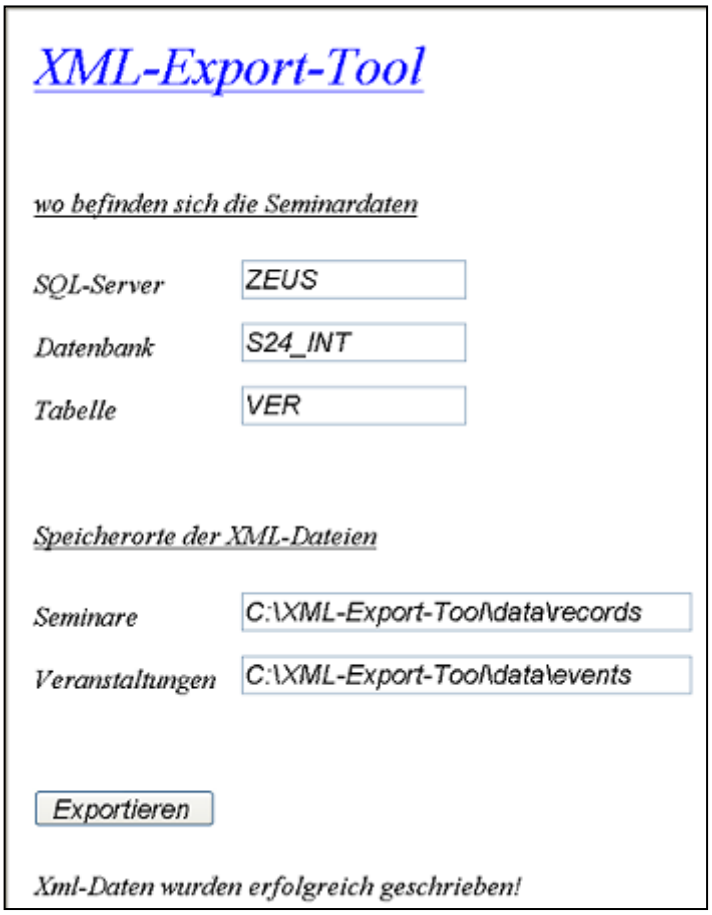
Damit die Seminardaten, deren Herkunftsort zuvor bestimmt wurde, in die Verzeichnisse, die ebenfalls vom Anwender zuvor festgelegt wurden, geschrieben werden, soll auf einen Button geklickt werden.

Nach erfolgreichem Schreiben der Seminardaten in die Xml-Dateien soll ein Meldungstext erscheinen, der den Benutzer davon in Kenntnis setzt.

8. Realisierung

8.1 Benutzeroberfläche

Die Benutzeroberfläche wird dem Entwurf entsprechend erstellt.



Namensvergabe der einzelnen Steuerelemente:

Die Namen für die einzelnen Steuerelemente bestehen aus einem Kürzel und einer kurzen Beschreibung der Funktion, die das Steuerelement erfüllen soll.

Z.B. soll in das Textfeld *, txtVerzeichnispfadSeminare'* der Verzeichnispfad für die Xml-Dateien, die die Seminarbeschreibungen beinhalten, eingegeben werden. Wenn im C#-Code auf den Inhalt dieses Steuerelements zugegriffen werden soll, weiß der Entwickler sofort und intuitiv, wenn er den Namen des Steuerelements betrachtet, was dieses Steuerelement für eine Aufgabe hat, warum und wie es verwendet werden soll.

Namenskürzel:

- Labels, Textfelder: *,lbl'*

- Eingabefelder: *‚txt'*
- Buttons: *‚btn'*

8.2 Namespaces einbinden

Für die im Programm verwendeten Klassen müssen die Namespaces am Anfang des C#-Quellcodes angegeben werden, damit die Klassen dem Compiler bekannt gemacht werden können.

8.3 Code-Bereiche verwenden

```
#region globale Variablendeklarationen

    Steuerelemente

    Methode 'OpenConnectionToDatabase()'

    #region Methode 'FillDataSet'

        private System.Data.SqlClient.SqlCommand cmd;
        private System.Data.SqlClient.SqlDataAdapter dataadapter;
        private System.Data.DataSet dataset;
        private string SqlQuery;

    #endregion

    Methode 'Set_SemVer_SemKuerzel_AnzahlVerProSem()'

    Methode 'WriteXmlData()'

#endregion
```

Beschreibung:

Der besseren Übersicht im C#-Quellcode halber, wurden Code-Fragmente, die logisch zueinander gehören, in einzelne Bereiche unterteilt. Ein Bereich kann mehrere Unterbereiche in sich vereinen, die wiederum mehrere Unterbereiche in sich haben können.

Vorteile:

- Ein Vorteil dieser Vorgehensweise liegt darin, dass der C#-Quellcode in seiner Gesamtheit betrachtet wesentlich übersichtlicher wird und dass Bereiche, die logisch miteinander in Verbindung stehen, zusammengefasst werden.
- Betrachtet ein anderer Entwickler diesen Code, kann er ihn Schritt für Schritt durchgehen und wird ihn mit Sicherheit viel schneller verstehen, als dies ohne die Anwendung dieses Programmierstils der Fall wäre.

8.4 Globale Variablendeklarationen

Fast alle im Programm verwendeten Variablen (außer die Zählvariablen in den Zählschleifen) werden global deklariert. Auf lokale Variablendeklarationen wurde vorausschauend verzichtet, da dies einige Probleme mit sich bringen kann.

Vorteile globaler Variablendeklarationen:

Wird in einer Methode eine Variable gebraucht, verwendet und ihr Inhalt geändert, müsste diese Variable bei einer lokalen Variablendeklaration von Methode zu Methode als Parameter übergeben werden, damit ihr Inhalt nicht verworfen wird. Das würde sich aber als äußerst unübersichtlich und auch unnötig erweisen. Deshalb bedient sich der Entwickler, der dies bedacht hat, globaler Variablendeklarationen. Wird der Inhalt der Variablen in einer Methode verändert, bleibt ihr Inhalt auch beim Verlassen der Methode erhalten.

8.5 Programmablauf

8.5.1 UML-Diagramm der Projektklasse

Eine Web-Applikation besteht in der Regel nur aus einer Klasse, die die verwendeten Eigenschaften, Methoden sowie die Ereignismethoden enthält. Dabei werden auch die Steuerelemente, die auf der Benutzeroberfläche platziert wurden, als Membervariablen deklariert.

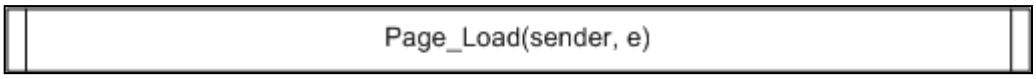
WebForm1

lblÜberschrift : System.Web.UI.WebControls.Label
lblSqlServer : System.Web.UI.WebControls.Label
lblDatenbank : System.Web.UI.WebControls.Label
lblTabelle : System.Web.UI.WebControls.Label
lblErfolgreichGeschrieben : System.Web.UI.WebControls.Label
lblSpeicherorteXmlDateien : System.Web.UI.WebControls.Label
lblVerzeichnispfadSeminare : System.Web.UI.WebControls.Label
lblVerzeichnispfadVeranstaltungen : System.Web.UI.WebControls.Label
lblQuelleSeminardaten : System.Web.UI.WebControls.Label
txtSqlServer : System.Web.UI.WebControls.TextBox
txtDatenbank : System.Web.UI.WebControls.TextBox
txtTabelle : System.Web.UI.WebControls.TextBox
txtVerzeichnispfadSeminare : System.Web.UI.WebControls.TextBox
txtVerzeichnispfadVeranstaltungen : System.Web.UI.WebControls.TextBox
btnExportieren : System.Web.UI.WebControls.Button
- ConnString : String
- conn : System.Data.SqlClient.SqlConnection
- cmd : System.Data.SqlClient.SqlCommand
- dataadapter : System.Data.SqlClient.SqlDataAdapter
- dataset : System.Data.DataSet
- SqlQuery : String
- SemVer : Array von String-Elementen
- SemKuerzel : Array von String-Elementen
- CurrentTempString : Array von String-Elementen
- AnzahlVerProSem : Array von Integer-Elementen
- counter : Integer
- Encoding : System.Text.UnicodeEncoding
- XmlSemWriter : System.Xml.XmlTextWriter
- XmlVerWriter : System.Xml.XmlTextWriter
- ArrayList alstReplace : System.Xml.XmlTextWriter
- row : DataRow
- Seminardateipfad : String
- Veranstaltungsdateipfad : String
- StringToEdit : String
- DateString : Array von String-Elementen
- Separator : Array von char-Elementen
- index : Integer
- InhaltLaenge : Integer
- oReg : Regex
- oMatch : Match
- decimalvalue : decimal
- MailString : Array von String-Elementen

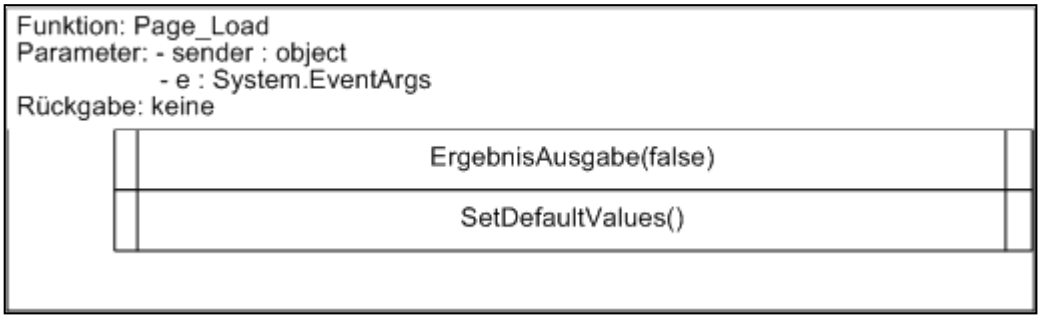
- Page_Load(object sender, System.EventArgs e) : void
- SetDefaultValues() : void
- ErgebnisAusgabe(bool sichtbar) : void
- OpenConnectionToDatabase() : void
- FillDataSet() : void
- Set_SemVer_SemKuerzel_AnzahlVerProSem() : void
- WriteXmlData() : void
- RemoveChars() : void
- ReplaceChars() : void
- CloseConnectionToDatabase() : void
- btnExportieren_Click(object sender, System.EventArgs e) : void
OnInit(EventArgs e) : void
- InitializeComponent() : void

8.5.2 Die *‚Page_Load'*-Ereignismethode

Wenn das Programm gestartet wird, wird zu allererst die Methode *‚Page_Load'* aufgerufen.



In ihr wird die Methode *‚ErgebnisAusgabe'* aufgerufen und gleichzeitig ein Parameter mit übergeben.



Diese Methode blendet die Ergebnismeldung „Xml-Daten wurden erfolgreich geschrieben!“ auf der Benutzeroberfläche in Abhängigkeit des zu übergebenden Parameters entweder ein oder aus. Wird wie in der *‚Page_Load'*-Methode der logische Operator *‚false'* übergeben, bleibt das Steuerelement unsichtbar. Es soll ja nur angezeigt werden, wenn die Xml-Dateien erfolgreich geschrieben wurden. Dazu wird der Eigenschaft *‚Visible'* des Label-Steuerelements entweder auf *‚true'* (wahr) oder *‚false'* (falsch) gesetzt, je nachdem, ob das Control angezeigt werden soll, oder nicht.

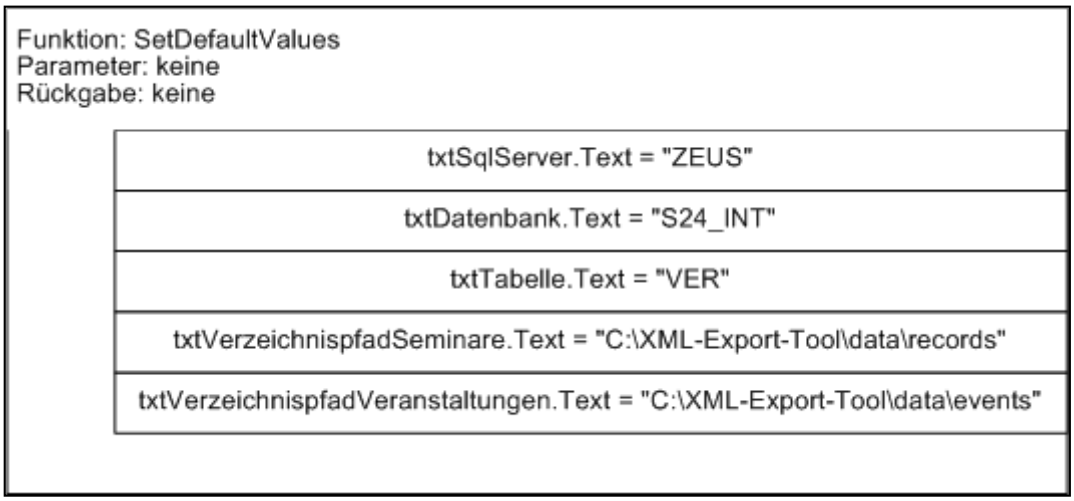
Vorteile von Methoden:

- Diese Methode wird zwei Mal in diesem Programm benötigt. Nehmen wir an, wir würden diese Methode hundert Mal in unserem Programm aufrufen. Würde der C#-Quellcode nicht in dies Methode ausgelagert werden, stände er hundert Mal an verschiedenen Stellen in unserem Programm. Im Quellcode stände hundert Mal derselbe Quelltext. Unser Programm wäre wirklich sehr umfangreich, nicht was die eingebaute Funktionalität angeht, sondern nur

in Bezug auf die Anzahl der geschriebenen Zeilen. Das wäre nicht weiter schlimm, gäbe es da nicht noch einen anderen, weitaus gewichtigeren Grund, häufig verwendeten Quelltext in Methoden auszulagern.

- Ändert sich ein Teil dieses Quelltextes, müsste an allen hundert Stellen im Programm die Änderung manuell vorgenommen werden. Das hätte eine erhöhte Zeitaufwendung und Fehleranfälligkeit (Vertippen) zur Folge. Es lohnt sich also durchaus, in sich abgeschlossene, öfter verwendete Funktionalität in einer Methode zu modularisieren, die später dann auch in anderen Projekten ausgelagert und wieder verwendet werden kann.

Nachdem die Methode *‚ErgebnisAusgabe'* beendet und zur aufrufenden Methode *‚Page_Load'* gewechselt wurde, wird die *‚SetDefaultValues'*-Methode aufgerufen.



Hier werden die Standard-Werte definiert, mit denen die Steuerelemente zu Beginn des Programms initialisiert werden sollen.

→ nun erscheint die Bildschirmausgabe der Web-Applikation

Jetzt können der Herkunftsort der Seminardaten (SQL-Server, Datenbank, Tabelle) und der Speicherort der Xml-Dateien (Seminarbeschreibungen und Veranstaltungsbeschreibungen) angegeben oder die bereits angezeigten Standard-Werte verwendet werden.

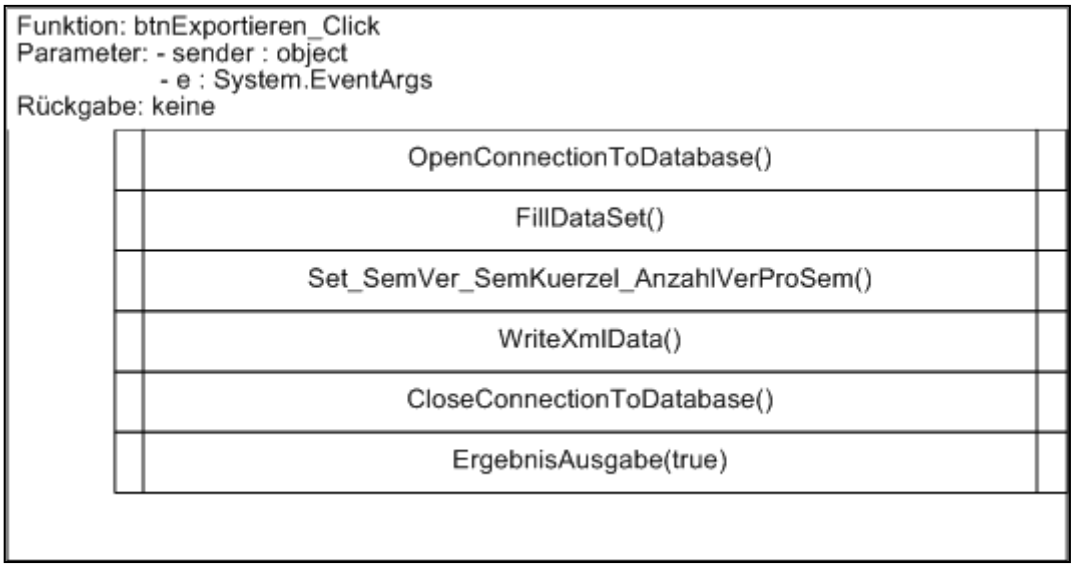
→ wird auf den *‚Exportieren'*-Button geklickt, sollen nun schließlich die Xml-Dateien erstellt werden.

8.5.3 Die Ereignisbehandlungsroutine *‚btnExportieren_Click'*

Wenn auf den Button ‚Exportieren' geklickt wird, werden nacheinander verschiedene Methoden aufgerufen, damit schließlich die Xml-Dateien geschrieben werden.

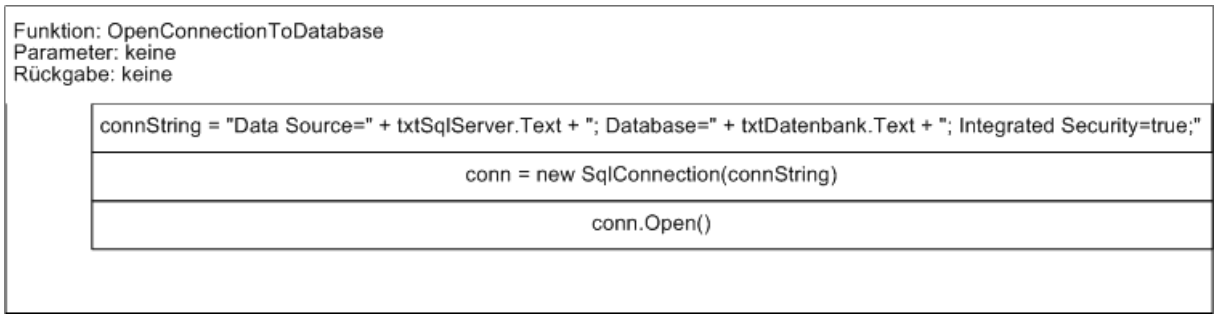
Namenvergabe der Methoden und der Variablen:

Grundsätzlich bemühen sich sorgfältige Entwickler darum, die Namen ihrer verwendeten Variablen und Methoden so zu wählen, dass man weiß, für welchen Zweck die Variable oder die Methode angelegt wurde. Man soll intuitiv und ohne Hilfe des Entwicklers des Programms erkennen können, für was die Variable oder die Methode verwendet werden kann.



Zuerst wird die Methode *‚OpenConnectionToDatabase'* aufgerufen.

8.5.4 Die Methode *‚OpenConnectionToDatabase'* – eine Verbindung zur Datenbank erstellen



Voreinstellungen:

Um eine Verbindung zu einem SQL-Server herzustellen, sind

diverse Vorabeinstellung notwendig.

1. Modifikationen in der Konfigurationsdatei *‚web.config'*:

```
<identity impersonate="true" />
<authentication mode="Windows" />
```

   → in IIS (Internet Information Services) soll die Windows-Authentifizierung integriert werden.

2. Die anonyme Benutzerauthentifizierung durch den IIS-Standard-Benutzer soll durch eine Windows-Authentifizierung ausgetauscht werden:

   Start -> Control Panel -> Administrative Tools -> Internet Information Services -> local computer -> Web Sites -> Default Web Site -> rechte Maustaste auf *'XML-Export-Tool'* -> Properties -> Registerkarte *'Directory Security'* -> Anonymous access and authentication control -> Button *'edit'* drücken -> Check-Box *'Anonymous access'* deaktivieren

Verbindungsaufbau zur Datenbank:

Zuerst muss eine Instanz der Klasse *‘SqlConnection’* erstellt werden. Hier wird ihr der Name *‚conn’* zugewiesen. Dem Konstruktor dieser Klasse wird ein so genannter *‚ConnectionString’* übergeben. In dem *‚ConnectionString’* wird der Name des SQL-Servers und der Datenbank angegeben. Des Weiteren ist darauf zu achten, dass auch hier die Windows-Authentifizierung verwendet wird.

```
connString = @"Data Source=" + txtSqlServer.Text + "; Database=" + txtDatenbank.Text + "; Integrated Security=true;"
```

Das @-Zeichen vor dem *‚ConnectionString’* gibt an, dass die Escape-Sequenzen in der nachfolgenden Zeichenkette vom Compiler nicht interpretiert, also ignoriert werden sollen. Zum Abschluss dieser Methode wird die zuvor definierte Verbindung geöffnet.

Gründe für die Windows-Authentifizierung:

Bevor eine Verbindung zu einer Datenbank aufgebaut werden kann, muss unbedingt angegeben werden, dass die Anmeldedaten des Benutzers, der das Programm gestartet hat, für die Authentifizierung verwendet werden sollen.

Es sollen nicht die IIS-Standard-Benutzerdaten verwendet werden, da diesen normalerweise keine Berechtigungen

eingeräumt wurden, auf eine sich im Netzwerk befindliche Datenbank zuzugreifen.

Damit auf der Benutzeroberfläche dynamisch der Herkunftsort der Seminardaten eingestellt werden kann, wird der *‚ConnectionString'* mit Hilfe der *‚Text'*-Eigenschaft der Steuerelemente zur Laufzeit angepasst.

8.5.5 Die Methode ‚FillDataSet'

Funktion: FillDataSet
Parameter: keine
Rückgabe: keine

SqlQuery = "SELECT * FROM " + txtTabelle.Text + " WHERE (NUMMER LIKE 'MOC%' OR NUMMER LIKE 'WSE%' OR NUMMER LIKE 'ISC%') " + "AND STATUSTEXT NOT LIKE 'abgesagt' " + "AND FIRMA_NAME1 = ''"
cmd = new SqlCommand(SqlQuery, conn)
dataadapter = new SqlDataAdapter()
dataadapter.SelectCommand = cmd
dataset = new DataSet()
dataadapter.Fill(dataset, "Seminardaten")

Nachdem eine Verbindung zur Datenbank hergestellt und diese geöffnet wurde, kann das Resultset einer Datenbankabfrage in ein *‚DataSet'* kopiert werden.

Zuerst muss die SQL-Query definiert werden, die gegen die Datenbank gefahren werden soll. Sie lautet folgendermaßen:

```
SqlQuery = "SELECT * " +
           "FROM " + txtTabelle.Text + " " +
           "WHERE (NUMMER LIKE 'MOC%' OR NUMMER LIKE 'WSE%' OR NUMMER LIKE 'ISC%') " +
           "AND STATUSTEXT NOT LIKE 'abgesagt' " +
           "AND FIRMA_NAME1 = ''";
```

Aufgabe dieser SQL-Abfrage:

Es sollen alle Datensätze mit allen vorhandenen Spalten von dieser Abfrage zurückgegeben werden, die sich in der Tabelle befinden, deren Name zuvor in dem Textfeld-Steuerelement eingegeben wurde (txtTabelle.Text) und die den drei angegebenen Bedingungen genügen:

1. In der Spalte *‚NUMMER'* steht das Seminarkürzel zusammen mit der Veranstaltungsnummer. Ein Seminar

kann hier also öfter aufgelistet sein, wenn es mehrmals abgehalten wird. Es sollen nur diejenigen Datensätze zurückgegeben werden, deren Seminarkürzel entweder mit *‚MOC'*, *‚WSE'* oder *‚ISC'* beginnen. Die anderen Seminare sollen den Kunden nicht zugänglich gemacht werden, da diese bei der anfragenden Firma stattfinden und deshalb anderen Kunden nicht zur Verfügung stehen.

2. Die Spalte *‚STATUSTEXT'* beinhaltet drei mögliche Werte: *‚abgesagt'*, *‚findet statt'* oder *‚geplant'*. Es macht keinen Sinn, die abgesagten Seminare mit einzubeziehen.
3. Die dritte Bedingung besagt, dass der Inhalt der Spalte *‚FIRMA_NAME1'* leer sein muss. Ist in dieser Spalte ein Firmenname eingetragen, so bedeutet das, dass dieses Seminar bei diesem Kunden stattfinden wird. Warum dies nicht erwünscht ist, kann aus der ersten Bedingung weiter oben erschlossen werden.

Durch diese Anweisung wird mit Hilfe der bestehenden Verbindung die SQL-Query gegen die Datenbank gefahren:

```
cmd = new SqlCommand(SqlQuery, conn);
```

Es wird ein Objekt *‚cmd'* von der Klasse *‚SqlCommand'* instanziiert, das nach dem Ausführen dieses Statements das Resultset der Datenbankabfrage enthält.

Das Dataset füllen:

Nun muss dieses Ergebnis der Abfrage in irgendeiner Form gespeichert werden. Dies kann durch ein *‚SqlDataAdapter'*-Objekt erfolgen, das eine Instanz der Klasse *‚DataSet'* mit dem Ergebnis der Datenbankabfrage füllt:

```
dataadapter = new SqlDataAdapter();
```

→ ein Objekt der Klasse *‚SqlDataAdapter'* wird erstellt

```
dataadapter.SelectCommand = cmd;
```

→ Die Instanz *‚dataadapter'* der Klasse *‚SqlDataAdapter'* wählt durch dessen Eigenschaft *‚SelectCommand'* die Query aus (Objekt *‚cmd'*).

```
dataset = new DataSet();
```

→ ein neues *‚DataSet'*-Objekt wird erstellt

```
dataadapter.Fill(dataset, "Seminardaten");
```

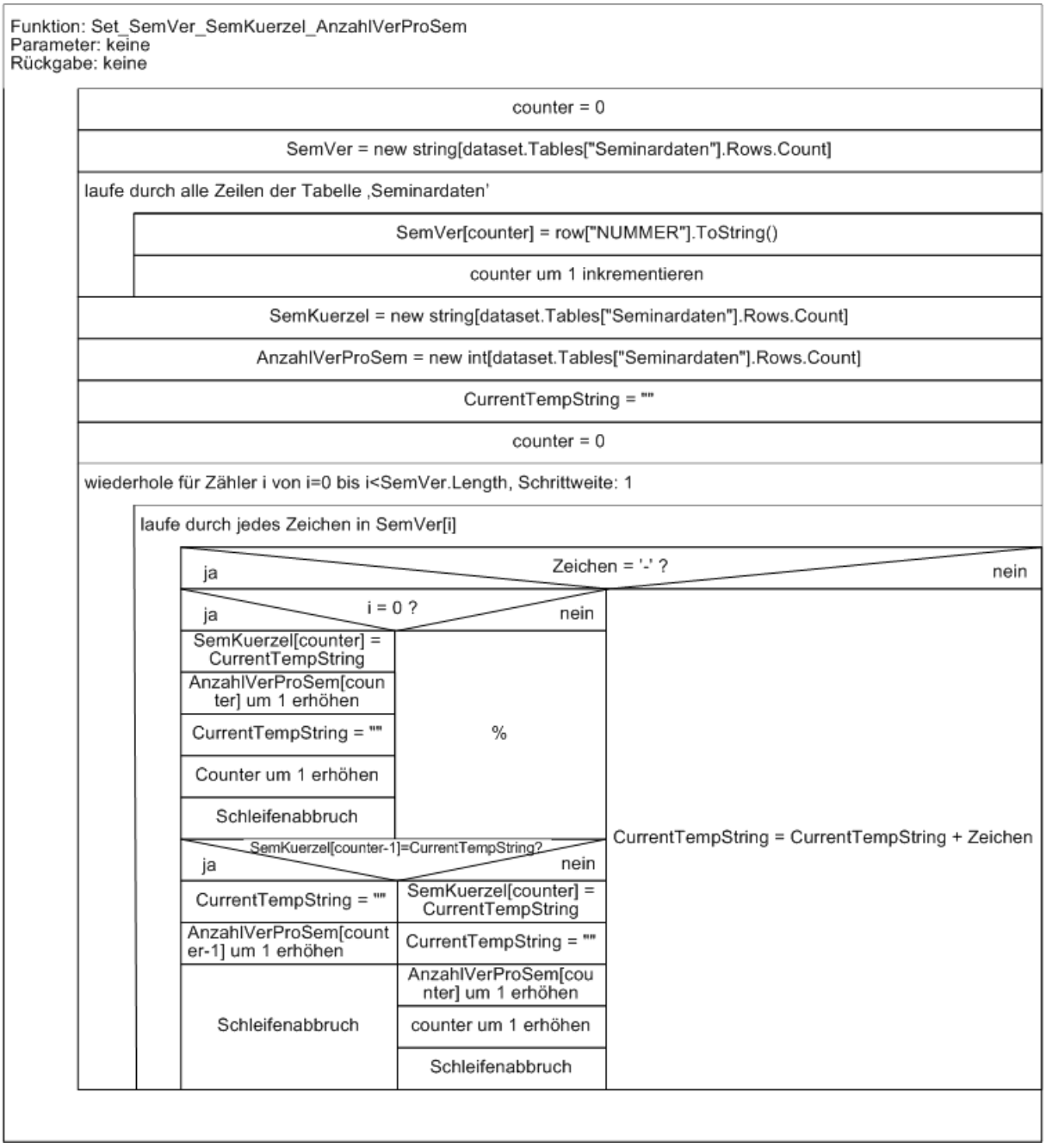
→ Das *‚SqlDataAdapter'*-Objekt *‚dataadapter'* füllt die *‚DataSet'*-Instanz *‚dataset'* mit den Ergebnisdatensätzen der SQL-Query und schreibt sie in die Tabelle *‚Seminardaten'* des *‚DataSet'*.

8.5.6 Methode 'Set_SemVer_SemKuerzel_AnzahlVerProSem

Aufgabe dieser Methode:

Diese Methode füllt die Zeichenkettenarrays *‚SemVer'* und *‚SemKuerzel'* und das Ganzzahlarray *‚AnzahlVerProSem'*.

- *‚SemVer'* → Kombination aus Seminarkürzel und Veranstaltungsnummer
- *‚SemKuerzel'* → Seminarkürzel
- *‚AnzahlVerProSem'* → Anzahl Veranstaltungen pro Seminar



Aufgaben der Methode
‚Set_SemVer_SemKuerzel_AnzahlVerProSem':

- Zuerst wird dynamisch zur Laufzeit ein neues *Stringarray ‚SemVer'* erstellt, das genauso viele Elemente beinhalten soll wie es Zeilen in der Tabelle *‚Seminardaten'* gibt. In diesem Stringarray sollen ja die Kombinationen aus Seminar- und Veranstaltungskürzel stehen.

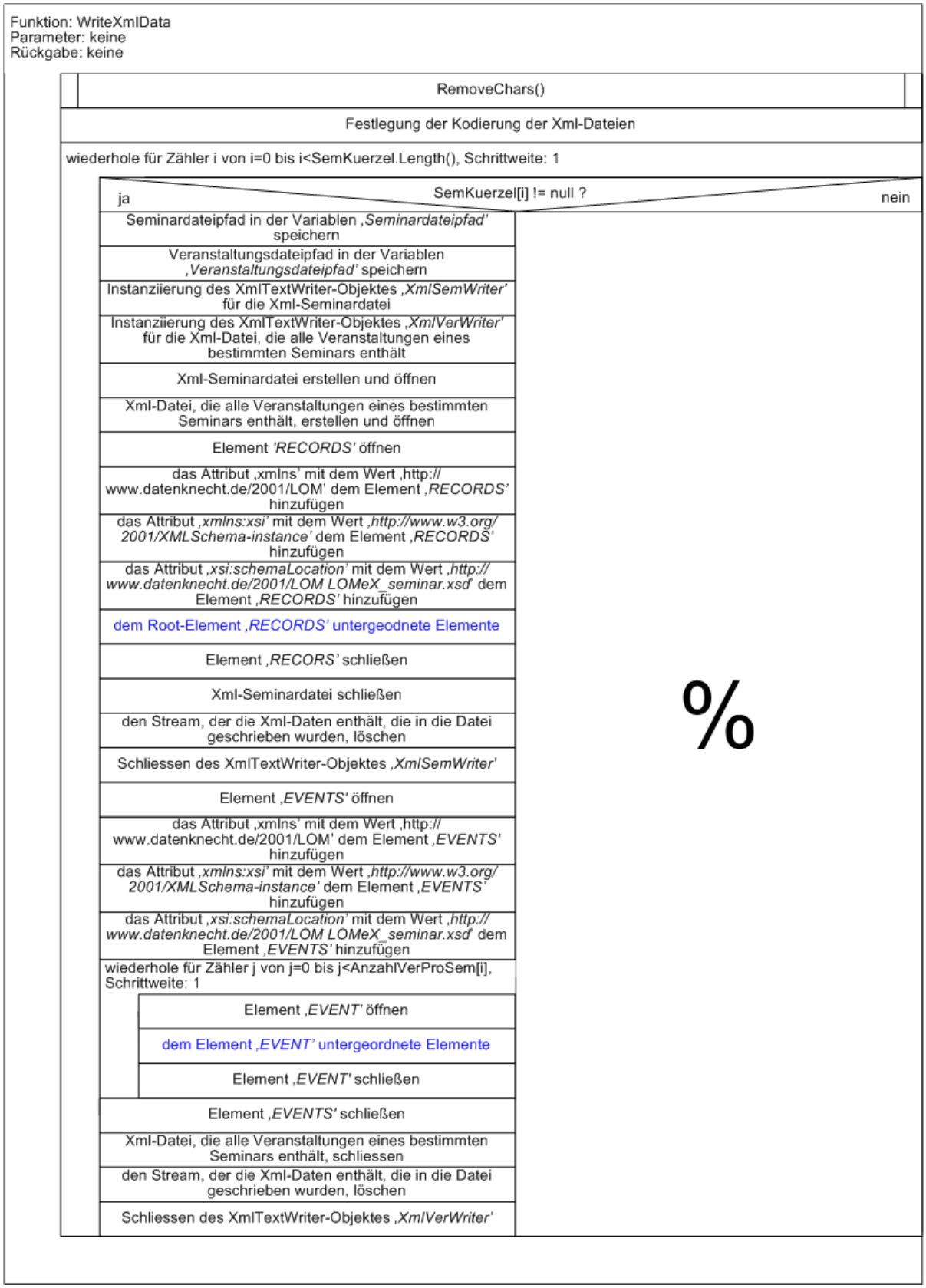
  Danach werden die einzelnen Zeilen dieser Tabelle durchlaufen. Die Kombinationen aus Seminar- und Veranstaltungsdauer der jeweiligen Zeilen der Tabelle

‚Seminardaten' werden im korrespondierenden Element des Stringarrays *‚SemVer'* gespeichert.

- Im zweiten logisch zusammenhängenden Teil dieser Methode werden zuerst die beiden Arrays *‚SemKuerzel'* und *‚AnzahlVerProSem'* dynamisch zur Laufzeit angelegt und mit der Anzahl der in der Tabelle *‚Seminardaten'* enthaltenen Datensätze dimensioniert.

  Im darauf folgenden Schritt werden die einzelnen Elemente des Zeichenkettenarrays *‚SemVer'* durchlaufen. Ein Element ist folgendermaßen aufgebaut: *‚Seminarkürzel-Veranstaltungsnummer'.* Am Ende dieser verschachtelten Schleifen sollen im Zeichenkettenarray *‚SemKuerzel'* nur die verschiedenen Seminarkürzel stehen (sie dürfen also nicht doppelt vorkommen) und im Ganzzahlfeld *‚AnzahlVerProSem'* soll die Anzahl der Veranstaltungen für jedes Seminar stehen. Diese zwei Arrays sind als so genannte *‚Parallelarrays'* zu bezeichnen, da, wenn sie beide z.B. über den Index 0 angesprochen werden, im Feld *‚SemKuerzel'* ein bestimmtes Seminarkürzel steht und im Feld *‚AnzahlVerProSem'* für genau dieses Seminar die Anzahl der Veranstaltungen.

8.5.6 Die Methode *‚WriteXmlData'*



Aufgabe der Methode:

- In dieser Methode findet die Umsetzung der eigentlichen Funktionalität statt – die Xml-Dateien werden geschrieben.

Erläuterung der einzelnen Anweisungsblöcke:

- Als erstes wird die Methode ‚*ReplaceChars*' aufgerufen, die eine ‚Arraylist' mit bestimmten Zeichen und Zeichenfolgen füllt, die später nicht in die Xml-Dateien geschrieben werden sollen. Diese Zeichen oder Zeichenfolgen stellen Formatierungs- und Druckzeichen dar, die nicht ausgegeben werden sollen.

- Als nächstes wird die Kodierung der Xml-Dateien festgelegt.
- In einer Zählschleife werden die einzelnen Seminare und innerhalb dieser werden in einer weiteren Zählschleife die einzelnen Veranstaltungen des jeweils aktuellen Seminars durchlaufen.
- Da das Stringarray *‚SemKuerzel'* wurde mit derselben Anzahl an möglichen Elementen dimensioniert wie das Zeichenkettenarray *‚SemVer'*. Da das Stringfeld *‚SemKuerzel'* weniger Elemente enthält als das Zeichenkettenfeld *‚SemVer'*, muss in der äußeren Zählschleife auf den Inhalt *‚NULL'* geprüft werden. Es soll nämlich nur in die Xml-Dateien geschrieben werden, wenn es noch zu schreibende Seminare gibt.
- Es muss jeweils für die Seminare und für die Veranstaltungen eines bestimmten Seminars ein eigenes *‚XmlTextWriter'*-Objekt erstellt werden, mit dessen Hilfe die im Programm generierten Xml-Daten in die Xml-Dateien geschrieben werden.

```
Seminardateipfad = txtVerzeichnispfadSeminare.Text + @"\" + SemKuerzel[i].ToString() + "-R.xml";
Encoding = new System.Text.UnicodeEncoding();
XmlSemWriter = new XmlTextWriter(Seminardateipfad, Encoding);
```

In diesem Codefragment wird veranschaulicht, wie das *‚XmlTextWriter'*-Objekt *‚XmlSemWriter'* für das Schreiben

der Xml-Dateien der Seminarbeschreibungen instanziiert wird. Das *‚XmlTextWriter'*-Objekt für das Schreiben der Veranstaltungs-Xml-Dateien wird in gleicher Weise erstellt. Dem Konstruktor müssen zwei Parameter übergeben werden. Einmal der Dateipfad der Xml-Datei und dann noch die Kodierungsart, die angibt, wie die Zeichen in der Xml-Datei kodiert werden sollen. Hier soll die Kodierungsart *‚Unicode'* innerhalb der Xml-Datei verwendet werden.

- Notation des Seminardateipfads: Verzeichnisdateipfad + Seminarkürzel + „-R.xml".
- Notation des Veranstaltungsdateipfads: Verzeichnisdateipfad + Seminarkürzel + „-E.xml"

  Beim Aufruf des Konstruktors wird die als Parameter angegebene Datei gleich geöffnet und überschrieben. Wenn sie noch nicht existiert, wird sie erstellt.

- Xml-Deklaration in die Xml-Dateien schreiben:

```
XmlSemWriter.WriteStartDocument();
```

  → es wird die Datei, die beim Erstellen *des ‚XmlTextWriter'*-Objekts spezifiziert wurde, entweder überschrieben oder neu erstellt, je nachdem, ob die Datei bereits existiert oder nicht.

  → es werden die beiden Attribute *‚version="1.0"'* und *‚encoding="ISO-8859-1"'* automatisch in die Xml-Datei geschrieben. Die Kodierungsart wurde bei der Instanziierung des *‚XmlTextWriter'*-Objektes auf *‚Unicode'* festgelegt.

  → die Xml-Deklaration für die Xml-Dateien der einzelnen Veranstaltungen wird analog, aber mit eigenem *‚XmlTextWriter'*-Objekt verwendet.

- Schreiben der Elemente, Attribute und Inhalte der Xml-Dateien:
  - grundsätzliche Vorgehensweise:

    Die deutsche Privatbank hat dem Unternehmen Cellent AG noch zusätzlich eine DTD für die zu erstellenden Xml-Dateien zugeschickt. Diese entspricht dem Xml-Schema, ist nur nicht hierarchisch aufgebaut und ist deshalb um ein vieles leichter zu lesen. Mit Hilfe dieser DTD kann erkannt werden, wie die Struktur der einzelnen Xml-Dateien aufgebaut ist und welche Elemente und Unterelemente definiert werden müssen. Ebenso sind die Attribute und die

Inhaltsbeschränkungen angegeben. So ist also genau ersichtlich, wann ein Element geschrieben werden muss, ob, welche und wie viele Unterelemente vorhanden sein dürfen.

```
<!ELEMENT RECORD (IDENTIFIER, YEAR?, TITLE, DESCRIPTION, CONTENTS?, GOALS?, METHODOLOGY, KEYWORDS?,
OFFERERID, LOCATION?, PLATFORMREQ*, DURATION, ENDUSER?, COST+, ENTRYTEST?, CERTIFICATE?)>
```

Dieses Beispiel stammt aus der DTD. Hier wird das Element *‚RECORD'* definiert, das die in der nachfolgenden Klammer stehenden Unterelemente sowie deren verbindliche Reihenfolge enthält. Diesen Unterelementen werden Verbindlichkeitsangaben angehängt (*, +, ?).

Verbindlichkeiten:

- keine Angabe → das Element muss einmal vorkommen.
- ? → die Angabe des Elements ist optional.
- + → das Element muss einmal, kann aber auch mehrmals vorkommen.
- * → das Element muss entweder keinmal, einmal oder mehrmals angegeben werden.

In der restlichen Dokumentation wird weitgehend darauf verzichtet, alle einzelnen Elemente, die zugehörigen Attribute und die erlaubten Inhalte zu beschreiben. Dies kann bei Bedarf ausführlich entweder in der DTD oder im Xml-Schema für die Xml-Dateien nachgeschlagen werden. Dies ist aber auch für das grundsätzliche Verständnis nicht wichtig.

- Immer wiederkehrender Aufbau, in die Xml-Dateien zu schreiben:

```
#region Element 'RECORD'

    XmlSemWriter.WriteStartElement("RECORD");
    XmlSemWriter.WriteAttributeString("LANGUAGE", "de");
    untergeordnete Elemente
    XmlSemWriter.WriteEndElement();

#endregion
```

- Methode *‚WriteStartElement'*:

  → das Element wird geöffnet

  → das Starttag wird geschrieben

- Methode *‚WriteAttributeString'*:

→ Der Methode werden zwei Parameter übergeben. Der erste repräsentiert den Attributnamen und der zweite den Wert des Attributs

- Codebereich *‚untergeordnete Elemente'*:

  → In diesem Codebereich sind hierarchisch die untergeordneten Elemente aufgelistet.

  → Jedes untergeordnete Element wird so nacheinander in die Xml-Datei geschrieben.

- Methode *‚WriteEndElement'*:

  → das Element wird geschlossen

  → das Endetag wird geschrieben

- Berechnung und Konvertierung verschiedener Inhalte von einzelnen Elementen gemäß dem Xml-Schema:

  Damit nur die Daten in die Xml-Dateien geschrieben werden, die den Anforderungen der Privatbank entsprechen, müssen oft die Inhalte der einzelnen Spalten der Seminardatentabelle umgewandelt werden.

- Xml-Datei schließen:

```
XmlSemWriter.WriteEndDocument();
```

- *‚XmlTextWriter'*-Objekt schließen':

  Bevor dasselbe *‚XmlTextWriter'*-Objekt zum Schreiben einer weiteren Xml-Datei verwendet wird, muss es geschlossen werden.

```
XmlSemWriter.Flush();
```

→ Diese Methode löscht den Stream, der zum Schreiben der Xml-Daten in die Xml-Datei verwendet wurde.

```
XmlSemWriter.Close();
```

→ Diese Methode schließt letztendlich die Instanz der *‚XmlTextWriter'*

Wenn jeweils eine Xml-Datei für die Veranstaltungen eines bestimmten Seminars geschrieben wurde, müssen ebenfalls diese zwei Methoden aufgerufen werden, damit das Schreiben in die Xml-Datei ordnungsgemäß abgeschlossen wird.

- Schreiben der Xml-Dateien der Veranstaltungen eines bestimmten Seminars:

  Innerhalb der Zählschleife, die die einzelnen Seminare durchläuft und diese in getrennte Xml-Dateien schreibt, müssen für das jeweils aktuelle Seminar alle Veranstaltungen in separate Xml-Dateien geschrieben werden. Dazu bedient man sich einer weiteren verschachtelten Zählschleife.

```
#region Element 'EVENTS'

    Beschreibung

    Element 'EVENTS' öffnen

    Element 'EVENTS' - Attribute

#endregion

for(int j=0;j<AnzahlVerProSem[i];j++)
{
    Element 'EVENT'
}

Element 'EVENTS' schliessen
```

- Erklärung des Codeausschnitts:

  Das Element *‚EVENTS'* ist nur ein Umschlagelement, das die Elemente *‚EVENT'* beinhalten kann. Ein Element *‚EVENT'* repräsentiert eine Veranstaltung. Da jeweils für ein bestimmtes Seminar, das in der äußeren Zählschleife behandelt wird, alle zu diesem gehörigen Veranstaltungen in einer vom eigentlichen Seminar getrennten Xml-Datei aufgelistet werden sollen, müssen die einzelnen Veranstaltungen jeweils in der Zählschleife durchlaufen und nacheinander in die Xml-Datei als Unterelemente des Umschlagelements *‚EVENTS'* geschrieben werden. Deshalb wird dieses Umschlagelement erst nach der For-Schleife beendet.

8.5.7 Die Verbindung zur Datenbank schließen – die Methode *‚CloseConnectionToDatabase'*

```
conn.Close();
```

8.5.8 Ergebnisse des Programms

Nachdem noch die letzte Methode *‚ErgebnisAusgabe'*

aufgerufen wurde, um auf das gelungene Erstellen der Xml-Dateien hinzuweisen, kann das Programm geschlossen werden. Die Xml-Dateien sind in das angegebene Verzeichnis geschrieben worden.

8. Qualitätssicherung und Testphase

Grundsätzliches zur Qualitätssicherung und zur Testphase:

Die Qualitätssicherung wurde einerseits nach der Fertigstellung des Projekts und andererseits auch schon während der Programmierung durchgeführt.

Teilnehmer an der Testphase nach der Fertigstellung des Projekts:

Herr Bosch	**Cellent AG, Hüttlingen**
Herr Gronbach	**Cellent AG, Hüttlingen**

Herr Gronbach von der Firma Cellent AG konnte die Testphase des Projektes mit durchführen. Dabei war er nicht nur passiver Betrachter der Qualitätssicherung, sondern testete auch selbst. Zum Abschluss der Qualitätssicherung konnte er sich selbst davon überzeugen, dass das Programm in der geforderten Art und Weise funktioniert und dem Kunden übergeben werden kann. Schließlich war er auch bei der Abnahme des Kunden dabei.

Überprüfung der korrekten C#-Syntax:

Die C#-Syntax konnte mit Hilfe des in der Microsoft Visual Studio .NET Entwicklungsumgebung integrierten Debuggers auf Fehler überprüft werden. Dieser Debugger war auch bei der Fehlerentfernung hilfreich.

‚XMLSPY 5 Professional':

Für die Testphase wurde das Xml-Betrachtungsprogramm *‚XMLSPY 5 Professional'* verwendet. Mit diesem Programm ist es möglich, verschiedene Dateiformate zu betrachten. Für die Zwecke dieses Programms genügt die Betrachtung der Xml-, DTD-, und Xml-Schema-Dateien.

Validierungsprogramm *‚Validator'* und die Validierungsfunktion des Programms *‚XMLSPY 5 Professional'*:

Als erstes mussten die zwei Arten von Xml-Dateien, und zwar diejenigen mit den Seminarbeschreibungen und diejenigen mit den Beschreibungen der einzelnen Veranstaltungen eines bestimmten Seminars, auf ihre Gültigkeit hin überprüft werden. Sind Xml-Dokumente gültig, sind sie implizit auch wohlgeformt.

Zur Überprüfung der Xml-Dokumente wurde neben der in dem Programm *‚XMLSPY 5 Professional'* eingebauten Validierungsfunktion auch das von der deutschen Privatbank mitgelieferte Validierungsprogramm *‚Validator'* verwendet. Beide Programme arbeiten in gleicher Weise.

Es wurden von den Validierungsprogrammen während der Testphase wie auch in der Entwicklung einige kleinere Fehler entdeckt. Einige Elemente wiesen unerlaubte Inhalte auf. Diese Inhalte mussten im C#-Code dem Xml-Schema entsprechend umgewandelt werden.

Sind die Inhalte der Xml-Dateien den jeweiligen Seminaren richtig zugeordnet?

Des Weiteren wurde auch überprüft, ob die Inhalte der Elemenete der Xml-Dateien auch zu den korrespondierenden Seminaren gehören. Dazu wurde einerseits der *‚Microsoft Enterprise Manager'* und andererseits der *‚Microsoft Query Analyzer'* verwendet. Mit Hilfe des *‚MS Query Analyzers'* konnten konkrete Abfragen gegen die Datenbank gefahren werden, um dadurch herauszufinden, ob die Inhalte der Elemente der Xml-Dateien mit den Inhalten der jeweiligen Tabellenspalten übereinstimmen. Unter Verwendung des *‚Microsoft Enterprise Managers'* konnte die ganze Tabelle betrachtet werden, um einen Gesamtüberblick über alle enthaltenen Spalten zu haben.

Es wurden ausnahmslos alle erstellten Xml-Dateien überprüft, ob sie die richtigen Inhalte aufweisen.

Druck- und Formatierungszeichen:

Einige Textfelder der Seminardatentabelle enthielten eine große Menge an kryptischen Zeichen, die in den Xml-Dateien nicht angezeigt werden sollen und deshalb im C#-Code durch die Methoden *‚ReplaceChars'* und *‚RemoveChars'* entweder ersetzt oder gelöscht wurden. Diese Zeichen bzw. Zeichenfolgen mussten in der Testphase natürlich auch aus den Xml-Dateien gefiltert werden, um sie mittels der C#-Anwendung löschen bzw. durch andere Zeichen oder Zeichenfolgen auszutauschen.

Nach erfolgreicher Qualitätssicherung:

Nachdem die Testphase erfolgreich abgeschlossen wurde, konnte die Funktionalität der Anwendung sichergestellt werden.

Abnahme des Kunden:

Nachdem alle Xml-Dateien auf ihre inhaltliche Korrektheit überprüft wurden, konnten sie der deutschen Privatbank in Form einer gebrannten CD gebracht werden. Herr Gronbach und ich

vereinbarten hierfür einen Termin mit Herrn Raupach, damit auch er sich davon überzeugen konnte, dass die Xml-Dateien den Anforderungen entsprechend erstellt wurden. Dieses Gespräch dauerte ungefähr zwanzig Minuten. Herr Raupach bedankte sich bei der Firma Cellent AG, aber auch speziell bei mir. Er erklärte sich sehr zufrieden über die Ergebnisse meiner Programmierarbeit.

9. Soll- / Ist-Vergleich

Das Projektziel der deutschen Privatbank wurde erreicht:

Alle Anforderungen der deutschen Privatbank wurden erfüllt.

Der Kunde hat seine Zufriedenheit in Form eines Feedbacks zum Ausdruck gebracht.

Die deutsche Privatbank hat die Xml-Dateien mit Hilfe des LOMeX-Konverters in ihr SAP-System übertragen.

Die Teamleiter und Vorgesetzten der deutschen Privatbank haben nun die Möglichkeit, in ihrem Kursplanungstool die Beschreibungen der Seminare und der Veranstaltungen der Firma Cellent AG durchzusehen und ihre Mitarbeiter für diese Seminare zu buchen.

Auch das Projektziel der Cellent AG wurde erreicht:

Die Forderungen der Cellent AG wurden ebenfalls erfüllt.

Durch dieses Projekt haben die Teamleiter und Vorgesetzten der deutschen Privatbank die Möglichkeit, ihre Mitarbeiter für die Seminare der Cellent AG zu buchen. Da die Gewinnspanne eines solchen Seminars sehr hoch ist, stellt sogar die Möglichkeit, dass ein Seminar gebucht wird, einen großen Erfolg dar.

Da dieses Projekt in sehr kurzer Zeit durchgeführt wurde, konnte mich die Cellent AG nach der Fertigstellung dieses Projekts in ein anderes integrieren. Somit hat meine Person einen eigenen Anteil an der Wertschöpfungskette der Cellent AG und die Ressource *‚Thomas Bosch'* kann wieder in gewinnbringenden Projekten eingesetzt werden.

Verwendung der Erkenntnisse dieses Projekts für die Zukunft:

In Zukunft sollen über dieses Einzelprojekt hinaus die Seminardaten in ein noch universelleres, nicht firmenspezifisches Format transferiert werden, um es auch anderen Unternehmungen zu ermöglichen, die Seminardaten der Firma Cellent AG in ihr Kursplanungssystem einzubinden.

Seither wurden die Seminarpläne in Form kleiner Bücher an alle denkbaren potentiellen Interessenten versandt. Diese Prozedur war mit einem hohen Personalbedarf, hohen Personalkosten, hohem Zeitaufwand, hohen Verpackungs- und Transportkosten und mit einem erheblichen Organisationsaufwand verbunden.

Wenn die Seminardaten generell in ein einheitliches Xml-Format gebracht werden, können die Kosten für das Papier gespart werden. Auch müssen die Seminarpläne nicht mehr mit der Post versandt werden, sondern können bequem per E-Mail an die entsprechenden Empfänger weitergeleitet werden. Der Personalbedarf und die Personalkosten werden deutlich sinken. Es müssen weniger Mitarbeiter eingesetzt werden. Und dadurch verringert sich auch der Organisationsaufwand. Wenn die Seminardaten per E-Mail versandt werden, entfallen auch die erheblichen Verpackungs- und Transportkosten.

10. Kostenplanung

Die Kosten und Abrechnung für diesen Auftrag sind nicht Bestandteil dieses Projekts und der Dokumentation, da ausschließlich die Geschäftsleitung solche Vorgänge abwickelt.

Kostenplanung und Kostendeckung der Firma Cellent AG:

- Dieses Projekt wurde in einem engen Zeitraum ausgeführt. Deshalb konnte ich nach Abschluss dieses Projekts sofort in andere Projekte integriert werden. Die Kosten, die der Cellent AG wegen der Ausbildungsvergütung anfallen, können so gedeckt werden.
- Die Seminarpreise rechen von 175 Euro bis 2500 Euro. Für die Cellent AG lohnt es sich also bereits ein oder wenige Seminare im Monat abzuhalten.

11. Projektverlauf / Zeitplan

Durchführungszeitraum: 13. April bis 23. April 2004

1. Tag Dienstag, 13. April 2004

08.00 – 10.00 Uhr:
Aufnahme des IST-Zustands.
Besprechung mit Herrn Gronbach, Herrn Leis und Herrn Stegmaier.

10.00 – 12.00 Uhr:
Aufnahme des SOLL-Zustands.
Besprechung mit Herrn Gronbach, Herrn Leis und Herrn Stegmaier.

13.00 – 14.30 Uhr:
Aufnahme des SOLL-Zustands.
Besprechung mit Herrn Gronbach, Herrn Leis und Herrn Stegmaier.

14.30 – 16.00 Uhr:
Vorhandene und erforderliche Datenbankinhalte abstimmen

16.00 – 17.00 Uhr:
Konzeption eines Entwurfs

Netto-Projektstd.: 8,0

2. Tag Mittwoch, 14. April 2004

08.00 – 12.00 Uhr:
Konzeption eines Entwurfs

13.00 – 16.00 Uhr:
Konzeption eines Entwurfs

16.00 – 17.00 Uhr:
Design der Benutzeroberfläche

Netto-Projektstd.: 8,0

3. Tag Donnerstag, 15. April 2004

08.00 – 08.30 Uhr:
Design der Benutzeroberfläche

08.30 – 10.00 Uhr:
Verbindung zur Datenbank aufbauen

10.00 – 11.00 Uhr:
Datenbankabfrage definieren

11.00– 12.00 Uhr:
‚DataSet' füllen

13.00– 17.00 Uhr:
Realisierung der Funktionalität *‚Schreiben der Xml-Dateien'*

Netto-Projektstd.: 8,0

4. Tag Freitag, 16. April 2004

08.00– 12.00 Uhr:
Realisierung der Funktionalität *‚Schreiben der Xml-Dateien'*

13.00– 17.00 Uhr:
Realisierung der Funktionalität *‚Schreiben der Xml-Dateien'*

Netto-Projektstd.: 8,0

5. Tag Montag, 19. April 2004

08.00– 12.00 Uhr:
Realisierung der Funktionalität *‚Schreiben der Xml-Dateien'*

13.00– 17.00 Uhr:
Realisierung der Funktionalität *‚Schreiben der Xml-Dateien'*

Netto-Projektstd.: 8,0

6. Tag Dienstag, 20. April 2004

08.00– 12.00 Uhr:
Qualitätssicherung, Testphase durchführen

13.00– 15.30 Uhr:
Qualitätssicherung, Testphase durchführen

15.30– 17.00 Uhr:
Ausarbeitung der Kundendokumentation

Netto-Projektstd.: 8,0

7. Tag Mittwoch, 21.April 2004

08.00– 09.00 Uhr:
Abnahme durch den Projektverantwortlichen Herrn Gronbach

09.00– 09.30 Uhr:
Abnahme durch Herrn Raupach von der deutschen Privatbank

09.30– 10.00 Uhr:
Einbindung der Xml-Dateien in das Kursplanungstool der deutschen Privatbank

10.00– 12.00 Uhr:
Ausarbeitung der Projektdokumentation

13.00– 17.00 Uhr:
Ausarbeitung der Projektdokumentation

Netto-Projektstd.: 8,0

8. Tag Donnerstag, 22.April 2004

08.00– 12.00 Uhr:
Ausarbeitung der Projektdokumentation

13.00– 17.00 Uhr:
Ausarbeitung der Projektdokumentation

Netto-Projektstd.: 8,0

9. Tag Freitag, 23.April 2004

08.00– 12.00 Uhr:
Ausarbeitung der Projektdokumentation

13.00– 13.30 Uhr:
Ausarbeitung der Projektdokumentation

13.30– 15.00 Uhr:
Projektnachbesprechung (Teilnehmer: Herr Gronbach, Herr Leis, Herr Stegmaier)

Netto-Projektstd.: 6,0

Gesamte Projektstunden: 70,0

12. Abweichnungen zum Projektantrag

Auflistung der Abweichungen zum Projektantrag:

	Projektantrag (Stunden)	**tatsächlich (Stunden)**	**Abweichung Projektantrag – tatsächlich (Stunden)**
IST-Analyse	5	2	- 3
SOLL-Analyse	10	3,5	-6,5
Entwurfsplanung	10	8	-2
Ermittlung/Formatierung der Daten	8	5	-3
Umwandlung ins Xml-Format	12	20	+ 8
Qualitätssicherung	6	6,5	+0,5
Abnahme durch den Kunden	In *‚Qualitätssicherung'* enthalten	0,5	+0,5

Einbinden in das Kursplanungstool	1	0,5	-0,5
Erstellen der Kundendokumentation	10	1,5	-8,5
Erstellen der Projektdokumentation	8	18,5	+10

Erklärungen zu den Abweichungen zum Projektantrag:

Die wohl gravierendsten Unterschiede in diesen aufgelisteten Abweichungen sind die vier Punkte *‚SOLL-Analyse'*, *‚Umwandlung ins Xml-Format'*, *‚Erstellen der Kundendokumentation'* und *‚Erstellen der Projektdokumentation'*. Die Durchführungszeiten der anderen Projektteile entsprechen in etwa der Vorabplanung.

Kundendokumentation:

Da die Kundendokumentation nicht sonderlich umfangreich ist und sehr leicht zu erstellen war, konnte sie in kurzer Zeit abgehandelt werden.

Die Kundendokumentation richtet sich an die Mutarbeiter der Cellent AG, die Xml-Dateien auf der Grundlage der Seminardaten der Cellent AG erstellen möchten.

Auf eine Kundendokumentation für die deutsche Privatbank wurde verzichtet, da die Xml-Dateien nach dem mitgelieferten Xml-Schema, also nach den Anforderungen der deutschen Privatbank erstellt wurden.

Umwandlung der Seminardaten in das universelle Xml-Format:

Da die Xml-Dateien gemäß dem mitgelieferten Xml-Schema geschrieben werden sollten, mussten die Inhalte der Seminardatenbank oft konvertiert und angepasst werden, damit gültige Xml-Dateien entstehen.

Dies konnte nur schrittweise und mit höchster Sorgfalt durchgeführt werden, da jeder ausprogrammierte Teil im Sinne der Qualitätssicherung direkt überprüft werden musste.

SOLL-Analyse:

Die SOLL-Analyse konnte in relativ kurzer Zeit durchgeführt werden. Die Kundenwünsche waren sehr präzise formuliert und die Seminardaten werden bereits in einer zentralen Stelle verwaltet. Außerdem sind die Datenbankinhalte sehr sorgfältig dokumentiert. Dieser Sachverhalt erleichterte es, die erforderlichen Daten zu finden. Auch waren alle wichtigen

Schnittstellen bei diesem Meeting anwesend.

Erstellen der Projektdokumentation:

Die Projektdokumentation wurde umfangreicher als geplant. An vielen Stellen mussten während und nach dem Schreiben der Projektdokumentation noch Verbesserungen eingebracht werden. Ebenfalls wurde die grundsätzliche Struktur mehrmals geändert, damit der Leser dieser Projektdokumentation einen roten Faden hat und die Übersicht nicht verliert.

13. Anlagen

SOLL- / IST-Projektplan (Anlage 1)

Kundendokumentation (Anlage 2)

Programmiercode – Projekt *‚Xml-Export-Tool'* (Anlage 3)

Glossar (Anlage 4)