.NET in der Praxis: Von der Idee bis zum Einsatz

Harald Haller und Alexander Ramisch

Dr. Harald Haller *ist Seniorberater bei der sd&m AG. Schwerpunkte seiner Arbeit in den vergangenen Jahren waren die Architektur verteilter Anwendungen, die Entwicklung eines Java-Frameworks und Projektleitung. Aktuell verantwortet er eines der .NET-Projekte von sd&m. Kontakt: harald.haller@sdm.de*

Alexander Ramisch *ist Technischer Berater bei der sd&m AG. Seit 6 Jahren ist er bei sd&m schwerpunktmäßig in OO-Design und der Realisierung großer Softwaresysteme tätig. Aktuell verantwortet er als technischer Chef-Designer eines großen .NET-Projektes die technische Gesamtarchitektur und die technischen Komponenten. Kontakt: alexander.ramisch@sdm.de*

1 Zusammenfassung

.NET ist die von Microsoft empfohlene Plattform für die Software-Entwicklung. Dieser Artikel zeigt auf, wie mit Hilfe von Basisarchitekturen eine neue Technologie für große Projekte rasch und sicher erschlossen und zum Einsatz gebracht werden konnte. Ferner enthält dieser Artikel einen Vergleich zwischen .NET und J2EE sowie Erfahrungen, die aus dem Einsatz des .NET Frameworks und des Visual Studio.NET gewonnen wurden und gibt praktische Tipps insbesondere für größere Projekte.

2 Microsoft verspricht einiges mit .NET

Der im Folgenden angeführte und in der Öffentlichkeit kontrovers diskutierte Vergleich von J2EE mit Microsoft .NET soll zunächst die Versprechen von Microsoft darstellen. Im vorliegenden Artikel werden wir dann von unseren Erfahrungen aus zwei großen .NET Projekten berichten.

Microsoft hat die Java-Beispielanwendung *Petstore* von SUN nach .NET portiert. Die von Microsoft veröffentlichten Ergebnisse zeigen dramatische Verbesserungen. So konnte der zu schreibende Code um 68% reduziert und die Laufzeit der Anwendung um den Faktor 28 zur vergleichbaren Java-Anwendung gesteigert werden, vgl. Abbildung 1, Details findet man unter [1].

In zwei größeren Projekten der sd&m AG wurde .NET als Plattform zur Entwicklung und als Laufzeitumgebung gewählt. In diesen Projekten entstanden Multi-Tier-Anwendungen, die Kernprozesse der Kunden unterstützen. Wir möchten nun von Erfahrungen mit .NET in realen Projekten berichten und versuchen die Frage zu beantworten, ob die Versprechen bzgl. .NET allgemeine Richtigkeit haben.

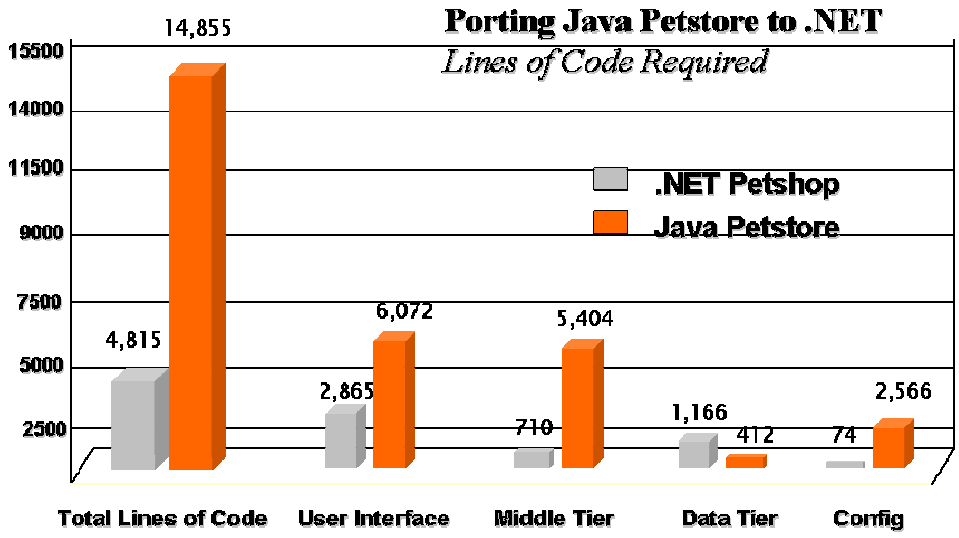

Abbildung 1 - Microsoft`s Petstore

3 Warum setzen wir in Projekten .NET ein?

Für zwei namhafte Unternehmen, die Münchener Rückversicherungsgesellschaft AG und die REAL I.S. Immobilienfonds AG, entwickelt die sd&m AG Systeme auf Basis von Microsoft .NET. Beide Kunden hatten bereits in den Jahren 2000 und 2001 unternehmensweit die strategische Entscheidung für .NET gefällt.

Für das Internet-Risikobewertungssystem *MIRA* der Münchener Rück entsteht in einem Team von bis zu 10 Mitarbeitern ein internationalisiertes, integriertes Pflegesystem mit Datenbank-Pflege, Dokumentenmanagement inklusive Verschlagwortung und umfangreicher Microsoft Office-Integration sowie Workflow-Unterstützung.

Das für die REAL I.S. entwickelte System *LEONARDO* bildet die Kernprozesse des Unternehmens zur Abwicklung aller Geschäftsvorfälle während der Fondslaufzeit ab. Nach der Einführung steht es heute allen Mitarbeitern für die tägliche Arbeit zur Verfügung und wird derzeit weiterentwickelt. Eine wichtige Anforderung war die starke Integration von Microsoft Office und die Verwendung des Active Directories zur Benutzerrechte-Verwaltung. LEONARDO entstand in einem Team aus bis zu 14 Mitarbeitern im Zeitraum von Mai 2001 bis Februar 2003.

Beide Projekte wurden vollständig in C# realisiert.

Beide Kunden entschieden sich bereits zu einem sehr frühen Zeitpunkt für die neue .NET-Plattform, vgl. Abbildung 2. Diese Entscheidung wurde getroffen, obwohl

- zu diesem Zeitpunkt nur Betaversionen des Frameworks und der Entwicklungsumgebung verfügbar
- und aufgrund mangelnder Referenzprojekte sowie fehlender Einschätzungen die Risiken für die neue Technologie hoch

waren.

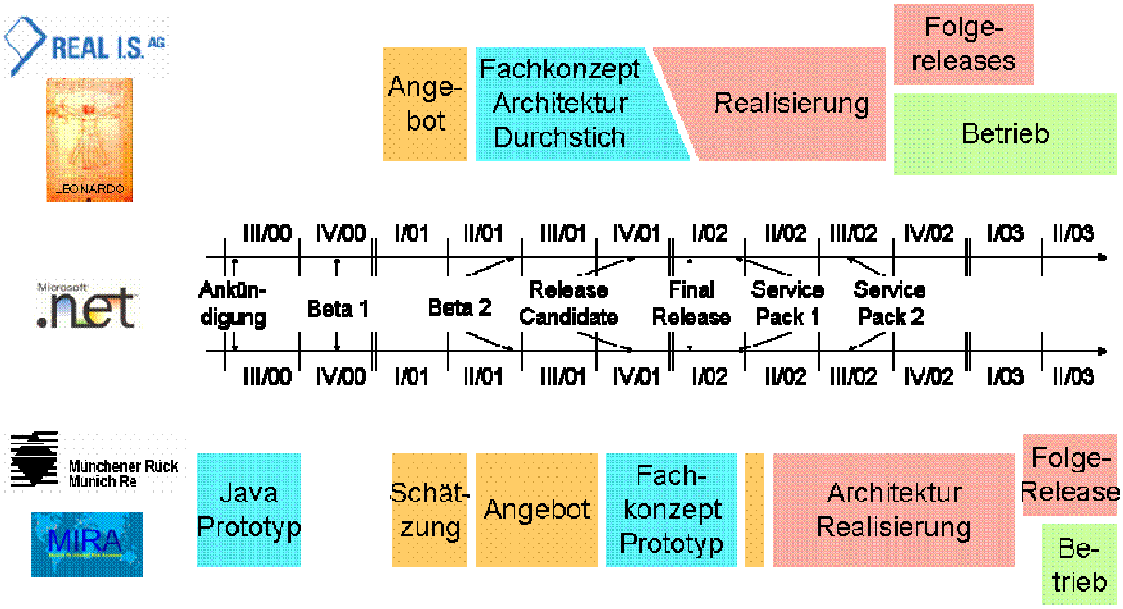

Abbildung 2 - Die Projekte von der Idee bis zum Einsatz

Warum sich unsere Kunden dennoch für die von Microsoft empfohlene Technologie entschieden haben, wollen wir im Folgenden darstellen.

Parallelen zwischen J2EE und .NET

Zunächst gibt es viele Parallelen zwischen J2EE und .NET (vgl. Abbildung 3) wobei sich die Vor- und Nachteile des Funktionsumfangs in etwa aufheben. So gibt es Vorteile für Java beim Funktionsumfang von Swing in Vergleich zu .NET, dafür Effizienzvorteile bei der Erstellung von Web-Clients oder Webservices zugunsten von .NET, da ihre Erstellung durch die Entwicklungsumgebung umfassend unterstützt wird.

Zweck	.NET	J2EE
Ausführungsumgebung	CLR (MSIL)	Java VM (Bytecode)
Basisbibliotheken	.NET Framework Class Library	Java Core API
Programmiersprache	C#, managed C++, VB.NET	Java
Web-Clients	ASP.NET	JSP
Native Clients	Windows Forms	Swing
Datenzugriff	ADO.NET	JDBC
Webservices	ASP.NET, .NET Services	SUN´s JXTA

Abbildung 3 – Gegenüberstellung .NET und J2EE

Dagegen ist ein wichtiger Pluspunkt von Java, dass sich der Einsatz von J2EE in vielen Projekten bewährt hat. Gravierend ist die Tatsache, dass .NET vor mehr als einem Jahr echtes Neuland war und sich mittlerweile etabliert.

Risiko des Vorreiters

Mit dem Einsatz einer brandneuen Technologie in einem für den Auftraggeber sehr wichtigen Projekt sind zahlreiche Risiken und Fragen verbunden:

- Trägt die neue Technologie z.B. bezüglich Performanz und Stabilität?
- Wie ausgereift sind Entwicklungs- und Laufzeitumgebung?
- Wie lange dauert die Einarbeitung?
- Welche Zusatzkosten entstehen dadurch, dass fertige, ausgereifte Komponenten fehlen?
- Kann sich die neue Technologie am Markt behaupten?[1]

Entscheidung für .NET

Trotz der genannten Risiken fiel die Entscheidung zugunsten von .NET. Der Hauptgrund bei beiden Kunden lag in ihrer jeweiligen Microsoft-Ausrichtung und darin, dass die strategische Plattform von Microsoft .NET ist.

Beim Einsatz brandneuer Technologien müssen Risiken minimiert werden

Im Vorfeld der Projekte galt es, sich den Risiken und potenziellen Gefahren zu stellen, die mit dem Einsatz der neuen Technologie einhergehen. Das Hauptproblem des fehlenden Know-hows wurde bereits genannt. Dieses Problem hat viele Facetten:

- keine Erfahrungen mit der neuen Entwicklungsumgebung, dem Framework und der Programmiersprache,
- keine Referenzprojekte zum Nachweis der Umsetzbarkeit,
- fehlende Sekundärliteratur, spärliche Online-Hilfe und Beispielcode,
- Fehler in Betaversionen der Entwicklungswerkzeuge.

Heute sieht es mittlerweile anders aus: Dokumentation, Onlinehilfe, Beispielcode und Sekundärliteratur sind zahlreich vorhanden, die Werkzeuge weitgehend ausgereift. Zu Projektbeginn vor über einem Jahr mussten konkrete Maßnahmen getroffen werden, um den Gefahren gewappnet zu sein.

Technischer Durchstich

So erstellten wir einen ersten technischen Durchstich unserer Anwendung auf Basis von .NET. Dieser sehr schmale Teil des zu bauenden Systems, der trotzdem die zentralen technischen Fragestellungen beinhaltete, zeigte, dass es möglich war unsere Basisarchitektur auf der neuen Plattform umzusetzen.

---

[1] vgl. IBM OS/2: Dieses System ist heute noch bei vielen Firmen im Einsatz und wird nicht mehr unterstützt.

Bei der Entwicklung des Durchstichs arbeitete sich ein kleines Teil-Team in das Framework und die Werkzeuge ein, um dieses Wissen im späteren Projektverlauf dem gesamten Team zur Verfügung zu stellen. So wurde das spätere 14-Personen-Team sehr schnell produktiv, da die wesentlichen Gesichtspunkte, die zur Entwicklung relevant waren, aufbereitet vorlagen.

Die Erprobung und Aufbereitung hat sich sehr bewährt. Es zeigte sich auch sehr deutlich, dass das gesamte Team unter Verwendung der eigenen Basisklassen sehr schnell produktiv wurde.

Iteratives Vorgehen

Um die Technik frühzeitig zu erproben und Performance-Messungen durchführen zu können, wurde im Projekt ein iteratives Vorgehensmodell gewählt.

Während der Spezifikationsphase, in der der überwiegende Teil des Projektteams zusammen mit dem Kunden fachliche Anforderungen erarbeitete und in Form von Konzepten niederschrieb, kümmerte sich ein kleines Team um die neue Technik. Es wurden Basisklassen implementiert, Komponenten von Drittherstellern getestet und eingebunden sowie die Entwicklungsumgebung für das Gesamtprojekt aufgebaut.

Zu Beginn der Realisierung lag demzufolge kein fertiges technisches Konzept vor. Stattdessen gab es Konzepte für technische Komponenten, Architekturpläne auf groben Level und Netto-Implementierungen[2] typischer aber einfacher Geschäftsvorfälle.

Während der Entwicklung fanden immer wieder Designrunden statt, in denen technische Fragen diskutiert und entschieden wurden. Diese wurden getestet und bei Bewährung in das DV-technische Konzept eingearbeitet. Durch die Vorleistung des kleinen Technik-Teams kam es nie zu größeren Wartezeiten.

Spezialisten

Sehr bewährt hat sich unsere Aufteilung von Aufgaben im Team. So gab es Spezialisten für die Entwicklungsumgebung, das Framework und die Programmiersprache C#, für Komponenten von Drittherstellern, für technische Fragestellungen[3] und für den Kontakt zu den Technologieberatern bei Microsoft.

[2] Unter Netto-Implementierung verstehen wir die einfache Implementierung der Basisfunktionalität ohne vollständige Fehlerbehandlung, ohne vollständigen Kommentar usw.

[3] COM-Zugriff, Internationalisierung, Datenbankzugriff, Kommunikation, Windows Forms.

4 Zur Entwicklung großer Anwendungen reichen .NET-Bordmittel nicht aus

In diesem Abschnitt werden wir motivieren, dass zur Entwicklung großer Systeme auch unter .NET eine Architektur benötigt wird. Hierfür ist einige Konzeptarbeit zu leisten, allerdings wird festgestellt, dass für andere OO-Plattformen konzipierte Architekturen weitgehend unter .NET eingesetzt werden können.

Technische Architektur

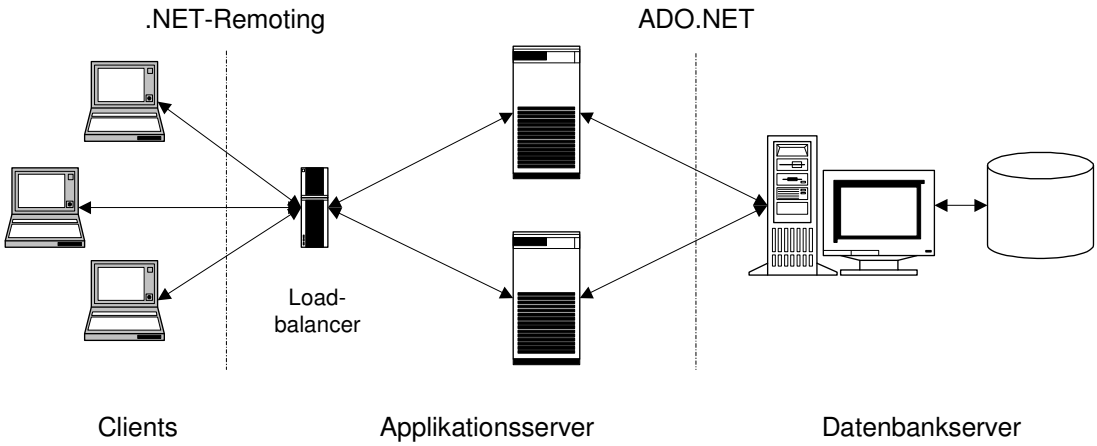

Abbildung 4 - Übersicht Technische Architektur

Abbildung 4 gibt einen Überblick über die technische Architektur der von der sd&m AG entwickelten Softwaresysteme. Es handelt sich um Systeme, die auf 3 Ebenen verteilt ablaufen. Die Systemarchitektur mit Thin-Clients und Applikationservern mit der Anwendungslogik ermöglicht Skalierung, erhöht die Ausfallsicherheit und erleichtert die Softwareverteilung.

Motivation für Architektur

Bei unseren Systemen handelt es sich um größere Anwendungen für zentrale Prozesse unserer Kunden. Hinzu kommt, dass die Softwaresysteme für unsere Kunden maßgeschneidert angefertigt wurden und dass von diesen erwartet wird, dass sie lange im Betrieb bleiben.

Das bedeutet, dass die Software explizit wartbar gebaut werden muss und dass die Systeme verständlich und in hohem Maße einheitlich strukturiert sind. Im Rahmen der langjährigen Wartung müssen neue Anforderungen der Kunden implementiert werden. Um die Erweiterungen und Änderungen einfach und sicher integrieren zu können, benötigt man eine Architektur, quasi einen Bauplan, an den sich jeder halten muss, um Wartbarkeit und damit Einheitlichkeit sowie Verständlichkeit langfristig garantieren zu können.

Was verstehen wir unter Architektur?

Für uns definiert die Architektur ein gemeinsames Verständnis von Software-Qualität. Es wird versucht, besonders wichtige Regeln und Mechanismen der Softwaretechnik verständlich zu beschreiben, zu präzisieren und konsistent umzusetzen.

Einige wichtige Leitlinien werden im Folgenden genannt.

- Trennung von Zuständigkeiten
  Dies ist eine, wenn nicht *die* Leitlinie für gute Software-Architektur: Klassen, die sich mit verschiedenen Dingen gleichzeitig befassen, sind schlecht. Der Alptraum jedes Programmierers sind Returncodes verschiedener technischer APIs (ist die Datenbank noch verfügbar?) vermischt mit Anwendungsproblemen (kann die Buchung noch storniert werden?).
- Denken in Komponenten und Schnittstellen
  Hiermit befasst sich der folgende Abschnitt *Komponentenorientierung*.
- Einheitlichkeit
  Einheitlicher Quellcode und einheitliche Strukturierung von Softwaresystemen erleichtern das Verständnis. Hiermit wird die Erstellung und Wartung erleichtert und somit auch sicherer. Um Einheitlichkeit zu erreichen, werden Konzepte erstellt und durchgängig beachtet z.B. zur Fehlerbehandlung, Datenbankzugriff, Datentypen, Tracing, Berechtigungen, Programmierrichtlinien und Styleguides, etc. Aber auch Quellcodemuster können als Vorlage zur Entwicklung dienen.
- Designentscheidungen
  Die Architektur eines Softwaresystems beinhaltet eine Menge an Designentscheidungen, die getroffen werden, um die Anforderungen wie Austauschbarkeit der Clientoberfläche, Ausbaubarkeit bei stark anwachsender Benutzerzahl und Umsetzung der Internationalisierung zu erfüllen.

Beim Entwurf der Anwendungsarchitektur werden frühzeitig wesentliche Grundlagen gelegt

Zu Beginn eines Projektes gilt es zunächst einmal die technische Systemarchitektur zu klären.

Bei der Wahl des Oberflächentyps, also *Webclient* oder *native Client* fiel die Entscheidung zu Gunsten eines nativen Clients. Da wir keine Probleme mit der Softwareverteilung erwarten, waren `Windows.Forms` auf Clientseite die richtige Wahl. Auch der Client/Serverschnitt wird mit dem ersten Argument beantwortet: Die Dialogsteuerung wurde auf dem native Client platziert.

Als Mittel zur Anbindung der Clients an den Server bietet .NET das Remoting Framework, was unseren Ansprüchen gerecht wurde.

Die Frage nach dem Komponentenmodell wird mit QUASAR beantwortet. Hierbei handelt es sich um die Basisarchitektur der sd&m AG, die problemlos unter .NET eingesetzt werden konnte. Weitere Informationen folgen in den nächsten Abschnitten.

Als Persistenzschnittstelle mussten wir eigene Zugriffsobjekte entwickeln, die für die fachlichen Objekte die Zugriffe auf die Datenbank kapseln. Der Datenbankzugang erfolgt über ADO.NET mittels managed Provider. Hier sieht man ein Manko von .NET gegenüber

J2EE: es gibt keine Unterstützung bei der Persistenz, wie sie mittels Container-Managed Persistence in J2EE geboten wird.

Komponentenorientierung

An dieser Stelle soll unser Verständnis des Begriffs *Komponente* erläutert werden, da viele verschiedene Vorstellungen von Komponenten existieren und Komponenten bei der .NET-Entwicklung eine zentrale Rolle spielen.

Eine Komponente ist eine softwaretechnische Einheit des Entwurfs, der Implementierung und der Planung. Sie bietet nach außen eine Schnittstelle in Form von Operationen an. Sie besteht meist aus mehreren Klassen, die untereinander Schnittstellen haben, die außerhalb der Komponente nicht sichtbar sind.

Im Zusammenhang mit modernen Komponentenarchitekturen wird der Komponentenbegriff allerdings enger gefasst. Eine wesentliche Eigenschaft einer solchen Komponente ist die Anpassbarkeit an unterschiedliche Systemumgebungen, bzw. die Möglichkeit die Komponente zu konfigurieren, ohne sie neu übersetzen zu müssen.

Damit wird eine Komponente wieder verwendbar und kann in unterschiedlichen Systemen eingesetzt werden. Unsere Komponenten haben selten diesen Anspruch. Warum wir dennoch Komponenten einsetzen, erläutern wir im folgenden Abschnitt.

Warum setzen wir Komponenten ein?

Komponenten bieten uns viele Vorteile, so können Softwareteile von Spezialisten gekauft und eingesetzt werden wie z.B. zum Tracing und für Tabellencontrols.

Komponenten sind bereits während der Entwicklung abgeschlossene Arbeitspakete, die parallel entwickelt werden können und somit schon per se Abhängigkeiten vermeiden.

Während der Wartung der Software fördern Komponenten die Änderbarkeit, denn wenn nur die Schnittstellen bekannt sind, kann die Implementierung einer Komponente geändert werden, solange der Vertrag der Schnittstellen eingehalten wird. Somit kann ein Softwaresystem an einer Stelle korrigiert und erweitert werden ohne Gefahr zu laufen, dass es sich danach an anderer Stelle nicht mehr korrekt verhält.

Unsere Komponenten verstecken fachliche oder technische Entitätsklassen, die die Aufgabe der Komponente erbringen. Diese Klassen sind dabei außerhalb der Komponente nur als Schnittstelle bekannt. Sie werden darum als `internal` und somit nur innerhalb der Komponente selbst (in der Assembly) sichtbar gemacht.

Wie erfolgt der Zugriff auf Komponenten?

Für den Zugriff auf Dienste einer Komponente gibt es den Komponentenverwalter, eine spezielle Klasse pro Komponente. Dieser Verwalter registriert sich selbst bei einer global bekannten und als Singleton realisierten Verwalterklasse (in Abbildung 5 der globale Komponentenverwalter), die die Schnittstellen der einzelnen Verwalter einer jeden Komponente kennt.

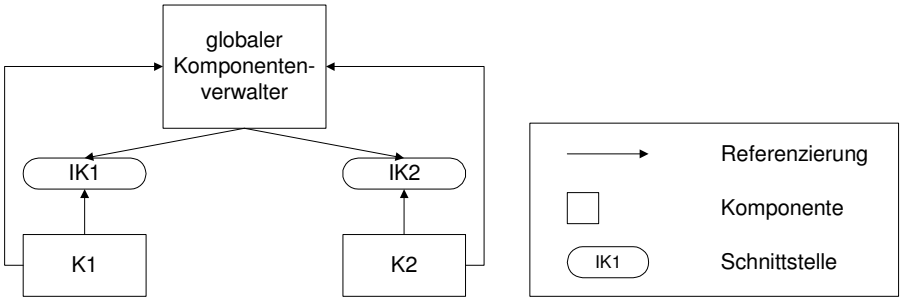

Abbildung 5 - Zugriff auf Komponenten über Verwalter

Beispiel
Die Komponente *Kontakt* enthält die Klasse `Kontaktverwalter`, die die Schnittstellen `IInternKontakte` und `IExternKontakte` implementiert. Hier sind Methoden enthalten, mit denen Kontakte angelegt, gesucht, gelöscht etc. werden können.

Wenn aus einer beliebigen Komponente wie z.B. *Fonds* heraus auf einen Kontakt zugegriffen werden muss, wird im Quellcode zunächst die Schnittstelle des Kontaktverwalters vom globalen Komponentenverwalter erfragt, um anschließend an diesem den Zugriff auf die Kontakte durchzuführen (vgl. Abbildung 6).

```
// In diesem Beispiel wird aus der Komponente Fonds heraus
// ein Kontakt gelesen

// Verwalter der Komponente Kontakt erfragen
IKontaktverwalter KontaktVerwalter =
   Komponentenverwalter.Instanz.IKontaktverwalter;

// gewünschten Kontakt lesen
IPerson = KontaktVerwalter.Lesen( strName );

...
```

Abbildung 6 - Zugriff auf Komponenten über den globalen Komponentenverwalter

Im Initialisierungsprozess wird die Anwendung konfiguriert, indem die einzelnen Komponentenverwalter erzeugt werden und sich mit ihren Schnittstellen am globalen Komponentenverwalter registrieren.

Unsere Komponenten verfügen meist über zwei verschiedene Schnittstellen: eine interne mit Diensten, die am Server verfügbar sind und eine externe Schnittstelle mit Diensten, die auch am Client bekannt sind.

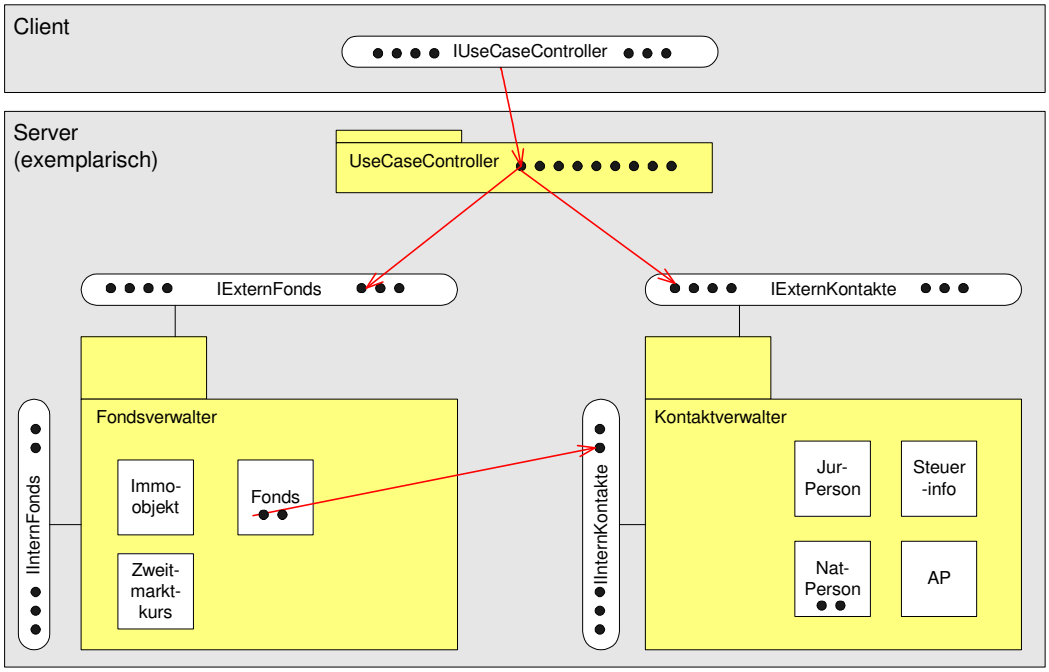

Abbildung 7 - Externe und Interne Schnittstellen

Das folgende Code-Beispiel (vgl. Abbildung 8) zeigt einen Ausschnitt der Komponente *Fonds* des LEONARDO-Systems. Hier sieht man den Verwalter der Komponente, der die beiden Schnittstellen `IAWKFondsExtern` und `IAWKFondsIntern` implementiert.

```
namespace leonardo.server.awk.fonds
{
   public class Fondsverwalter : IKAwkFondsExtern,
                                 IKAwkFondsIntern
   {
      private Fondsverwalter(){}

      public static void RegistrierenInstanz()
      {
         if( Fondsverwalter.Instanz == null )
            Fondsverwalter.Instanz = new Fondsverwalter();
         Komponentenverwalter.Instanz.IKAwkFondsExtern =
            Fondsverwalter.Instanz;
         Komponentenverwalter.Instanz.IKAwkFondsIntern =
            Fondsverwalter.Instanz;
      }
   }
  internal class Fonds : IFonds
  {
     //...
  }
}
```

Abbildung 8 - Codebeispiel Ausschnitt der Komponente Fonds

Am Fondsverwalter dient die statische Methode `RegistrierenInstanz()` dazu, die öffentlichen Schnittstellen am globalen Komponentenverwalter einzutragen; sie werden dazu als Property am globalen Komponentenverwalter gesetzt.

Client-Architektur

In Abbildung 9 sind die zentralen Komponenten des Clients veranschaulicht. Grundlage des Designs ist das MVC-Pattern (Model-View-Controller).

Die Aufgaben der einzelnen Komponenten wollen wir im Folgenden erläutern:

- Die **Windows Forms-Klassen** sind die Basis unserer GUI-Controls. Für die Verwendung der Controls wurden Wrapper-Klassen erstellt, die einheitliche Methoden zur Steuerung sowie Methoden für Prüfungen bieten. Damit ist eine komfortable und einheitliche Nutzung gegeben, was die Austauschbarkeit bei eventuellen Weiterentwicklungen der Framework-Klassen erhöht.

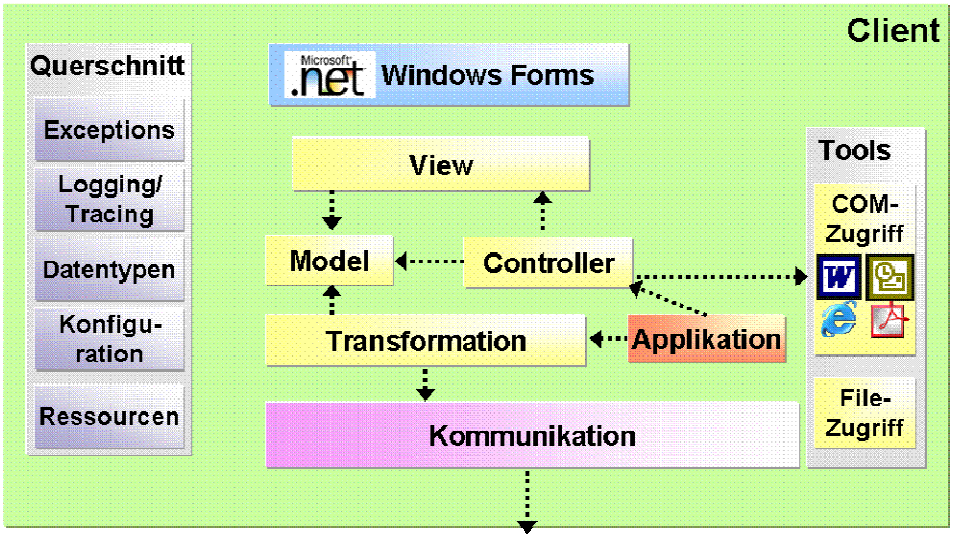

Abbildung 9 - Client-Architektur

Obwohl die Framework-Klassen schon umfangreiche Funktionalität bieten, waren Funktionserweiterungen erforderlich. Es wurden z.B. in den Wrappern die .NET Methoden `ToString()` zur korrekten Ausgabe und `Clone()` zum Kopieren von Objekten überschrieben. Ferner wurden die Klassen dahingehend erweitert, dass Benutzeraktionen synchronisiert werden, damit Aktionen, die ein Benutzer parallel zu laufenden Aktionen auslöst, korrekt abgearbeitet werden.

- Die **View** ist für den sichtbaren Teil der grafischen Oberfläche verantwortlich. Diese setzen die Oberflächen aus den Wrapperklassen für die Windows Forms zusammen und holen sich die notwendigen Daten aus dem Model.
- Das **Model** beinhaltet die Daten, die auf der Oberfläche dargestellt und eingegeben werden bzw. für die Oberflächendarstellung notwendig sind.
- Der **Controller** verarbeitet die Benutzeraktionen, beinhaltet die Dialogsteuerung mit Zustandsautomat (Aktivierung und Deaktivierung von Controls), triggert die Serveraktionen und öffnet parametrisierte Subdialoge.

- Die Komponente **Transformation** führt die Umsetzungen zwischen Client- und Server-Datenmodell und umgekehrt durch.
- Die **Kommunikation**skomponente ist zuständig für den Datenaustausch mit dem Server und benutzt .NET Remoting.
- Die **Ressourcen** beinhalten Daten für die Konfiguration der Dialoge und Lokalisierungen für verschiedene Sprach- und Marktversionen (vgl. Abschnitt 5.3). Ferner stellen sie die auf den Oberflächen angezeigten Bilder und Icons zur Verfügung.
- Den Start der Client-Anwendung und der Dialoge übernimmt die Komponente **Applikation**. Sie ist zuständig für die Initialisierung des Clients, der Dienste und zentralen Client-Komponenten. Ferner lädt sie die lokale Konfigurationen und Ressourcen vom Server.

Wir konnten zu einer Vielzahl weiterer Themen Erfahrungen sammeln, die wir im Rahmen dieses Artikels nicht ausführen können. Hierzu zählen z.B.

- Dialogsteuerung,
- Datenübergabe,
- Berechtigungen und Sicherheit,
- Speicher- und Ressourcenverwaltung,
- Anpassungen für Loadbalancing (Methoden Equals() und Hashcode()),
- Dokumentenmanagement,
- Workflow-Unterstützung,
- Hilfesystem,
- Clipboard.

Auch hier gilt die Aussage, Konzepte für andere OO-Programmiersprachen lassen sich weitgehend übertragen, teilweise sind spezifische Anpassungen im Detail sinnvoll.

5 In den Projekten wurden zahlreiche interessante Erfahrungen gewonnen

5.1 .NET Remoting

.NET Remoting ist ein Framework für den nahtlosen Austausch von Objekten zwischen verschiedenen Anwendungen, Prozessen, Rechnern und Domänen. Es entspricht in der Funktionalität und der Benutzung dem RMI von J2EE. Einen guten Überblick bietet [2].

Zur Kommunikation zwischen Client und Server hat .NET Remoting die notwendigen Anforderungen in unseren Projekten bezüglich Funktionalität und Antwortzeiten erfüllt. Allerdings sind bei der Nutzung einige Aspekte zu beachten, damit es nicht zu Performanzproblemen kommt. Dies wollen wir an einem Beispiel veranschaulichen.

.NET Remoting bietet standardmäßig zwei Serialisierungsverfahren:

- standardisierte Kommunikation über XML/SOAP via HTTP-Channel,
- binäre (proprietäre) Kommunikation via TCP-Channel.

In typischen Anwendungsfällen konnten wir bei der XML/SOAP-Kommunikation je nach Einzelfall deutlich längere Antwortzeiten messen (Faktor 2-10 gegenüber der binären Variante). Daher haben wir uns für die binäre Variante entschieden und werden uns im Folgenden auf diese beschränken.[4]

Wir möchten im Beispiel eine typische Tabelle vom Server zum Client übertragen. Diese hat in etwa folgende Datensätze:

Country	**Capital**	**Area**	**Number1**	**Number 2**	**Number 3**
Germany	Berlin	Europe	8234	3575	634,52
China	Bejing	Asia	2345	243	265,34
...	...	...	...	...	...

Es gibt vielfältige Möglichkeiten, die zu übertragenden Daten zum Transport in Objekte zu packen. Wir wollen drei Alternativen beleuchten:

- **DataSet**[5]
  Dabei werden die Daten aus der Datenbank gelesen und nach Empfehlung von Microsoft in ein DataSet geladen, welches bei Bedarf mittels Remoting zum Client gesendet wird.
- **Transportobjekt mit einer Hashtable [TO (Objects)]**
  Die Daten werden zeilenweise in ein Objekt gepackt. Diese *Zeilenobjekte* werden anschließend mit der Zeilennummer als Schlüssel in einer Hashtable verwaltet.
- **Transportobjekt mit einem zwei-dimensionalen Array**
  Die Daten werden in **einer** Matrix der Form

  ```
  object[,] matrix
  ```

  abgelegt, d.h. eine Zeile in der Tabelle wird in eine Zeile der Matrix abgebildet, jeder Zelleninhalt ist genau ein Eintrag in der Matrix.

[4] XML/SOAP ist zu verwenden, wenn entweder HTTP als Kommunikationsprotokoll vorgeschrieben ist bzw. mit Nicht-.NET-Anwendungen kommuniziert wird.

[5] Ein DataSet ist eine relationale Hauptspeicherdatenbank, auf die mittels SQL zugrgriffen werden kann. Ein DataSet kann mehrere Tabellen enthalten, die auch untereinander in Beziehung stehen. Dafür werden zusätzliche Meta-Information über die enthaltenen Daten im DataSet vorgehalten (Datentypen, Bezug der Tabellen untereinander etc.).

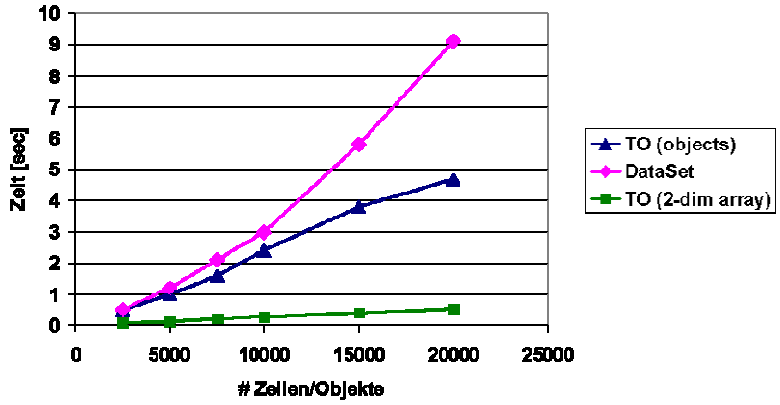

Abbildung 10 - Serialisierung

Abbildung 10 zeigt, dass die Serialisierung der Objekte, also die Umwandlung der Objektstruktur in den Datenstrom unterschiedlich viel Zeit in Anspruch nimmt. Die Umwandlung erfolgt am schnellsten beim zweidimensionalen Array und dauert bei der Verwendung von DataSets am längsten. Bei großen Datenmengen unterscheiden sich die Zeiten um einen Faktor größer 10.

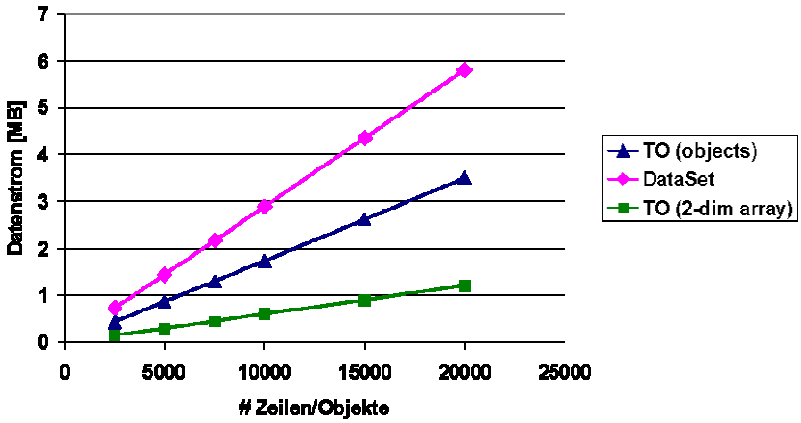

Abbildung 11 – Netzlast

Ebenso sind bei der Netzlast beträchtliche Unterschiede festzustellen, das zweidimensionale Array schneidet deutlich am besten ab, vgl. Abbildung 11.

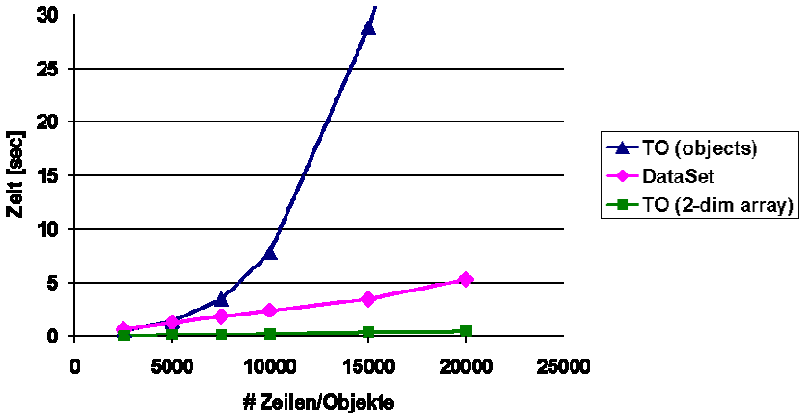

Abbildung 12 – Deserialisierung

Bei der Deserialisierung, also der Erzeugung von Objekten aus dem Datenstrom, ist das zweidimensionale Transportobjekt am schnellsten. Es mag überraschen, dass hier das Transportobjekt, in dem die Zeilen als Objekte in der Hashtable liegen, so schlecht abschneidet.

Wie lässt sich dieses Verhalten erklären?

Die zwei-dimensionalen Arrays haben die einfachste Struktur, lassen sich daher sehr schnell zwischen Datenstrom und Objekt umwandeln, denn es werden nur die notwendigen Daten übertragen.

DataSets mit ihrer umfangreichen Funktionalität sowie der Metainformation über die Datenbanktabellen besitzen in unserem Beispiel die komplexesten Strukturen. Folglich benötigen sie die meiste Zeit für die Serialisierung und erzeugen die größte Netzlast.

Der deutliche Overhead bei der Deserialisierung der Hashtables liegt zum einen darin, dass eine Vielzahl von Objekten erzeugt werden muss.

Daher haben wir uns zumindest bei zeitkritischen Anwendungsfällen für die binäre Kommunikation mittels zweidimensionaler Arrays entschieden. In Folge haben wir für den Datenbankzugriff DataReader anstatt DataSet eingesetzt.

In unseren Projekten hat sich gezeigt, dass Remoting für die Kommunikation sehr gut geeignet ist und sich leicht integrieren lässt, allerdings muss man sich die Struktur der zu sendenden Objekte genau überlegen und frühzeitig testen, damit man ein gutes Antwortzeitverhalten bekommt.

Generell gilt die Aussage: *Je einfacher die Struktur, desto schneller die Kommunikation.*

5.2 COM-Anbindung

Häufig müssen .NET-Anwendungen auf bestehende Programme zugreifen. Dies können selbst entwickelte Softwareprogramme oder vorhandene Produkte sein.

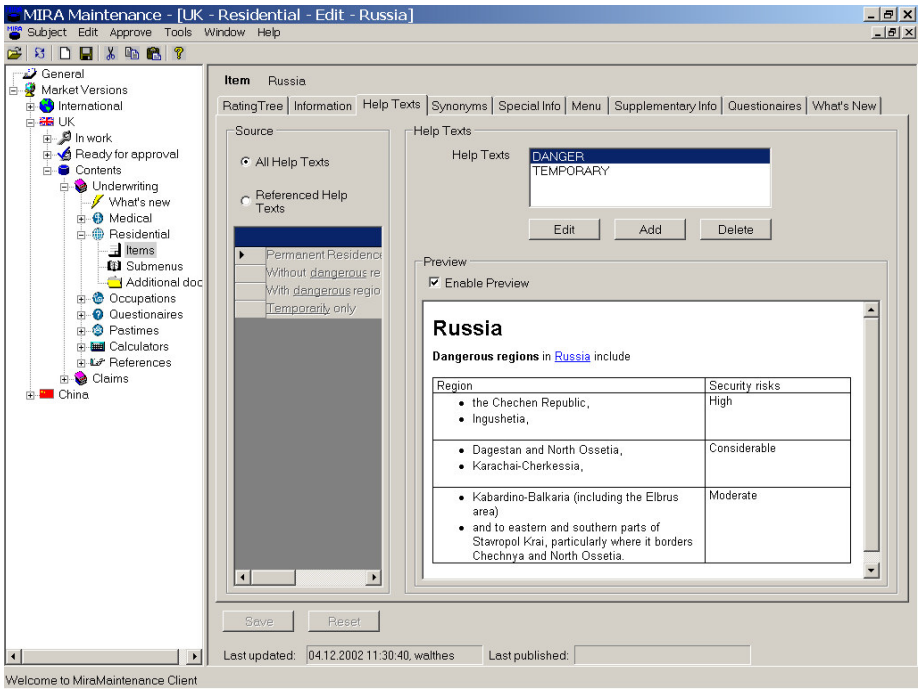

Abbildung 13 - Oberfläche mit Internet Explorer als Plug-In

In unseren Projekten mussten wir innerhalb der .NET-Applikation auf Word, Outlook, Internet Explorer und Acrobat Reader zugreifen. Während Outlook nur aufgerufen wurde, mussten Internet Explorer und Acrobat Reader für die Darstellung von HTML- und PDF-Dokumenten als Plug-In direkt innerhalb unserer Windows Forms-Oberflächen integriert werden, siehe Abbildung 13.

Noch komplexer war der Zugriff auf Word. Word wurde von unserem Client aus gestartet, damit der Benutzer Dokumente bearbeiten kann. Dabei haben wir auf bestimmte Aktionen des Anwenders innerhalb von Word reagiert, indem wir .NET-Dialoge in Word integriert angezeigt haben oder im Hintergrund Aktionen durchgeführt haben. Auf der anderen Seite wurde beim Speichern des Dokuments über COM die Dokumentstruktur ausgewertet und das Dokument unter anderem nach projektspezifischen Vorgaben nach HTML umgewandelt, die Konsistenz von enthaltenen Links geprüft und mit Schlagwörtern für die Suche versehen. Das folgende Beispiel zeigt, wie einfach man ein Dokument öffnen und auf einen Bereich (z.B. den aktuell selektieren Text) zugreifen kann:

```
using Word;
...

  try
  {
    Application wordApp  = new ApplicationClass();
    Document doc         = wordApp.Documents.Open(...);
    Selection sel        = wordApp.Selection;
  }
  catch( Exception e ){ ... }
}
```

Abbildung 14 - Codesnipplet zum COM-Zugriff auf Word

Ähnlich kann man über das COM-Modell von Word und damit der Struktur des Dokuments auf jeden Bereich, wie zum Beispiel Aufzählungen, Tabellenzellen, Grafiken und Links explizit zugreifen, Inhalte und Formatierungen auslesen und ändern. Eine Einführung in COM-Interop findet man unter [3].

Insgesamt ließen sich die COM-Zugriffe auf Word, Outlook, Internet Explorer und Acrobat Reader problemlos realisieren und selbst die Antwortzeiten waren überraschend gut.[6]

5.3 Internationalisierung

.NET verspricht eine weitgehende Unterstützung der Internationalisierung von Anwendungen. Dies war für uns eine zentrale Anforderung mit folgenden konkreten Punkten:

- Anzeige der Oberflächenelemente in Englisch,
- Daten, die vom System angezeigt und gepflegt werden, liegen in verschiedenen Sprachen vor. Dabei ist der Standard-Zeichensatz ebenso zu unterstützen wie spezielle Zeichensätze für Russisch, Chinesisch und Hebräisch,
- sprachabhängige Sortierungsregeln und Algorithmen für den Stringvergleich,
- länderspezifische Formatierung für Zahlen, Datum und Uhrzeit,
- unterschiedliche Schreibrichtungen,
- innerhalb der Anwendung und sogar innerhalb einzelner Dialoge sind mehrere Sprachen gleichzeitig zu unterstützen,
- Implementierung von kulturabhängig unterschiedlichen fachlichen Regeln.

Zu typischen Anforderungen für die Internationalisierung siehe [4].

Im Namespace `System.Globalization` bietet .NET zahlreiche Funktionen zur Internationalisierung einer Anwendung an (vgl. [5]). Hierzu zählen z.B.

[6] Die Integration von Word als Plug-In innerhalb von Windows Forms ist derzeit nur über Umwege (z.B. Internet Explorer) möglich und damit unter Umständen mit Einschränkungen verbunden, z.B. kann nicht auf alle Word Menüs zugegriffen werden.

- der Lokalisierungsmechanismus, der die Informationen zur Sprache, zum Land bzw. zur Region und zum Kalender zur Verfügung stellt (`System.Globalization.CultureInfo`),
- die lokalisierte Behandlung von Datum, Uhrzeit und Zahlen (`DateTimeFormatInfo`, `NumberFormatInfo` und `Calender`),
- die sprachabhängige Sortierung und Stringvergleich (`System.Globalization.SortKey` bzw. `System.Globalization.CompareInfo`).

Ressourcen für die Lokalisierung finden sich im Namespace `System.Resources` und können über die Klasse `System.Resources.ResourceManager` abgerufen werden. UNICODE -Support ist durchgängig im .NET Framework gegeben und die Konvertierung zwischen den Zeichensätzen kann man bei Bedarf in den Klassen `System.IO.StreamReader` bzw. `System.IO.StreamWriter` durchführen.

Diese umfangreiche Funktionalität wird im Projekt genutzt. Die interne Datenhaltung erfolgt vollständig in UNICODE, die Komponenten, die sprachspezifische Funktionen implementieren, benutzen die Klassen aus den Namespaces `System.Globalization` und `System.Resources`. Dennoch sind projektspezifische Erweiterungen in aller Regel für internationale Projekte erforderlich:

Konkret war die Zuordnung der Länder, Sprachen und Kulturen nicht mit der Standardzuordnung identisch. Griechenland hat derzeit z.B. keine eigene griechische Version sondern benutzt eine englische Version. Implizit in unserer GUI war zu berücksichtigen, dass die Sprache/Kultur nicht anwendungsspezifisch war, sondern dialogspezifisch, teilweise sogar feldspezifisch. Es wurden mit dem System von einem Anwender parallel die Daten mehrerer Versionen gepflegt. Folglich musste in den Windows Forms-Komponenten die Sprache explizit gesetzt werden. Einzelne Controls, wie zum Beispiel der `DatePicker`, waren davon jedoch unbeeindruckt und blieben in der Sprache des Betriebssystems. Dort waren Workarounds ebenso wie bei der Unterstützung verschiedener Schreibrichtungen innerhalb von Controls notwendig.

Zusammenfassend lässt sich sagen, dass das Framework einem bei der Internationalisierung von Anwendungen viel Arbeit abnimmt, projektspezifische Anpassungen und Erweiterungen durchaus notwendig werden können.

5.4 Entwicklungsumgebung

In beiden Projekten kommt eine sehr ähnliche Software-Entwicklungs-Umgebung zum Einsatz:

- Visual Studio.NET,
- Visual Source Safe,
- Modellierungstool Rational Rose bzw. Rational XDE,
- selbst entwickelte Umgebung zum automatisierten *Global Build* der gesamten Anwendung,
- Werkzeuge von Drittherstellern:
  NUnit als Testframework, NAnt als Build-Tool, NDoc zur Generierung von Online-Dokumentation, DLL-Viewer, Monitoring Tools (AQTime.NET).

Motivation für die Global Build Umgebung

Regeln zur Steuerung des Build-Prozesses wie in Makefiles üblich, werden vom Visual Studio.Net automatisch erzeugt. Hierzu definiert der Entwickler sogenannte *Projects* mit den Quellen, die zur Erstellung einer Komponente oder *Assembly* benötigt werden. Alle Projects, die untereinander in Abhängigkeit stehen, weil sie sich referenzieren, werden in so genannte *Solutions* gruppiert, damit das Visual Studio.Net die Build-Reihenfolge festlegen und sicherstellen kann, dass keine zirkulären Referenzen bestehen.

Man kann alle Quellen des zu bauenden Systems in Projekten innerhalb einer Solution (Single Solution-Ansatz) gruppieren und erhält damit die oben genannten Vorzüge, also automatischer Build in richtiger Reihenfolge und die Vermeidung von Zirkeln.

Das bedeutet allerdings, dass alle Quellen des Gesamtsystems auf allen Entwicklungsrechnern vorliegen müssen. Wir entwickelten im größeren Team über 1300 C#-Klassen, weshalb dieser Ansatz für uns nicht in Frage kam.

Die Alternative, pro Komponente eine eigene Solution anzulegen hat den Vorteil, dass nur die Quellen der aktuell entwickelten Komponenten lokal vorliegen müssen und alle übrigen Assemblies nicht als Quellcode, sondern als binäre Assemblies ausreichen. Der Nachteil dieses Ansatzes ist, dass der Entwickler selbst darauf achten muss, dass er keine zirkulären Referenzen aufbaut und sicherstellen muss, dass seine Komponenten in der richtigen Reihenfolge gebaut werden.

Da es nahezu unmöglich ist in einem realistischem Zeitaufwand die benötigten Komponenten nach einer Änderung/Erweiterung in richtiger Reihenfolge zu bauen, entwickelten wir zu diesem Zweck unsere Global Build Umgebung.

Eigenschaften der Global Build Umgebung

Diese Umgebung wurde mit standard Unix Tools (awk, sed, grep, find) unter CYGWIN realisiert und greift auf das Visual Source Safe über die Kommandozeilenschnittstelle zu.

Der Build-Server wird automatisch jede Nacht gestartet und kann zusätzlich jederzeit von Hand angestoßen werden, um das Gesamtsystem in sich konsistent in den neuesten Stand zu übersetzen.

Entwickler können mittels Label angeben, welche Quellen in den Global Build einfließen sollen. Es werden alle benötigten Quellen in der korrekten Version aus dem Source Safe angefordert, um diese in das makefile einzutragen, das anschließend von NAnt dazu genutzt wird, mit Hilfe des C#-Compilers den Quellcode zu übersetzen. Dabei werden auch Resourcencompiler und Lizenzgeneratoren angestoßen.

Nach dem Übersetzen werden die frisch gebauten Assemblies in entsprechende globale Verzeichnisse kopiert und die Entwickler über Emails auf den neuen Stand und ggf. aufgetretene Probleme aufmerksam gemacht.

Es gibt noch eine Reihe weiterer Details, die den Rahmen dieses Artikels sprengen würden. Zusammenfassend lässt sich aber feststellen, dass der Aufbau dieser Umgebung für die Entwicklung im großen Team nötig sein kann und dessen Aufwand nicht zu unterschätzen ist, siehe Abschnitt 5.6.

Vorzüge der Entwicklungsumgebung

Allgemein lässt sich sagen, dass die Werkzeuge eine mächtige Entwicklungsumgebung darstellen. Wie in den früheren Entwicklungsumgebungen von Microsoft integriert sich Source-Safe zun Check-In und Check-Out sowie zum visuellen Versionsvergleich von Quellcode.

Der Editor im Visual Studio.Net bietet die erwartete, hilfreiche Funktionalität zur Entwicklung wie zum Beispiel Code-Vervollständigung, Syntax-Highlighting, ein- und ausblenden von Implementierungsdetails mittels Regions sowie die Erzeugung von XML-Kommentaren.

Die Integration anderer Werkzeuge wie den Disassembler `ildasm.exe` oder Shellskripten ist einfach möglich.

Probleme mit der Entwicklungsumgebung

Seit der Final-Version des Visual Studios sind die Fehler seltener, aber immer noch vorhanden. Hauptkritikpunkt ist der Visual Designer, der auch heute noch ungefragt Controls auf Dialogen entfernt und grafische Elemente unerwartet mit ungewollten Werten vorbelegt. Der Ansatz, für grafische Elemente Quellcode zu generieren ist sehr gelungen (eine gute Verbesserung gegenüber VB 5), allerdings ist diese Codeerzeugung nicht immer fehlerfrei.

Es waren Konventionen nötig, um im größeren Team an gemeinsamen Komponenten zu arbeiten. So musste ein globales Assembly-Verzeichnis angelegt werden, das auf jedem Rechner mit dem selben logischen Laufwerksnamen gemappt werden musste. Denn bei der Referenzierung von Komponenten speichert das Visual Studio.Net den Pfad auf die Referenz. Um eigene Assemblies referenzieren zu können, war es nötig einen Eintrag in der Registry vorzunehmen, in dem der Assembly-Pfad angegeben wurde.

Das Öffnen großer Solutions, die unter Kontrolle des Visual Source Safe stehen braucht relativ lange Zeit.

Ein Kritikpunkt an der Entwicklungsumgebung ist die mangelnde Unterstützung beim Build-Prozess bei der Entwicklung von Komponenten im Team. Zu diesem Zweck entstand unsere eigene Global Build Umgebung.

5.5 Betrieb

Eines der von der sd&m AG entwickelten Softwaresystemen ist derzeit produktiv im Einsatz. Bei dessen Inbetriebnahme und dem laufenden Betrieb wurden folgende Erfahrungen gemacht:

Installation

Für die Installation musste auf den Windows 2000-Servern das .NET-Framework inklusive Servicepack 2 sowie unsere Anwendung installiert werden. Die Installation des Frameworks gestaltete sich dabei problemlos; in wenigen Minuten war die .NET-Infrastruktur auf den Servern eingerichtet.

Auch die Installation der Anwendung, die als Windows-Dienst arbeitet, konnte innerhalb kurzer Zeit vorgenommen werden. Es musste lediglich die Assemblies (DLLs) kopiert, und anschließend der Dienst mittels `installutil.exe` eingerichtet zu werden.

Die Software-Verteilung auf die Clients bereitete allerdings Probleme. Das Framework sowie die hinzugekaufte Komponente Crystal-Reports ließen sich nicht so einfach mittels NetInstall verteilen.[7]

Performance

Die beiden Server sind mit jeweils zwei Pentium 4, 2,4 GHz, 1GB RAM ausgestattet. Unsere Anwendung läuft ohne Performanceprobleme mit Antwortzeiten in der Regel unter 1 Sekunde. Clientseitig ist beim ersten Laden eines Dialoges eine Verzögerung festzustellen, die auf den JIT hindeutet, der mit großen Dialogen wohl doch einiges zu tun bekommt. Die Hauptlast der Anwendung liegt wie erwartet auf dem Datenbankserver.

Stabilität

Bezüglich der Stabilität des Systems gibt es derzeit keinerlei Probleme. Die Garbage Collection des Servers bzw. eventuelle Speicherlecks konnten wir nicht beobachten, allerdings ist dazu zu bemerken, dass unser Serverdienst täglich neu gestartet wird.

5.6 Als Pionier hat man Mehraufwand

Microsoft verspricht bei der Petstore Portierung eine deutliche Reduktion der Entwicklungszeiten, vgl. Abschnitt 1. Stimmen diese Aussagen?

Zunächst hat man Zusatzaufwand von etwa 3 Bearbeitertagen (BT) pro Mitarbeiter bei der Einarbeitung in die Programmiersprache C#, sofern die Mitarbeiter/innen bereits über Java

[7] Konkret ließ sich die erste Version des Frameworks nicht im *quiet mode* installieren. Crystal Reports konnte nicht mit XCopy-Deployment installiert werden, da zur Lizensierung eine MSI-Datei installiert werden musste, für die zudem Administrator-Rechte benötigt werden.

oder C++ Knowhow verfügen. Hinzu kommt das Studium der Klassenbibliotheken des .NET Frameworks. Hier hängt der Aufwand von der Komplexität der jeweiligen Teilbibliotheken und von den konkreten Aufgaben ab, nämlich davon, ob man die Klassen nur verwendet oder anpassen und erweitern muss. Weiterer Aufwand ist für die Installation der Entwicklungsumgebung mit der Erstellung von Nutzungskonzepten vorgesehen.

Unsere Erfahrungswerte sind:

- Installation Basiskomponenten (1 BT pro Mitarbeiter),
- Einarbeitung in Tools (ca. 2 BT pro Mitarbeiter),
- Nutzungskonzept (3 BT),
- Eventuell globaler Build: Hier ist der Aufwand schwer abschätzbar, je nach Komfort muss mit 5 bis 20 BT gerechnet werden.

Zusätzlicher Aufwand entstand durch interne Bugs, die sich aber für eine erste Version in Grenzen halten. Für konkrete Projekte ist zu bedenken, dass im Vergleich zu Java zurzeit nur wenige ausgereifte Komponenten von Drittherstellern wie O-R-Mapper oder Tabellencontrols existieren. Alles in allem kann man sich in der neue Technologie sehr schnell zurecht finden.

Es ist sinnvoll, wie bereits oben dargestellt, zunächst eine Infrastruktur / Basiskomponenten zu entwickeln und darauf aufbauend die fachlichen Anwendungen zu realisieren. Der Aufwand für den Aufbau der Infrastruktur hängt sehr stark davon ab, welche Funktionalität man zur Verfügung stellen will und was man allgemein für die Anwendungen benötigt. Hier ist je nach Umfang mit einer Größenordnung von 200-400 Bearbeitertagen zu rechnen. Anschließend kann die Entwicklung sehr effizient durchgeführt werden.

In Summe lässt sich sagen, dass man beim ersten .NET Projekt als Pionier mit Mehraufwand rechnen muss. Nach der Erstellung der Basis ist die Entwicklung sehr effizient möglich. Einsparungen gegenüber J2EE lassen sich bei der Erstellung von Web Clients, bei Webservices und bei der Anbindung von Microsoft Produkten erzielen, da die Entwicklungsumgebung den Entwickler sehr gut unterstützt. Es ist jedoch zu erwarten, dass die Java-Entwicklungsumgebungen in Kürze nachziehen werden.

6 Fazit

.NET trägt, wie wir aus der Erfahrung der beiden .NET-Projekte für die Real I.S. und die Münchener Rück behaupten können. Wir haben keine Probleme im Betrieb bei Performanz und Stabilität bzgl. Last und Verfügbarkeit.

Das .NET Framework bietet eine gute Basis für die Entwicklung, wie an den Beispielen Windows Forms, Remoting, COM-Zugriff und Internationalisierung dargestellt wurde. Trotzdem sind Erweiterungen des Frameworks und die Erstellung einer Infrastruktur mit Basiskomponenten gerade für größere Projekte sinnvoll. Beim konkreten Einsatz empfehlen wir, frühzeitig eine Referenzarchitektur zu entwerfen und zu erproben sowie die Entwicklungsprozesse klar zu definieren. Ferner ist Aufwand für die Einarbeitung und für Problemlösungen einzuplanen. Ansonsten ist der Aufwand mit dem bei der Entwicklung mit J2EE vergleichbar, sobald die Kinderkrankheiten beseitigt sind und stabile Komponenten von

Drittherstellern in größerem Maße zur Verfügung stehen. Einsparungen lassen sind bei der Erstellung von Web-Anwendungen, Webservices und bei der Integration von Microsoft Produkten erzielen.

7 Literaturhinweise

[1] http://www.gotdotnet.com/compare

[2] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp

[3] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconIntroductionToCOMInteroperability.asp

[4] Fuchs, Marc; Fuchs, Dirk; Haller, Harald: *Internationalisierung von Java-Anwendungen*, Java-Spektrum, Ausgabe 5/2001

[5] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxconWhyDesignInternationalSoftware.asp

[6] Ramisch, Alexander; Haller, Harald: *Erfahrungsbericht: .NET in der Praxis*, Professional Services, Ausgabe 02/2002

8 Unternehmen

Mit 906 Mitarbeitern und 138 Mio. Euro Umsatz (2001) gehört die **sd&m AG - software design & management** - zu den großen deutschen Software- und Beratungsunternehmen. Die sd&m AG ist Teil der Cap Gemini Ernst & Young-Gruppe, einem der größten IT-Dienstleister weltweit, mit gleichermaßen starker Präsenz in Europa und den USA. Die Betreuung ihrer Kunden vor Ort gewährleistet sd&m durch Niederlassungen in Hamburg, Berlin, Düsseldorf, Köln/Bonn, Frankfurt, Stuttgart, München und Zürich. Kunden sind die Allianz, BMW, die Commerzbank, DaimlerChrysler, die Deutsche Bahn, die Deutsche Telekom, Münchener Rück, Thomas Cook u.a.

Die sd&m AG konzipiert und entwickelt maßgeschneiderte Informations- und Internetsysteme für unternehmenskritische Prozesse. sd&m berät in allen Fragen der Informationstechnik. Dem dient die Kernkompetenz, das Software-Engineering, das sd&m zum E-Business-Engineering ausgebaut hat. sd&m macht große Projekte für unterschiedlichste Anwendungen auf allen gängigen Plattformen und integriert die Systemlandschaft.