Analysis of Performance and Synchronisation Aspects of Microsoft’s Dataset Technology in a Web-based Application

Hannes Silbernagl

14th October 2004
Abstract

My work as a trainee was to build an online booking system for a travel agency which allows to manage each aspect of one day trips. A web application generates dynamic Web pages containing various types of markup language (HTML, XML, and so on) in response to requests. One of the most advanced techniques to create web applications is probably ASP.Net from Microsoft, which was used for this work. A new platform offers always great chances in offering new possibilities, but not without leading to new problems. The two problems which are addressed in the second part of this thesis are performance in a web application and how this is somewhere related to the frequency that concurrency exceptions will arise. This paper is also intended to show various possible implementations for the same problem and what has to be considered by building a sound, solid and reliable web application.
Contents

1 Introduction 3

2 The Project "Martin Reisen" 4
   2.1 The Customer Wishes .................................. 4
   2.2 Technology chosen ..................................... 5
   2.3 The Online Booking System .............................. 5
   2.4 End of the Project .................................... 8

3 Performance in ADO.Net 10
   3.1 Basics .................................................. 10
   3.2 Data Retrieval Classes ................................. 11
      3.2.1 Datasets ........................................... 11
      3.2.2 Data Readers ...................................... 11
   3.3 Known Problems ........................................ 12
   3.4 Performance Analysis .................................. 12
      3.4.1 Performance with Data Readers .................. 12
      3.4.2 Performance with an incrementally filled Dataset .. 14
      3.4.3 Performance having one initially filled Dataset .... 14

4 Concurrency Exceptions in ADO.Net 15
   4.1 Basics .................................................. 15
   4.2 Types of Concurrency Control .......................... 15
      4.2.1 Pessimistic Concurrency .......................... 15
      4.2.2 Optimistic Concurrency ........................... 16
      4.2.3 Last in Wins ....................................... 16
   4.3 Concurrency Control in ADO.NET and Visual Studio.NET .. 16
   4.4 Concurrency Exceptions ................................. 17

5 Concluding Remarks 20
Chapter 1

Introduction

A online booking system has to deal with many things. Besides the obligatory login section it's about trip statistics, bills, administration of the trips, the customers and the partners, moreover offers various printouts and some other functionalities. Programming the web has slight differences than programming for a stand alone machine. Two of this differences are dealt with in chapter 3 and chapter 4.

Performance in a web application is quite critical. Studies showed that a standard internet user will leave the site after waiting for one second. Using an application in the web and therefore waiting not only once but many times will be quite annoying for the customer and maybe this will decide on the willingness of the user to use the system or not. Differences in performance are showed in this thesis by using different data retrieval classes. In the world of Microsoft there are two of them; the datareader and the dataset. The use of one of those has a deep impact of how many concurrency exceptions may occur.

Concurrency exceptions will appear more often, the older the data to update is. They can arise quite easily in my program, as maybe two customers will try to book the same seat at the same time. In practice, the handling of a concurrency exception depends on the probability how often one may appear and exactly this probability may be strongly influenced by the use of either the datareader or the dataset. Here the circle of the tight dependency between one of the two data retrieval classes and the arise of a concurrency exception closes.
Chapter 2

The Project ”Martin Reisen“

My internship took place in the time between the 16.02 and the 14.05.2004. I worked for the company ”Disaster OHG“ located in Tscherms. The customer was the travel agency ”Martin Reisen“ located in Lana. The initial situation was that they had a personalized software for their bus booking system which worked quite well and which was in use for around 3 years. The limitation of the system was that the programmer who wrote this application was no longer willing to give support and moreover, given the old programming language and old database system the decision was taken to replace this application with a complete new one. The biggest enhancement the new application should provide was the possibility to allow bookings directly into the system over the internet by providing a password and a username to the allowed users.

It was decided that the work will be divided on two programmers. For me the work was decided to be fulltime and for the second programmer the amount of time should only occupy half the day. But after the planning of the database, which was done in teamwork, the other programmer left the project and so the whole project was finished by me. This includes the contact to the customer, the design of the web pages, the further enhancement of the database, the development of the application and lastly the presentation in front of about 50 people.

2.1 The Customer Wishes

The main characteristics of the project are:

- The project is a web application.
- Other travel agencies can book trips at seat basis after a login. The price is auto calculated, the print out equals the voucher. The bus is represented graphically to ease the booking operation. Moreover, each travel agency can change or delete its own bookings.
• The customer has the same functionality as described above but without fewer constraints, i.e. he can modify or delete each booking, not only his own; he can see every trip, including those in the past, whereas the others can only see those in the future.

• Moreover, the customer has some more functions like inputting new trips, trip statistics, trip overviews, everything concerning bills, input of data of resellers like travel agencies, hotels and tourism offices and various printouts.

2.2 Technology chosen

The solution was realised under Microsoft’s Visual Studio .Net. The program is written in ASP.Net with C#. The database used is the open source database ”Firebird“, ”IBExpert“, also known from ”Interbase“ was used to administrate Firebird. Microsoft with its Visual Studio .Net seems to offer the best programming environment for the moment. The only similar product is the Borland C# Builder which is based on the Microsoft .Net Framework but this is the real first version and it seemed me to risky to use it for building a real business application. Java was also taken into consideration and had been probably also a good decision but I think that you have to program much longer for the same amount of work to be done. ASP.Net on its own is very powerful because it is not a script language but a real programming language. This means that it is more difficult to program but also much more powerful. You can program either in a .Net language or in HTML. With .Net the source code will be automatically translated in HTML. Moreover in contrast to Java you don’t have to work with the system settings, you just install the Internet Information Server called IIS and you’re done. The database I originally planned to used was MS Access, but strange enough this gave me some problems accessing the Db from within Visual Studio. Fortunately I read an article about open source databases and the winner has been Firebird. I informed myself a bit about Firebird and when I saw that they also provided .Net drivers on the website on http://firebird.sourceforge.net/ the decision was taken. IBExpert was formerly only known together with Interbase but works fine also with Firebird. The personal edition which only has a few constraints can be downloaded for free.

2.3 The Online Booking System

As the possibility to book seats online was the biggest news for the customers we decided to work on this first. The provided proposal consisted of 4 pages. These are the login section, a second site where you choose destination
The first things to do were to establish a connection with the database and to provide me an overview of ASP.Net. Then I started to fill my first dropdown lists and data grids with datasets. The next steps were to introduce myself to some web relevant techniques as parameter passing between round trips, the autopostback method and stuff like this.

When I first showed the program to the customer he changed or precised his ideas. My program at this time had two textboxes where to enter the period of time and then after the resulting SQL query only the possible trip destinations were filled into the dropdown list. This seemed to me to be the best solution because the main customers of this one day trips are Germans, Italians and other guests which are here only for a limited time and so they probably go to the travel agencies and ask which trips are offered within the next 2 weeks. Anyway, the customer wanted it the other way round and the customer is always right. So, the first change was that first you select the destination and then the dropdown list is filled with the dates the trips take place. The big change concerned the main booking site. On it you can see all relevant data about the trip, you can enter the customers information, you select the desired entry and you can click on single seats to reserve them. At that time my drawn bus was a rectangle created automatically with 4 buttons per row and the columns depending whether the bus has 16, 48, 50, 52 or 70 places. Mr. Plattner Martin, the boss of "Martin Reisen" explained me that the drawing of the bus must be different, because people
want to know exactly where there is the toilet, where the staircase, where there are the tables and whether the seats are in driving direction or vice versa. Moreover they want to see where there is the driver and the courier. This kind of drawing is hard to do dynamically except you save the fixed position of each button and label in a database, so I preferred to draw them once by hand. The disadvantage of this solution is that once you have to change something you have to change it on the site of each bus.

The database has been changed later on, because in the beginning each trip had one starting point with the entry having a plus or minus time relevant with trip time. The change was that a trip hasn’t a time anymore but each entry has its own time. This certainly makes sense but it was a lot of changes for me to do in one week. Finally the presentation was a success, everyone was happy and I told the audience that the bad news is that the program can be used not before another month of work. They agreed to that and I continued my work.

One important change that complicated my life a lot has been discussed during the presentation. The price is auto calculated regarding the number of seats. No problems with that. The price increases if you add seats to a formerly booked trip and the price is reduced if you delete seats. Still no problems with that. The textbox with the final price is set to editable because there are two special prices. First you have to know that the price per seat is composed by the trip price and the extra price which is for
example the price for a ticket or something like this. This distinction is important as the external online bookers are paid only on the amount of the trip price. The two exceptions however are children under 4 years and girls and boys between 4 and 16 years. Children under 4 have trip price zero and full extra price, children between 4 and 16 have half the trip price and full extra price. This was really a lot of work until it worked fine, because you always have to know whether you are currently working on a new reservation, if you are a modifying a reservation or if you are deleting from a reservation. Then you have to know how many seats the customer currently owns because the number of children together can not exceed the number of seats, because this would cause a negative price. Altogether the complexity of the price calculation is really high because it depends on many factors and avoiding errors in that was a lot of testing, trying and programming. The thing that complicated it a lot was the autopostback functionality, because this made it necessary that the price is auto calculated on each round trip and not only at the end, but on the other side the customer liked it, because it gives an immediate feedback.

After the insertion of the data each of the possible buttons should bring you to a confirmation page, nevertheless whether you are clicking on delete, change or save. One requirement was that they want to provide a printout of the booking which was equivalent to the voucher they used until now and the external booker should be constrained to do the printout. So the best solution I could find was to automatically bring up the print dialog each time you press the delete, change or save button. This first was a technical problem because I used JavaScript for the print command and the Response.Redirect() method for the transfer to the next page. When I tried it, everything seemed to work well but the print window didn’t appear. As I found out this was because the redirect() method has the priority over the JavaScript method and so I had to change it in Appendheader() with the possibility to have a time delay between the appearance of the print window and the redirection. I created a method printview() where all the non relevant commands are set to invisible and so this was a good solution for this requirement.

2.4 End of the Project

At that time the customer knew a comparable solution of a competing travel agency which is only text based and much more error prone because of the need to input much data by hand. One thing we changed in the end is that we provided the printouts in PDF because they look nicer and moreover offer the possibility that they can be stored. One drawback we experienced is that the program didn’t work on all computers. The problem was that on the graphical representation of the bus there is an overlaying panel which
gives the buttons the shape of the bus. On some computers there was no possibility to click on the buttons, it seemed that the clicks reached only the overlaying panel. I found out that the whole ASP.Net technology is only fully compatible with the Internet Explorer Version 6 or higher. I solved the problem by reading out the information of the browser and if the browser wasn’t IE Version 6 or higher the program denies the entry even with a valid username and password.
Chapter 3

Performance in ADO.Net

3.1 Basics

ActiveX data objects for the .Net Framework (Ado.Net) is the latest database access technology from Microsoft. Ado.Net uses .Net Common Language Runtime (CLR) services to manage the library, and it utilizes Extensible Markup Language (XML) to cache the data and exchange data among applications over the Internet. It’s a set of object-oriented classes that provides a rich set of data components to create high-performance, reliable, and scaleable database applications for client-server applications. In the Ado.Net model applications connect to the data sources when they are reading or updating the data. After that, the connection closes. This is important because in client-server or distributed applications, having connection resources open all the time is one of the most resource-consuming parts. You don’t have to connect to a data source all the time; the only time you need to connect to a data source is when you are reading and writing final changes to data source. Ado.Net uses SQL queries and stored procedures to read, write, update, and delete data from a data source. You use SQL queries through the Ado.Net command object, which returns data in the form of DataReader or DataSet objects which we will see are two important classes when dealing with performance. After that connection closes, you use DataSet objects to work with the data and connect to the data source again when you need to update the data source. A typed dataset, because of its underlying XML schema can be treated as the database itself having all the constraints also on the dataset. This prevents from working with data which in the end cannot be updated because one of the constraints weren’t satisfied.
3.2 Data Retrieval Classes

3.2.1 Datasets

Datasets store data in a disconnected cache. The structure of a dataset is similar to that of a relational database; it exposes a hierarchical object model of tables, rows, and columns. In addition, it contains constraints and relationships defined for the dataset. The fundamental parts of a dataset are exposed to you through standard programming constructs such as properties and collections. For example:

- The DataSet class includes the Tables collection of data tables and the Relations collection of DataRelation objects.
- The DataTable class includes the Rows collection of table rows, the Columns collection of data columns, and the ChildRelations and ParentRelations collections of data relations.
- The DataRow class includes the RowState property, whose values indicate whether and how the row has been changed since the data table was first loaded from the database. Possible values for the RowState property include Deleted, Modified, New, and Unchanged.

An ADO.NET dataset is one view - a relational view - of data that can be represented in XML.

3.2.2 Data Readers

You can use the ADO.NET DataReader to retrieve a read-only, forward-only stream of data from a database. Results are returned as the query executes, and are stored in the network buffer on the client until you request them using the Read method of the DataReader. Using the DataReader can increase application performance both by retrieving data as soon as it is available, rather than waiting for the entire results of the query to be returned, and (by default) storing only one row at a time in memory, reducing system overhead.

You use the Read method of the DataReader object to obtain a row from the results of the query. You can access each column of the returned row by passing the name or ordinal reference of the column to the DataReader. The DataReader provides an unbuffered stream of data that allows procedural logic to efficiently process results from a data source sequentially. The DataReader is a good choice when retrieving large amounts of data because the data is not cached in memory.

Note that while a DataReader is open, the Connection is in use exclusively by that DataReader. You will not be able to execute any commands for the Connection, including creating another DataReader, until the original DataReader is closed.
3.3 Known Problems

My program leads you through 3 pages, the login section, the site where you choose destination and date and finally the main booking site. This small application is optimal for testing purposes. Nevertheless, it is always possible that for some reason a website will become well known and so has to handle a big increase in requests, or some times the application will be expanded or maybe the page is offering different, bigger data with the same system where performance will have a new importance. So it is very important that performance should been taken into account when planning a web project, even if it seems not relevant for the moment. Moreover, also the freshness of data plays an important role. On the one hand the use of a dataset reduces dramatically the amount of database accesses, but leads to older data if the dataset is not refreshed in each drawback to the server and on the other hand older data leads to an increased effort to cope with exception handling deriving from invalid database updates.

3.4 Performance Analysis

3.4.1 Performance with Datareaders

This is the scenario where we don’t use datasets at all as you can see in figure 3.1. For each need of data a datareader is created and the data is read out. Clearly the datareader will be the fastest way to retrieve data from the database. In plus, there will be no data around, neither on the server, nor on the client side. A drawback are the many database accesses, taking into consideration the many post backs in the booking site you come up to tens of accesses for each single booking. Also with an intelligent use of placing the first retrievals in the non auto post back region, heavy traffic is caused between the database and the web server. You also have the biggest effort in programming. Beside the effect that you have to repeatedly write connections to the database, you also have to write update, insert or delete commands. If you are filling a dataset with an appropriate select command,
Figure 3.2: Incremented Dataset

Figure 3.3: 1 big dataset
the insert, update and delete commands will be auto generated for you. Especially a typed dataset, which will become a first class object, is directly accessible not only as a collection. The last great advantage of using data readers is that you use them exactly when you are in need of this data, so the data will always be fresh. This means that you can practically reduce possible concurrency problems to a minimum.

3.4.2 Performance with an incrementally filled Dataset

Here we do use a dataset. We create it in the first site and expand it site by site with the information needed; this is shown in figure 3.2 With datasets the data retrieval will be certainly slower that with data readers, because beside the raw data, also the metadata is written out and with this knowledge data is organised as collection of rows which are stored in collections of tables. Depending whether the dataset is typed or not they become first class objects, meaning that you can access them directly, or remain normal collection where you have to walk through. Anyway performance would not be too bad, because the data you extract is the minimal one needed for that page. You will not use server memory because you send the dataset to the client to store it there, with the result that with slow connections or big datasets the speed of the page load will decrease dramatically. On the positive side we have that we have reduced database accesses, normally one per page and an additional one if you are updating to the data source and it is also easier to program because all you have to do at the end of the application is to call the data set update method, which will create the appropriate SQL query on its own. Moreover you have a high probability of avoiding concurrency problems because the needed data is retrieved only on the page where you need it.

3.4.3 Performance having one initially filled Dataset

Here you will have one big dataset created as soon as you passed the login section as shown in figure 3.3 You as programmer know in advance which data is needed in the sequent pages and this whole amount of data is captured in one step. This causes problems if the dataset is really big together with a high number of users, because the web server will run out of memory. The page load remains fast, because only the needed data is sent to the client and what is really important is that you have the lowest amount of database accesses. This may be a good solution if you have more web servers than database servers, passing so some amount of work from the database to the web server. In my opinion this solution is only suitable if data is not opposed to great changes in short time, if the dataset is rather small and most of all if the page flow is linear and does not go through to many pages.
Chapter 4

Concurrency Exceptions in ADO.Net

4.1 Basics

When multiple users attempt to modify data at the same time, controls need to be established in order to prevent one user’s modifications from adversely affecting modifications from simultaneous users. The system of handling what happens in this situation is called concurrency control.

4.2 Types of Concurrency Control

In general, there are three common ways to manage concurrency in a database:

- **Pessimistic Concurrency Control** - a row is unavailable to users from the time the record is fetched until it is updated in the database.

- **Optimistic Concurrency Control** - a row is unavailable to other users only while the data is actually being updated. The update examines the row in the database and determines whether any changes have been made. Attempting to update a record that has already been changed results in a concurrency violation.

- **"Last in Wins"** - a row is unavailable to other users only while the data is actually being updated. However, no effort is made to compare updates against the original record; the record is simply written out, potentially overwriting any changes made by other users since you last refreshed the records.

4.2.1 Pessimistic Concurrency

Pessimistic concurrency is typically used for two reasons. First, in some situations there is high contention for the same records. The cost of plac-
ing locks on the data is less than the cost of rolling back changes when concurrency conflicts occur.

Pessimistic concurrency is also useful for situations where it is detrimental for the record to change during the course of a transaction.

However, pessimistic concurrency control is not possible in a disconnected architecture. Connections are open only long enough to read the data or to update it, so locks cannot be sustained for long periods. Moreover, an application that holds onto locks for long periods is not scalable.

If your underlying data source supports transactions, you can simulate pessimistic concurrency by updating your data within a transaction.

4.2.2 Optimistic Concurrency

In optimistic concurrency, locks are set and held only while the database is being accessed. The locks prevent other users from attempting to update records at the same instant. The data is always available except for the exact moment that an update is taking place.

When an update is attempted, the original version of a changed row is compared against the existing row in the database. If the two are different, the update fails with a concurrency error. It is up to you at that point to reconcile the two rows, using business logic that you create.

4.2.3 Last in Wins

With "last in wins", no check of the original data is made and the update is simply written to the database. It is understood that the following scenario can occur:

User A fetches a record from the database. User B fetches the same record from the database, modifies it, and writes the updated record back to the database. User A modifies the 'old' record and writes it back to the database. In the above scenario, the changes User B made were never seen by User A. In a normal program this situation is never acceptable.

4.3 Concurrency Control in ADO.NET and Visual Studio .NET

ADO.NET and Visual Studio .NET use optimistic concurrency, because the data architecture is based on disconnected data. Therefore, you need to add business logic to resolve issues with optimistic concurrency. How should this business logic be? Clearly it should depend on the number of the times such an exception may occur. And this is were the connection with the data retrieval classes lay on. Using datareaders you only read out data when you need it and concurrency exceptions will occur very seldom. Using one big dataset will reduce the number of connections, increasing hence the
availability of the database server but with the cost of having much more concurrency exceptions, which will create more work for the webserver in handling this exceptions.

4.4 Concurrency Exceptions

Concurrency exceptions in my application can appear quite easily. Take the following as an example. As you already know bookings are done at seat basis. Now, when the page first loads the dataset is filled and represented graphically. Unbooked seats are grey, seats booked by the logged in external booker are orange and all other reserved seats are red. You can better see this in figure 2.1

- older data means that concurrency controls will take place more often
- handling of exceptions is not the same for problems if they arise quite often in contrast that they appear only once
- if they appear often, how should they be handled?

Suppose now that in the drawing of the bus seats 20, 21, 22, 23 are grey and therefore free. I click on them and then I start to fill in the information of my customer, such as name, location, phone, entry point etc. Then I click the save button. In the meantime another external booker reserves seats 22, 23, 24, 25 fills out the information required and presses the save button just one second before I can do this. So there will be no problems for the first user which presses the save button. But what happens in my case? The save mechanism works the following: First it copies the value of the text field where the seats are written and separated by a space character in the database as a string in the row of the customer. In a second step it splits this string and creates a single row for each individual seat and saves it in a table which is called "Sitzezuordnung". This is necessary in order to find the customer and the related seats when you are just clicking an orange seat in the application. This makes it possible to find the seats of this booking and mark them green. So what does really happen? First it will save the string to the customer row. Then it splits the seats and starts to save them row by row in the second table. This works fine for seat, 20, 21, but it will fail for seat 22 which is already reserved. This is checked by the unique constraint I set which assures that there will be only one time the same seat per trip.

The program will give an exception, you close it and restart it. The problem is that the string value is still saved and in fact it is the string value where the information comes from to draw the colours of the seats.

When you as Martin Reisen now log in you see each seat orange and no one red as you have the rights to change everything. You will certainly see
seats 20 to 25 marked orange. Here the problems begin. When you click on seat 20 or 21 you will see seats 20 - 23 marked green, when you click on seat 24 or 25 you will see seats 22 - 25 marked green. But when you click seats 22 or 23 you will also see seats 22 - 25 marked green because as the database passes row per row in the table it will first find this booking and will show this related information. The solution here is to use a transaction object which guarantees that both operations come to an end or that both are undone. But this is not in what I am interested in, also because the solution is not that simple, because the customer must be saved first, in order that I have a number of the booking which I need to couple with the seat and trip number.

What is interesting to me is to cope with the concurrency violation in the program. To do this one has to know how often this will happen. Obviously there is a solution where each exception is handled is the best possible way, but time pressure was high and also exception handling is nothing the customer is willing to pay. So it is a solution to estimate how often which exception will occur and do the slightest acceptable work to cope with this specific exception. The simplest way to cope with the violation is to show a message box where you tell about an internal error and that the user should retry. The page is reloaded, the seats are marked in the right colour and you’re done. If this happens once each 1000 or even more bookings it will work fine, but if this happens more often, users will stop to use the program because they will not lose their time by filling out forms two or even more times.

When will this happen more often? If you use the technique of filling the complete dataset in the beginning you have the lowest database accesses but also the oldest data. After you have logged in, the dataset is filled with all data necessary for the rest of the application, so also the reserved seats are read out quickly in the beginning. This means that your data is the same from the login to the end when you press the save button. One could mention that in the application here there is a difference on just the time by clicking through two additional pages. Well this is true, but my reasoning is intended to be general and moreover you forgot the laziness of the customers. Users always try to avoid work and so what will they do? Imagine that they do their first booking at 9 o’clock in the morning. Then they do not use it the whole day until lets say 6 o’clock in the evening. To avoid to do the login twice the let the site open. This means that when they use at the evening, the data they work with is 9 hours old and therefore not considering the bookings of a whole day. A concurrency violation is therefore highly predictable.

Is there a possibility to escape this dilemma? Well, surely there exists a solution. You can create the dataset each time the page is reloaded, meaning on each post back, therefore whenever you click on a seat. There are many other solutions but their meaning beyond is that you can not use
the initially used dataset. Normally this would not be a problem. You know your application and how it is used and by whom. If you have a small tree structure or an a bit larger flow of pages and only a few users and you have to care of database connections because for example another application is heavingly using the database this implementation may be the best but you still have to cope with the concurrency violation problem. The solution would be something like an advertiser that alerts you if data that is locally saved on your Pc has been changed. But how can the system know when to alert whom? The solution is the application level of a program. You can have variables on application level that will tell you how many instances of the program are open, basically it’s the current count of users in a webpage. You can insert some check in each page and when a new page is loaded the program can check whether your variable has the same of the page, if not, the dataset will be discarded and refilled or just updated.

Let’s see an example for a better understanding: you log in, create the dataset and get dataset variable 1. Variable 1 can be something like a timestamp of the database when it was last updated. User 2 has logged on earlier and saves his booking. After the database is updated, the application variable is updated, say set to the current date time. When user 1 reaches a post back of the site, the variables are compared and if different the dataset is reloaded. The question here is: Must the user be advertised of this process? I think you have to know where he is in the page flow. Assuming that only the number of booked seats have been changed and the user is on the site where it is choosing the date of the trip and therefore cannot see the data that has been changed. Here, an advertising would certainly only annoy the user and would give it no value.

The thing is completely different if the user is already on the main booking site. You click a seat and by reloading the dataset is updated. Now there are two possibilities: - the seats that have changed are not the ones which were already clicked on - one or more of the seats which were clicked on are now marked red In the first case an advertisement must happen because it would look strange for the customer if seats are changing colour while it is on the way of book. In the second case I think the information in the textboxes should be saved between the round trip, but all by me marked buttons should be enabled again and those which are reserved are marked red now. Still you have to explain the problem to the user in order to convince him that the fault was not his and that know he can go on.
Chapter 5

Concluding Remarks

This thesis is meant as a starting point for the basic problems of an application and for a web application in general: performance and concurrency exceptions. Given the huge amount of different requirements, implementation possibilities and underlying hardware structures it was not possible to cope with each one individually. Nevertheless, it seems that the usage of one of the data retrieval classes in a distributed environment such as the world wide web affects the number of concurrency exceptions which will be thrown. This is showed graphically in figure 5.1

Whereas in normal windows programs no one will care whether to use a dataset or a datareader for performance reason, the story is quite different for programming in the web. Clearly everyone prefers to work with datasets given their direct accessibility and the underlying XML schema and maybe in the future where bandwidth between users and servers has grown enough; in Japan they already have 26MBit/s lines, also this barrier will fall. For the moment the better choice is to base its business logic on datareaders, especially for data that is used only once as for example data which is only used to fill a dropdownlist from which the user chooses once or when you read out a huge amount of data and you need data as soon as possible, not only when the whole data is read out.

Basically you can choose to pass work from the webserver to the database server and vice versa. The use of datareaders will cause heavy traffic on the database server because of the huge amount of connections whereas the use of datasets will dramatically decrease the number of connections but leading to more concurrency violations which than have to be handled by the webserver. It is therefore quite important that the programmer is informed of the hardware the program should run on and take it into consideration by planning an optimal solution. It is important here that optimal is meant in a practical meaning, id est that the program satisfies all needs of the customer with the minimal amount of work in the shortest time possible.

Clearly there is no best way in absolute, but it’s worth to reason a bit of
Figure 5.1: My vicious circle
it, even for small scale programs. It depends on your hardware architecture, the line speed, the type of data and therefore the size of data you provide, the data retrieval class used and the number of users which use your system. As already said, take care of performance from the beginning, because ASP.Net, by providing many useful things like the autopostback method seems to be not very fast on its own.
Bibliography


Since most data reside in the data sources, and to achieve high performance, the core of the system is a main-memory DBMS having a storage manager, query optimizer, transactions, client-server interface, disk backup, etc. The AMOS II data manager is optimized for main-memory access and is extensible so that new data types and query operators can be added or implemented in some external programming language. Active database technology uses active rules, called Event-Condition-Action (ECA) rules, as an integral feature of application development. This paper illustrates DeltaBeans in a simple application example and motivates the use of deltas to trigger active rules that specify application integration. View. Show abstract.