Dynamic Server-Based Presentation Logic

Niklas Röstberg
Dynamic Server-Based Presentation Logic

Niklas Röstberg

Examiner:
Lektor Stefan Mankefors

Supervisor:
Dr. Stanislav Belenki

Department of Informatics and Mathematics

Trollhättan, 2003

2003:PM09
Sammanfattning

Bland distribuerade system så är N-lagers arkitekturen den mest använda designen. När denna arkitekturen används så är underhållet av systemet enkelt så länge som det är företags logiken eller data logiken som behöver uppdateras. Om det är presentations logiken som behöver uppdateras så blir underhållet mer problematiskt och tidskrävande eftersom dessa uppdateringar innebär förändringar på varje klient. I denna artikel så undersöks det om det är möjligt att placera delar av presentations logiken på server-sidan i ett system. Focus i projektet är att låta servern hantera utplaceringen av grafiska objekt på clienten. Detta har visat sig vara genomförbart men det är inte möjligt att placera all presentations logik på servern. Det har visat sig att svårigheten med att skapa en dynamisk klient på detta sätt ligger i att hantera händelser som inträffar på de objekt som är utplacerade av servern. Så slutsatsen av projektet är att det är möjligt att använda sig av ”Dynamisk Server-Baserad Presentations Logik” men inte utan vissa sidoeffekter. Sido effekterna av denna typen av design är att applikationen blir långsamt och program-koden tenderar att bli komplex och svår att underhålla.
Förord

Detta arbete har bestått av två delar, den ena delen var att skapa en applikation och den andra delen var att skriva en artikel. Applikationen skapades tillsammans med Andreas Grahn och Marcus Ankar medan artikeln är skriven av enbart en författare. Jag vill härmed tacka dessa personer samt Dr. Stanislav Belenki och Assoc. Prof. Stefan Mankefors för deras hjälp och handledning vid artikelskrivningen.
Innehållsförteckning

Sammanfattning ................................................................................................................ ii

Förord ............................................................................................................................... iii

1 Introduction ................................................................................................................. 5

2 Distributed Software Design ...................................................................................... 6
   2.1 Designing Distributed Applications ...................................................................... 6
   2.2 Distributed Architectures ...................................................................................... 6
   2.3 Distributed Technologies ...................................................................................... 7

3 Dynamic Server-Based Presentation Logic: Goals ...................................................... 8

4 Methodology ................................................................................................................ 8
   4.1 Technical platform ................................................................................................ 9
   4.2 Modeling ............................................................................................................... 9

5 Dynamic Server-Based Presentation Logic: Prototypes ............................................ 10
   5.1 Prototype 1 .......................................................................................................... 10
   5.2 Prototype 2 .......................................................................................................... 10
   5.3 Web Shop Builder ................................................................................................ 14
   5.4 Prototype 3 .......................................................................................................... 14
   5.5 The use of a Dynamic Client Solution ................................................................. 16

6 Discussion .................................................................................................................. 17

7 Conclusion .................................................................................................................. 17

8 Future work ................................................................................................................ 17

9 Acknowledgements .................................................................................................... 17
Dynamic Server-Based Presentation Logic

Niklas Röstberg  
Address  
Hullsjögatan 16 A  
461 42 Trollhättan  
Communicating author: niklas.rostberg@telia.com

Abstract

Among distributed systems today, the N-tier architecture is the most commonly used design. When using this architecture the maintenance of the system is easy as long as it is either the business logic or the data logic that is being updated. But if it is the presentation logic that has to be updated, the maintenance becomes more problematic and time-consuming since each client has to be updated. In this paper, it is examined if parts of the presentation logic can be placed on the server side of the system. The main focus of the project is to place graphical objects on the client from the server. This has proven to be possible, but not all presentation logic can be placed on the server. It is found that the difficult part in creating this dynamic client is to handle events on the remotely placed objects. So the conclusion of this project is that it is possible to use “Dynamic Server-Based Presentation Logic”, but not without some side-effects. The side-effects of this design are that the application becomes slow and that the code has a tendency to get complex and difficult to maintain.

1 Introduction

Today distributed systems are commonly used in a large number of ways. The most common reason for distributing a system is to share the workload between several computers. The technologies that are used to distribute a system have become more and more developed and easier to use.

Distributed programming has gone through some big enhancements during the last years. First came COM/DCOM second came Enterprise Java Beans and now we have the support for remoting in the .NET framework.

A distributed application is often divided into different logical layers. These layers are presentation logic, business logic and data logic. The presentation logic is what the regular user of the application sees as the actual application, and it often resides on the computer where the user is located i.e. “the client”. When this is the case, a change in the presentation logic means that all clients have to be updated.

In this paper, it will be examined if it is possible to change this design.

By locating some of the presentation logic on the server side, hopefully a flexible application can be created that has the possibility to be updated by making changes only on the server, and by that changes give the clients different design and functionality.

It is expected that this kind of application could have some significant advantages:

- The need for upgrades and patches on the client can be reduced, and by that hopefully make the customers more satisfied.
- Customers of the application sometimes discover some errors or miss some functions. Some of these problems can be corrected directly on the server, i.e. the flexibility of the application is enhanced.
- This kind of architecture can open up new ways of selling and getting paid for services. Consider a dynamic client is created and delivered to a number of customers. These customers are given access to a server that holds certain functionality. If another server then is created with another or increased functionality, the customers are able to get access to this server as well, for a fee.
- The possibility of making illegal copies of the application is greatly reduced since the client is useless without the server and the access to the server is limited in a controlled way.
All of the above advantages are only valid if this kind of application can be created in an efficient way.

In this project .Net will be used as framework and C# as programming language. The evolutionary prototyping model will be used to create a number of prototypes. These prototypes are created to gain understanding of the possibilities of distributed programming in C#. Using this knowledge, an application that creates a web-shop will be built to test this experience in a more real life situation. This application will be created together with Andreas Grahn and Marcus Ankar which in their turn will use the application to perform tests in their studies.

Organization of the rest of the paper is as follows; section 2 describes the basics of distributed Software Design. In section 3 the goals of the project is listed. Section 4 is a description of which methods that were used during this project. Section 5 contains the technical information about the results in this project. In section 6 there is a discussion about the results of the project and in section 7 a conclusion is made.

2 Distributed Software Design

In software development today it has become more and more common to distribute the application. Distributing means that the application is divided into a number of components that are separated, either physically or logically.

Designing a distributed application differs in some parts from regular object oriented design. There are some aspects to take into consideration when designing distributed applications.

There are several distributed architectures and technologies that can be used when creating a distributed application.

These issues are described in the following sections.

2.1 Designing Distributed Applications

Most business applications consist of three sets of logic: presentation, business and data source. Barnaby [2] describes these layers as follows:

- Presentation logic – This is the application in the eyes of the user. This is where the end user can enter orders, search in a database etc.
- Business logic – This is the heart of the application, it includes business rules and calculations. This is the most time consuming part for the developers.
- Data source logic – This is where the data is stored.

The recommended way to design a distributed application is to start by separating the application in to these layers. [2]

2.1.1 The five principles of distributed design

Barnaby [2] names five principles to consider when designing a distributed application:

- Principle 1: Distribute sparingly - This principle is based on the fact that invoking a method on an object in a different process is much slower than invoking a method on an in-process object. Thus distributing should only be used when there is more to gain then to loose.
- Principle 2: Localize related concerns - Ensure that components that interact frequently are kept close to avoid overhead in the network and long response time of the application.
- Principle 3: Chunky instead of chatty interfaces - Keep the invocations of methods located in distributed components as low as possible. It is better to have one invocation containing a lot of data then several invocations that hold a small amount of data. This allows for keeping the traffic and the response time as low as possible.
- Principle 4: Stateless rather than stateful objects - Stateful objects can negatively affect scalability for two reasons. First, a stateful object typically exists on the server for an extended period of time. During this time, it can consume server resources and prevent other objects to use these resources. Second, stateful objects minimize the effectiveness of duplicating and load balancing an application across several servers.
- Principle 5: Program to an Interface, not to an Implementation - Interfaces provides easier versioning and a convenient way to deploy type information to a client.

2.2 Distributed Architectures

According to Barnaby [2] there have been three main architectures in the history of distributed programming: Centralized computing, Two-tier Client/Server architecture and N-Tier Client/Server architecture.
There is also a fourth architecture, namely the WEB architecture. [2] These four architectures are described in the following sections.

2.2.1 Centralized computing

In the beginning, applications were built around a large mainframe computer which the user interacted with through a “dumb terminal”. The mainframe provided the terminal with everything, including presentation.

The benefit of centralized computing is: [2]
• Upgrades are made only on the mainframe, which saves time and money in maintenance.
• As soon as an upgrade is applied, all users can benefit from it.

The disadvantages are that: [2]
• All processing burden are placed on one computer.
• The application that runs on the mainframe has a tendency to become very large and this makes it difficult to maintain
• Integration with other applications is very difficult because of the size of the application.

2.2.2 Two-tier Client/Server Architecture

This architecture became popular when hardware prices began to decrease. This meant that the users were provided with personal computers, which were far more powerful than dumb terminals. This made it possible to use graphical user interfaces, which were more intuitive than the text-based interfaces on the dumb terminals. In two-tier systems, the client often executes both the presentation and the business logic, while the server provides access to the data. [2]

The advantages of two-tier architecture are: [2]
• That it is cheap and applications can be built quickly.
• Single point of access for the clients

The disadvantages are: [2]
• Maintenance becomes difficult when both presentation and business logic is placed on the client.
• Updates on the presentation or business logic must be done on potentially great number of clients.
• It is impossible for clients to share resources like database connections.

2.2.3 Three- or N-tier Client/Server architecture

In three or N-tier architecture, the business logic is logically separated from the presentation and data layers. This means that all the layers at minimum are logically separated, but quite often they are also physically separated on different computers.

The advantages of three or N-tier architecture are:
• Deployment and maintenance become easier because updates to the business logic only need to be applied to the server.
• Single point of access for the clients. (Same as in the Two-tier architecture)
• Load balancing becomes easier. [2]

2.2.4 WEB architecture

The WEB has played an important part in the evolution of distributed programming. On the WEB, there are servers that hold information that the clients can display by using browsers. Web servers can host business logic and databases just like the server in the N-tier architecture. Unlike the N-tier server, it also builds the user interface by HTML and scripts. In this way the WEB architecture is very similar to the “Centralized computing” architecture.

The advantages of WEB architecture are:
• Updates on the system often only need to be implemented on the server.
• Browsers make it easy to create simple user interfaces.

The side effects are:
• It is difficult to predict how many concurrent users to expect.
• It is difficult to create really sophisticated user interfaces with scripting and HTML.
• Sophisticated user interfaces are often browser dependant. This means that the platform independency of the system will decrease. [2]

2.3 Distributed Technologies

There are several technologies that make the distributing easier. In the following sections two of the technologies, COM/DCOM and .NET, are briefly presented.

2.3.1 COM/DCOM

During the last decade, the main technology in distributing has been COM/DCOM. COM (Component Object Model) is an object-based programming model that allows two or more software components to interact between different
processes on the same computer. [5]. DCOM (Distributed COM) is an extension of COM that is needed to enable communication between objects located on different computers [7] [8].

COM/DCOM components interact with each other through interfaces. An interface is a collection of functions/methods that provides a type of contract between the components. [5]

The advantages of COM/DCOM are:

- There is no difference in method calls to remote objects and local objects. This gives location transparency.

The disadvantages of COM/DCOM are:

- Something called “dll-hell” which occurs when an old version of the dll-file was to be replaced with a new one. This could often lead to errors in applications using this file.
- It is not platform independent; it primarily works on Windows platforms.

2.3.2 .NET

The .NET Framework provides a remoting framework that allows object or components to interact even if they are located in different assemblies on separate computers [9]. When it comes to distributing .NET has the same goals as COM/DCOM.

The advantages in .NET are: [2]

- .NET is language independent. The components in .NET follow CTS (Common Type System), and this makes component interoperation possible using several different languages. One can use any language that follows CLS (Common Language Specification), and there are about 20 different languages to choose from.
- Code accessing a local object is identical to the code that accesses a remote object. This gives location transparency in the same way as COM/DCOM does.
- .NET provides a new safer way of handling versions. Assemblies are either local or shared. Local assemblies are stored with the rest of the application and shared assemblies are stored in the “Global Assembly Cache”. Every shared assembly must have a “strong name”. A “strong name” includes the name of the assembly, version number, public key and a digital signature. This means that “dll-hell” no longer is a problem.
- The .NET framework is based on XML this makes it platform independent.

3 Dynamic Server-Based Presentation Logic: Goals

By looking at the four architectures in section 2.2 the main goal is to find a way to combine some of them and keep as many advantages as possible. The intention is to use the N-tier architecture (see section 2.2.3) and modify it to become a little more like Centralized Computing (see section 2.2.1). The plan is to keep the business logic and data logic on the server, but move some of the presentation logic from the client to the server, much like in the WEB architecture (see section 2.2.4). By letting the server handle much of the presentation logic, hopefully a more flexible application can be created than if the pure N-tier architecture had been used. The three layers will now seem like four with business logic, data logic and two parts of presentation logic, one part on the client and one part on the server (see figure 3.1).

![Figure 3.1 An illustration on how the different logical layers are located physically on the client and the server. To the right some of the presentation logic is extracted from the client and placed on the server.](image)

The second goal is to come up with a recommendation on what benefits and side effects there are if one decides to move this type of logic to a server instead of letting the client handle it.

To test this design an application is created. This application is named “Web Shop Builder” and is supposed to help inexperienced users to create a customized web shop. The creation of this application is expected to give important feedback on the use of “Dynamic Server-Based Presentation Logic”.

4 Methodology

This project started with a literature study where material written about related topics was investigated. In this study nothing was found about
creating a dynamic client in the way that was intended in this project.

In this project evolutionary prototyping will be used. This means that the development will start with an initial prototype implementing the basic features. In the next steps the initial prototype will be expanded by adding new features to it. When a working design is achieved through the prototypes this knowledge will be used by implementing it in an application called "Web Shop Builder".

4.1 Technical platform

Counting all the advantages of .NET this framework was chosen to be the distributing framework of choice in this project and C# was chosen as programming language.

4.2 Modeling

To describe the prototypes produced in this project UML has been used and the figures are created with the tool "Rational Rose"[10]. In UML there are a number of different diagrams used to describe the design of an application. In this paper only two of these diagrams are being used, namely the Class diagram and the Sequence diagram.

4.2.1 Class Diagram

Class Diagrams are used to describe the classes and relationships between the classes. The name, public attributes and public methods are shown in boxes in this diagram. There are several kinds of relationships that can be described in a class diagram but the relationships used in this paper are realize and implements. (See figure 4.1)

- Realize, is when a class knows or keeps an instance of another class and is described with a dashed arrow.
- Implements, is when one class implements another and by that is forced to implement all methods of this class. This is described with an unfilled arrow with a continuous line.

4.2.2 Sequence Diagram

Sequence diagrams are used to describe the interaction between the classes. In this diagram each involved class is shown together with a timeline to illustrate how long each class is involved in the operation. Between the classes each interaction is shown in form of an arrow and a describing text. There are two types of arrows that are used in a Sequence diagram, first there is the arrow that represents the method call and second there is the arrow that represents the returned value. (See figure 4.2)

- The method call is represented with a regular arrow with a continuous line.
- The returned value or object is represented with a regular arrow with a dashed line.

![Figure 4.1 An example of a Class diagram. The class ServerLogic implements the Interface IRemote and by that also must implement the method GetObject. The class ClientLogic Realizes the Interface IRemote and by that it knows all the public methods in this class.](image)

![Figure 4.2 An example of a Sequence diagram. The Gui class makes a call to the GetObject method located in the Logic class. This method returns an ArrayList, after this the Gui class makes a call to an internal method named Display, along with the call an object is sent.](image)
5 Dynamic Server-Based Presentation Logic: Prototypes

During this project there were a number of prototypes produced. These prototypes were produced to test the possibilities of moving presentation logic away from the client.

The purpose of Prototype 1 was to get a correct syntax to create graphical objects like labels and buttons in one class and display them in another.

In Prototype 2, the graphical objects were created in an assembly placed on one computer and displayed by an assembly located on another computer.

To be able to test the design used in Prototype 2 an application called “Web Shop Builder” was created. This application is supposed to help inexperienced users create a web shop.

The goal of Prototype 3 was to find a way to handle events that took place on objects that have been placed on the client remotely. This was found necessary when the “Web Shop Builder” was created. In this application it was discovered that there should be a dynamic way to handle these kinds of events to be able to take full advantage of the functionality that is tested in this project.

All of the prototypes and the application above are described in the following sections.

5.1 Prototype 1

The goal of Prototype 1 was to find a working way to create graphical objects in one class and get another class to display these objects, and in this way extract some of the presentation logic.

This prototype included two classes, which were located in the same assembly (see figure 5.1):

- The first class Gui was only an empty Form and it inherited from “System.Windows.Form”.
- The second class Logic was the class that held all information about what objects to display.

At start-up, the Gui class invoked a method called GetObject in the Logic class. This method created a number of different objects by calling the appropriate local methods; these methods are shown in figure 5.1. All the objects that were created were stored in an ArrayList and returned to the Gui class. When this ArrayList was received by the Gui class, each object that was stored in it was extracted and displayed on the screen. All the interaction is displayed in figure 5.2

The result of this design is that from the beginning, the Gui is just an empty frame which is waiting for objects to display. What objects to display are decided in the GetObject method which has the possibility to create any number of labels, textboxes, comboBoxes or buttons.

5.2 Prototype 2

The goal of Prototype 2 was to separate the presentation logic further. By placing the logic that was located in the Logic class in Prototype 1 in another assembly, placed on another computer.

This prototype included three assemblies which were named: Client, Server and Middleware. The Client assembly included three classes; Gui, ClientLogic and ObjectCreator. The Server assembly also included three classes which were named; ServerLogic, GuiMaker and ServListener. Finally, the Middleware assembly held one Interface named IRemote. All the assemblies and classes are displayed in figure 5.3.
Finally there was the interface `IRemote`, this interface was implemented by the `ServerLogic` class. In this way the `Client` had an interface to the `Server` to know what methods it had the possibility to use (see figure 5.3).

The functions of all the classes in the prototype were:

- The `Gui` class is almost the same as in the last prototype, with the exception that the actual creation of the objects is now passed to the `ObjectCreator` class.
- The `ServerLogic` class was the class that handled the remoting on the server side.
- The class `ClientLogic` was needed only to handle the remoting on the Client side.
- The class `ServListener` was needed to establish a connection between the two assemblies `Client` and `Server`.
- The class `GuiMaker` was the class that should hold the extracted presentation logic, and it was responsible for creating all the objects that were to be displayed.
- The class `ObjectCreator` class was the one responsible for creating all the different objects that were sent from the `Server`.

Figure 5.3 Classes and their relationships in Prototype 2
As shown in figure 5.4 the Gui makes a call to the method GetObject which is finally located on the Server in the class GuiMaker. This class holds the settings of all the objects that are to be displayed. For each object that is to be displayed a call to a method is made. If for example a label is to be displayed the method CreateLabel is invoked (see figure 5.4). When this method is called, a number of values are sent along with the call. These values are then stored in an ArrayList which represents the actual object.

```java
private ArrayList CreateLabel(int x, int y, int w, int h, Color c, string text, string name)
{
    ArrayList aList=new ArrayList();
    string s = "Label";
    aList.Add(s);
    aList.Add(x);
    aList.Add(y);
    aList.Add(w);
    aList.Add(h);
    aList.Add(c);
    aList.Add(text);
    aList.Add(name);
    return aList;
}
```

Figure 5.4 The values that is sent to the CreateLabel method are stored in an ArrayList. In the first position in the ArrayList is a string which identifies the type of the stored object.

The ArrayList that is created in the above example is returned to the GetObject method. In this method each of these ArrayLists are stored in another ArrayList (see figure 5.5).
Figure 5.5 Content of the ArrayList which holds all the objects that are to be displayed on the client. This ArrayList is created at the Server and sent to the Client

As shown in the above figure this ArrayList now holds a number of ArrayLists that represents all the objects that are to be displayed at the Client. When all these objects are stored in the ArrayList it is returned to the Client. In this way the number of interactions between the Client and the Server can be reduced. This follows the third principle Barnaby [2] recommends when designing distributed systems (see section 2.1.1)

When this ArrayList has been returned to the Gui, each ArrayList that represents the objects is extracted. Depending on what type of object that is represented, these ArrayLists are sent to different methods that are located in the ObjectCreator class (as shown in figure 5.3). These methods use all the information in the ArrayLists to create the objects (see figure 5.6). When each object is created it is returned to the Gui class where it is displayed.

Figure 5.6 A label is created from the information that is sent from the server. After the creation the label is returned to the Gui class where it is displayed.

This prototype shows that it is possible to move presentation logic away from the client. Using this design makes it possible to add, change or remove objects to display on the client, just by applying these changes on the server.

The way that the objects are sent and created in this prototype is not the most natural way of doing this. The most natural way would have been to create an instance of the object on the server and send it to the client. This is unfortunately not possible since graphical objects like labels and buttons are members of the "System.Windows.Form" class, and this class derives from "MarshalByRefObj". This makes these objects nonserializable, which means that they can not be sent to the client as a local copy. [4] Since they cannot be sent as a local copy they are sent as a reference, and this means that the client only receives a reference/pointer to the object that still resides at the server.

```java
public Label MakeLabel(ArrayList a) {
    Label l = new Label();
    int x = (int)a[1];
    int y = (int)a[2];
    l.Location = new System.Drawing.Point(x, y);
    int w = (int)a[3];
    int h = (int)a[4];
    l.Size = new System.Drawing.Size(w, h);
    Color color = (Color)a[5];
    l.BackColor = color;
    string text = (string)a[6];
    l.Text = text;
    string name = (string)a[7];
    l.Name = name;
    return l;
}
```
Dynamic Server-Based Presentation Logic

5.3 Web Shop Builder

To be able to test the experience that was gained from the prototypes, an application was created. This application is called; “Web Shop Builder”, and is supposed to help inexperienced users to create a web shop. The “Web Shop Builder” was implemented as a client/server solution as shown in figure 5.7.

![Diagram of client/server solution](image)

Figure 5.7 General view of the application (Web shop builder)

At start-up the client connects to the server and the server sends information about the graphical objects which are to be displayed. After this the user can enter information and make selections. All this information is then sent to the server. The server then copies a predefined XML-file and a predefined database into a personal directory created for the user and stores the information sent into either the XML-file or the database. In the XML-file all the settings that are needed to set the layout are stored. In the database all the products that are to be sold in the web shop are stored.

Based on the selections the user have made the server then sends back more information about what objects the client should display. This procedure continues until the user has entered all the information needed to create the web shop.

After this the web shop, which consists of a number of ASP pages, can use the information stored in the XML-file and the database to display a fully customized web shop.

When this application was created it was found that when objects like buttons, which generally are used to trigger some kind of event, were added, it often required an update of the Client. This problem is addressed in Prototype 3 and is not included in this application.

5.4 Prototype 3

Displaying the object is not the only task that has to be accomplished to create a dynamic client. There are objects that are used to interact with the user and these interactions need to be taken care of. Prototype 3 was created to find a way to handle these interactions dynamically. This prototype extends Prototype 2 and the classes in this prototype were the same as in Prototype 2, which are shown in figure 5.2.

5.4.1 Problems with events on interactive objects

Some objects are used to interact with the user. Because of this it is difficult to add an interactive object in a dynamic way. The main reason for this is that the dynamically placed object is not known to the client when the client is created. This leads to a number of problems:

- The client does not know what action to take when an object is activated (i.e. a button is clicked)
- The object can not be directly accessed in the code on the client.

To address these problems the following actions are taken:

- Implement a general eventListener for each type of interactive object. For example; an eventListener that is connected to every Button placed on the Client.
- Every interactive object created and placed on the client must be added to an array to make it possible to get access to every single object on the client by using the index in the array. Each object must be placed in its own array, for example buttons in a ButtonArray.

When these preparations have been made it is now possible to begin to handle the event. The first step when a button is pressed is to convert the “object sender” in the ButtonClick method into a button object to get access to all of its properties and see which button was pressed (see Figure 5.8).
In the example above the object sender is converted into a button and the name of the button is accessed. This name, which at the creation is set to an integer on the server, is then stored in the variable “buttonIndex”. This variable can then be used as an index in an Array of any kind. The name and usage of the variable are decided by the developer depending on the function of the button.

5.4.2 Example of solution

This solution is shown in the Sequence Diagram in Figure 5.9. The solution is just an example of how handling of an event can be designed. In this example a button is clicked on the client and the wanted action for this button is that information in a number of TextBoxes is to be collected and sent to the server. The numbered events in the figure are described later in the text.

A general eventListener is connected to every button that is placed on the client. This eventListener method is named ButtonClick in this prototype. First, the sender is converted into a button and the button’s name is stored in a variable, similar to the example in figure 5.8. In the next step a call (1) (2) (3) is made to the method called ButtonClick which is placed on the server. When this method is called, the name of the pressed button is passed on with the method call.

In the ButtonClick method the name of the button is used to determine what actions to take, in this case an ArrayList is created (4) that in its first element hold information about the action to be taken by the client. The rest of the elements it holds are the names of several TextBox objects that the server wants to know the content of. After this the ArrayList is returned (5) (6) (7) to the client that makes a call to a method. The method that is called depends on the name stored in the first element of the ArrayList that was returned from the Server. In this example the GetText method (8) is called. This method uses the rest of the ArrayList to get the text in specific TextBoxes. It knows which TextBoxes to read from since the names of them are stored in

```csharp
private void ButtonClick(object sender, System.EventArgs e)
{
    Button colorButton = (Button) sender;
    int buttonIndex = Int32.Parse(colorButton.Name);
}
```

Figure 5.9 Flow of events when a remotely placed button is pressed. The name of the button is used by the Server to take the corresponding actions. In this case an ArrayList is created holding information on what kind of action that is to performed on the Client and what objects this action should include, in this case a number of TextBoxes. This ArrayList is then used by the Client to see what action to take. In this example it reads the information held in the textboxes and sends the information in form of an ArrayList to the server.
the ArrayList. After this the content of each textbox is sent to the server in another ArrayList (9).

When this design is used, some new objects like buttons can be added without having to update the Client. This depends, however, on the fact that the type of event performed by the button is predefined and coded when the client is created.

5.5 The use of a Dynamic Client Solution

During the project it was found that it was hard to take full advantage of the separate architectures and that there are some benefits that can be gained, but also some side effects that arise when using this type of design.

5.5.1 Benefits

To be able to change the appearance and function on the client without actually changing client can be useful in many cases.

• The upgrade only needs to be implemented in one place instead of updating numerous clients. This means that maintenance costs of an application can be reduced.
• If the customers don’t need to download and install the update themselves, this can lead to more satisfied customers.
• Changes due to suggestions from the customers can be implemented faster which can encourage them to give more feedback in the future [6].

Using a client/server solution makes it possible to give the customers a client for free and let them pay for getting access to the server. If the same client has different appearance and function depending on how the server is configured, it is possible to use the same client for different kinds of services, just by giving them access to different servers.

This design makes it hard to illegally use the application. The only part of the application that is accessible to the users is the client. Since the client is more or less like an empty shell it is useless without the connection to the server.

5.5.2 Disadvantages

It was found that different objects were more or less suitable to be placed remotely on the client. Objects like labels and pictures which are just displayed on the screen are quite simple to distribute, while active objects like Buttons and ComboBoxes are really hard to add without changing the code on the client. It is possible to add even these kinds of objects, but if an event is to be connected to the object, the client must be prepared in advance to handle the object and the event. This problem is described more in detail in section 5.4.

The client has to be prepared to handle different kind of objects and events and there are many method calls between the client and the server. There are also several steps in different classes to create the objects and handle the events. These factors make the code in the application more complex and difficult to maintain.

The clients have to make calls to the server, which returns the objects that the client displays. This traffic is additional to the normal traffic in a client server application. Due to this traffic, the application becomes slower as compared with the traditional case.
6 Discussion

The first and the most important goal of this project was to find out if it was possible to use the N-tier architecture (see section 2.2.3) in a new way. In particular, one of the objectives was to try to make the changes in appearance and function of the application easier and less time-consuming.

This goal was met by looking at the design in the Centralized Computing architecture (see section 2.2.1), where the presentation logic is located at the server. This project has shown that it is possible to modify the N-tier architecture in such a way that the presentation logic is controlled by the server.

When an application is designed in this way modifying appearance and functions becomes easier since the changes often involve only updates on the server. In this way there is no need to update every single client, which in some cases may be several hundreds. On the other hand the maintenance of the application as a whole becomes more difficult since the code has a tendency to become very complex.

Other positive features of this design are the reduced risk of illegal copying of the application and that there is an opportunity to use the same client for different purposes.

The solution presented in this paper involves a lot of traffic between the client and the server. This makes this kind of application unsuitable for time-critical systems. It is also important to have quite high bandwidth on the connection between the client and the server, because otherwise the customers might find the application to slow.

Based on the benefits and side effects in this design it is really hard to make any real recommendation. One recommendation is that this design should not be used without considering the side effects.

7 Conclusion

This paper shows that it is possible to place parts of the presentation logic on a server. It is also stated that this design of the architecture is not fully supported by the .NET framework since graphical objects like buttons and labels can not be sent between two computers. Sending information about each object in a structured way from the server to the client solved this problem.

It is also stated in this paper that handling events that take place on remotely placed objects is difficult. All events that can occur on these objects must be prepared in advance, otherwise one have to update the client to take care of these unpredicted events.

By implementing “Dynamic Server-Based Presentation Logic”, the application gets slightly more flexible and adjustable. Even other positive features arise like improved copy protection and new interesting ways to charge the customer for the services of the application.

There are also some consequences one has to consider when this design is used: The code gets complex and difficult to maintain. The application gets slow since many calls have to be maid between the client and the server.

The conclusion of this project, after considering all these facts, must be that it is possible to use “Dynamic Server-Based presentation Logic”, but it is not recommended. There are certainly some applications that can benefit from this design as it looks now, but general applications probably would not.

8 Future work

The .NET framework as it looks now does not support the sending of “Windows.Form” objects by remoting. This is because these objects can not be serialized to be marshalled by value. Since the .Net framework is extensible, it could be interesting to try to find a way to make the serialization of these objects possible. This would also give possibility for easier handling of events that take place on remotely placed objects.

In the design proposed in this paper, the server does not really know anything about the client. This makes the handling of events very difficult and not so flexible, as shown in section 5.4. If the server in some way could keep an instance of the client it would have been much easier to handle many of the events.

9 Acknowledgements

The author wishes to thank the examiner of the project Dr. Stanislav Belenki for his help with this paper. Also, I would like to thank the two students that took part of the development of the application, Andreas Grahn and Marcus Ankar.
8. References


